Part 3

Classical and Quantum Information:

Cryptology and Computing

Peter Fortune

Part 1 of this four part series reviewed the history, development, and interpretation of quantum mechanics. This was done in a nonmathematical fashion appropriate to a general background of the field.

Part 2 reviewed some of the details of quantum theoretical methods. The objective was to lay out the gist of the field with a minimal level of mathematics.

Part 3 reviews issues in Classical and Quantum Information Theory, focusing on Cryptology and Computing

Part 4 is a Technical Appendix

November 2012

Contents

Classical Computing

We are all (too) familiar with classical computing: it is on our desktops, in our briefcases, and constantly within touch. Classical computing is based on the *binary number system* in which the only information is zeros and ones. In this section we cover the bare basics of classical computing.

Elements of Classical Computing

A *bit* is one piece of information (a zero or a one). The bit is implemented in a computer by voltage differences: a low voltage "ground state" is read as a zero, a high voltage "excited state" is a one. The bits are embedded in *semiconductors* that let electrons flow only if the voltage is sufficient to bridge a gap; the semiconductors contain switches called *transistors* that can be set to 1 (switch open) or 0 (switch closed). The basic unit of classical computing is a *byte*. A byte is a group of eight bits that represents a character, a number, a letter, a punctuation mark, or a symbol. Since each bit can have two pieces of information (0 or 1), a byte can refer to as many as $2^8 = 256$ characters.

Thus, voltage differences determine the string of zeros and ones in a byte, and these are referenced to a character map with 256 characters (the *Extended ASCII map*). For example, the byte $1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 150$ (decimal) refers to character number 150 (•) in the Extended ASCII map. The maximum number in a byte, 11111111, $2^8 = 256$, refers to the character **ÿ** in the Extended ASCII map.[1] In earlier days, the ASCII map consisted of only 128 characters; a table of that earlier ASCII character map is shown below. The 128 characters of the original ASCII character map are hardwired to the keys on your computer keyboard; the additional 128 characters in the Extended ASCII map are implemented through software.

The classical bit was a concept developed in the late 1940s by Claude Shannon, a researcher at AT&T's Bell Labs and the founder of information theory. Shannon was interested in the question, "how much information is contained in data?" Shannon's insight was that *information is surprise*: if you can accurately predict what the next bit will be, it is not information. Suppose you have 100 bits of data and you have processed 25 of those bits. If the

[1] Zero is a character in ASCII so the 256 characters in Extended ASCII are 0 through 255.

next 75 bits follow a fixed pattern so that they can be accurately predicted from the first 25 bits, they contain no surprises and, therefore, no information. But if each of the 100 bits is perfectly random, each bit contains information. So we have a counterintuitive result: *information is randomness; information is disorder, not order.*

Shannon's notion is akin to the second law of thermodynamic: *the natural progression of all physical systems is toward increasing entropy*, that is, increasing *dis*order. For example, the universe began in a very highly ordered state (zero entropy) at the Big Bang, when mass and energy were all compressed into an infinitesimally small space. But over the ensuing 15 billion years the distribution of mass and energy has diffused and become less ordered. Matter became clumped into stars, planets, and other objects so there was still order in the universe, but the amount of order declined. The ultimate destiny of the universe is complete randomness of matter: no galaxies, no planets.

Shannon Entropy, Shannon's measure of the information content of a string of bits, is the average entropy of the string. Shannon Entropy is $E = -\sum_1^N p(x_i) log_2(1 - p_i(x_i))$ where x_i is the i^{th} possible value of the (yet to be defined) random variable x, and $p(x_i)$ is the probability density function defining the probability that x_i will take a specific value. In information theory x_i has two possible values : zero and one, which can stand for on and off, up and down, right and left, etc. For example, suppose that each new bit has probability $p(x)$ = ½ of containing information and there are N = 100 bits of data. Then E = 50, that is, ½ of the bits have information. If, on the other hand, you are at a cocktail party talking with a repetitive drunk with a 0.1 probability of information content in each word, Shannon Entropy is 1.52 for 100 words: not much surprise there! Thus, the greater is Shannon Entropy the more information is in the next bit of data.[2]

In classical computing there are three basic places to store bits of data. One storage site is a chip that permanently stores a fixed set of data in Read-Only Memory (ROM); this allows routine operations to be performed without user input. An example is the BIOS chip that controls the computer's startup procedures so that the computer "boots" properly—turn the power on and the BIOS wakes up and goes through the code necessary to make the computer

[2] Information theory has advanced considerably over time, and new definitions of entropy have evolved (among them, Von Neumann Entropy), but the basic premise is the same—entropy is an inverse measure of order; the higher the entropy number, the lower is the amount of order in a system.

usable. ROM data cannot be changed by the user and, being permanently embedded in transistors, is "nonvolatile," that is, it remains after power has been turned off.

Another storage site is Random Access Memory (RAM) which holds the data from software designed to do specific operations. Examples are the Operating System information in, say, Windows 8 on a PC or OS X on an Apple computer. The OS is automatically loaded into RAM at the end of the boot procedure; it controls the computer's interactions both internally (as with the keyboard, mouse, or hard drive) and externally (as with the network, printers, and so on). RAM is "volatile," meaning that the information content disappears when the computer is shut down.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
0	0 NUL	1 SOH	2 STX	3 ETX	4 EOT	5 ENQ	6 ACK	7 BEL	8 BS	
1	10 NL	11 VT	12 NP	13 CR	14 SO	15 SI	16 DLE	17 DC1	18 DC2	
2	20 DC4	21 NAK	22 SYN	23 ETB	24 CAN	25 EM	26 SUB	27 ESC	28 FS	
3	30 RS	31 US	32	33 !	34 "	35 #	36 $	37 %	38 &	
4	40 (	41)	42 *	43 +	44 ,	45 -	46 .	47 /	48 0	
5	50 2	51 3	52 4	53 5	54 6	55 7	56 8	57 9	58 :	
6	60 <	61 =	62 >	63 ?	64 @	65 A	66 B	67 C	68 D	
7	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	
8	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	
9	90 Z	91 [	92 \	93]	94 ^	95 _	96 `	97 a	98 b	
10	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	
11	110 n	111 o	112 p	113 q	114 r	115 s	116 t	117 u	118 v	
12	120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 DEL	128	

ASCII Code Equivalents

A third part of the firmware in a computer is the Central Processing Unit (CPU). The CPU consists of transistors designed to undertake specific activities like interactions with peripheral devices (keyboard, mouse, printer, network) and mathematical operations. The CPU is ROM but it interacts with data in RAM. The ROM chips controls the operations on RAM data, things like mathematical operations, and outputs the results to RAM. Thus, the ROM portions of the CPU are not volatile but the portions that get data input and assign data output are volatile.

Finally, punchcards, hard disk drives, flash drives and so on provide permanent data storage. It is there that classical computing has a major advantage that we will discuss later: *copiability*: classical bits and bytes can be copied for safekeeping or for distribution with little danger of corruption.

We all know that there have been enormous changes in computing technology. When I programmed and operated an IBM 1401 in the early 1960s it had only 16 kilo*bytes* (KB) or 128 kilo*bits* (Kb) of RAM—just enough to hold the ASCII character code at that time. Data storage was on punch cards (called Hollerith cards) that were heavy, easily put out of sequence (by, say, dropping the card box), and easily damaged. There was no ROM so booting and operating system installation was done by punch cards. The tiny RAM put a premium on programming efficiency, and even for simple operations the computer was quite slow, doing mathematical operations in hundreds of flops (floating-point operations[3] per second) while the speed of modern computing is measured in Teraflops—trillions of cycles per second. Today a computer with 32 gigabytes (32 billion bytes) of RAM and hard drives in the terabytes can be put in a briefcase.

These advances are based on the 1947 development of the transistor at Bell Labs. Transistor density, measured by the number of transistors on a semiconductor chip, has conformed to *Moore's Law* stating that the number of computing components (transistors) on an integrated chip doubles every 1½ years. First proposed as a casual prediction in 1965, Moore's law has been remarkably accurate. It says that the 16KB "chips" of 1965 would now, 48 years later, hold 2^{32} KB, times the bits of an IBM 1401, or almost 7 billion bytes (7 gigabytes, or GB). The laptop this is written on has 8Gb of RAM.

But Moore's Law is scheduled to fail for a number of reasons. Heat is a chip killer and the heat generated by integrated circuits with such dense packing of memory chips is extremely

[3] A floating-point operation is an arithmetic operation which has a remainder, like 10/3 = 3.333333.

difficult to dissipate. Also, the speed of processing is running up against physical limits. Nothing travels faster than light, and electrons used in modern computers are coming ever closer to that limit. Advances like superconductivity can forestall the limits of classical computing by eliminating the electrical resistance that causes heat, but superconductivity has immense energy requirements. For example, superconductive materials like rubidium must be cooled to almost zero degrees Kelvin—absolute zero—to become superconductive.

Among the efforts to overcome those limitations are parallel processing, biological computing, and quantum computing. Parallel processing uses multiple computers simultaneously to process mathematical calculations. The coordination of many computers is complex, and each is still subject to the physical limits just discussed. Biological computing uses chemical interactions to represent bits of data. Quantum computing uses the quantum states of particles like electrons to represent data.

Binary Arithmetic

Arithmetic operations in computers occur in r*egisters* that record and update the operations. The smallest register—one byte or eight bits—can hold binary numbers up to 256 (decimal); the two byte (16-bit) computer of the 1960s could hold a number up to 65,535; today's four byte (32-bit) desktop computer can handle a number in the tens of billions; and modern high-end desktop 64-bit computers with eight-byte registers can handle numbers in the quintillions. As chip sizes have declined, register sizes have increased dramatically, and as register size has grown the precision of calculations has increased exponentially. The dramatic increase in precision has made scientific computing extremely accurate.

