Quantum Computing

Leon and Mark

Contents

§1 Introduction

If computers that you build are quantum,
Then spies everywhere will all want 'em.
Our codes will all fail,
And they'll read our email,
Till we get crypto that's quantum, and daunt 'em.
– Jennifer and Peter Shor

§2 Shor's Algorithm

Currently the fastest classical prime factoring algorithm is the general number factoring sieve with a runtime of approximately $\exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}\right) = L_n[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}]$. One the other, Shor's Algorithm has a runtime slightly slower than $O(n^3)$[1]

§2.1 Shor's Algorithm

In this article I will present how Shor's Algorithm backwards from how is it usually presented. Instead of introducing the tools then using the tools to construct Shor's Algorithm, I will use elements of the final alogrithm to motivate the each part.

Shor's Algorithm looks to factor numbers of the form $N = a \cdot b$ where a and b are

[1]optimizations can get this down to $\mathrm{O}(\mathrm{n}^2 \ln n \ln\ln n)$

prime. It turns out that RSA is most secure when a and b are both prime so N can be as large as possible. If a or b were instead factorable themselves into smaller primes, elementary number theory would easily be able to use the smallest prime factor to find the rest of the factors and break the key.

We begin by guessing a number g. If g is factor of N, we're down. If g shares a factor with one of the factors of N, then we can run Euclid's Algorithm to find the GCD which runs very fast relative to factoring itself on a classical number and we are done.

However, it is extremely unlikely that our guess g will be either a factor or share factors with a factor of N. Shor's algorithm gives a way to turn our crappy guess g to one that is much more likely to share factors with N and here is how.

§2.2 Quantum Fourier Transform

So how exactly do we find this phase. You may remember that the Fourier Transform can be used to decompose sums of sine waves into the frequencies of each of the individual sine wave. The Quantum fourier transform (QFT) is in turn a version of the Discrete Fourier Transform (DFT). Instead of sampling the entire function, DFT samples our function at regularly spaced intervals to approximate the frequency of the underlining functions.

The DFT is defined that for the vector input $x = (x_0, x_2, x_3 \ldots x_{n-1}$ the output $y = (y_0, y_2, y_3 \ldots y_{n=1}$ is where n is the number of elements. In our case, each element of x and y is a bit that is either 0 or 1. Thus, x and y are n-bit strings that can represent 2^n numbers. We then define $N = 2^n$ and for simplicity we will only deal with unsigned positive integers. We thus define the DFT by

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i jk/N} \tag{1}$$

We use our definition of the DFT to define the QFT in an analogous way. For a quantum input state $|j\rangle = |j_0 j_1 j_2\rangle \ldots j_{n-1}$ which represent the individual state of the qubits that make up $|x\rangle$ and a similar defined quantum output state $|k\rangle = |k_0 k_1 k_2 \ldots k_{n-1}\rangle$, the QFT is defined by

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i jk/N} |k\rangle \tag{2}$$

This can be written simply as

$$\sum_{j=0}^{N-1} x_j |j\rangle \mapsto \sum_{k=0}^{N-1} y_k |k\rangle \tag{3}$$

with a bit of algebra we can show that

$$
\begin{aligned}
|j\rangle \to \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi ijk/2^n} |k\rangle && (4)\\
= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi ij(\sum_{l=1}^{n} k_l 2^{-l})} |k_1 \ldots k_n\rangle && (5)\\
= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} e^{2\pi ijk_l 2^{-l}} |k_l\rangle && (6)\\
= \frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} \left[\sum_{k_l=0}^{1} e^{2\pi ijk_l 2^{-l}} |k_l\rangle \right] && (7)\\
= \frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} [|0\rangle + e^{2\pi ij 2^{-l}} |1\rangle] && (8)\\
= \frac{1}{2^{n/2}} [(|0\rangle + e^{2\pi ij/2}) |1\rangle) \otimes (|0\rangle + e^{2\pi ij/2^2} |1\rangle) \otimes \ldots] && (9)
\end{aligned}
$$

This defintion of the QFT is so useful you may consider it to be the defintion of the QFT.

