16 *Quantum Computing*

16.1 What is a Quantum Computer

Quantum mechanics predicts that it should be possible to build a computer that can do certain calculations much faster than would be possible on a conventional (classical) computer. Aspects of quantum mechanics may seem nonintuitive, but all evidence supports it. In this chapter we describe how a quantum computer differs from a classical computer, and give intuitive descriptions of the quantum algorithms most relevant to cryptography.

There are entire books about quantum mechanics. Our goal isn't to pack years of physics and math into a few pages, but to give some intuition into the concepts, terminology, and notation, and the algorithms that run on quantum computers. And for readers who want to study the topic more deeply, hopefully this introduction will make it easier to understand the formal literature and books.

When I_2 challenged me_4 to write something "really short" about what a quantum computer is, I_4 wrote "a quantum computer is a magic box that factors numbers quickly". We want to go into more detail than that, but keep the chapter reasonably intuitive and short.

16.1.1 A preview of the conclusions

First we'll hint at the conclusions, and then we'll explain why they are true.

A common misconception is that a quantum computer is faster than a classical computer, and because Moore's law is slowing down, eventually all computers will be replaced by quantum computers. This is definitely not true. There is only a narrow set of problems for which a quantum computer would be faster. The property of a quantum computer that makes it excel at some tasks is that it can compute, with only storage size *n*, as if it were operating on 2^n values in parallel. However, as we'll see later in this chapter, it also has serious limitations, such as if you read the state you will see only one value and the others disappear. The magic of a typical quantum program is for the quantum computation to raise the likelihood that when you finally read a result, it will be a useful value.

Another common misconception is that a quantum computer can solve NP-hard problems (such as the traveling salesman problem) in polynomial time. This is almost certainly not true. Although nobody has proven that it is impossible, no known quantum algorithm for solving such problems is that powerful. Indeed, a quantum algorithm that powerful would be nearly as surprising, given our current state of knowledge, as a classical algorithm that was that powerful.

Although a quantum computer can, in principle, do any calculation that a classical computer can do, for most calculations quantum computers would be no faster than conventional computers, and in practice, quantum computers are likely to be much more expensive to build and operate. For example, most designs would need to operate at temperatures very close to absolute zero. So, it is not likely that quantum computers will ever serve more than a small niche market doing extremely CPU intensive calculations on tiny amounts of data. However, there are two important quantum algorithms that are relevant to cryptography.

- Grover's algorithm, which makes brute force search faster. This might seem worrisome for things like encryption or hashes, where a brute force search can find a key or a pre-image, but the limit of Grover's algorithm is that it only reduces the search time to the square root of the size of the search space. Squaring the size of the space being searched (for instance, using an encryption key twice as long) is more than adequate to protect against Grover's algorithm.
- Shor's algorithm, which can efficiently factor numbers and calculate discrete logs. Shor's algorithm, run on a sufficiently large quantum computer, would break all of our widely used public key algorithms (e.g., RSA, Diffie-Hellman, elliptic curve cryptography, ElGamal). At the time of this writing, no quantum computer sufficiently large to break the currently deployed public keys has ever been publicly demonstrated, and some experts remain skeptical that one ever will be built. There are a number of difficult engineering challenges that remain, and it may never be economically viable to overcome them. But because such a computer might be possible, it is important to convert to quantum-safe public key algorithms well before a quantum computer (of sufficient size) might exist. The cryptographic community is actively developing and standardizing such algorithms, which we describe in chapter XXX.

16.1.2 First, what is a classical computer?

Classical computers compute on information stored in bits. Each bit stores either a 0 or a 1. A classical computer with n bits can be in one of 2^n states. For instance, with 3 bits, the possible states are 000, 001, 010, 011, 100, 101, 110, 111. A classical computer operates on bits using "gates" (such as AND or NOT), which take some number of bits as input, and output some number of bits. A classical gate can be described with a table that indicates, given the values of the input bit(s), what the value of the output bit(s) are.

For instance, the classical AND gate's table is:

Input	Output
00	0
01	0
10	0
11	1

In quantum computation, there is an intuitively similar notion of a gate. However, whereas a classical gate has inputs and outputs, a quantum gate operates on a collection of qubits and changes the state of those qubits. We'll describe more about that later.

16.1.3 Qubits and superposition

Instead of bits, a quantum computer uses *qubits*, where a qubit's state can be a mixture of a 0 and a 1. This is known as **superposition**. The standard notation for describing superposition is known as *ket* notation. A ket is a symbol or value written between a vertical bar and a right angle bracket representing a state. The state of one or more qubits is usually denoted by $|\psi\rangle$. The standard notation for a single qubit's state is $|\psi\rangle = \alpha|0\rangle+\beta|1\rangle$. The coefficients ($\alpha$ and β) determine how likely the measured value will be 0 vs 1. Measuring a qubit (reading it) destroys the superposition information, and the qubit takes on the value read—either completely 0 (written as $1|0\rangle+0|1\rangle$, or simply $|0\rangle$) or completely 1 (written as $0|0\rangle+1|1\rangle$, or simply $|1\rangle$).

The standard notation for the state of two qubits is $|\psi\rangle = \alpha|00\rangle+\beta|01\rangle+\gamma|10\rangle+\delta|11\rangle$. The coefficients ($\alpha$, β, γ, δ) determine how likely the measured value will be 00, 01,10, or 11.

Once measured, a qubit is no different from a classical bit—it's either 0 or 1. How does a qubit become a mixture of 0 and 1? A quantum computer can compute with gates that create superposition, as we will show in §16.1.4.

The coefficients (α and β) of the qubit state are called **probability amplitudes**, and their squared absolute values are probabilities. Given that the total probability must be 1, the sum $|\alpha|^2+|\beta|^2 = 1$. The probability that a qubit in state $\alpha|0\rangle+\beta|1\rangle$ would be read as 0 is $|\alpha|^2$, and the probability that it would be read as 1 is $|\beta|^2$. For example, if $\alpha = \frac{1}{\sqrt{2}}$ and $\beta = -\frac{1}{\sqrt{2}}$, then the probability of the state $\alpha|0\rangle + \beta|1\rangle$ being read as 0 is ½, and the probability of it being read as 1 is also ½. In most literature the coefficients are referred to as amplitudes.[1]

The coefficients are actually complex numbers. But, for ease of drawing a figure, and because until we explain Shor's algorithm we only need to use real coefficients, assume for now that the coefficients are real numbers. Given that the probabilities $|\alpha|^2+|\beta|^2$ need to add up to 1, if you were to graph all the potential (real) values of ⟨α,β⟩, you'd wind up with a circle of radius 1.

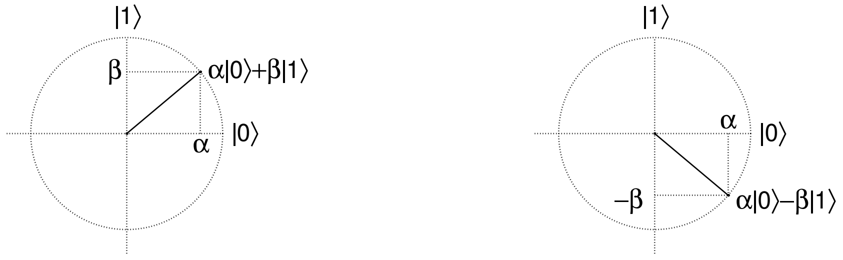

Figure 16-1. Real Coefficients

(see Figure 16-1). Note that if you change the sign of α or β, the probability of reading 0 or 1 is not changed. For instance, in the left half of Figure 16-1, we show α|0⟩+β|1⟩. And in the right half of Figure 16-1, we show α|0⟩−β|1⟩. In both cases, the probability that the qubit would be measured as 0 is $|\alpha|^2$, and the probability that the qubit would be measured as 1 is $|\beta|^2$.

If two coefficients are different, but they have the same absolute value, the two coefficients are said to have different **phases**. If the coefficient is a real number, the phase is just a choice of plus or minus. For complex numbers there is a continuum of possible phases. (If we plot a complex coefficient $x+yi$ as a vector in two dimensions, its phase is the angle the vector makes with the real axis, and its absolute value is the vector's length. See Figure 16-2 and note that a coefficient can have absolute value no more than 1 and so can only exist within the radius-1 outer circle; any coefficient on the radius-m inner circle has absolute value m.)

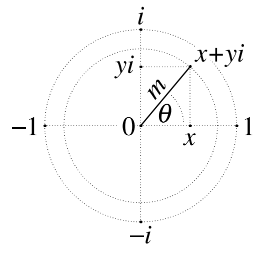

Figure 16-2. A complex number $x+yi$ with absolute value (*aka* magnitude) m and phase θ

Why should we care about phase of a coefficient if it doesn't change the probability of that value being read? The answer is that in useful quantum computations the state is repeatedly altered by transformations called **quantum gates** before the state is measured, and the probability of reading a certain value after the gates are applied generally depends on what the phases of the coeffi-

1. For a while, we'll use the term *coefficient* rather than *amplitude* because α in the expression α|0⟩ describes both the magnitude and the phase (see Figure 16-2), and the words *amplitude* and *magnitude* are often used as synonyms in nontechnical English.

cients were before the gates were applied. We will hear more about quantum gates later in the chapter.

16.1.3.1 Example of a Qubit

A photon behaves as a qubit, where the polarization of the photon can be thought of as its quantum state. It can be polarized to be up/down (which we'll interpret as 1), right/left (which we'll interpret as 0), or anything in between. If a polarizing filter is exactly aligned with the photon, the photon will definitely pass through the filter. If the filter is 90° off from the photon's polarization, the photon will definitely not pass through. If the filter is 45° off from the polarization of the photon, the photon will pass through with probability ½. More generally, if we set the x axis to be aligned with the polarizer, and the photon is polarized along a line through the origin that crosses the unit circle at $\langle\alpha, \beta\rangle$, then the probability that the photon passes through the filter is $|\alpha|^2$.

What's happening is that the polarizing filter is measuring the photon. The values 0 and 1 are relative to the alignment of the filter. If the photon passes through, it means it has been measured as 0 relative to the polarizer. And since measurement destroys the 1 component of the photon, the photon will now be exactly aligned with the filter.

This behavior can easily be demonstrated in real life with three polarizing filters. If you put two filters aligned 90° from each other on top of each other, no light passes through (none of the photons get through), but if you insert the third filter between those two, and align it 45° from the others, light will pass through the combination of the three filters. (½ will pass through the first filter, then of those, ½ will pass through the second filter, and then of those, ½ will pass through the third filter). (See Homework Problem 1.)