How is binary arithmetic done in a computer? Suppose we want to add the eight-bit binary numbers **a** and **b**, that is, perform the operation **a** + **b**. If **a** = 10011110 and **b** = 00010001 this requires bitwise addition: add the two bits in the rightmost column and if the result is odd (a 1) record the result (sum) as a 1 for that column; if the result is even (a 0 or two 1's) record a zero in that column and carry a 1 to the next left position. Repeat until all eight bits have been added.

```
   Carry    1
R1:  10011110  (= 158 dec)
R2:+ 00010001  (=  17 dec)
     10101111  (= 175 dec)
```

This is shown in the above operation, where **a** is in the first row (Register 1) and **b** is in Register 2. The two red digits are those affected by the carry operation. Note the fifth position: in binary addition 1 + 1 = 0 and a 1 is carried to the next column. This is, or course, exactly like the decimal addition with which we are all familiar, shown below,

```
Carry    1
        158
      +  17
      = 175
```

Multiplication in binary numbers is also performed in the standard fashion, starting with the multiplier's rightmost bit, then shifting one bit to the left and again doing bitwise multiplication, and so on.

```
    10011110   (= 158 dec)
  × 00010001   (=  17 dec)
    10011110
  + 00000000
 + 00000000
+ 00000000
 + 10011110
+ 00000000
 + 00000000
+ 00000000
= 0001010001111110  (= 2,686 dec)
```

So nothing changes when we go from decimal to binary arithmetic except the base of the number system.[4]

These operations are done in either software—with a program dictating the bitwise operations, or in hardware (chips) with the operations permanently embedded in the chips. As time has passed, the need for speed has given the nod to firmware: embedding the procedures in a chip is far faster than using software.

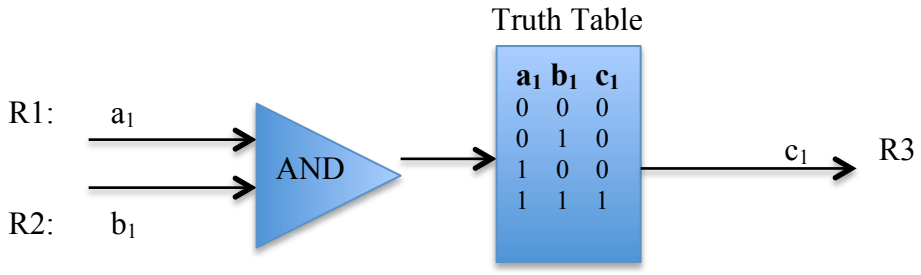
Not all arithmetic operations conform to our familiar rules. Computer science makes frequent use of Modular Arithmetic (see Part 4: Technical Appendix). We won't venture into that field here except to say that a somewhat strange result is that in modular arithmetic using

[4] Subtraction and division (which is repeated subtraction) need not be demonstrated as they also follow the standard rules.

base 2 (called *modulo 2 arithmetic*) the sum of 1 + 1 is zero, as before, but there is no carry bit. Thus, in the above example of binary addition the answer using modulo 2 is 10101111, with no carry from the red bit.

Logic Gates and Digital Circuits

Computers use *digital circuits* to process data in a more efficient manner than is shown above. The essence of these circuits is *logic gates* responsible for bitwise operations on register data. Logic gates employ *Boolean Algebra*—an algebra designed to determine the "truth" of binary results with 0 = false and 1 = true. The simplest example is the AND gate, shown below.

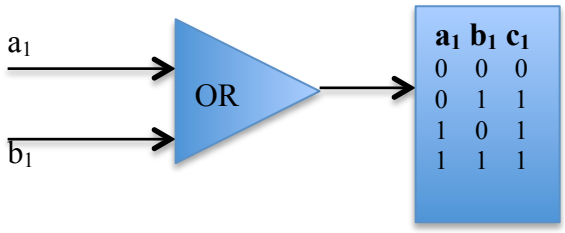


The AND Gate

The AND gate is designed to determine if two inputs are both "true," meaning that two switches are both "on" or that two bits are both 1. It takes bitwise input from two registers: Register 1 holds the bits for number **a**, register 2 holds the bits for number **b**. In the schematic above, $\mathbf{a}_1$ and $\mathbf{b}_1$ are the 0 or 1 bits from the first position in each register. The AND operation outputs a single bit **c** to Register 3. The value of that bit conforms to the rule "1 if $\mathbf{a}_1$ and $\mathbf{b}_1$ are *both* 1; 0 otherwise." This output is shown in the truth table. The operation is then repeated for all bits.

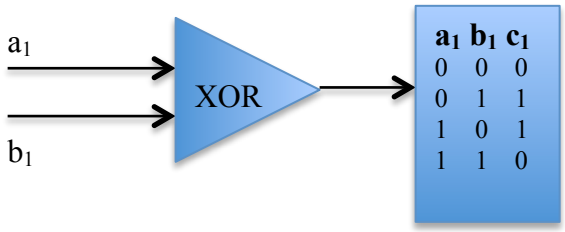
A NAND (Not-AND) gate simply reverses the outputs in the truth table: if both bits are "on," the result is a zero; if none or only one of the bits is on, the result is 1.

Yet another gate is the OR gate: the output is 1 if either **or** b_1 *or both* are 1; 0 otherwise.



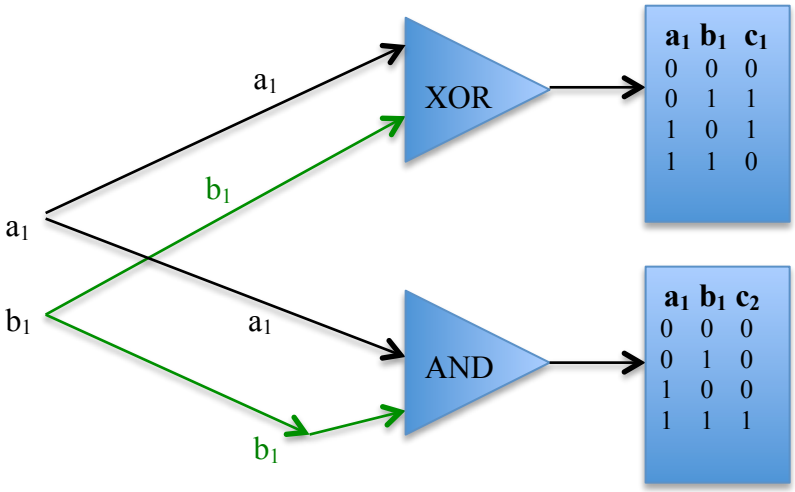
The OR Gate

A related gate is the XOR gate. This gate determines whether **a** *or* **b** but *not both* is on. It tests the logical "exclusive or" status. The XOR gate is useful in certain operations involving modular arithmetic, a staple of computer science. It implements the rule that 1 + 1 = 0. The XOR gate is



The XOR Gate

Multiple logic gates can be defined for more complex operations. A simple example is the HALF ADDER.



An XOR + AND Gate

Error Correction

We think of classical computers as reading and writing registers once—and flawlessly—then going on to the next step. In fact, each time a register's data are created, manipulated, or read, errors can creep in because of fluctuations in the voltage differences that mark a 1 from a 0: the same bit might be read as a 1 on one trial and a 0 on another.

In order to minimize the chances of incorrect reading that corrupts the calculation in progress and any further calculations based on it, the calculation is run repeatedly. Each time the result is recorded and used to build up a probability distribution of the result. When the probability that the output bit is (say) a 1 is sufficiently high, the computer takes 1 as a fact and moves on.

This creates two problems. First, calculation is slowed down by the need to repeat operations. Second, the sampling procedure increases the heat that builds up in the computer because some, if not many, operations are literally thrown away. As we know, heat is the ultimate limit on the number of transistors on a chip, hence on the validity of Moore's Law.

The need for repetition is due to a significant limit of classical computing: *operations are irreversible*. This means that you can't reconstruct the input bit string from the output bit string. If you could, error analysis would be easier and quicker: you could take the output string, determine which input string it came from, and compare that to the actual input string. If they are the same, no repetition is needed. Processing speed would be increased and heat buildup reduced.

The irreversibility of classical computation comes from the use of digital circuits to implement logic gates. We will see that quantum calculations are reversible, at least in principle, because they don't require digital circuits. Instead, quantum logic gates are in the form of matrices used to modify quantum states. These matrices are invertible, allowing the computation to be reversed. As a result, each operation is done once and immediately tested. If no inconsistency with the inputs is found, it moves on.

Quantum Computing

Quantum computing has many of the same characteristics as classical computing. The basic unit of information is still zeros and ones, but these are now in the quantum states of particles called *qubits*.[5] A qubit might be an electron, a photon, or any other quantum particle; the characteristic of the qubit that defines the 0 or 1 status might be the direction of polarization of a photon (vertical, horizontal, circular) or the quantum spin direction of an electron (up, down, right or left). We will generally use z-axis spin: the two basis states are | 1 > if an electron's z-axis spin is up (i.e., | ↑ >) and | 0 > if its z-spin is down (i.e., | ↓ >).

Qubits store and manipulate information. Recall that the quantum state of any system is a superposition of its basis states, that is, it is $|\psi> = \mathbf{a}|\ 0> + \mathbf{b}|\ 1>$ where $|\psi>$ represents the qubit's quantum state, **a** and **b** are the (possibly complex) amplitudes of each state, $|\mathbf{a}|^2$ and $|\mathbf{b}|^2$ are the probabilities of each state, and $|\mathbf{a}|^2 + |\mathbf{b}|^2 = 1$.

Also recall that all possibilities in the superposition exist *simultaneously*—a qubit is simultaneously *both* 0 *and* 1 until a measurement of the system is made; at that time the system irretrievably collapses to either |0 > and | 1 > with respective probabilities $|\mathbf{a}|^2$ and $|\mathbf{b}|^2$. Repeated trials will allow the observer to estimate the probabilities, but on each single trial the result is random.

It is the quality of superposition that gives quantum computing its potential. A classical bit carries either a zero or a one at any time, but a qubit carries both a zero and a one! A classical byte holds *one* of 256 numbers at any time, a qubyte holds *all* 256 numbers simultaneously!