Like all quantum gates, the QFT is unitary. This proves to be important as it means its eigenvalue is real. Remember the eigenvalue is used to estimate the phase. This will be important later on.

§3 Post-Quantum Safety

With the discovery of Shor's algorithm and the invention of quantum computers, it is reasonable to question if there will be defenses against it. Currently, the encryption methods that are used are quantum proof because even with the best implementation of Shor's algorithm, it can only calculate the factor of numbers that is a few digits long using a couple quantum bits. This is not even close to enough to decrypt some of the smaller numbers used in encryption. However, there are research on how we can better our encryptions against the threat of quantum computers. There are four different methods.

§3.1 Lattice-Based Encryption

Lattice is defined as any graph of regular square points that goes to infinity. A vector is simply a point and it will have 2 coordinates on a 2-dimensional plane and 3 coordinates in the 3-dimensional space. If a vector is "long" from the origin, the vector is far from the origin and if it is "short" from the origin, the vector is close to the origin. Since lattices are infinite, but computers have a finite storage space, we need to use a basis of a lattice.a small amount of vectors that can represent the lattice. In order to generate one of these lattices, pick two points that would not be on the same line going through the origin. Next, choose 2 whole numbers and multiply the first number with the first coordinate and the second number with the second coordinate. Put the two coordinates together and you will get the third point and rinse and repeat. With this method, we can generate a whole grid or lattice of evenly spaced points. However, a given lattice can have multiple starting pairs of coordinates. For example, a lattice created using the points (2, 0) and (0,2) can be generated with (-2, 0) and (0, -2) or (4, 2) and (2, 2) as well. A basis that is short consists of short vectors and a basis that is long consists of long vectors. The short bases are the more useful one in lattice problems. One of the most important problems in lattice-based cryptography is the "Short Vector Problem." This problem gives us a long basis in lattice L and asks us to find a grid point in L that is as close to the origin as possible. What makes this question difficult is that we are given a basis with long vectors so it's not easy to find which short vectors will result in the long vector. In addition, because it is in cryptography, these lattices will be in higher dimensions so they will have tens of thousands of coordinates rather than just two or three coordinates.

§3.2 Multivariate Cryptography

This type of cryptography includes cryptographic systems such as the Rainbow Scheme, also called the Unbalance Oil and Vinegar(UOV) scheme. This is a modified version of the Oil and Vinegar Scheme designed by J. Patarin. This cryptography's security relies on an NP-hard mathematical problem. To create and validate signatures a minimal quadratic equation system has to be solved. Solving m equations with n variables is very difficult for quantum computers to be able to solve.

As stated before, the UOV relies on solving m equations with n variables, with the equation system as the public key. Because a standard computer can not process n variables in a reasonable time and would need a lot of resources to do it. This is where the private key come into play. The key consist of three parts: two Affine transformations T and S and polynomial vector P. The transformations are used to transform elements in certain groups. The P provided certain tools for the equations creation.

To create a valid signature, a list of y, which will be our message that needs to be signed, that each y value in the list will be obtained by running functions on our valid signature which will be our list of x values. In order for a given y to be signed. a series of steps need to be performed on it. First, the message needs to be transformed to fit the equation system which is what our T will be used for. It will spilt the message into acceptable pieces which will be a list of y. Next the equation will need to be build and every equation will have the same form:

$$y_i = \sum \gamma_{ijk}\alpha_j\alpha_k^{'} + \sum \lambda_{ijk}\alpha_j^{'}\alpha_k^{'} + \sum \xi_{ij}\alpha_j + \sum \xi_{ij}^{'}\alpha_j^{'} + \delta_i \tag{10}$$

The next steps sign the given message y and the valid signature, x, will be the result. First the coefficients $(\gamma_{ijk}, \lambda_{ijk}, \xi_{ij}, \xi_{ij}^{'}, \delta_i)$ must be chosen secretly. The vinegar variables $(a_j^{'})$ are chosen randomly. The result linear equation system gets solved for the oil variable (a_i). The variables create the pre-signature $A = (a_1...., a_n, a_1^{'}, a_v^{'})$. Finally A get transformed by the private affine transformation S to x.