Note that what you define as 0 and 1 is arbitrary, as long as what is defined as 0 is orthogonal to what is defined as 1.

16.1.3.2 Multi-qubit states and entanglement

Another strange concept in quantum computers is *entanglement*; where a collection of qubits has a state that can be described collectively, but cannot be described by specifying the states of the individual qubits. For instance, three qubits can be in a superposition of all 8 possible classical states: 000, 001, 010, 011, 100, 101, 110, 111. To express the state $|\psi\rangle$ of the set of three qubits, the notation would be

$$|\psi\rangle = \alpha|000\rangle+\beta|001\rangle+\gamma|010\rangle+\delta|011\rangle+\varepsilon|100\rangle+\zeta|101\rangle+\eta|110\rangle+\theta|111\rangle.$$

The probability is 1 that the set will be in one of those 8 states. So,

$$|\alpha|^2+|\beta|^2+|\gamma|^2+|\delta|^2+|\varepsilon|^2+|\zeta|^2+|\eta|^2+|\theta|^2 = 1.$$

And the probability that the set of three qubits would be measured as, say, 011, is $|\delta|^2$.

It requires 2^n complex numbers (coefficients) to express the state of n entangled qubits. However, if the qubits are not entangled, i.e., are independent of each other, their state can be expressed more compactly, with only $2n$ coefficients. The state of the first qubit can be expressed as $\alpha_1|0\rangle+\beta_1|1\rangle$. The state of the second qubit can be expressed as $\alpha_2|0\rangle+\beta_2|1\rangle$. The state of the third qubit can be expressed as $\alpha_3|0\rangle+\beta_3|1\rangle$, and so on. So, for example, suppose there are ten entangled qubits. It requires 1024 coefficients to specify the state of ten entangled qubits (a coefficient for each of the 2^{10} possible values of ten bits) but if the qubits are unentangled, the state of each of the ten qubits can be expressed with two coefficients, requiring only twenty coefficients.

While the compact notation using $2n$ coefficients cannot be used to describe an entangled state, the less compact notation using 2^n coefficients can be used to describe unentangled states. In an unentangled state, you can derive the coefficients of all 2^n classical states from the coefficients of $|0\rangle$ and $|1\rangle$ in each of the n unentangled qubit states. For instance, with three unentangled qubits in states $\alpha_1|0\rangle+\beta_1|1\rangle$, $\alpha_2|0\rangle+\beta_2|1\rangle$, $\alpha_3|0\rangle+\beta_3|1\rangle$, respectively, the coefficient of state $|000\rangle$ in the collective state of three qubits would be $\alpha_1\alpha_2\alpha_3$. Likewise the coefficient of $|001\rangle$ in the collective state would be $\alpha_1\alpha_2\beta_3$, the coefficient of $|010\rangle$ would be $\alpha_1\beta_2\alpha_3$, and so on. (See Homework Problem 2).

16.1.4 Becoming superposed and entangled

Entanglement and superposition are what can make a quantum computer more powerful than a classical computer. How do qubits become superposed and entangled? You'd likely start a computation on a quantum computer by initializing the qubits to a known state, which would be done by measuring all the qubits. Now none of them are entangled, and they are all firmly 0s or 1s.

There are various operations (gates) one can apply to qubits that will cause superposition and/or entanglement. For instance, superposition can be created on a single qubit using a quantum gate known as a Hadamard gate, which takes a qubit that is solidly zero or solidly one, and sets it to be an equal mixture of 0 and 1. The Hadamard gate operation is given by

initial state	final state
$\|0\rangle$	$\frac{1}{\sqrt{2}}\|0\rangle+\frac{1}{\sqrt{2}}\|1\rangle$
$\|1\rangle$	$\frac{1}{\sqrt{2}}\|0\rangle-\frac{1}{\sqrt{2}}\|1\rangle$

truth table representation

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

matrix representation

Gates that operate on a set of qubits can entangle the set. For instance, if you have unentangled qubits x, y, and z, and operate on x and y, then x and y may become entangled. If you then operate on y and z, then all three qubits (x, y, and z) may now be entangled. A gate might also unentangle previously entangled qubits.

States and Gates as Vectors and Matrices

The quantum state of a collection of qubits can be represented as a column vector of coefficients, one for each bit combination. By convention, we'll order the coefficients according to the corresponding bit combinations, so, for one qubit the order is just 0,1; for two qubits it's 00,01,10,11, and so on. You can think of this as a shorthand to avoid writing out the bit combinations every time. For example, we can write $\alpha|0\rangle+\beta|1\rangle$ as

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

A quantum gate can be represented as a matrix of multipliers, with each column corresponding to an initial state and each row corresponding to a final state. So, for a single-qubit gate, the first column gives the coefficients of final states $|0\rangle$ and $|1\rangle$ when the initial state is $|0\rangle$, while the second column gives the final state coefficients when the initial state is $|1\rangle$. Again, this is a shorthand to avoid writing out the bit combinations. When we describe gates, we'll show their matrix representations as well. For example, we can write the NOT gate as

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

The beauty of this representation is that the application of a gate to a state is accomplished by multiplying the gate matrix times the state vector. For instance, to apply the NOT gate to a qubit in state $\alpha|0\rangle+\beta|1\rangle$, you'd multiply the NOT matrix by the column vector representing the qubit state.

The result of applying a sequence of gates is given by the product of the gate matrices in right-to-left order.

The truth table representation shows what the Hadamard gate does to a qubit initialized to 0 or 1. If the qubit started out with a superposed state, for instance, $\alpha|0\rangle+\beta|1\rangle$, then to calculate the output of the Hadamard on $\alpha|0\rangle+\beta|1\rangle$, you add the outputs of $|0\rangle$ and $|1\rangle$ in proportion to the coefficients of the input state. Since the coefficient of $|0\rangle$ in the input is α, and the output of $|0\rangle$ is $\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle$, you multiply the output of $|0\rangle$ by α to obtain $\frac{\alpha}{\sqrt{2}}|0\rangle+\frac{\alpha}{\sqrt{2}}|1\rangle$. Likewise, you multiply the output of $|1\rangle$ (which is $\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$) by β.

Adding the two results, the Hadamard gate's output on a qubit with state $\alpha|0\rangle + \beta|1\rangle$, is $\frac{\alpha+\beta}{\sqrt{2}}|0\rangle+\frac{\alpha-\beta}{\sqrt{2}}|1\rangle$.

This can also be calculated by multiplying the matrix representation of the Hadamard gate by the column vector representing the qubit state $\alpha|0\rangle+\beta|1\rangle$.

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

The reason we can multiply the output of a state by the coefficient of that superposed state in the input is due to one of the properties of quantum gates known as *linearity*, described in the next section.

Now, for something really fascinating, what happens when you apply a Hadamard gate twice? Hadamard is its own inverse! (See Homework Problem 7.)

16.1.5 Linearity

Quantum gates satisfy the property of **linearity**. The intuition for linearity is that if you know how a gate will operate on various input states, and the state is a superposition of those input states, you can multiply the output of each input state by the coefficient of the input state, and add all the weighted outputs to get the quantum state that is the result.

Linearity is nice, because it means that we can describe n-qubit quantum gates by tables that only describe the outputs produced when the input is one of the 2^n classical states. The table is sufficient to determine what the gate will do to any input state, because all the possible quantum states can be expressed as a superposition (linear combination) of the 2^n classical states. For example, there is a 2-qubit gate known as **CNOT** (*controlled not*), which flips the second qubit between 0 and 1 if the first qubit is 1. It can be defined by how it acts on $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$.

initial state	final state
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|11\rangle$
$\|11\rangle$	$\|10\rangle$

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

Then if the input state of two qubits is $\alpha|00\rangle+\beta|01\rangle+\gamma|10\rangle+\delta|11\rangle$, linearity implies that the output of a CNOT gate on those two qubits is $\alpha|00\rangle+\beta|01\rangle+\delta|10\rangle+\gamma|11\rangle$.

16.1.5.1 No Cloning Theorem

An interesting consequence of linearity is that it is impossible to clone a qubit (or set of entangled qubits). If you had a qubit in state $\alpha|0\rangle+\beta|1\rangle$, you might think you could XOR it into a qubit in state $|0\rangle$, and wind up with two qubits, each in state $\alpha|0\rangle+\beta|1\rangle$. But instead you'd wind up with two entangled qubits, in state $\alpha|00\rangle+\beta|11\rangle$. (See Homework Problem 8.)

An important implication of the no cloning theorem is that you can't read a quantum state more than once. If you could clone the state, you could make lots of copies, and by reading them, derive the probabilities of the various superposed states. But you can't. Reading the quantum state destroys the superposition, and because of no cloning, you can only read it once.

16.1.6 Operating on Entangled Qubits

Quantum gates seldom operate on more than one or two qubits, because the engineering challenge of building a quantum gate increases wildly with the number of qubits it is acting upon. One could imagine a quantum gate that operates on *n* qubits, but in practice it would almost certainly be built out of gates that operate on one or two qubits at a time. The running time of a quantum algorithm employing a virtual *n*-qubit gate is adjusted to account for the actual gate configuration. So, given that we will only be using 1- or 2-qubit gates, what happens if you operate on a subset of entangled qubits?

For instance, suppose there were three entangled qubits with state $\alpha|000\rangle+\beta|011\rangle+\gamma|100\rangle$. Now suppose we use a NOT gate on the first qubit. The result would be $\alpha|100\rangle+\beta|111\rangle+\gamma|000\rangle$. See Homework Problem 5 for a more complicated example.

Measurement typically happens one qubit at a time. Suppose you measure one qubit of an entangled set of qubits and get the value 1? A side effect of the operation of measuring that qubit is that the coefficients of all of the states that don't match the measured value go to 0 and the coefficients of all the states that do match the measured value are increased linearly to make the sum of their squared absolute values equal to 1. Note that this linear scaling does not change the phases of the remaining coefficients. (See Homework Problem 6.)

16.1.7 Unitarity

Linearity (§16.1.5) constrains the possible things a quantum gate can do, for example, by preventing cloning qubits. Another constraint on quantum gates is that they must be **unitary**. Unitarity is the property of a linear gate that says, if the input state of the gate is normalized (i.e., the sum of the squared absolute values of the coefficients is 1), then the output state is normalized. These constraints are not arbitrarily imposed by us, or by some committee. Instead, they result from the known laws of physics.