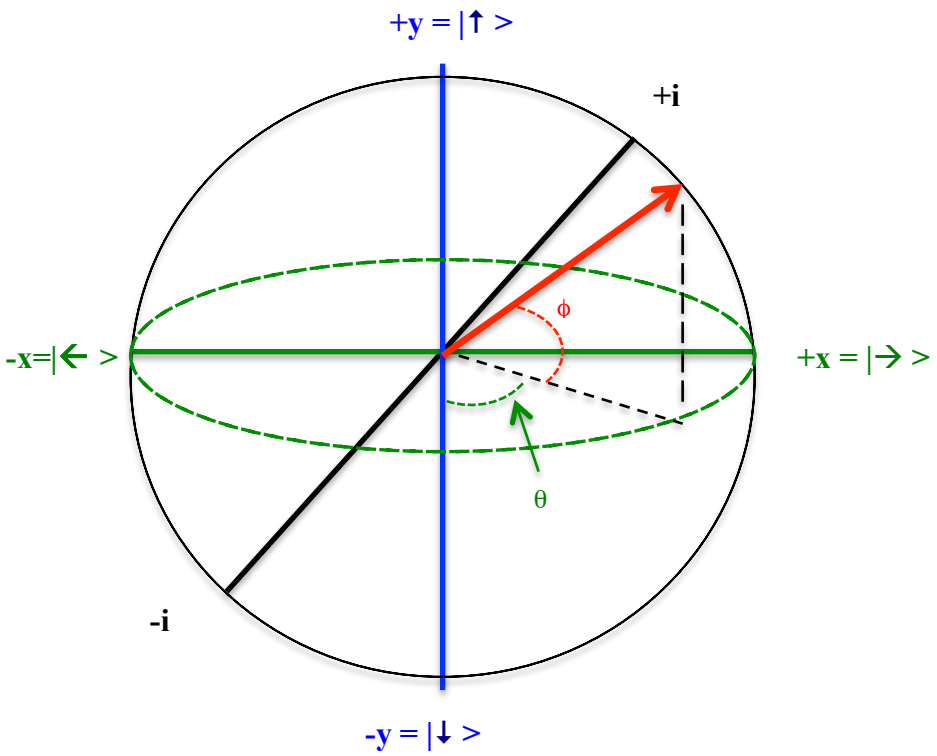
The Feral Qubit

Thus far we have focused on a qubit as a quantum particle with two possible states, say |U > and |D > for spin up and spin down, each with a specific probability. But the qubit is a bit more complicated. This section looks at the feral qubit in its native environment.

The *Bloch Sphere* below shows a qubit as a *unit sphere* (a sphere with radius 1) and three axes: a blue vertical axis for |↑ > and |↓ >, a green horizontal axis for |→ > and |← >, and a black "in-out" axis for the imaginary number *i*. The quantum state of the particle can be *any*

[5] The qubit was so named as a play on the word *cubit*, an ancient measure of length based on the length of a forearm, about 18 inches.

point on the outer surface of the sphere. The red arrow is a vector describing one specific state. The vector's magnitude is 1 because the sphere's radius is 1; the vector's direction is determined by two angles, θ and ϕ. The angle θ is a counterclockwise rotation in the "flat" x, i plane between the x axis and the *i* axis; the angle ϕ is a vertical rotation in the x, y plane, from the y axis toward the x axis.



A Quantum Particle as a Qubit

A qubit in its *pure* (unmeasured) state must have a vector with magnitude 1and terminate on the surface of a Bloch sphere. This is not true of a qubit in a *mixed* state, that is, a qubit that has interacted with the external system or with another qubit and has (at least partially) decohered: it's magnitude (the state's amplitude) must be < 1 so its position must be *inside* the sphere.

Any qubit state vector can be described by three numbers: the magnitude (always 1), θ, and ϕ. As we show in the Technical Appendix, a complex number in a space with one real dimension can be written in several ways—as $(a_0 + a_1 i)$, as $|l|(\cos(\theta) + i\sin(\theta))$, or as $|l|e^{i\theta}$ where e is Euler's constant ($e = 2.7183\ldots$). Suppose that there are M qubits in the space of the three-

dimensional Bloch Sphere shown above, with $|l| = 1$. Then a general form of the k^{th} qubit's state is

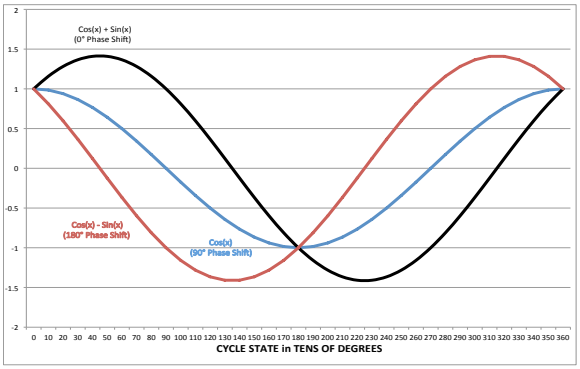
$$|\psi_k> = \cos(\theta_k) + e^{i\phi_k}\sin(\theta_k) \qquad k = 1, 2, 3, \ldots, M$$

in which θ_k is, as before, the vector's angle between the x axis and the *i* axis achieved by a horizontal rotation in the x plane, and ϕ_k is the *complex phase shift* angle arising from a vertical rotation in the xi plane.

The pace of angle θ_k is the qubit's horizontal rotational velocity—its pulse. The speed of horizontal rotation can be fast or slow, depending on the energy in the qubit: a high energy particle's heart beats fast, a low energy particle has a slower heartbeat. The speed of rotation can be tailored by the energy level imparted to the qubit by the computer.[6]

The complex phase shift ϕ measures the extent to which cos(θ) and sin(θ) are "out of phase, that is, the extent to which there is constructive or destructive interference: when $\phi = 0°$ then $e^{i\phi} = +1$ and they are "in phase;" when $\phi = \pi/4$ then $e^{i\phi} = i$ and they are 90° out of phase; and when $\phi = \pi/2$ they are 180° out of phase and $e^{i\phi} = -1$. The complex phase shift can also be shaped by energy inputs. The figure below shows the effect over a single cycle.

A Qubit Cycle with Different Phase Shifts
$|\psi> = \mathbf{Cos(\theta) + e^{i\phi}Sin(\theta)}$
$\phi =$ 0°, 90°, and 180°



[6] Suppose that ω is the speed of rotation; i.e., the frequency of a cycle (say, 650 terahertz for red light), and t is the time in nanoseconds from the start of a cycle. Then $\theta = \omega t$ gives the angle in terms of the frequency of the cycle (650 trillion cycles per second) and the clock time of the cycle.

Note that with a 90° phase shift $e^{i\phi} = i$ and $|\psi> = \cos(\theta)$; the $\sin(\theta)$ component disappears because imaginary numbers are not measurable. This is true whenever $e^{i\phi}$ is not a real number.

Quantum Computing Basics

Quantum computing is a brand new field initiated in the 1980s by (among others) the ubiquitous Richard Feynman. It is just coming out of the pure research stage—a 128-qubit quantum computer was produced and sold in 2011. However, it is designed for special tasks. No general-purpose quantum computer is in use, but its day will come.

A difference between a quantum computer and a classical computer is that one defines a piece of information as a string of bits indicated by either a "high" or "low" voltage, while the other defines it as a string of quantum qubits, each in a superposition of basis states (say 0 or 1). The data in a qubyte is in the form of eight qubits, each in a superposition taking the well-known form $(\mathbf{a}_i|0> + \mathbf{b}_i)|1>)$; each is represented by a vector in the Bloch Sphere, with each vector showing the composition between the $|0>$ and $|1>$: the more southward the vector points, the more likely is a $|0>$ reading. It is useful to remember that *every superposition is a Schroedinger probability wave showing the probability an outcome (0 or 1) of the qubit's quantum state.* Thus, when the computer varies the states of its qubits it is acting on the quantum system's probability waves.

As noted above, in quantum computing each qubit is in a superposition with value *both* 0 *and* 1). What this means is that in an n-qubit quantum computer, all 2^n possible numbers exist simultaneously and can be "discovered" by suitable rotations of the state vectors).

Quantum and classical computing share much of the same language: a quantum computer consists of registers, logic gates, and algorithms that execute operations. A register of eight qubits (one qubyte) represents up to $2^8 = 256$ binary numbers, as does a classical byte. But simultaneity of quantum states is what makes a quantum computer massively parallel: a classical byte can hold only *one* number at a time; a qubyte can hold *all* numbers simultaneously. The trick is to maintain the superposition long enough to allow algorithms to employ logic gates to process all of those numbers simultaneously. An 8-bit quantum register has 256 possible combinations of 0 and 1; A 30-bit quantum register simultaneously holds 1,073,741,824 combinations!

The basic raw material of a quantum computer is a *quantum register* formed by placing a number of qubits together. This is not a simple task—remember that if a qubit interacts with its external system—including another qubit—it decoheres. So the qubits must be put into proximity but prevented from interaction.

The Appendix gives a detailed description of a 3-qubit quantum computer's calculation of a very simple operation: *modulo-2* addition. Our "computer" has a register with three qubits: q, q, and q. Qubits q and q are input qubits, q is an output qubit. The states of the three qubits are $|\psi> = (a_0|0> + a_1|1>)$, $|\psi> = (b_0|0> + b_1|1>)$, and $|\psi> = (c_0|0> + c_0|1>)$. This generates four possible basis states for the input qubits: $|0>\otimes|0>$, $|0>\otimes|1>$, $|1>\otimes|0>$, an $|1>\otimes|1>$, with respective amplitudes a_0b_0, a_0b_1, a_1b_0, and a_1b_1.

The following steps are required for an n-qubit computer's quantum calculation:

- *Initialize the Register*: Set the n qubit states to some arbitrary values, like all $|0>$. This clears out the result of any previous calculation.

- *Combine the Register Qubit Basis States:* Create combinations (tensor products) of all 2^n possible basis states for the input and output qubits, leaving the output qubits in their original initialized states. This links the qubits together in an expanded product state space.

- *Entangle the Register Qubit Basis States:* Use quantum logic gates to create a superposition of abbreviated (orthogonal) product states in which the unknown amplitudes are eliminated and the individual qubit basis states are highly correlated.

- *Complete the Calculations Using the Entangled States:* This requires repeated use of quantum logic gates to manipulate the entangled input qubits, placing the results into the outout qubits.

- *Read the Results*

Some important aspects of this series of operations are the *entanglement of qubit states* and *the use of quantum logic gates*, and *error correction*. Entanglement will be covered more fully under our section on quantum information, but the basic idea is to convert pure independent qubit states into highly independent qubit states. For example, the combined product space for q and q is $|\psi>\otimes|\psi>$, that is, $a_0b_0|0>\otimes|0> + a_0b_1|0>\otimes|1> + a_1b_0|1>\otimes|0> + a_1b_1|1>\otimes|1>$; This has four basis states and four unknown amplitudes. But after entanglement this product state is

$$|\psi>\otimes|\psi> = \sqrt{½}(|0>\otimes|0> + |1>\otimes|1>).$$

so two important things have happened: the unknown amplitudes have been replaced by a known amplitude √½, and the two intermediate basis states have disappeared, leaving the remaining states in perfect correlation: q and q always have the same spin direction!