In order to valid a signature, we need use a equation system that is only a slightly modified version of the one we used to create the signature. Only this time every equation has to solved in order for a signature to be valid. If an attacker is tries to break the encryption, they would be able to do any thing because of the fact that they do not know the oil and vinegar variables and the secret coefficients. If the y_i is equal to the corresponding part of the original message then the verification is done.

However, one drawback is the key-length. Because the public key is literally a whole equations system that can take up spaces up to some kilobytes, if someone wants to verify a signature they need to all the equations in order to compute the encrypted message and compare that to the original message.

§3.3 Hash-based Cryptography

Hash-based cryptography uses an input string and produces a fixed-size output. There are multiple security requirements to make a cryptographic hash, but we will focus on three here.

One of these is preimage resistance which is to make the output time-comsuming to find that the input is so that attackers cannot just brute-force their way in. A second-preimage resistance is to make sure that there is not a second input that can result in the output. The last one is collision resistance which make sure that the when a function is applied to to the input, the result should not be able to be reached by passing a different number through the function. In other words, f(x) should be be able to equal to f(h)

The first hashed-based signature scheme was created in 1979 by Leslie Lamport. He found that by using a one-way function it is possible to create a powerful signature scheme. For example, if we want to sign 256 bit messages, we need a secret key and a private key. We first generate 512 separate random bit strings and arrange them into two separate lists and will be referred to by their index and is what we will be signing with. To create the public key, we will hash each of those random string with the function H(•) which will turn the two secret key lists into two public key lists. The public key can be shared or whatever we want to do with it. Now if we want to sign something with the secret key, we take the index of each bit in the message and correspond it with either the index of a bit in the first secret key list of the index of a bit in the second secret key list. Once we finish assigning, we can concatenate the result and the message would be encrypted. If the message gets to the intended recipient, then they would be able to just take the message and hash everything back and match it with which ever list or index it is suppose to be.

This methods seems alright but one crucial problem is that this method can only be able to sign one message. This caused by the fact that the signatures shows one of the two secret key values at each position. This way if two messages were signed and differs at any point it would reveal the secret key for the position. This would allow someone else to sign a message that the original owner did not sign and how much they would be able to affect the message depends on the amount of messages the attacker has to work with.

So how can we solve this problem of the inability to sign multiple message? One idea to this problem is to just increase the length of the secret key and public key every time we sign a new message. This method is obviously terrible because of how large the keys will end up being if this method is used. Here is where Merkle comes in.

He proposed that first to generate n amount of Lamport keypairs. Next place each public key at one leaf of a Merkle Hash tree and calculate the root of the tree. The root will serve as the master public key. To get this root, we take the two lists of public keys and concatenate the result after hash each of them. Repeat this step for another two lists and finally, concatenate the result after hashing each of the new lists and the result of this will be the root of the tree.

Basically how it works is that Computer A sends a hash file to Computer B. Computer B would then check the file against the root of the tree. If there is no differences, then it is all good. Otherwise, the computer will requests the roots of the subtree of the hash. Computer A would create the request hash and sent it back to Computer B. The last two steps would repeat until it matches.

As said before Merkle's method still has the downside of getting too big. However, what if we just do not sign all of the messages? What if we just sign the message bits that equal to 1. This would cut the space needed by half due to an entire list being rid off. However, this is not a very good method because if someone wanted to change a signed message from "1111" to "0000", they would just have to delete some parts of the signature. To fix this hole, we can just check the number of zero bits in the message. If the number of zero bits does not match the original number of zero bits, the signature would be rejected. The checksum is in binary and is also signed with the message so messing with the checksum would not be possible. The method above does a good job at reducing the size, but some signatures and public key size are still thousands of bits long.