Unitarity seems like a sensible property. It seems rather absurd to imagine that the probability of all measurement outcomes could add to something other than 1, after you do a physically possible operation. Nonetheless, there are some operations that are forbidden by this constraint that seem like they should be legal operations. For example, suppose we tried to make a linear gate out of the operation that sets a qubit to 0. We will call this the "zeroize" gate and its truth table will be

Input	Output
0	0
1	0

Such a gate would violate unitarity since, when we apply the rule for linearity to the above truth table, we find the gate would take, for example, the normalized state, $\frac{3}{5}|0\rangle+\frac{4}{5}|1\rangle$, to the unnormalized state, $\frac{3}{5}|0\rangle+\frac{4}{5}|0\rangle = \frac{7}{5}|0\rangle$. (Yes, $\frac{3}{5}|0\rangle+\frac{4}{5}|1\rangle$ is normalized because $3^2+4^2 = 5^2$.)

Classical operations such as the zeroize gate that can't be straightforwardly turned into a unitary gate are called *irreversible* operations and what they have in common is that they take at least two different inputs to the same output. In contrast, *reversible* classical operations, those that take every possible input to a different output, can be straightforwardly turned into valid, unitary quantum gates. In the next two sections, we will show how to implement even irreversible classical operations with quantum components, first through measurement, and next, through converting the irreversible classical operation into a reversible classical operation.

16.1.8 Doing irreversible operations by measurement

One way to perform classical operations that are irreversible (from quantum building blocks) is to use measurement. For instance, the zeroization operation can be applied to a qubit, as follows:

1. Measure the qubit
2. If you measure a 1, apply the NOT gate: $|0\rangle\to|1\rangle$, $|1\rangle\to|0\rangle$. If you measure a 0 do nothing.

However, this way of implementing classical operations has a major drawback. When you measure a qubit in a superposition of multiple classical states, you only get information about one of the classical states in that superposition. Moreover, if the qubit you measured was entangled with some other qubits, it won't be entangled with them after you make the measurement. Since superposition and entanglement are necessary for quantum computing to be more powerful than classical computing, it would be unfortunate if measurement was the only way to do certain classical computations you wanted to use as part of a quantum algorithm.

16.1.9 Making irreversible classical operations reversible

When we want to incorporate an irreversible classical computation into a quantum algorithm, it is typical to add some extra qubits to the operation in order to make it reversible (and therefore unitary).

Suppose you wanted to implement an irreversible function that, in the classical world, used i input bits and o output bits. One way that this can be made into a reversible function is by modifying the function so that it takes $i+o$ input bits, computes the irreversible function on the first i bits and XORs the result into the last o bits. If the o bits are initialized to 0, the computation is the same, and the new function is clearly reversible since if you execute it twice you get the original input.

In a quantum version of this, if the *i* input qubits are in a superposition and the *o* output qubits are 0s, then after the function is executed, the *i+o* qubits will be in a superposition of states, with coefficients corresponding to those of the original superposition of the *i* input qubits. For each classical state in the superposition, the *i* input qubits will hold a classical input, and the *o* output qubits will hold the result of the computation for that input. There are known algorithms for converting any computable classical function into a similarly efficient quantum circuit that acts in the way we just described.

16.1.10 Universal Gate Sets

The terms circuit and gate used in quantum computing were chosen to sound familiar to people used to conventional computers. In principle, all classical operations could be built out of NAND gates, and a circuit would carry out a more complex calculation (such as adding together two 32-bit quantities) using a large number of physical gates wired together.

Similarly, in a quantum computer, all circuits can be approximated with a relatively small number of gate types. An example of a **universal quantum gate set** (a collection of gates from which any desired circuit can be constructed) is a Hadamard gate, a CNOT gate, and a π/8 gate. These gates can be defined as follows:

Hadamard gate

initial state	final state
$\|0\rangle$	$\frac{1}{\sqrt{2}}\|0\rangle+\frac{1}{\sqrt{2}}\|1\rangle$
$\|1\rangle$	$\frac{1}{\sqrt{2}}\|0\rangle-\frac{1}{\sqrt{2}}\|1\rangle$

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

CNOT gate

initial state	final state
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|11\rangle$
$\|11\rangle$	$\|10\rangle$

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

π/8 gate

initial state	final state
$\|0\rangle$	$\|0\rangle$
$\|1\rangle$	$\frac{1+i}{\sqrt{2}}\|1\rangle$

$$\begin{bmatrix} 1 & 0 \\ 0 & \frac{1+i}{\sqrt{2}} \end{bmatrix}$$

The kinds of gates implemented in an actual quantum computer would depend on engineering tradeoffs between the cost of having more gates and the performance penalty of requiring more steps to implement a circuit.

The description above is a little misleading because there is a continuum of possible quantum gates, and almost none of them can be constructed from finite sequences of these universal gates. But any possible quantum gate can be approximated as closely as desired with a finite sequence of universal gates.

State and Gate Geometry

In Figure 16-1, we showed the state of a qubit as a point on a unit circle. More generally, since amplitudes can be complex numbers, the state of a qubit is actually a point on a 4-dimensional unit sphere, with the real and imaginary parts of each complex amplitude considered as separate coordinates. Similarly, the state of a collection of n qubits is a point on a 2^{n+1}-dimensional unit sphere. It's a *unit* sphere because states are normalized, i.e., the sum of the squared absolute values of all the amplitudes must be 1.

A quantum gate must be unitary, which means that it is linear and maps states to states. The vector from the origin to any state has length 1, so the gate maps that vector to another vector of length 1. Because it is linear, the gate must map the vector *between* any two states to another of the same length. In other words, the gate preserves the distance between states. So the sphere of states remains rigid under gate application, which means that it can only rotate or flip. And if a set of states lie in a plane, they will still lie in a (likely different) plane after gate application.

16.2 GROVER'S ALGORITHM

Grover's algorithm is a quantum algorithm that can do brute force search in the square root of the time it would take on a classical computer. For instance, assume we want to know which n-bit secret key k encrypts plaintext m to ciphertext c. On a classical computer, we could try all possible (N=2^n) keys, and on average, it would take $\frac{N}{2}$ guesses to find the right key (and worst case, N). On a quantum computer running Grover's algorithm, the number of iterations to get n qubits into a state where it is very probable that the state will be read as the key k is proportional to $\sqrt{N} = 2^{n/2}$.

Grover's algorithm begins by initializing n qubits such that if the set were read to produce an n-bit value, each of the N=2^n possible values would be equally likely. It then iteratively boosts the probability of reading the value k (the answer we are searching for), little by little, until after roughly $\sqrt{N} = 2^{n/2}$ iterations the likelihood of reading k will be close to 1.

To superpose all $N{=}2^n$ possible n-bit values onto n qubits, we begin by zeroing each qubit (measuring it, and if it's 1, inverting it) and then applying a Hadamard gate to each qubit. The leftmost part of Figure 16-3 show the amplitudes of each of the $N{=}2^n$ superposed values at this point, which will be $\frac{1}{\sqrt{N}}{=}2^{-n/2}$ for each.

Now we will loop over the following two operations (the rest of Figure 16-3):

1. Multiply the amplitude of $|k\rangle$ by -1
2. Reflect the amplitudes of all $N{=}2^n$ states across the mean of all the amplitudes

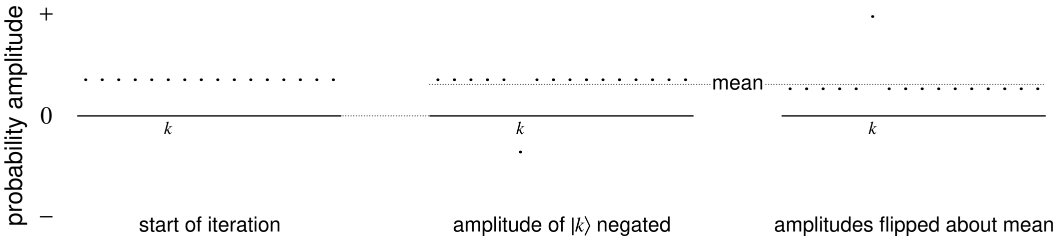

Figure 16-3. First iteration of Grover's Algorithm, $n{=}4$, N=16, $k{=}5$

Note that negating the amplitude of one state ($|k\rangle$) is a unitary operation (the probabilities still add up to 1 afterwards) since the squared absolute value of the amplitude of $|k\rangle$ will not change when the amplitude is multiplied by -1. It may be somewhat mysterious at this point how to create a quantum circuit that will recognize and negate the amplitude of $|k\rangle$, but we'll explain how to do that in §16.2.2.

Now we will reflect each of the amplitudes across the mean of all of the amplitudes. The mean will be very close to the common amplitude of every state except $|k\rangle$, because there are $N{-}1$ of those and only one instance of $|k\rangle$. The mean will be slightly below all of them, because the amplitude of $|k\rangle$ (which is now negative) will bring down the average. You might be skeptical that this operation is unitary, but it is. In the next section, we will give an alternate description of this operation as a sequence of simple unitary operations.

The rightmost part of Figure 16-3 shows the amplitudes after the first iteration. Notice that we've now made all the amplitudes (other than $|k\rangle$'s) smaller by a tiny amount (because the mean was so close to the amplitude of every other state). But we've made the amplitude of $|k\rangle$ a lot bigger, because it was further from the mean. Now, the amplitude of $|k\rangle$ is bigger by approximately a factor of 3. Because the probability is the square of the amplitude, the chances of measuring k goes up by approximately a factor of 9 after the first iteration. Despite that, it will still be very improbable, if we read the n qubits, that we'll get k (assuming n is large). To make the figure legible, we've used $n{=}4$ qubits, but to make Grover's useful, a value of $n{=}128$ would be more realistic.

The next time we do the two operations

1. Multiply the amplitude of $|k\rangle$ by -1

2. Reflect the amplitudes of all $N=2^n$ states across the mean of all the amplitudes

all the amplitudes (other than $|k\rangle$'s) again get a little bit smaller, and $|k\rangle$'s increases (but the increase is a little less than before since the mean is a little smaller). After iterating the optimal number of times, the amplitudes of each of the states (other than $|k\rangle$) are nearly 0, and the amplitude of $|k\rangle$ is nearly 1.

Interestingly, if we continue iterating beyond the optimal number of times, the mean becomes negative, so when we reflect across the mean, the amplitude of $|k\rangle$ decreases, and the amplitude of the other states increases. If we overshoot the optimal number of iterations, then for a while it will become less and less likely that when we measure the qubits we'll read k. Figure 16-4 shows how the probability of reading k changes as the algorithm proceeds. Note that the probability of reading k increases to a maximum, then decreases as more iterations are performed, and then cycles. So it's important to know when to stop iterating in order to have the best chance of reading the correct result. You only get one chance to read a result.