The quantum logic gates are (as was noted above) not unidirectional digital circuits as in classical computing. Instead, they are unitary matrices which can be implemented quickly with minimal chance of error, and which are bidirectional: just as the inputs are used to produce an output, the output can be used to reproduce the inputs.

This last property is extremely important in error correction. Like a classical computer, a quantum computer can generate errors. But unlike a classical computer, which repeats the operation until it is assured of a correct result, a quantum computer can reverse the logical gate and immediately determine if there is an error.

Advantages of Quantum Computing

What do we get from a quantum computer that we don't get from a classical computer? The property that all states simultaneously exist in a superposition gives quantum computing its power: a quantum computer processing superpositions with a large number of basis states would be a "massively parallel" computer that can simultaneously do millions of operations in the time a classical computer takes to do just one operation. Thus, quantum computing's primary advantage over classical computing is that it is faster.

Quantum computation is also *reversible*: you can start with an output string from a register and go back to the input string that generated it. The ability to recreate a facsimile of the original state and to check it against the actual original state means that you only need one step to identify an error. This avoids the repetition of operations in classical computers and the adverse consequences of repetition: computational redundancy leading to reduction in processing speed and heat buildup.

The reversibility of quantum computations arises from the linearity of quantum mechanics and the use of unitary gates (matrices) to process data. A matrix can be inverted to reproduce the input state, while the nonlinear wiring in classical computers prevents reversal of operations.

The speed advantages of quantum computers show up in a variety of applications. One is in *search algorithms*. A simple example is reverse phone number lookup. Suppose that there are N phone numbers in a zip code and you want to find the name and address of the owner of number 555-1212. With classical computing you require an average of N/2 lookups, and as many as N lookups might be needed. For New York City, with (say) ten million residents (assuming each has a listed phone) there would be an average of five million calculations, and as many as ten million. It has been established that a quantum computer would require only 10,000 calculations.

The increase in speed also enhances *code-breaking* (a two-edged sword). We saw above that a Public Key Encryption Protocol consists of a very large number with, say, N digits; that number and an exponent make up the *public key* because they can be made accessible to anyone. The first step in decoding a message is to find the two prime factors of N (call them p and q). This *prime factorization* is a *one-way function* because it is easy to multiply p and q to get N, but it is almost impossible to reverse that to find the values of p and q when N is very large. For example, if N = 200 (which is not "very large") classical computing would require hundreds of computers working in parallel for a year. If N = 500 classical computers can never complete the prime factorization. But a quantum computer can do a 500-digit number in only 16 times the time required by a classical computer for a 200-digit number. Recent PKE methods use N with 1,024 bits, way beyond the scope of classical computing.

Finally, quantum computing is a boon to the scientist beyond the increase in speed. Quantum computers can be programmed to create *Oracles*. An Oracle is a process that takes a question as an input and gives a yes/no answer. A simple example is *function identification*. Suppose that a function f(x) exists where there are N possible integer values of x (x = 1, 2, …, N). You know that f(x) is all zeros, all ones, or a mix of zeros and ones, and you want to know which is true. Classical computing would require that you do as many as N calculations to determine the answer: the average required number of calculations is 1 + N/2. So if f(x) is a 512-bit binary number a classical computer operating on each bit would require an average of 257 operations, and a maximum of 512 operations. A quantum computer with an Oracle can answer the question with just one calculation.

Difficulties of Quantum Computing

A quantum computer derives its power from quantum superposition, but, as noted above, this also is a source of difficulties. One difficulty is *maintaining the superposition*. We know that measuring a quantum system causes the superposition to vanish, leaving only one state: if a superposition of | 0 > and | 1> is measured, it collapses to either | 0 > with probability $|\mathbf{a}|^2$ or to | 1 > with probability $|\mathbf{b}|^2$. A dilemma exists: the operation of a quantum computer requires superpositions, but the detection of a result destroys the superpositions. Furthermore, the basis state that is revealed is random: if you do a measurement you get one state in proportion $|\mathbf{a}|^2$ of the detections and the other state in proportion $|\mathbf{b}|^2$. Because the superposition collapses at the first measurement, it can't be used for a sequence of calculations; and because the measured result is random, it can't be used to do calculations.

How do you detect results without losing the superposition? There are ways to do this (for example, using quantum entanglement) but they increase the complexity of the machine. The 2012 Nobel Prize in Physics was given to two physicists who measured the quantum state of photons without destroying their superpositions, so progress is being made on this difficulty.

A second difficulty is the *copiability problem*. Classical computing has the property that data can be copied to a permanent storage device and used for backup or shared. A classical copier simply takes each character on a page and transmits it to the same space on a blank page, outputting both the original page and its copy. But, as we will see, the *No-Cloning Theorem* of quantum computing says that *a quantum computer's data (qubits) can not be copied.*

The Copiability Problem: No Cloning Allowed!

As noted above, the No-Cloning Theorem says that quantum data can not be copied. By "copy" we mean the creation of a second qubit or string of qubits that is an exact replica of the first *without destroying the first*. If you are satisfied with losing the original qubits the issue is not copiability because you are not copying the original state, you are merely sending it to another place (perhaps a storage device or another computer); this is called *teleportation*.

Suppose you did have a copying machine that takes a qubit as input and creates the output of a copy plus the original qubit, just as a Xerox machine outputs the original page and a copy. The original qubit has two basis states, | ↑ > and | ↓ >, with superposition

$|\psi> = \mathbf{a}|\uparrow> + \mathbf{b}|\downarrow>$. We want our quantum copier to give a superposition with two down spins and two up spins, that is, since basis states are combined by tensor products, we want $\mathbf{a}^2(|\uparrow>\otimes|\uparrow>) + \mathbf{b}^2(|\downarrow>\otimes|\downarrow>)$. But this is not what you will get. Creation of a superposition with itself is the tensor product $|\psi>\otimes|\psi>$, resulting in the original superposition and a copy. But $|\psi>\otimes|\psi> = \mathbf{a}^2(|\uparrow>\otimes|\uparrow>) + \mathbf{ab}(|\uparrow>\otimes|\downarrow>) + \mathbf{ba}(|\downarrow>\otimes|\uparrow>) + \mathbf{b}^2(|\downarrow>\otimes|\downarrow>)$. The correct tensor product includes the two opposed spins (up/down and down/up). Only if those opposed spin states are zero czn you copy a quibit. That occurs when (1) either a or b is zero, i.e. only one spin can possibly occur, or (2) $(|\uparrow>\otimes|\downarrow>) = (|\downarrow>\otimes|\uparrow>) = 0$, i.e. opposed spins can not occur. Condition (2) requires the two spin states to be "orthogonal," meaning that the direction of one qubit's spin is independent of the other qubit's spin direction.

Thus, there *are* circumstances that allow quantum copying, but in general it is not possible. When we look at *Dense Coding* we will see that the Bell States that it relies on can be copied because there are no interaction terms. But in general, quantum states can not be copied.

Classical Cryptology

In this section we address an important application of classical computing: *cryptology*. The field of *Cryptology*—the science of communicating in codes—consists of two closely related but distinct subfields: *cryptography*, the writing of coded messages, and *cryptanalysis*, the reading of coded messages. A cryptanalyst's job is to decode a message that a cryptographer has created. The cryptanalyst must uncover the cryptographer's key if he is not already privy to it. The cryptographer is, of course, privy to his key, and is determined to design it in a way that stumps the unauthorized cryptanalyst.

There is a very long history to the dance between coding and decoding, dating back to the first time a third person came onto the planet when Adam asked Eve to bring a basket of those big red berries (apples). Codes have ranged from very simple (pig latin, alphabet substitutions) to very complex (the German Enigma code of World War II; the code embedded in a sculpture at the CIA's Langley complex, still not fully decoded even by the CIA). Virtually every code has been broken, often at great effort and expense, and each broken code stimulates new efforts to restore secrecy.

The crucial element in any code is a *key* that allows a plaintext message to be encoded and decoded. As we will see, the key for coding might not be the key for decoding, but the two must be related in order to have intelligible messages. There are two crucial problems associated with the key: distribution and authentication. The *key distribution problem* is simply the problem of maintaining the key's secrecy—only the sender and the intended receiver should have the key. Two ways to keep a key opaque to third parties are to ensure that it is not distributed outside the inner circle, and to never reuse the same key. The *authentication problem* is that the receiver must have a way to determine that the message received, even if constructed with the proper key, was sent by the proper sender and not by third party who wants to send misinformation.

We will discuss two important applications of classical cryptology, the O*ne-Time Pad*—considered the only method that, done properly, can not be decoded—and *Public Key Encryption*, a method used in a variety of important everyday applications that can be broken but only at great expense.

The One-Time Pad

The One-Time Pad (OTP) is the only known classical coding system that ensures secrecy when properly used. It consists of a randomly selected string of letters or characters—the key—known only to the sender and the intended receiver. This string is often printed on a pad of paper with each page containing a string of random letters; one page is used for each message (modern computers use binary numbers). The sender and receiver both have the same pad and share a way to determine which page on the pad is used to code the message: either each party destroys a page after use and the next message is encoded/decoded with the following page, or a code is assigned that refers to the page used. The pad's text is often broken into groups of five letters so that short words could not be identified and used to break the code.

A Russian One Time Pad

Suppose that H at headquarters wants to send the message "Attack at Dawn" to F, the field commander. The message "Attack at Dawn" is only 12 letters, so three Q's are attached to fill out a 15-letter message. The OTP approach works as in the following table.

Line 1 is the plaintext of the message. The first cryptographic step—line 2—is to assign a number to each letter; our example uses the order of the English alphabet: 01 for A, 02 for B, and

so on up to 26 for Z. The encoder then refers to the pad of randomly selected (the letters on line 3). We assume that *R D Y K Z Q M F K O D W B N U* are the first 15 letters on the pad page. On line 4 he assigns the number for each pad code letter (R is letter 18 in the alphabet, and so on).