Robert Winternitz proposed an idea: what if, instead of signing with bits we use bytes, which can store 8-bits of information. His idea was that since storing and distributing 265 random lists is too storage consuming, what if we only generate the lists when we need them? This method required us to first generate a list that will serve as the initial secret key. To get the next list, we just hash each element forward to get the next set of secret key and so on and so forth. To get the public key, Winternitz suggested that the public key would just be the last list or the 256th list. This method is very clever way that allows the fact that any given secret key, if we hash it forward, we would get the public key. The way we would use this is that we check the value of the message byte and we would use the corresponding secret key list. To prevent forgeries, Winternitz came up with another trick. The Winternitz checksum works by summing the difference between 255 and the value of each message bytes signed. The sum would also be in base-265 and would be added to the message to be signed. For example, if a message was (0, 0, 0, 0) the checksum would be 1020 which in base-256 is (3, 252).

Even though these methods were developed in the late 1900s, most hash-base signatures are not vulnerable to Shor's algorithm. It could can lower the effectiveness of the security but simply increasing the capacity and output size of the hash function.

§3.4 Supersingular Elliptic Curve Isogeny Cryptography

Elliptic curve cryptography is similar to the RSA in terms of security except elliptic curve cryptography used fewer bits. Using ecliptic curve required less power, smaller chip size, and increase in speed. There is two type of elliptic curve cryptography.

The Diffie-Hellman Key Exchange Encryption works as followed. Say two people want to make a common key so that they can exchange messages secretly. Assuming that the only way the two people had not contact each other beforehand and can only contact through a public channel. Now we use Diffie-Hellman Key Exchange to get the private key. The two people would agree on an elliptic curve, E over a finite field, $\mathbf{F}_q$ such that the discrete logarithm problem is contained in $E(\mathbf{F}_q)$. A point P that is contained on $E(\mathbf{F}_q)$ such that the subclass generated by P has a large order. These are usually chosen such that the order is a large prime number. Person A would chose a secret number a

and compute aP and sends that to the other person. Person B would do the same so that Person A would get bP and Person B would have aP. They would apply their secret number to the recently obtained number and they both would get abP and now would use abP to create a key.

Let's say there is a third person that has been listening to the conversation. The only information this person would have is the curve E, the finite field, $\mathbf{F}_q$, the points P, aP, bP. They would have to use these information to compute what abP. The only problem that is there is no way of doing it without solving the discrete log problem for our curve. There is a proof to show abP is hard to find, but it is quite complicated.

The second elliptic curve cryptography is the Massey-Omura Encryption. Say the two people wanted to make another encryption method. They would agree on another elliptic curve E over a finite field, $\mathbf{F}_q$ such that the discrete logarithm problem is contained in $E(\mathbf{F}_q)$ and lets say N = $E(\mathbf{F}_q)$. Basically Person A will choose a point in $E(\mathbf{F}_q)$, M, to hide their message. They will then choose a secret number, m_A, where the greatest common multiple between m_A and N is 1. They will find the value for m_AM and send that to Person B. Person B would choose his own secret number m_B, where the greatest common multiple between m_B and N is 1 and calculate m_Bm_AM. They would sent this back to Person A where they would calculate m_a^{-1} and be sent back to person B. Person B would calculate m_b^{-1} and reads the message. In simpler terms, suppose Person A places a message in a box and locks it with their lock and give it to Person B. Person B would put their own lock on it and give it back to Person A who will open their lock and send it back again. Now the only lock on the box is the one Person B had placed so they can just open the box and read the message.

Although cryptography using elliptic curve has been around for 20 years and is considered the best concept of cryptography, but there are some problems with this cryptography method. One major roadblock has been the intellectual property environment surrounding elliptic curves that prevented the implementations and use. However, as we go into the future, it would be used more and more and will become a part of our normal encryption methods.

§4 References

Oceania, E. (2020, November 30). Beating the code breakers: How quantum computing changes everything. Retrieved April 01, 2021, from https://www.ey.com/en$_au/consulting/how-quantum-computing-changes-everything$