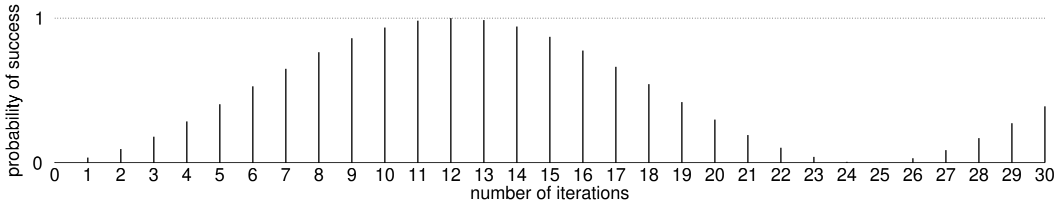

Figure 16-4. Overview of Grover's Algorithm (n=8 shown)

16.2.1 Geometric Description

In general, a quantum state of n qubits, superposing $N=2^n$ classical states, is represented by a point on a $2N$ dimensional unit sphere. The reason it's $2N$ (rather than N) is that the amplitudes, in the general case, are complex numbers. However, for Grover, we can focus on just one unit circle on that $2N$ dimensional sphere. That circle lies in a plane we'll call Plane K (because the y coordinate is the amplitude of $|k\rangle$). The amplitude of |anything-but-$k\rangle$ (an equal superposition of all classical states other than $|k\rangle$) is the x coordinate.

The initial state in Plane K (after initializing each qubit to 0, and then applying Hadamard to each) is shown in Figure 16-5. The amplitude of $|k\rangle$ is tiny ($\frac{1}{\sqrt{N}}$). So the state is just slightly above $\langle 1,0\rangle$, the rightmost point of the circle. Let's call that angle a.

So, the state in Plane K will start at angle a. Each iteration of Grover will rotate the state counter-clockwise by an angle $2a$. After $\frac{\pi}{4a}$ iterations, the state in Plane K will have traveled roughly $\frac{\pi}{2}$ around the circle, arriving near $\langle 0,1\rangle$, so the probability of reading k will be nearly 1.

If we were using Grover to search for a 128-bit key, angle a would be way too small to show in a diagram, so we've chosen n=6 ($N=2^6=64$) for Figure 16-6.

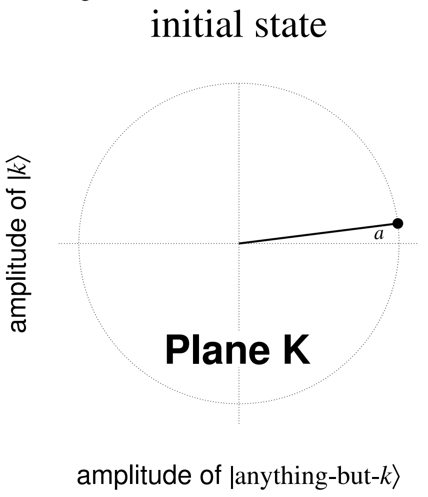

Figure 16-5. Grover's Plane K Initial State

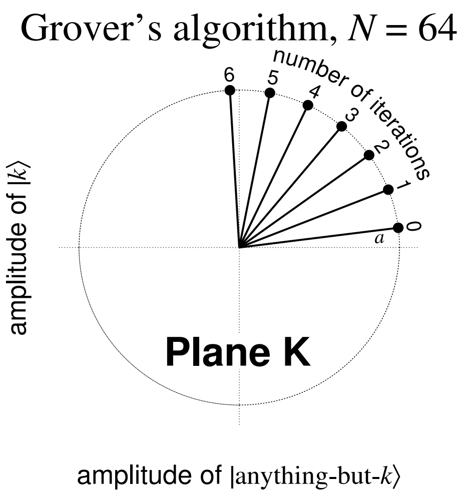

Figure 16-6. Grover's Plane K State Progression

- Since the arc of the circle is nearly vertical at the initial state, and levels off to near horizontal near the top of the circle, the amplitude of $|k\rangle$ (the y coordinate) increases more at the beginning than it does when the state nears $\langle 0,1 \rangle$.
- If there are too many iterations (going past the top of the circle), the amplitude of $|k\rangle$ starts decreasing. If you kept doing iterations, you'd cycle around the circle, with the probability of success decreasing to near 0 before increasing again once the amplitude of $|k\rangle$ goes negative.

That is the high level summary. Now we will explain how we can manage to do these operations.

16.2.2 How to negate the amplitude of $|k\rangle$

We assume that there is some efficiently computable function f that takes an n-bit classical bitstring s, and returns 1 if s has the correct value and 0 otherwise. For example, if we were searching for an s for which hash(s)=h, then $f(s)$ would return 1 if hash(s)=h and return 0 otherwise. If we want to know what n-bit secret key encrypts plaintext p to ciphertext c, then $f(s)$ would return 1 if p encrypted with key s is c, and return 0 otherwise.

For simplicity, we will assume that we know that s is an n-bit integer, and that there is exactly one value, k, such that $f(k)=1$. That is, $f(s)=1$ if $s=k$ and $f(s)=0$ otherwise.

In order to construct a quantum version of f, say f_Q, we will use an extra qubit, which we'll refer to as the **ancilla** qubit, and a quantum circuit that will operate on $n+1$ qubits (the n qubits of s plus the one ancilla). With the notation $s;b$ meaning the value s together with the ancilla qubit b, f_Q flips the ancilla between 0 and 1 when $s=k$, and leaves the ancilla alone for all other values of s.

After performing f_Q, we will have $n+1$ entangled qubits. If we read the qubits after only a few iterations of Grover, we'd almost certainly read a value s different from k in the first n qubits, and 0 in the ancilla. But if we were lucky enough to read k, we'd get $k;1$.

We now describe how to negate the amplitude of $|k;1\rangle$, and leave the other amplitudes unchanged. We will need a quantum gate called a Z gate, which acts on a single qubit and takes $|0\rangle$ to $|0\rangle$ and $|1\rangle$ to $-|1\rangle$.

To negate the amplitude of $|k;1\rangle$, apply a Z gate to the ancilla. Since the ancilla is entangled with the other qubits, this multiplies the amplitude of state $|k;1\rangle$ by -1, and does not change any of the amplitudes for $|s;0\rangle$. (All states other than $s=k$ will have the ancilla equal to 0).

Now apply f_Q again. This leaves the ancilla unchanged for all values except k, but flips the ancilla from 1 back to 0 for $|k\rangle$. Now the ancilla is solidly 0 for all states and is therefore no longer entangled with the other qubits. (See Homework Problem 11.)

The above circuit is not the most efficient way to negate the amplitude of $|k\rangle$, since it requires us to compute f_Q twice. There is a cleverer way to negate the amplitude of $|k\rangle$. It turns out that if we set the ancilla qubit to Hadamard$(|1\rangle)=\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$, then we only need to apply f_Q once. It's not obvious, but it works (see Homework Problem 12).

So now we know how to negate the amplitude of $|k\rangle$, or to put it another way, we know how to reflect the amplitude of a selected state (in this case $|k\rangle$) across the x-axis.

16.2.3 How to reflect all the amplitudes across the mean

This is accomplished in a way very similar to negating the amplitude of $|k\rangle$, but is slightly more complicated to explain. What we really want to accomplish is reflecting the state across the line in Plane K at angle a, as shown in Figure 16-5. We know from the previous section (§16.2.2) how to reflect an amplitude across the x-axis in a plane. But that line in Figure 16-5 that we want to reflect across is not the x-axis.

The trick we will use is to visit another plane, which we'll call Plane Z (for "zero"), in which the x-axis is the amplitude of $|0\rangle$, and the y-axis is the amplitude of $|\text{anything-other-than-0}\rangle$. The line we wanted to reflect across in Plane K will be the x-axis in Plane Z.

The initial state at the beginning of Grover, when all the qubits are initialized to zero, is shown in Figure 16-7 as the point (1,0) in Plane Z, because the amplitude of $|0\rangle$ is 1, and the amplitude of all other states is 0.

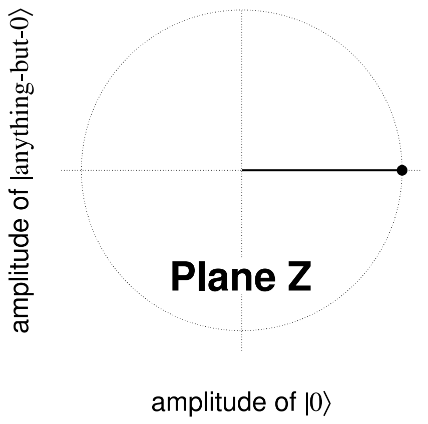

Figure 16-7. Grover's Plane Z Initial State

To travel between Plane Z and Plane K, we apply a Hadamard to each of the *n* qubits. As the state travels from Plane Z to Plane K, it rotates counterclockwise by angle *a*. If we apply Hadamards while in Plane K, it will return to Plane Z, rotated clockwise by angle *a*.

The line we wanted to reflect across in Plane K is now the x-axis in Plane Z, and we know how to reflect states across the x-axis.

We mark which states we wish to reflect across the x-axis by flipping the ancilla between 0 and 1 for all states other than $|0\rangle$. We use a quantum function, say g_Q, which flips the ancilla between 0 and 1 when the value of the *n* qubits is not equal to 0, and leaves the ancilla alone for $|0\rangle$.

Now we apply a Z gate to the ancilla and we've reflected all values (other than $|0\rangle$) across the x-axis. Then we need to apply g_Q again to reset the ancilla to 0 for all states.

Now we return to Plane K by applying Hadamards to the n qubits. The combination of the 3 operations

- apply Hadamards to get from Plane K to Plane Z
- reflect all states other than $|0\rangle$ across the x-axis
- apply Hadamards to get back to Plane K

accomplishes the goal of reflecting all states across the mean of all the amplitudes.

This is depicted graphically in Figure 16-8.

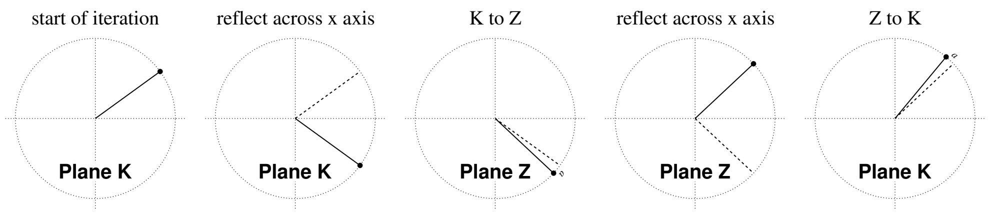

Figure 16-8. One Iteration of Grover's

16.2.4 Parallelizing Grover's Algorithm

Grover's algorithm speeds up key search quadratically. Does that mean that we need to double our key sizes for secret key algorithms? Not necessarily. Apart from the very real possibility that quantum gates might be more difficult to build than their classical counterparts, there is a more fundamental difference between Grover's algorithm and classical key search algorithms that makes Grover's less practical—Grover's algorithm doesn't parallelize well.