Line 5 is the sum of lines 2 and 4. If this sum exceeds 26 (the number of letters in the alphabet) then 26 is subtracted from that sum and line 6 is derived; this is the list of letter numbers that will be sent.[7] Line 7 is the actual message sent: SXSLC BNZOP AKSEL.

One-Time Pad Example

Coding the Message

1	**Text**		**A**	**T**	**T**	**A**	**C**		**K**	**A**	**T**	**D**	**A**		**W**	**N**	**Q**	**Q**	**Q**
2	Text Num		01	20	20	01	03		11	01	20	04	01		23	14	17	17	17
3	Pad Letter		R	D	Y	K	Z		Q	M	F	K	O		D	W	B	N	U
4	Pad Num		18	04	25	11	26		17	13	06	11	15		04	23	02	14	21
5	(2 + 4)		19	24	19	12	*29*		*28*	14	26	15	16		27	37	19	31	38
6	SUB 26 if ln5 >26?		19	24	19	12	03		02	14	26	15	16		01	11	19	05	12
7	**Coded Text**		**S**	**X**	**S**	**L**	**C**		**B**	**N**	**Z**	**O**	**P**		**A**	**K**	**S**	**E**	**L**

Note: Underscored numbers on line 6 have 26 subtracted from line 5

Decoding the Message

8	**Cipher Rec'd**		**S**	**X**	**S**	**L**	**C**		**B**	**N**	**Z**	**O**	**P**		**A**	**K**	**S**	**E**	**L**
9	Text Num		19	24	19	12	03		02	14	26	15	16		01	11	19	05	12
10	Pad Letter		R	D	Y	K	Z		Q	M	F	K	O		D	W	B	N	U
11	Pad Num		18	04	25	11	26		17	13	06	11	15		04	23	02	14	21
12	(ln 11 - ln 9)		01	20	06	01	*23*		*15*	01	20	04	01		*03*	*12*	17	*09*	*09*
13	ADD 26?		01	20	20	01	03		11	01	20	04	01		23	14	17	17	17
14	**Decoded Text**		**A**	**T**	**T**	**A**	**C**		**K**	**A**	**T**	**D**	**A**		**W**	**N**	**Q**	**Q**	**Q**

Note: Red numbers in line 12 are negative and 26 is added to them to get line 13

F receives the message (line 8) and reverses the encoding method. He first translates each letter received (line 8) to its appropriate number (line 9). Then he refers to the page on his pad for the pad code (line 10) and assigns numbers to each of those letters (line 11). He then

[7] The letter recorded is calculated *modulo 26*: after 26 the sequence is started again at 1.

subtracts line 11 from line 9 (reversing H's addition of lines 2 and 4). Some of the numbers so derived will be negative, so F *adds* 26 to those to get line 13. Finally, line 14 reveals the message "Attack at Dawn."

The One-Time Pad was invented in the 1880s. It has several variations, but in all cases it has the same weaknesses: developing a completely random string of letters or other symbols for letters is not as simple as it seems, and ensuring that the key is available only to authorized parties is difficult. When a one-time pad code has been broken it is often because the key has been reused or the letters are not truly random. Barring these two security breaches, the method is known to be perfectly secure.

The OTP's excellent security arises because it is impossible to reconstruct the pad code from the coded text, and because there is typically more than one sequence of pad code letters that reveals a sensible message. Not only must the intercepting agent decode an intelligible message, he must also select the correct message from among the several intelligible messages.

Modern OTP's use computers and binary codes of zeros and ones to encode and decode messages. For example, the text might be 10111 and the pad code might be 01001. The sum is 11110 (note the last digit—1 + 1 = 0 in binary arithmetic *modulo 2*). The procedure is the same, but a computer is fast and allows the cryptanalyst to rapidly encode a message (and the hacker to rapidly search for binary numbers to decode it).

As noted above, the OTP has *perfect security*, a term coined by Claude Shannon. It can not be broken so long as the pad code is completely random and the pad is securely distributed. It does, however, have two weaknesses:

- The OTP has an *authentication problem*. A third party can intercept the original encoded message and substitute a new pad code that gives a different decoded message, then pass the message on to the authorized party.

- The OTP is an *inconvenient and insecure way to distribute the code keys*, especially for long messages. In the old days, heavy pads of paper were used (though technology allowed increasingly small pads read with magnifying glasses); with modern computing, flash drives and CDs can be used. But none of these are impervious to interception.

Public Key Encryption (PKE)

In the 1870s the British mathematician and economist William Stanley Jevons (discoverer of the sunspot theory of business cycles) investigated the application of "one-way

functions" to cryptography. A one-way function is a mathematical function that is easy to solve in one direction but extremely difficult to solve in the reverse direction. A simple example is he factoring of a number into two prime numbers: it is easy to multiply two known prime numbers to get a larger (not necessarily prime) number. But it is difficult to take a large number and find two unique prime factors.

This is the basis of the most frequently used PKE system. In 1977 three MIT scientists—Ronald **R**ivest, Adi **S**hamir, and Leonard **A**delman—proposed the RSA Method of Public Key Encryption. The RSA algorithm is based on number theory, particularly prime numbers and modular arithmetic. Its security is due to the extreme difficulty of factoring large numbers into two prime numbers. However, in recent years there have been advances in number theory that have made prime factoring of large numbers easier; these have weakened RSA's security.

A very simple example is the two binary numbers 11 and 10001; in decimal notation these are the prime numbers 3 and 17. Their product is 110011 (the number 51). Constructing the product 110011 from the two known prime numbers is a snap, but if one knows only the number 110011 it takes some time to find that its two prime factors 11 and 10001. The time required increases at least exponentially as the number length increases, and with larger numbers there can be multiple prime factors.

To use a more complicated—and more realistic example—consider a 512-bit number in binary code. The largest possible value occurs when all 512 bits are ones; that value is $\sum_0^{511} 2^x = 1.341\text{x}10^{154}$; it has 155 digits and is, as the astronomer Carl Sagan would say, "…billions and billions and billions…and billions." The task of factoring a 155 digit number into two primes is almost overwhelmingly complicated, and there can be many prime factors to choose from.

A simple example of the RSA method of PKE is given below. H wants to send F the single letter "A" for "ATTACK AT DAWN" (Desktop computers can not implement more than one letter messages because they don't record enough significant digits, so the example chosen is simply a one-letter message.)

The PKEP-RSA Method

Step 1: Choose the private and public keys

1. Choose two prime numbers, p and q: ***p = 23, q= 41***
2. Compute the product n = p•q*:* ***n = 943***
3. Compute the number r = (p-1)(q-1): ***r = 880***
4. Find a number k that is 1mod(r) (see appendix):[8] **k = 2641**
5. Compute the prime factors of k, e and d, for which e*d = 1mod r): **e = 3, d = 587**

The numbers n and e are the public key components used to encode the message.
The number d is the private key required (along with the public key) to decode.

Step 2: Encode the message

6. Convert the plaintext message to ASCII Code:

PLAINtext = A ➡ ASCIItext = 65

7. Encodes the ASCIItext as CODEtext = $(\text{ASCIItext})^{\mathbf{e}}$mod(n):

CODEtext = $\text{ASCItext}^{\mathbf{e}}$ mod(n) = $(65)^{3}$mod(943) = 212

8. Send CODEtext 123 to recipient with private key under separate cover

Step 3: Decode the Message

9. Receiver Decodes as (ASCIItext) = $(\text{Codetext})^{\mathbf{d}}$mod(n) :

ASCIItext = $(212)^{\mathbf{d}}$ mod(943)

= 65

10. Receiver converts ASCIItext to PLAINtext : **A**

PKE has a variety of uses beyond spywork. Although we don't realize it, PKE is encountered every day. Commercial banks use it to verify the authenticity of online access bank accounts; *digital signatures* are used to verify the authenticity of electronic documents; *digital certification* is used to verify that your computer has an authorized relationship with a website (you have undoubtedly been asked by your computer to verify a "certificate").

[8] 2641 = 2(880) + 1 so it is a multiple of 880 with remainder 1, that is, it is 1mod(880).

PKE has several advantageous properties: it is *easily coded and decoded by computer;* It is *extremely difficult to break*, although algorithms developed in recent years have made it less secure; and *the keys can be reused*. But there are weaknesses: PKE is *not perfectly secure* and can be broken, though not without great effort and expense; PKE is subject to the *key distribution problem*, that is, if the private key falls into an evildoer's hands, the code is broken because he has both the public key used to encode the message and the private key required to decode the message; and PKE also has the *authenticity problem*—if the private key is known to an intruder he can construct misleading messages that will seem legitimate to the intended recipient, thus engaging in disinformation.

One approach to the key distribution problem is to have H encode a message using the public key ($\mathbf{e_H, n_H}$), then send the message to F; the message is simple, it is the public key. H keeps the private key, $\mathbf{d_H}$, that is required for decription. Thereafter, any message F wants to send to H can be encrypted with H's public key, and H can decrypt it with his private key ($\mathbf{d_H}$). The private key is never out of H's hands so it is more secure than if H had sent the private key to F. This works only in one direction—F can encode a message and send it to H, but if H uses the same public key to send messages to F, F must have the private key to decode, and the distribution problem returns. But two-way communication can be easily restored if F constructs his own public key ($\mathbf{e_F, n_F}$) and private key ($\mathbf{d_F}$), sends the public key to H so H can encode messages to F, then uses his private key to decode H's messages. This allows both H and F to communicate by RSA codes with no key distribution problem.

Quantum Cryptology

We have investigated two popular forms of classical cryptology: the One-Time Pad system and the RSV method of Public Key Encryption. We have also seen that one of the difficult issues in both is the key distribution problem: an intruder can obtain the key required to decrypt a message either by intercepting it when it is sent to the code recipient, or by theft. Indeed, if the key distribution problem can be solved, a One-Time Pad system is perfectly secure.

So the question is, "Is there a quantum method to solve the key distribution problem?" Can A find a way to give B the key without fear of an eavesdropper intercepting the key and either decoding messages or creating false messages to pass on to B. The answer is "yes."

The BB84 Key Distribution Method

Developed in 1984 by Charles Bennett and Gilles Bressard, the BB84 method is surprisingly straightforward. Messages are sent in 0's and 1's, as in classical cryptography, and the key is also in zeros and ones.

First, A sends B a batch of quantum particles, say electrons, that have been measured so that the spin states are known to A. A has measured some particles for z-axis spin, others for x-axis spin; the spin axis for each particle has been randomly determined. It is essential that the number of qubits sent is sufficiently greater than the number of qubits needed to form a key. If the message is, say 800 bits long, the number of qubits sent by A must be (much) greater than 800.

Second, B measures the spin of each qubit. He does not know which axis A measured, so he randomly chooses either the x or z axis. Line 3 shows his choice of axis, and line 4 shows the result he gets.

The BBQ84 Method of Key Distribution

A Measures	$\|\uparrow\rangle$	$\|\rightarrow\rangle$	$\|\leftarrow\rangle$	$\|\uparrow\rangle$	$\|\downarrow\rangle$	$\|\leftarrow\rangle$	$\|\leftarrow\rangle$	$\|\rightarrow\rangle$
A Knows	+	+	-	+	-	-	-	+
B Measures	z	z	x	Z	X	x	z	x
B Finds	+	-	-	+	-	-	-	+
B Says	Z	z	x	Z	X	x	z	Z
A Responds	Yes	No	Yes	Yes	No	Yes	No	Yes
Key	1	na	0	1	na	0	na	1

Third, A and B have a phone conversation; it can be on an open line so an eavesdropper (E) can listen in. For each particle B reports the spin axis he tested, and A then agrees or disagrees that that was the spin axis she found for that particle. For example, for the third particle B reports "I tested the x axis", and A says "Yes, that was the axis I measured." For the second particle B reports "I tested the x axis" and A responds "No, I sent a z-axis spin."

Finally, B discards the particles on which he and A tested different axes (particles 2, 5, and 7), leaving particles 1, 3 , 4, 6 and 8. Both A and B have agreed that the spin directions for these particles are + – + – +, so they agree that the first five bits of a key are 10101. If the number of qubits sent is enough, they can find 800 qubits necessary to form a key.

So now both A and B have agreed on a key. Suppose a third party, E, has listened in on the phone call. All E knows is the positions of the qubits on which A and B measured the same axis; this tells her nothing about the spin states found on those axes. So the key has been distributed with perfect security.

Suppose the message to be sent by A is 00111. A encodes this message (recall that 1 + 1 = 0 with carry of 1; 0 – 1 = 0 with carry -1) as follows

Text Message	0 0 1 1 1
+ Key	1 0 1 0 1
= Coded Message	1 0 1 0 0

and B decodes the message by subtracting the key from the coded message:

Coded Message	1 0 1 0 0
+ Key	1 0 1 0 1
= Text Message	1 0 1 0 0

The BB84 method allows joint selection of a key with no possibility that an intruder can learn the key. It has the added advantage that A and B can determine whether an eavesdropper has intercepted the particles and attempted to decode the message or attempted to modify the message before it was received by B. Suppose that E has intercepted the particles and measured their spins, then passed the particles on to B. With two spin axes, there is a 50% chance that she will have measured the wrong axis on any given particle. This error will show up as a "reset" of the correct axis and there will be a 50% chance that B's measurement of the correct axis (when A

and B agrred on that axis) will be wrong. So for 25 percent of the qubits E will have inserted an error and the message B decodes will be unintelligible. Thus, E announces her presence!

The BB84 method allows restoration of the simple One-Time Pad system with resolution of both the key distribution problem and the message authentication problem. But the same key (string of qubits measured by A) can not be reused. So for each message a new string of qubits must be sent. Inconvenient, but very feasible.

Quantum Information

In this section we review some important features of quantum information theory. What types of quantum information exist? How many quantum bits of information are needed to equal a classical information bit? Is quantum information "more efficient" than classical information?

Bits, Qubits, and Ebits: Bennett's Rules

There are three basic types of information: *bits* (classical 0, 1 data), *qubits* (quantum two-state particles) and *ebits*. Bits and qubits have been discussed above. The addition to information theory is the ebit.

An ebit is a quantum particle with total zero spin (TZS = 0). It might have z-spin, x-spin, *and* y-spin, but its total spin—the "average" of the three spins—is zero. An example is the photon—the particle that carries electromagnetic radiation. Even after a photon decays, the result—a positron and an electron—each having opposite characteristics, thereby maintaining TSZ. These are entangled particles because they share a single probability wave: if the z-spin of the electron is measured as ↑ the z-spin of the positron must be ↓.

We have also seen in Part 2 that entangled particles remain entangled even at great distances from each other. The example given was creation of a positron-electron pair with the electron sent to A and the positron sent 300,000 km (one light second) away to B. When the particles are received, their spins are unknown, but if A measures her electron's z-spin and finds that it is ↑ then B will, with certainty, find his positron's z-spin is ↓.

Charles Bennet, a leading information scientist and the co-founder of the BB84 key distribution method, has investigated the equivalences between bits, qubits, and ebits in terms of their ability to carry bits of information. Using the notation ≥ to mean "at least as much as," he proposes the following rules:

1. ***1 qubit ≥ 1 bit***: as we have seen, a qubit can always carry one bit of information.
2. ***1 ebit ≥ 1 qubit***: an ebit can carry at least as much information as a qubit.
3. ***1 ebit + 1 qubit ≥ 2 bits***. An ebit and a qubit carry at least as much information as two bits. This is called the *Dense Code Rule*; it is discussed below.
4. ***1 ebit + 2 bits ≥ 1 qubit.*** This is the *Quantum Teleportation Rule*, also discussed below.

Rule 1 is that a qubit can always carry at least one classical bit of information. A qubit's quantum state until it is measured is the superposition **a**| ↑ > + **b**| ↓ >. Suppose A wants to form the number "1" with his quantum computer using the code 1 = | ↑ > and 0 = | ↓ >. He can do this by measuring the spins of several particles and selecting one with an up spin. Once measured, the particle retains that spin unless it is remeasured on another axis, at which time a superposition returns to the first axis measured. So A can select a qubit with up spin on the z-axis, send it to B along with information on which axis was measured, and B can remeasure the spin on the z-axis, retrieving a 1. Thus, one qubit can carry one bit of information. In this way any binary sequence can be formed with a number of qubits equal to the number of binary digits required.

It would be more efficient to use one qubit to carry two (or more) bits of information. Suppose we use the quantum code 00 =|↑↑ >, 01 = |↓↓ >, 10 = |→ → > 10, and 11 = |← ← >; this requires two qubits. To create the number 10 (decimal 2) in a single qubit requires a particle whose quantum state is |→ → > = 10. In Part 2 we saw that a right spin is a superposition of up and down spins, that is, (|→ >) = √½(|↑ > + |↓ >). So by the Composite Product Rule two right spins |→>|→> is the square of one right spin, $|\rightarrow\rangle)^2$, or ½(|↑↑> + |↑↓> + |↓↑> |↓↓ >). Thus |→ → > is *not* |↓↓ >. The important lesson is that *a qubit can not carry more than one bit of information*.

Rule 2 is that an ebit can always carry at least one qubit of information. From Rule 1 we see that the if spins are orthogonal (in the sense defined above) then an entangled bit with z-spin has superposition ½(|↑↑> + |↓↓ >)—equivalent to two right spins and, therefore, giving the two bit code 10. So one ebit can carry *two* bits of information. An ebit appears to be "more efficient" that a qubit. But is it really more efficient? The answer is negative—the ebit itself consists of two qubits, which Rule 1 says *should* carry two qubits of information.

Bennett Rule 3 is that one qubit plus 1 ebit can carry at least two bits of information. This is the rule underlying Dense Coding, and we will see that it is correct but that *no more than two bits of information can be carried by a qubit and an ebit*.

Bennett Rule 4—that one ebit plus one qubit carries at least two classical bits is the foundation of Teleportation, which we cover after dense coding.

Before discussing these rules we need to solidify the notion of entanglement. This brings us to the maximal entanglement of qubits, called Bell States after the physicist John Bell.

Maximal Entanglement (Bell States)

Dense Coding and Teleportation, the subjects of Rules 3 and 4—and of the next two sections—rely on a particular form of entanglement—*maximal entanglement*—in which two qubits always show either the same or opposite values (spin, polarization, etc). We focus on the simple case of two entangled qubits.

Suppose that A owns qubit q_A and B owns qubit q_B, each qubit having basis states $| 0 >$ and $| 1 >$. With two qubits there are four joint states: $| 0_A0_B >$, $| 0_A1_B >$, $|1_A0_B >$, and $|1_A1_B >$, and the system is a superposition of these four unentangled states.

Now suppose the two qubits are maximally entangled. The result is that the system has four *Bell States*. The Bell states are the four quantum states $| B_0 >, \ldots, | B_3 >$ shown below.

$$| B_0 > = \sqrt{½}(| 0_A0_B > + |1_A1_B >)$$
$$| B_1 > = \sqrt{½}(| 0_A0_B > - |1_A1_B >)$$
$$| B_2 > = \sqrt{½}(| 0_A1_B > + |1_A0_B >)$$
$$| B_3 > = \sqrt{½}(| 0_A1_B > - |1_A0_B >)$$

State $| B_0 >$ can be interpreted as "if A's qubit is zero, then B's qubit is zero; if A's qubit is 1, B's qubit also is 1." In two of the states ($|B_0 >$ and $|B_1 >$) A and B hold qubits with the same spin, and in the other two Bells states A and b have qubits with opposite spin. Each Bell state is said to be maximally entangled because the qubit states are perfectly correlated—either they are both the same or they are both opposite.

Note that in both $| B_1 >$ and $| B_3 >$ the second joint state is an *antistate*. In Part 2 we discussed the formation of antistates by rotating particles around specific axes. The amount of rotation required to convert a basis state to its antistate depends on the type of particle—boson or fermion—and the axis of rotation. In any event, transforming to an antistate inverts the probability function.

Any of the Bell states can be derived from the others by suitable *Bell Rotations*. For example, $| B_1 >$ can be derived by a 180° rotation of $| B_0 >$ around its z axis, $| B_2 >$ can be derived by a 180° rotation of $| B_0 >$ around its x axis, and $| B_3 >$ can be derived by a 180° rotation of $| B_0 >$ around its y axis.