To provide a concrete example, DES with its 56-bit key is considered to be completely insecure. However, if all you had was a single CPU, trying one key after another, it would take decades to get through all the keys, even if it was a pretty good CPU. Nonetheless, DES keys are routinely broken in a matter of hours. This is done by using multiple CPUs (or graphics cards, or specialized hardware) to try many keys at once. For example, if you can try 100000 keys at the same time, you can break a DES key in a few hours. There is no similar way to speed up Grover's algorithm.

You could of course divide the key space into segments. For example, if you tried to search for a 128-bit key using a million quantum processors, each with a couple hundred qubits, you could get the first processor to run Grover's algorithm under the assumption that the first twenty bits of the key were 0…00, you could get the next one to assume the first twenty bits were 0…01, etc. However doing this would only reduce the number of iterations of Grover's algorithm by a factor of about a thousand (from about 2^{64} to about 2^{54}.) You made your computer a million times bigger, but it only made the search run a thousand times faster. One might hope that there would be a more efficient way to parallelize Grover's algorithm. However, Zalka [ZALK99] showed that there isn't one.

Therefore, doubling the size of secret keys and hashes is more than sufficient to defend against the threat of quantum computers running Grover's algorithm.

16.3 SHOR'S ALGORITHM

Shor's algorithm [SHOR95], which can efficiently factor integers and find discrete logarithms, is the most important application of quantum computing for cryptography. Shor's algorithm, combined with a general purpose quantum computer of a few thousand logical qubits and a quantum program that applies billions of logical gates (see §16.6 *Quantum Error Correction*), would render all currently widely used public key cryptography insecure. Shor's algorithm finds the period of a periodic function. We'll explain in §16.3.1 how RSA with a modulus n relates to a periodic function, and explain in §16.3.2 why knowing the period of that function helps us factor n.

Then starting in §16.3.3, we describe how Shor's algorithm can factor a b-bit RSA modulus n. Typically, RSA moduli are 2048 bits long, but to make the numbers easy, we'll use b=2000 in examples. The version of Shor's algorithm which finds discrete logs is a little more complicated, but conceptually similar to the one for factoring.

16.3.1 Why exponentiation mod *n* is a periodic function

An RSA modulus n is (usually) the product of two primes p and q: $n=pq$. As we described in §6.2 *Modular Arithmetic*, $\phi(n)$ (the totient function) is the number of positive integers less than n that are relatively prime to n. For $n=pq$, $\phi(n)=(p-1)(q-1)$. Euler's theorem states that for any a relatively prime to n, $a^{\phi(n)}=1$ mod n. So for any value x, a^x mod $n=a^{x+\phi(n)}=a^{x+2\phi(n)}=a^{x+3\phi(n)}$, and in general, for any integer k, a^x mod n equals $a^{x+k\phi(n)}$. In other words, a^x mod n is a periodic function of x that repeats every $\phi(n)$.

In fact, although a^x mod n indeed repeats every $\phi(n)$, the actual period will be a divisor of $\phi(n)$. The period of a^x mod n will be no bigger than the least common multiple of $p-1$ and $q-1$, lcm($p-1$,$q-1$), which is smaller than $\phi(n)$ because common factors of $p-1$ and $q-1$ will be divided out. For instance, since p and q will be odd numbers, both $p-1$ and $q-1$ will be even (having a common factor of 2), so lcm($p-1$,$q-1$) can be at most half of $\phi(n)$. Also, if a is, for instance, a square mod n, a^x mod n will only cycle through half the numbers, so the period might again be halved. Shor's algorithm will compute the period of a^x mod n. We'll call the period d.

16.3.2 How knowing *d* lets you factor *n*

First we'll show how, knowing d (the period of a^x mod n), you can find some number y that is a *nontrivial square root* of 1 mod n, (a number, other than ±1 mod n, such that y^2=1 mod n). Then we'll explain why knowing y lets you factor n.

Because d is the period of a^x mod n, we know that a^d mod n=1. If a^d mod n=1, then $a^{d/2}$ mod n is a square root of 1. By the Chinese remainder theorem, there are four ways that a number y can be a square root of 1 mod n, where n=pq:

- y is 1 mod p and 1 mod q. In this case, y will be 1 mod n.
- y is −1 mod p and −1 mod q. In this case, y will be −1 mod n.
- y is 1 mod p and −1 mod q. In this case, y will be a nontrivial square root of 1 mod n.
- y is −1 mod p and 1 mod q. In this case, y will be a nontrivial square root of 1 mod n.

In the first case, you can divide the exponent by 2 again, to get $a^{d/4}$ mod n, and hope to get a nontrivial square root of 1. However, in the second case ($y = -1$ mod n), you can't go further, and you need to choose a different number to exponentiate instead of a.

An implementation wouldn't start out by calculating $a^{d/2}$ mod n, and then, if that's 1, calculating $a^{d/4}$ mod n, etc. You calculated $a^{d/2}$ mod n by repeated squaring, so you would have already calculated $a^{d/4}$ mod n, $a^{d/8}$ mod n, etc.

Instead of starting with $a^{d/2}$ mod n, you would first find the largest odd divisor of d by dividing d by 2 until getting an odd number, say w. Then you calculate a^w mod n, and keep squaring the result until you get 1 or −1. If you get −1, choose a different number (instead of a) to exponentiate. If the first time you get 1 is after calculating, say, a^{16w} mod n, then a^{8w} mod n is your nontrivial square root of 1 mod n.

So now you've found a nontrivial square root of n. Let's call that y. By definition of being a square root of 1, y^2=1, or equivalently, y^2−1=0. Factoring y^2−1, the equation becomes $(y+1)(y-1)$ mod n=0.

If the product of two numbers mod n is 0, there are two cases:

- One of the numbers (y+1 or y−1) is 0 mod n.
- One of the numbers is a multiple of p, and the other is a multiple of q.

But since y is a nontrivial square root of 1, we know it is the second case. Therefore, taking gcd(y+1,n) or gcd(y−1,n), you'll get p or q. And of course, once you know one of the factors of n, you can divide by that to get the other.

16.3.3 Overview of Shor's Algorithm

For factoring a b-bit RSA modulus n, we'll need 3b qubits, where the first 2b qubits, (which we'll call the "exponent qubits") will be initialized to hold an equal superposition of all 2^{2b} classical values of t, (we are using a double-sized exponent t so that the periodic function a^t mod n will repeat many times), and b additional qubits, (that we'll call the "result qubits"), initialized to 0. We will XOR a^t mod n into the b result qubits. We will then have $3b$ entangled qubits.

Assuming a 2000-bit modulus, that would be 6000 qubits—4000 exponent qubits to hold t, plus 2000 result qubits to hold a^t mod n. We'll use N to denote the number of classical values of the exponent t. So $N = 2^{2b}$, (or 2^{4000} for a 2000-bit n).

Initialize all $3b$ qubits to 0 (measure each and invert if not 0). Then set the first $2b$ qubits to an equal superposition of all N possible classical values of t by applying a Hadamard to each one.

Next, we randomly choose an x relatively prime to n, and use a quantum circuit that calculates x^t mod n and XORs the result into the result qubits. Now we have $3b$ entangled qubits, where the $2b$ exponent qubits hold an equal superposition of all N classical values of t, and the b result qubits hold x^t mod n for each possible value of t.

In other words, if we were to measure the $3b$ qubits at this point, we'd get some random number j in the $2b$ exponent qubits, and x^j mod n in the b result qubits. That would not be very useful. Using a classical computer, we could have picked a random number j, computed x^j mod n, and gotten the same information. So we're not going to do that.

Instead of measuring all $3b$ qubits at this point, we are only going to measure the b result qubits. We'll get some random uninteresting number u. Because the $2b$ exponent qubits are entangled with the b result qubits, the result of reading the result qubits and getting the value u is that the amplitudes of all the classical states of t for which x^t mod n is not equal to u go to zero. The resulting graph of the amplitudes of all the superposed values of t will be a periodic function, with spikes every d. For reasons that will become clear later, we call this graph (Figure 16-9) the *time graph*. If

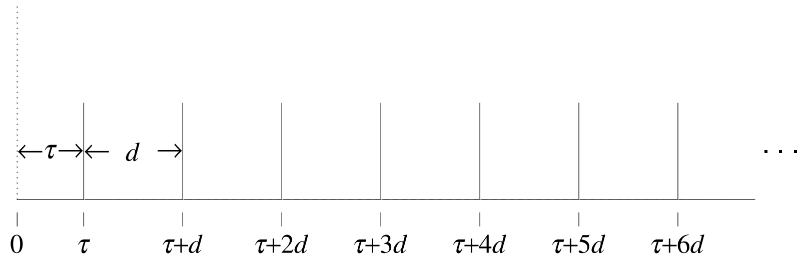

Figure 16-9. Time Graph

we could now somehow read the value of d, we'd be able to factor n. Although the spikes will be d apart, the location of the first spike will be at some random value τ, which is the first value of t for which x^t mod $n = u$. So reading the qubits at this point will not let us find d.

If we could get the locations of two of the spikes, we could take the difference, which would be a multiple of d, and that would be useful. But, once we measure, all the other amplitude spikes disappear, and due to the no-cloning theorem (§16.1.5.1 *No Cloning Theorem*) we can't copy the state before measuring.

Shor (inspired by Fourier transforms) observed that there is a way to convert the quantum state from what we have in the time graph, into a new quantum state whose graph (Figure 16-10), we'll call the *frequency graph* (for reasons that will become clear later). The nice thing about the

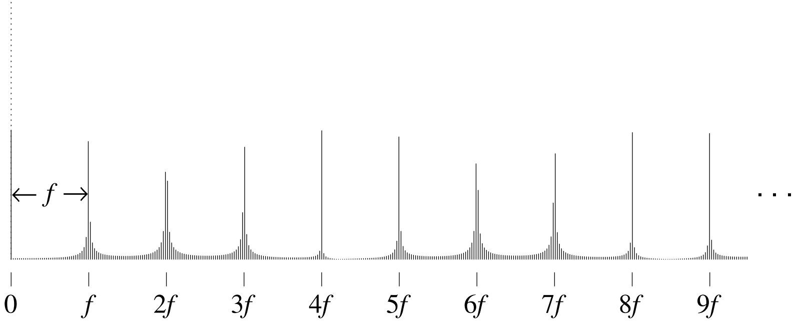

Figure 16-10. Frequency Graph

frequency graph is that it also has spikes at regular intervals, ($f{=}N/d$ apart), but the first spike occurs at the value 0, so each spike is at an integer multiple of f. That means that if we read the state at this point, we'll almost certainly get a number which is a multiple of f.