The four Bell states can also be added or subtracted to obtain the following *Bell Additions*

$$| B_0 > + | B_1 > = \sqrt{1/2}(| 0_A 0_B >)$$

$$| B_0 > - | B_1 > = \sqrt{1/2}(| 1_A 1_B >)$$

$$| B_2 > + | B_3 > = \sqrt{1/2}(| 0_A 1_B >)$$

$$| B_2 > - | B_3 > = \sqrt{1/2}(| 1_A 0_B >)$$

Thus, each possible state of the two qubits can be derived from addition or subtraction of Bell States, As we will see, much can be done with Bell States, Bell Rotations, and Bell Additions.

Bell states have a number of desirable properties. Bell states are *mutually orthogonal*, meaning that the chance of $| B_2 >$ occurring does not depend on whether another Bell State occurs. This is why Bell States are maximally entangled: there can be entangled states in which two states are entangled so that the chances of those two states occurring are only loosely correlated; in that case you can't definitively know one state from knowledge of the other. But with Bell States you can know with certainty what the states are.

Another nice property is that Bell states can be copied. This comes directly out of their orthogonality. Thus, the tensor product $| B_i >^2$ creates two qubits with state $| B_i >$; the No Cloning Theorem does not apply. Another desireable feature is that *Bell states can be measured without decoherence (superposition collapse)*. This allows the state of an ebit to be determined without destroying the ebit.

These properties underlie the central role that Bell states play in Dense Coding and Teleportation in which qubits and Bell state ebits are combined to achieve certain goals that are otherwise impossible under the laws of quantum mechanics.

<u>Dense Coding</u>

The goal of Dense Coding is to overcome Bennett's Rule 1 by allowing one qubit to send two classical bits of information,. But dense coding has a somewhat strange procedural component requiring a great deal of preparation and communication between A and B, and its advantages depend on the congestion and expense of operating a quantum communications channel to transmit/receive qubits.

Suppose that A and B work in adjacent offices in the SCA (SpyCraft Agency). Each has their own qubit with unknown states $|q_A > = \mathbf{a}_A|0 > + b_A|1 >$ and $|q_B > \mathbf{a}_B|0 > + \mathbf{b}_B|1 >$. A and B

get together to create an ebit in a specific Bell State, say $| B_0 >_{AB} = \sqrt{1/2}(| 0_A0_B > + |1_A1_B >)$; thus, q_A and q_B have been fashioned to always have the same spin direction. Both A and B know that 180° rotations along an appropriate axis will convert this Bell state to any other Bell state.

A and B have agreed on a two-bit code based on Bell states,

$$| B_0 >_{AB} = \sqrt{1/2}(| 0_A0_B > + |1_A1_B >) \quad \Rightarrow \quad 00$$
$$| B_1 >_{AB} = \sqrt{1/2}(| 0_A0_B > - |1_A1_B >) \quad \Rightarrow \quad 01$$
$$| B_2 >_{AB} = \sqrt{1/2}(| 0_A1_B > + |1_A0_B >) \quad \Rightarrow \quad 10$$
$$| B_3 >_{AB} = \sqrt{1/2}(| 0_A1_B > - |1_A0_B >) \quad \Rightarrow \quad 11$$

A and her qubit are sent off to do field work in a galaxy far far away, armed with her qubit q_A. B stays at headquarters with his qubit q_B. They have a quantum channel through which they can communicate via qubits. At some point A wants to send the message 01 to B. To do this she wants to send her qubit q_A to B with characteristics that will tell B that $| B_1 >$ is the relevant Bell state, i.e. 01 is the two-bit message.

A puts q_A into her digital circuits and rotates it on an appropriate axis. Because q_A is entangled with q_B, and because they were entangled so that they would have the same spin directions, her rotation of q_A induces the same rotation in q_B even though it is in that galaxy far far away. A knows that if she wants to send 00 she does nothing to her qubit, if she wants to send 01 she rotates it x-axis by 180°, to send 10 she rotates the y-axis by 180°, and to send 11 she rotates the z-axis by 180°. So A rotates q_A on its x-axis by 180° and sends it to B. B doesn't know what the spin state is, but he puts q_A into his Bell State Machine along with q_B. He determines that the two qubits have the same spin direction, so the message is either $|B_1 >$ or $|B_1 >$. He also finds that the Bell state involves an antistate, so the message must be $|B_1 >$. Voila! Message received!

So sending one qubit has done the work of two qubits, breaking Bennet's Rule 1. But has it really? Recall that the dense code process began with two qubits (q_A and q_B) so it really took two qubits (plus a lot of intermediate fiddling) to send two classical bits of information. However, only one qubit needed to be *transmitted* from the field to send those two bits. If the quantum channel used to transmit qubits is congested, dense coding could be an improvement. But if use of the quantum channel is free, dense coding appears to have no advantage over

simply sending two qubits with appropriate spins. However, Bennett's Rule 3 is satisfied: it took one qubit plus one ebit to *transmit* two bits of information. One can't do better.

Quantum Teleportation

Quantum Teleportation is the converse of Dense Coding: instead of using a qubit to send a two-bit message, it uses a two-bit message to send a qubit. Recall that the No Cloning Theorem says that you can't make a copy of a qubit *and* keep the original. Teleportation allows transmitting a qubit in a fashion that destroys the original qubit, thus bypassing the No Cloning issue.

The qubit could, of course, simply be transmitted from A to B via Qubit Xpress, a quantum channel that allows A and B to directly communicate. This would be subject to the law of Special Relativity: nothing can travel faster than light speed. The advantage of teleportation is that it happens instantly: the qubit is not "sent," it is created instantly at B's location.

The Teleportation protocol starts from the Dense Coding protocol: A and B each have a qubit: A has q_A with unknown state $| q_A > = \mathbf{a}_A| 0 > + \mathbf{b}_A| 1 >$; B has q_B in unknown state $| q_B > = \mathbf{a}_B| 0 > + \mathbf{b}_B| 1 >$. A also has a second qubit, q_C, with state $|q_C > = \mathbf{a}_C| 0_C > + \mathbf{b}_C| 1_C >$.

Step 1 in the teleportation scheme occurs at headquarters. q_A and q_B are maximally entangled to create an ebit in one of the four Bell states; let's say that the chosen Bell state is $| B_0 >_{AB} = \sqrt{½}(| 0_A0_B > + |1_A1_B >$. A and B then reclaim their two (now entangled) qubits. At this point A has two qubits (q_A and q_C) and B has qubit q_B. After Step 1 there is a system of three qubits: q_A and q_B are entangled; q_C is not entangled but is still part of the quantum system. The three cubit system has a quantum state described as the tensor product of $| B_0>_{AB}$ and $|q_C >$.

Step 2 occurs after A goes into the field (to that galaxy far far away) with her two qubits, leaving B at headquarters with his qubit q_B. After settling into her new lab, A learns that B's lab is experiencing a shortage of type-C qubits. So she decides to "send" B a cubit with the exact characteristics of her extra qubit, q_C. Of course, there is a problem: she doesn't know those characteristics, and if she tries to measure them the superposition of states in q_C collapses. She might send it by Qubit Express, but that is ruled out. What is a SpyCraft agent to do?

A decides to entangle her two qubits in one of the four Bell states $|B_0 >_{AC}$, $|B_1 >_{AC}$, $|B_2 >_{AC}$, or $|B_3 >_{AC}$. By entangling q_C with q_A—which is already entangled with q_B—she has very cleverly entangled q_C with q_B! The resulting three qubit state is the tensor product

$$| q_A q_B q_C > = \sqrt{½}\{|B_0>_{AC} \otimes (\mathbf{a}_B| 0_B > + \mathbf{b}_B| 1_B >) + |B_1>_{AC} \otimes (\mathbf{a}_B| 0_B > - \mathbf{b}_B| 1_B >)$$
$$+ |B_2>_{AC} \otimes (\mathbf{b}_B| 0_B > + \mathbf{a}_B| 1_B >) + |B_3>_{AC} \otimes (\mathbf{b}_B| 0_B > - \mathbf{a}_B| 1_B >)\}$$

so A knows that the new three qubit state is a superposition of one of the following four equally probable states:

$$| B_0>_{AC} \otimes (\mathbf{a}_B| 0_B > + \mathbf{b}_B| 1_B >)$$
$$|B_1>_{AC} \otimes (\mathbf{a}_B| 0_B > - \mathbf{b}_B| 1_B >)$$
$$|B_2>_{AC} \otimes (\mathbf{b}_B| 0_B > + \mathbf{a}_B| 1_B >)$$
$$|B_3>_{AC} \otimes (\mathbf{b}_B| 0_B > - \mathbf{a}_B| 1_B >)\}$$

A uses her Bell State Machine to measure the Bell state in which q_A and q_C are entangled—remember that Bell states can be measured without decoherence. So now she knows which of the four states the system is in. Suppose it is the third: $B_2>_{AC} \otimes (\mathbf{b}_B| 0_B > + \mathbf{a}_B| 1_B >)$.

Having gotten all the information she needs, A measures q_A and q_C, causing their superpositions to collapse and effectively destroying them (why is this necessary?). If she can tell B that $B_2>_{AC} \otimes (\mathbf{b}_B| 0_B > + \mathbf{a}_B| 1_B >)$ is the relevant state, he can take q_B and put it into his Bell State Machine along with a "blank" D-type qubit, q_D, and reverse the operations to give q_D the exact characteristics of q_C—he will have created an exact copy of q_C! And he will have it instantly because it never had to be "sent."

But how can she get the information to B? The only way is to use a classical communication channel to send two bits of information to B. Those two bits are 11, indicating state 3. Thus, one qubit has been "sent" without mailing it. To do this two qubits were destroyed (q_A and the "real" q_C , which formed the ebit in A's lab) and two classical bits were sent. Bennett Rule 4 is satisfied: *one ebit plus two classical bits created one qubit*. And the No Cloning Restriction is also satisfied since q_C was created in B's lab but its original was destroyed.

This is amazingly clever, but there is less here than meets the eye. When would this ever be the best way to send a qubit? First, it saves time relative to Qubit Express only because the classical communication can occur at the speed of light, while a physical spacecraft can not. But what if Qubit Express is simply a quantum communication channel that also operates at light speed. Then no time is saved—A spends less time on the quantum channel but uses all of that

saving on the classical channel. The only advantage appears to be when the quantum channel is congested or in some other way (perhaps price) more expensive than the classical channel.

Quantum Teleportation does work: it has been used to "send" qubits ozens of miles. But the state of that science is very rudimentary and, of course, without practical value at this stage.

Appendix
Computing with a 3-Qubit Quantum Computer

As an example of a quantum computer with three color-coded qubits, q, q, and q. The qubit states are $|\psi> = a_0|0> + a_1|1>$, $|\psi> = b_0|0> + b_1|1>$, and $|\psi> = c_0|0> + c_1|1>$. The squared amplitudes (probabilities) are normalized to sum to 1, so $a_0^2 + a_1^2 = 1$, and $b_0^2 + b_1^2 = 1$, and $c_0^2 + c_1^2 = 1$.

The Setup

The two-qubit input register is loaded with q and q having states $|\psi>$ and $|\psi>$. The one qubit output register is loaded with **q**, having state $|\psi>$. The basis vectors (kets) defining the *computational basis* of the system are

$$|0> = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{and} \quad |1> = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

so that $|\psi> = a_0|0> + a_1|1> = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$ $|\psi> = b_0|0> + b_1|1> = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$

and $|\psi> = c_0|0> + c_1|1> = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$

The tensor product of the two-qubit input register is the 2x2 matrix

$$|\psi>\otimes|\psi> = \begin{bmatrix} |0>\otimes|0> & |0>\otimes|1> \\ |1>\otimes|0> & |1>\otimes|1> \end{bmatrix} = \begin{bmatrix} a_0b_0 & a_0b_1 \\ a_1b_0 & a_1b_1 \end{bmatrix}$$

and the tensor product of the three qubits is the 2x4 matrix

$$|\psi>\otimes|\psi>\otimes|\psi> = \begin{bmatrix} |0>\otimes|0>\otimes|0> & |0>\otimes|1>\otimes|0> & |0>\otimes|0>\otimes|1> & |0>\otimes|1>\otimes|1> \\ |1>\otimes|0>\otimes|0> & |1>\otimes|1>\otimes|0> & |1>\otimes|0>\otimes|1> & |1>\otimes|1>\otimes|1> \end{bmatrix}$$

$$= \begin{bmatrix} a_0b_0c_0 & a_0b_1c_0 & a_0b_0c_1 & a_0b_1c_1 \\ a_1b_0c_0 & a_1b_1c_0 & a_1b_0c_1 & a_1b_1c_1 \end{bmatrix}$$

The basis vectors for the tensor product spaces are the

for the two-qubit inputs:

$$|0\rangle\otimes|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad |0\rangle\otimes|1\rangle = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$|1\rangle\otimes|0\rangle = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad |1\rangle\otimes|1\rangle = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

for the 3-qubit register

$$|0\rangle\otimes|0\rangle|0\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad |0\rangle\otimes|1\rangle|0\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$|0\rangle\otimes|0\rangle|1\rangle = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad |0\rangle\otimes|1\rangle|1\rangle = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$|1\rangle\otimes|0\rangle|0\rangle = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad |1\rangle\otimes|1\rangle|0\rangle = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$|1\rangle\otimes|0\rangle|1\rangle = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad |1\rangle\otimes|1\rangle|1\rangle = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The tensor products contains all of the possible basis states for the two- and three-qubit systems. If you knew the amplitudes you can immediately identify the numbers and do any calculations. For example, using only q and q, decimal 4 = binary 10 + 11 = $a_1b_0 + a_1b_1$. Unfortunately, the amplitudes are unknowable until the system has been measured—and measurement destroys the information.

A Simple Quantum Calculation: *Modulo-2* Addition

Step 1: Initialize Qubits

Before any calculation the input register qubits q and q, and the output qubit q, must be refreshed to purge them of any residue from previous calculations. This is done by setting the three qubits to some arbitrary basis state, say "spin -½ or |0 >, |0 >, and |0 >.

Step 2: Rotation via Quantum Gates

Any quantum calculations require passing the qubits q, and q and q through one or more *quantum gates*, just as we would pass bits through logical gates in a classical computer; the result is new states for the three qubits. But quantum gates are not electrical circuits, they are matrices called *unitary operators* that simultaneously rotate the just-initialized qubits to put them into suitable states for the next stages of the calculation.

The first quantum gate is an R Gate. This is a unitary matrix that takes q and q and transforms them to superpositions with known amplitudes √½. Its effect is shown below

R GATE

IN	*OUT*
|0 >	√½(|0 > + |1 >)
|1 >	√½(|0 > - |1 >)
|0 >	√½(|0 > + |1 >)
|1 >	√½(|0 > - |1 >)

Thus, the single-qubit state values are transformed to superpositions with amplitude √½ and with antistates (180° rotations) for |1 >.

The three-quibit register then has the following four tensor product states

½(|0 >⊗|0 >⊗|0 >)

+ ½(|0 >⊗|1 >⊗|0 >)

+ √½(|1 >⊗|0 >⊗|0 >)

+ ½(|1 >⊗|1 >⊗|0 >)

Step 3: Entanglement

The superposition just derived consists of tensor products of independent quantum states: each qubit's state is independent of the states of the other qubits. As explained in the text, entanglement exists when some of the states are orthogonal, meaning that they are perfectly correlated. For example, if $|0\rangle \otimes |1\rangle = 0$ and $|1\rangle \otimes |0\rangle$ then the tensor product $|\psi\rangle \otimes |\psi\rangle$ is

$$|\psi\rangle \otimes |\psi\rangle = \begin{bmatrix} |0\rangle \otimes |0\rangle & 0 \\ 0 & |1\rangle \otimes |1\rangle \end{bmatrix}$$

Thus, the only possible product states are perfectly correlated: either both qubits are 0 or both are 1. The states are said to be *maximally entangled*.

Maximal entanglement of the three qubit states is done via a D Gate, which is a unitary matrix that does the following operation

D GATE

IN	*OUT*
$\|0\rangle \otimes \|0\rangle \otimes \|0\rangle$	$\|0\rangle \otimes \|0\rangle \otimes \|0\rangle$
$\|0\rangle \otimes \|0\rangle \otimes \|1\rangle$	$\|0\rangle \otimes \|0\rangle \otimes \|1\rangle$
$\|0\rangle \otimes \|1\rangle \otimes \|0\rangle$	$\|0\rangle \otimes \|1\rangle \otimes \|1\rangle$
$\|0\rangle \otimes \|1\rangle \otimes \|1\rangle$	$\|0\rangle \otimes \|1\rangle \otimes \|0\rangle$
$\|1\rangle \otimes \|0\rangle \otimes \|0\rangle$	$\|1\rangle \otimes \|0\rangle \otimes \|1\rangle$
$\|1\rangle \otimes \|0\rangle \otimes \|1\rangle$	$\|1\rangle \otimes \|0\rangle \otimes \|0\rangle$
$\|1\rangle \otimes \|1\rangle \otimes \|0\rangle$	$\|1\rangle \otimes \|1\rangle \otimes \|0\rangle$
$\|1\rangle \otimes \|1\rangle \otimes \|1\rangle$	$\|1\rangle \otimes \|1\rangle \otimes \|1\rangle$

The D-Gate takes each of the eight ($= 2^3$) possible register states and performs the following: *if the input state state of q is* $|0\rangle$ *then the output state of q is the mod-2 sum of the input states of q and q; if the input state state of q is* $|1\rangle$ *then the output state of q is 1 plus the mod-2 sum of the input states of q and q (recall that 1 + 1 = 0).*

But only four of those eight output states are relevant—those in which **q**'s input state was $|0\rangle$. The four output states with $|1\rangle$ as qs's initial state can't occur because **q** was initialized to $|0\rangle$. So after both the D Gate and the R Gate the state of the register is

$$\tfrac{1}{2}(|0\rangle\otimes|0\rangle\otimes|0\rangle)$$
$$+\tfrac{1}{2}(|0\rangle\otimes|1\rangle\otimes|1\rangle)$$
$$+\tfrac{1}{2}(|1\rangle\otimes|0\rangle\otimes|1\rangle)$$
$$+\tfrac{1}{2}(|1\rangle\otimes|1\rangle\otimes|0\rangle)$$

In each of these four tensor product states **q** contains the modulo-2 sum of *q and q*. The four states are said to be entangled *mod-2*, meaning that the third qubit must always be the *mod-2* sum of the other two qubits.

Step 4: Read the result

The calculation has computed the mod-2 sum of every possible combination of *q and q*. The combinations all exist simultaneously, each having probability ¼ of being observed when the register is measured. Suppose you read the register and find **0 >**, then you know that the other two qubits are either 1 and 1 or 0 and 0; if you get **1 >**, then the other qubits are either 1 and 0 or 0 and 1. It turns out that 50 percent of the time a measurement would yield **0 >** and 50 percent of those times it would also indicate 1 and 1.

This is a "calculation," but it is a very trivial calculation because you knew the answer without using a computer. It illustrates the steps in a quantum calculation without indicating the power of a quantum computer. Perhaps more useful would be the answer to the question "if a specific string of 0's and q's is added *mod-2* to another specific string of 0's and 1's, what would be the sum?" That answer can be derived by multiple-qubit quantum computers that iterate (use repeated trials) to get an answer. It has been shown that the time to do any binary arithmetic calculation is a low-order polynomial function of the number of binary digits; with a classical computer the time increases exponentially with the number of iterations.

My apologies to Richard Feynman!

References

Altepeter, Joseph B. "A Tale of Two Qubits: How Quantum Computers Work," Google *Ars Technica* Blog, 2010.

Cox, Brian and Jeff Foreshaw, *The Quantum Universe*, DeCapo Press, 2012

Dahlsten, Oscar. "An Introduction to Entanglement [Transfer Essay I]," 2005 *(www.phys.ethz.ch/~dahlsten/entanglementintro.pdf)*

Feynman, Richard. *Quantum Electrodynamics*, Princeton University Press, 1985.

Feynman, Richard. "Quantum Mechanical Computers," *Foundations of Physics*, Vol. 16 No. 6, 1986.

Gleick, Paul. *The Information: A History, A Theory, A Flood.* Pantheon Books, 2011.

Griffiths, David. *Introduction to Quantum Theory*. Pearson Publishing, 2005.

Phillips, Tony. "The Mathematics Behind Quantum Computing, Part II," American Mathematical Society *(www.ams.org/samplings/feature-column/fcarc-quantum.two)*

Schumacher, Benjamin. *Quantum Mechanics: The Physics of the Microscopic World*, Lectures at Kenyon College, The Great Courses DVD Series.