There are a few subtleties. Unfortunately, N/d will almost certainly not be an integer, since N will almost certainly not be a multiple of d. As a result, the spikes in the frequency graph are actually spread over a few values, as you can see from Figure 16-10. But that's OK. If we read the qubits, with high probability we'll read a value sufficiently close to a multiple of $f = N/d$ that we will be able to compute d with a simple classical algorithm known as *continued fractions*.

Another subtlety is that, in the time graph, all the amplitudes are positive real numbers. In the frequency graph, the amplitudes are complex numbers. In figure Figure 16-10 we're only showing the absolute values of the spikes in the frequency graph, because that's all that matters. The only thing we do once the quantum state is as represented by the frequency graph is read the qubits.

The next section will give a *classical* algorithm, hopefully optimized for comprehensibility, for converting the time graph into a frequency graph. This is not how a quantum computer would do it, but rather how a classical computer might simulate what the quantum computer is doing. The description involves doing things a quantum computer can't do (like reading the amplitudes), and a thoroughly impractical number of operations (on the order of 2^{8000} to factor a 2000-bit number.). On a quantum computer, Shor's algorithm for factoring a b-bit number would run in time proportional to b^3. In fact, the most expensive part of Shor's algorithm is the modular exponentiation, not the way a quantum computer converts the time graph to a frequency graph.

16.3.4 Converting to the Frequency Graph—introduction

We'll call call the values plotted by the x axis in the time graph t, and the values plotted by the x axis in the frequency graph s. Both t and s are integers between 0 and N–1. When we start, the quantum state of the 2b qubits of t are represented by the time graph. Later, the quantum state of the same 2b qubits will be represented by the frequency graph.

Throughout this section, we will use vectors to represent complex numbers. A complex number $x+yi$ can also be described as a vector $\langle x,y\rangle$. The vector $\langle x,y\rangle$ starts somewhere on a plane, and goes x to the right and y up.

The amplitude at each s in the frequency graph will be the sum of a bunch of vectors, as we'll describe shortly, normalized so that the probabilities of all the s's add to 1. The phase of the sum (the direction in which the vector points) does not affect the probability of reading the value. However, Shor's algorithm demonstrates how an amplitude can be a complex number. If the normalized sum vector of s is, say, the vector $\langle a,b\rangle$, that means the amplitude of $|s\rangle$ is the complex number $a+bi$.

To add a set of vectors, you add the x values to get the x value of the sum, and add the y values to get the y value of the sum e.g., $\langle x_1,y_1\rangle+\langle x_2,y_2\rangle+\langle x_3,y_3\rangle=\langle x_1+x_2+x_3, y_1+y_2+y_3\rangle$. In our case, all the vectors being added into the sum for a given s will have the same magnitude, but may be pointing in different directions. If n vectors point in the same direction, their sum will be a vector n times as long (see Figure 16-11a). If two vectors point in opposite directions, say $\langle x,y\rangle$ and $\langle -x,-y\rangle$, they will cancel each other out, resulting in a zero length vector. Likewise, if n equal sized vectors are equally spaced around a circle, they will cancel each other out (see Figure 16-11c). If the n vectors are equally spaced within an arc of a circle, then the direction of the sum vector will be in the middle of the arc (see Figure 16-11b), and the length of the sum will depend on how large the arc is. If the arc is smaller, the sum is larger.

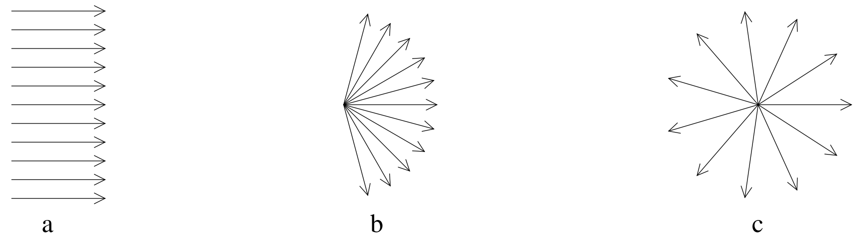

Figure 16-11. Adding Vectors

16.3.5 The mechanics of converting to the frequency graph

Imagine the time graph as a giant springy string with spikes corresponding to the spikes in the time graph. To calculate the amplitude at s in the frequency graph:

- initialize the sum vector S to $\langle 0,0\rangle$
- stretch the time graph string so that it wraps around the circle exactly s times
- for each spike in the string, add to S the vector pointing in the same direction as the spike, and of length equal to the size of the spike (in our case, all the spikes are equal length).
- normalize S by dividing by $\sqrt{N}$ so that the sum of the squared absolute values of the Ss is 1.

For $s=0$, the string is shrunk to a point, so the spikes all point in the same direction. See Figure 16-12. Note the vector inside the circle in Figure 16-12 is the normalized sum of the spikes.)

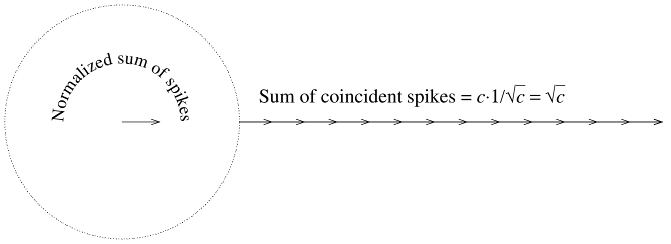

Figure 16-12. Frequency Graph Calculation for Stretch Factor $s = 0$

For $s=1$, the string wraps around the circle exactly once. Since the spikes are spaced symmetrically around the circle, as vectors they will essentially cancel each other out and the resulting amplitude will be very nearly 0. (See Figure 16-13.)

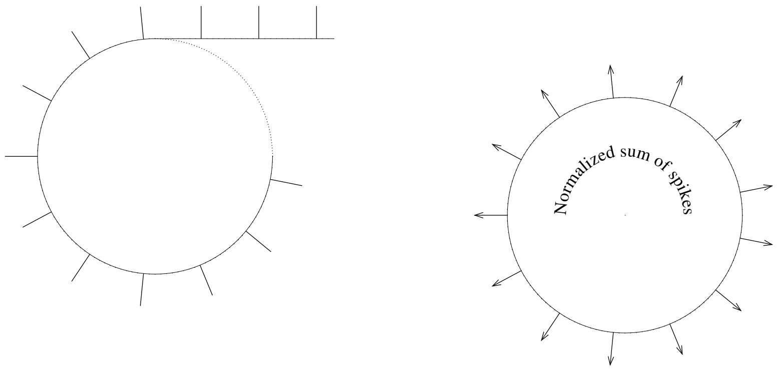

Figure 16-13. Frequency Graph Calculation for Stretch Factor $s = 1$

Similarly for $s=2$. (See Figure 16-14.) The normalized sum of spikes is basically zero, so all we see inside the circle in Figure 16-14 is a point.

But for $s \approx f$ (or s being close to an integer multiple of f) the spike hits the circle in approximately the same place on each wrap. Since f is not usually an integer, s (when s is close to a multiple of f) is either a little smaller than a multiple of f or a little larger than a multiple of f. If smaller, each successive spike will undershoot the previous spike a little, while if larger, each successive

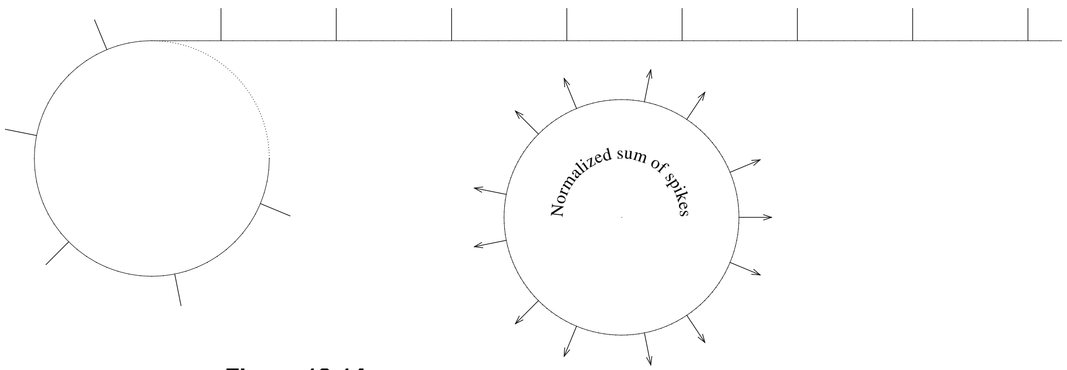

Figure 16-14. Frequency Graph Calculation for Stretch Factor $s = 2$

spike will overshoot the previous spike a little. In either case, the spikes will fall on an arc of the circle. The length of the sum vector will be somewhat smaller than if they were pointing in exactly the same direction. (See Figure 16-15.) The case where the height of the spike will be smallest is when

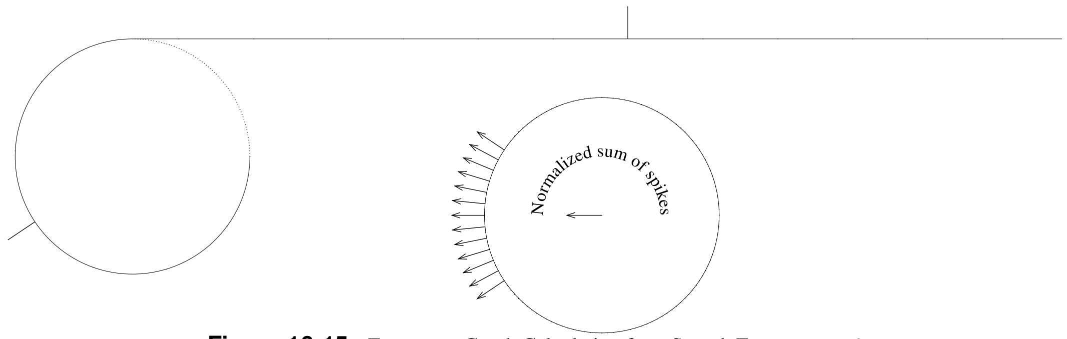

Figure 16-15. Frequency Graph Calculation for a Stretch Factor s near f

a multiple of f is a half integer so that the spikes span half the circle, reducing the magnitude of the sum by a factor of roughly $\frac{2}{\pi}$. See Figure 16-16.) Even for values of s for which the spikes go com-

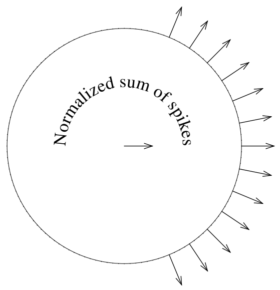

Figure 16-16. Frequency Graph Calculation for a Stretch Factor s nearly ½ away from a multiple of f

pletely around the circle, they might not exactly cancel out, so the sum could be very small, but nonzero.

When we've computed the amplitude for each s, we have the frequency graph. As we've seen, it will have spikes near each integer multiple of f. The differences between the time graph (Figure 16-9) and the frequency graph (Figure 16-10) are

- The time graph starts with a spike at a random place that depends on the value read from the result qubits. The frequency graph starts with a spike at 0.
- The spikes in the time graph are spaced d apart. The spikes in the frequency graph are spaced $f = \frac{N}{d}$ apart.
- Each spike in the time graph is a single value of t. Each spike in the frequency graph comprises a few values of s near an integer multiple of f, with the values of s closest to such a multiple providing the largest amplitudes.
- The time graph amplitude at each non-spike t is 0. The frequency graph amplitudes away from each spike are nearly 0.
- The spikes in the time graph all have identical amplitudes (both magnitude and phase). The spikes in the frequency graph vary in magnitude and phase.

16.3.6 Calculating the period

With high probability when we read the qubits in the state represented by the frequency graph, we will get a value v that is close to an integer multiple of f, where $f = \frac{N}{d}$. What we want is the denominator d.

There is a small probability that the v that we read will be 0, which is not useful, but given that there are so many other spikes with similar sizes, we almost certainly won't be that unlucky. There's also a small probability that we'll read a value that isn't near a spike, because, although small, those still have nonzero amplitudes. If we manage to read a non-useful value, we just run the algorithm again, (starting with randomly choosing a number to exponentiate).

Once we read a value v that is close to an integer multiple of f, we need to calculate d. Remember that $f = \frac{N}{d}$. We expand $\frac{N}{v}$ as a continued fraction and use that to get good rational approximations. The numerator of one of those rational approximations is usually going to be d, although it might occasionally be a divisor of d.

The choice to use a $2b$-bit exponent to factor a b-bit modulus is made because that makes it very likely that the measured v is close enough to an integer multiple of f for the continued fraction technique to efficiently find d.

16.3.7 Quantum Fourier Transform

The algorithm we've described for transforming the time graph into the frequency graph is actually performing a Fourier transform. Although $I_{3,4}$ cannot imagine anyone that isn't completely comfortable with Fourier transforms, $I_{1,2}$ spent a lot of energy on the previous section tricking potentially math-phobic readers into understanding Fourier transforms before surprising them by telling them, in this section, that they now understand them.

A Fourier transform computes a frequency graph from a time graph using a formula that might look intimidating at first glance.

$$\tilde{\alpha}(s) = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} \alpha(t) \cdot e^{-2\pi i \frac{st}{N}}$$

But let's break the formula down. It is describing the computation we described in §16.3.4. The amplitude of each s (written as $\tilde{\alpha}(s)$ to the left of the = sign) is the sum of N values, each of which depends on $\alpha(t)$ (the amplitude of t in the time graph) from $t=0$ to $t=N-1$.

We are going to travel around a unit circle and for each t, calculate a complex number to add to our sum. The amount that we travel through, for each successive value of t, is the fraction s/N of the circle. For instance, for $s=1$ we travel around the circle exactly once, and for $s=3$, we travel around the circle exactly 3 times. Also, recall that the circumference of a unit circle is 2π, and the units of distance along the circumference that add up to 2π (on a unit circle) are known as *radians*. That means that for a given value of s, we want to travel the fraction s/N of the circle for each increment of t, which is $2\pi s/N$ radians on each increment of t. So after t steps we've traveled $2\pi st/N$ radians.

We've explained most of the exponent of e in the formula. The exponent is negative, but that's the convention for the Fourier transform. If the exponent were positive, it would be equivalent to wrapping around the circle counterclockwise. Each $\tilde{\alpha}(s)$ would have the same magnitude whether the exponent were positive or negative, but its phase would be different, since it would be reflected across the x axis. For our purposes, only the magnitude matters.

But what is i, the square root of -1, doing in the exponent? This is an application of Euler's formula, which states that for any real number θ, $e^{i\theta} = \cos\theta + i \sin\theta$. What Euler's formula says is that there are three ways of expressing a point on a unit circle

- as coordinates: $\langle x,y\rangle=\langle\cos\theta, \sin\theta\rangle$
- as $e^{i\theta}$
- as the complex number $\cos\theta + i\sin\theta$

Each value we add into the sum is a complex number consisting of the complex number representing the angle at which t hits the unit circle times the amplitude at t in the time graph.

The only remaining part of the Fourier formula is dividing by $\sqrt{N}$, to make the probabilities add up to 1.

16.4 QUANTUM KEY DISTRIBUTION (AKA QUANTUM CRYPTOGRAPHY)

Quantum key distribution (QKD) is sometimes referred to in the literature as *quantum cryptography* or *quantum encryption*.

In 1984, Bennett and Brassard [BENN84] came up with a way for two parties, Alice and Bob, to leverage an existing shared key and a medium (such as a fiber optic link) for sending quantum information (such as a stream of photons), to establish a new secret that would be impossible for an eavesdropper to see. The goal is similar to using authenticated Diffie-Hellman, leveraging an existing secret, to create a new secret for perfect forward secrecy.

The theory behind quantum key distribution is that if an eavesdropper Eve attempted to read the secret bits Alice is sending to Bob, Eve would introduce enough errors in the information stream that Alice and Bob would detect the interference based on seeing more than the expected number of errors (and they would fail to agree on a new secret key).

The idea is fairly simple. Alice will carefully send a single photon at regular time intervals, randomly polarizing that photon to be up (|), 45° (⁄), –45° (\), or sideways (–).

Bob will read the photon after passing it through a polarizing filter in one of two positions, randomly chosen at each time interval by Bob; vertical or 45° diagonal. If Bob chooses to read one of the photons with a vertical filter, then

- if Alice sent |, Bob will see the photon;
- if Alice sent –, Bob will definitely not see the photon;
- if Alice sent \ or ⁄, then Bob has probability ½ of seeing the photon.

Similarly, if Bob chooses to use a filter tilted 45° diagonal, then

- if Alice sent | or –, Bob has probability ½ of seeing the photon;
- if Alice sent ⁄, Bob will see the photon;
- if Alice sent \, Bob will not see the photon.

Next, Bob tells Alice which sequence of filter positions he used, say ||⁄|||⁄⁄|⁄|⁄. This message (where Bob reports the filter positions he chose) must be authenticated using the pre-shared secret we are assuming Alice and Bob have been configured with before this exchange. Otherwise, our eavesdropper, Eve, can simply do this protocol acting as a man-in-the-middle, establishing a separate secret between Bob-Eve and Eve-Alice.

When Bob reads with a filter 45° off from the polarization of the photon, it's totally random whether he sees a photon or not, so those bits are useless. Therefore, Alice checks the sequence of filter positions that Bob reports, and tells Bob to ignore bits for which his filter was 45° off of the polarization of the photon Alice transmitted.

If Alice sent | or – and Bob chose ⁄ for that time slice, Alice tells Bob to ignore that bit. Similarly, if Alice sent ⁄ or \ and Bob chose | for the filter, Alice tells Bob to ignore that bit.

For the remaining bits, Bob (or Alice) sends a checksum computed using an error-correcting code. Since sending single photons will be a somewhat noisy channel under the best of circumstances (no eavesdropper), they will expect a certain number of errors, and the error-correcting code should be able to correct more than the expected number of benign errors, but not as many errors as would be introduced by an eavesdropper.

Why would eavesdropper Eve introduce errors? She might try to read the photons, and allow a photon that makes it through her filter to continue on to Bob. However, she has to guess what configuration to set her filter. If the photon passes through her filter, and the filter is not exactly aligned with the photon, she'll have twisted the photon, so that even if Bob's filter is aligned with Alice's choice, Bob may not be able to read the now-twisted photon. Or if Bob should have read a 0, the now-twisted photon might pass through Bob's filter.

16.4.1 Why it's sometimes called Quantum Encryption

Why do people worry about making high performance QKD (as described above)? If QKD were only used to send, say, 256 bits, to be used as a secret key to encrypt data over a traditional communication channel, high performance of the quantum channel is unnecessary.

A system has **information-theoretic security** if even an attacker with unlimited compute power cannot get any information from seeing ciphertext (i.e., there are no attacks based on going through all possible keys and recognizing plaintext when the correct key is tried). ⊕ with a one-time pad has that property. If the goal is to use the quantum channel to establish a one-time pad (and achieve information-theoretic security), the name *quantum encryption* makes sense. The one-time pad needs to be as long as the amount of data to be sent. High performance is required in this case, because the quantum communication channel (over which the one-time pad will be established) will need to have several times the desired bandwidth of the channel over which the encrypted data will be sent, because of all the bits that will need to be discarded, and error correction.

16.4.2 Is Quantum Key Distribution Important?

Quantum key distribution has been highly advertised as an important breakthrough with guaranteed security, but in our opinion, it's not all that useful or as secure as claimed.

QKD depends on having established a pre-shared secret. If you believe that cryptography works, any mechanism described elsewhere in this book for Alice and Bob to securely communicate will work. And these traditional mechanisms can work over a traditional multi-hop network, whereas QKD requires a very expensive special direct link.

16.5 HOW HARD ARE QUANTUM COMPUTERS TO BUILD?

We haven't talked much about how a quantum computer might be implemented. We have implicitly promised that all these superposed and entangled manipulations of quantum information are consistent with physics as we know it, but this raises the question: If all this is consistent with physics, why hasn't anyone built a quantum computer big enough to break 2048-bit RSA? What's so hard about it? What are the challenges involved and how might they be overcome?

The first challenge is coming up with physical systems that can reliably store and manipulate qubits. Then the qubits must be isolated from their environment, because any interaction between a qubit and its environment affects the state of the qubit. But, in order to apply gates, it is necessary to interact with the qubits in a carefully controlled way.

One way of isolating qubits from their environment is to choose qubits that interact very weakly with their environment e.g., the polarization of photons in empty space or fiber optics, but this makes the qubits very hard to manipulate, especially when we need to apply two-qubit gates that require the qubits to interact with each other. Often, reducing unwanted interactions between qubits and the environment requires extreme cooling. At normal temperatures, there are too many particles (at least photons) bouncing around to allow qubits to stay unmeasured for very long. Qubits encoded in very small physical systems like atoms and ions can sometimes deal with higher temperatures provided they are separated from other atoms by a significantly larger region of high quality vacuum.

Unfortunately, even without a noisy environment, most qubits will decay (or "decohere") on their own. If the $|1\rangle$ state is slightly more energetic than the $|0\rangle$ state, for example, it will have a certain nonzero probability of decaying to the $|0\rangle$ state in any given interval of time, and that probability will be directly proportional to the energy difference between the two states. The downside, however, of having a small energy gap between possible states of the qubit, is that it tends to make applying gates to the qubit very slow.

Despite all these challenges, there are a number of ways to create qubits that do the right thing about 99% of the time. Serious proposals have been based on photons traveling through an obstacle course of half silvered mirrors and polarizers (optical quantum computers), manipulating the spins of atomic nuclei using radio waves (NMR quantum computers), manipulating the states of valence electrons in trapped ions using lasers (ion trap quantum computers), nanoscale circuits involving semiconductors (single electron transistors, quantum dots) or superconductors (superconducting quantum computing), and pushing around excitations of planar solids in braid-like trajectories (topological quantum computers.)

At the time of this writing, the most promising candidate is superconducting quantum computing. Here the qubits are represented by tiny oscillating currents in superconducting circuits. The circuits need to be cooled to around .01K. The qubits can interact using resonators and they can be manipulated by hitting them with lasers, or simply by running a pulse of current near them.

So, qubits that can be manipulated to do the right thing 99% of the time have been produced (with some difficulty) but this on its own is not good enough. A cryptographically relevant quantum computation like Shor's algorithm involves billions of gates. Doing something a billion times that has a 1% chance of completely ruining your computation is pretty much guaranteed to ruin your computation. Solving this problem has led to the development of advanced schemes for achieving fault tolerance using error correcting codes (see §16.6 *Quantum Error Correction*), where a group of flaky physical qubits act as a single well-behaved qubit known as a *logical* qubit.

The challenges involved in building a quantum computer appear daunting, but not insurmountable. In fact, it seems increasingly likely that a cryptographically relevant quantum computer will be built in the next few decades, and if built, such a computer would render all widely used public key cryptography insecure. We live in interesting times.

16.6 QUANTUM ERROR CORRECTION

Quantum error correction is an active area of research. With all known technology, qubits have a very high error rate. Therefore, the only hope for doing complex computations is to have a set of intrinsically flaky *physical* qubits act as a group to behave as a more stable *logical* qubit. A quantum error correction scheme is characterized by a *threshold*, which is the maximum error rate for physical qubits that will result in the logical qubits created being more reliable than the physical qubits they are based on. The quantum error correction algorithm (and the size of the physical qubit group creating a logical qubit), depends on the fidelity (e.g., stability) of the physical qubits, what types of gates need to operate on the logical qubit, and how many gates the logical qubit needs to experience while keeping the correct resulting quantum state sufficiently accurately.

When building conventional computers, we can build circuitry whose error rate is almost arbitrarily low, but sometimes it is more cost effective to build something with a higher error rate and use error correcting codes to compensate for the small expected number of errors. In traditional communications and classical memory, typically less than 10% overhead is needed for error correction bits. Some classical technologies, such as DRAM, need to be refreshed by being read and rewritten periodically, because otherwise the state decays.

Quantum error correction is much harder, for various reasons.

- The intrinsic error rate is much higher (at least given currently known qubit and quantum gate technology). While a conventional computer might have one error bit in a million, all known quantum gates have error rates no better than one in a thousand.
- A quantum state cannot be read and rewritten like you can with classical DRAM to refresh the state before it decays beyond repair.
- A conventional computer bit can only have one kind of error (a bit is flipped). In contrast a qubit can drift into an infinite number of states and its entanglement can change.

It is somewhat surprising that quantum error correction is possible at all.

Acting on one or two logical qubits with a logical gate, to perform the equivalent of acting on physical qubits with a physical gate, requires more physical gates. Therefore, the physical qubits must have have high enough fidelity to withstand the extra gates required to have the logical qubit group be operated upon by at least one logical gate. If acting on a logical qubit with a logical gate has sufficient fidelity, then the error correction algorithm can be applied recursively, to produce a higher level of logical qubits with higher fidelity still. This in theory results in logical qubits with arbitrarily high fidelity (though at the expense of needing exponentially growing numbers of physical qubits to implement them). There is a trade-off between the fidelity that can be achieved by the physical qubits and the number of them needed. The frontiers of quantum computer design involve increasing both the numbers of qubits and their fidelity.

For a simplified example of a quantum error correction algorithm that will correct for a single qubit's bit flip (flipping between $|0\rangle$ and $|1\rangle$), but not correct for other types of errors, such as changes in phase (where the coefficient of $|0\rangle$ or $|1\rangle$ is multiplied by a complex number of absolute value 1), use 3 physical qubits (which we'll call *a*, *b*, and *c*) to represent one logical qubit. The correct state of the 3 qubits are that they are supposed to be equal, so the two superposed states are $|000\rangle$ (indicating the logical qubit is $|0\rangle$) and $|111\rangle$ (indicating the logical qubit is $|1\rangle$). So the quantum state of the entangled group of 3 qubits might be $\alpha_1|000\rangle+\beta_1|111\rangle$. To apply a single qubit gate to the logical qubit, apply the gate to all 3 qubits. If there were no errors, a, b, and c should all be identical after the gate, so the only two superposed states should still be $|000\rangle+|111\rangle$, possibly with different coefficients, say $\alpha_2|000\rangle+\beta_2|111\rangle$.

We assume the probability of more than one bit flip is very low. Suppose qubit *b* suffers a bit flip, so now the state might be $\alpha_2|010\rangle+\beta_2|101\rangle$. If we read any of the qubits; we'll get 0 or 1 and destroy the superposition. So instead, there is a clever way of detecting which of the qubits disagrees, by measuring both $a \oplus b$ and $b \oplus c$ (in a nondestructive way[1]). If there was no error, both measurements ($a \oplus b$ and $b \oplus c$) will be 0, if bit a flipped then we will measure 1 and 0, if bit b flipped we will measure 1 and 1, and if bit c flipped we will measure 0 and 1. Thus, we can apply a NOT gate to flip the affected physical qubit, and the state of the logical qubit will be repaired to be $\alpha_2|000\rangle+\beta_2|111\rangle$.

Unfortunately, there are ways a single qubit can be damaged other than a simple bit flip, which is why the triplet scheme above does not work in practice. For example, an error could turn $|0\rangle$ into $.6|0\rangle-.8|1\rangle$, and $|1\rangle$ into $.8|0\rangle+.6|1\rangle$. Or it could turn $\alpha|0\rangle+\beta|1\rangle$ into $\alpha|0\rangle$- $\beta|1\rangle$. The error correction schemes that work correctly are somewhat more complicated to explain, and use more qubits for a logical qubit. Those error correction schemes limit the variety of 1- and 2-qubit gates that can be fault tolerant. However, there are error correction schemes that allow a "universal" gate set. That is to say, they cannot produce any 2-qubit gate you might write a truth table for, but they can, with logarithmic overhead, produce something which is close enough to the desired gate that, even if many of these gates are required, the overall computation will succeed with high probability.

16.7 HOMEWORK

1. Suppose you use four polarizing filters, with each one 30° off from the previous one. What percentage of photons would pass through the four filters? (see section §16.1.3.1)
2. Suppose there are three unentangled qubits with qubit 1 having state $\alpha_1|0\rangle+\beta_1|1\rangle$, qubit 2 having state $\alpha_2|0\rangle+\beta_2|1\rangle$, and qubit 3 having state $\alpha_3|0\rangle+\beta_3|1\rangle$. Write out the amplitudes of each of the 8 possible values for the 3 qubits, expressed in terms of α_1,β_1, $\alpha_2,\beta_2,\alpha_3,\beta_3$.
3. Suppose we make a 2-qubit gate that (on classical inputs) XORs the value of qubit 1 into qubit 2, while leaving qubit 1 alone. Write out the truth table for this 2-qubit gate. Have we seen it before? What is the output of this gate when qubit 1 and qubit 2 are initialized to Hadamard($|0\rangle$)=$\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle$ and Hadamard($|1\rangle$)=$\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$ respectively? Are the two qubits entangled in the output state? Did the state of qubit 1 change? Did the state of qubit 2 change?

1. Note that to measure $a \oplus b$, you use an extra qubit, let's call it d, initialized to 0. First apply a CNOT gate to a and d, then apply a CNOT gate to b and d.

4. Consider a Hadamard gate operating on a qubit with real amplitudes α and β, so the state of the qubit is a point on a unit circle. The Hadamard gate reflects the state around a line through the origin. At what angle is this line? (Hint: where does the Hadamard move the state $|0\rangle$? Where does it move the state $|1\rangle$?)

5. Suppose we have three entangled qubits, in state $\alpha|001\rangle+\beta|010\rangle+\gamma|100\rangle$. What would the state of the qubits be after operating on the first qubit with a Hadamard gate? Hint: compute the result for each of the three superposed states and add the results (in proporation to their coefficients). For instance, operating on the first qubit of $\alpha|001\rangle$ with a Hadamard results in $\frac{\alpha}{\sqrt{2}}|001\rangle+\frac{\alpha}{\sqrt{2}}|101\rangle$.

6. Suppose we have the same entangled set of three qubits, in state $\alpha|001\rangle+\beta|010\rangle+\gamma|100\rangle$. What is the state of the set of three qubits after we measure the first qubit and get 1? What would the state of the set of three qubits be if we measured the first qubit and got 0?

7. For each of the following states of a qubit, show the result of applying Hadamard once, then show the result of applying Hadamard twice:
   a. $1|0\rangle$
   b. $1|1\rangle$
   c. $\alpha|0\rangle+\beta|1\rangle$

8. What is the state of two unentangled qubits, each in state $\alpha|0\rangle+\beta|1\rangle$ expressed as coefficients of $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$?

9. Show that any linear quantum gate on n qubits whose truth table is a one-to-one mapping of classical input states to classical output states is unitary.

10. If you choose two random 128-bit blocks $\langle x,y\rangle$, is it always true that there is exactly one 128-bit AES key k which maps plaintext x to ciphertext y? How about with 256-bit AES?

11. In §16.2.2, what would happen if instead of performing f_Q a second time, you simply read the ancilla, and if it's 0, then leave it as 0, and if it's 1, then perform NOT on the ancilla?

12. Show that the optimization in §16.2.2 (initializing the ancilla to Hadamard($|1\rangle$) and performing f_Q once) indeed negates the amplitude of $|k\rangle$.