Intuitive Understanding of Quantum Computation and Post-Quantum Cryptography (Chapter 1)

Nguyen Thoi, Minh Quan [*†]

Abstract

Post-quantum cryptography is inevitable. National Institute of Standards and Technology (NIST) starts standardizing quantum-resistant public-key cryptography (aka post-quantum cryptography). The reason is that investment in quantum computing is blooming which poses significant threats to our currently deployed cryptographic algorithms. As a security engineer, to prepare for the apocalypse in advance, I've been watching the development of quantum computers and post-quantum cryptography closely. Never mind, I simply made up an excuse to study these fascinating scientific fields :) However, they are extremely hard to understand, at least to an amateur like me. This article shares with you my notes with the hope that you will have an *intuitive understanding* of the beautiful and mind-blowing quantum algorithms and post-quantum cryptography.

Contents

Introduction

To understand post-quantum cryptography, we have to understand quantum mechanics and quantum computers. Therefore, before studying post-quantum cryptography, I bought the classic book "Quantum Computation and Computation Information" by Michael Nielsen and Isaac Chuang [1]. The book was too advanced to me, so I took an excellent approachable quantum computation course by Umesh Vazirani [2]. In fact, I learned most quantum computation from Vazirani's course. Later on, I've realized that understanding quantum computers has nothing to do with understanding post-quantum cryptography. It was too late. I couldn't unlearn what I have learned. Therefore, I'll describe both quantum computers and post-quantum cryptography to make sure that you will make the same mistake as I did :) Joking aside, as a security engineer

[*] https://www.linkedin.com/in/quan-nguyen-a3209817, https://scholar.google.com/citations?user=9uUqJ9IAAAAJ, msuntmquan@gmail.com

[†] When asked, "How to say your name, Quan?", I answer, "It's the prefix of Quantum :)".

who is trained with "trust, but verify" mindset, I feel guilty to blind trust that Shor's quantum algorithms [3] break our current cryptographic protocols.

After "wasting" our time studying quantum algorithms, we'll study post-quantum cryptography. As Shor's algorithms solve factoring and discrete log problems in polynomial time, cryptographers had to find alternative cryptographic constructions that are presumably safe against quantum computers. The following cryptographic constructions are selected to advance to the 2nd round in NIST's post-quantum cryptography competition [4]: lattice-based cryptography, hash-based digital signature, code-based cryptography, multivariate public key cryptography and supersingular elliptic curve isogeny. I feel a headache just by reading these names :) They're independent of each other and each topic deserves its own research, so I'll describe them one-by-one in later chapters. They're are all difficult to understand, but the most challenging obstacle is to overcome our fear in dealing with them. No worries, if you can't understand them, blame me for not explaining them well :)

1 Quantum Computation

Have you ever played computer games? American player Kyle Giersdorf won 3 million on the Fortnite game. Computer games are ruling the world, so I recommend you stop reading this article, instead go and play games :) Computer games are a strange world where games' creators invent rules and players follow with no questions asked. In the same spirit, we'll follow quantum computers's rules, play along and design quantum algorithms based on its rules. The rules are strange but they're not stranger than computer games' rules. Furthermore, we'll study quantum computation without saying a word about quantum physics. I don't even try to understand quantum physics because I don't want to be crazy :)

1.1 Quantum computers

To describe a classical computational system, we define its state, how to change its state and how to measure its state. For instance:

- State: n bits $x = x_1, x_2 \cdots, x_n$ represented by n transistors.
- Classical logic gates such as NOT, AND, OR, NAND, etc are used to change $x_1, x_2, \cdots, x_n$.
- Measurement: measure the transistors, based on transistors' voltages, we'll get n output bits.

In a similar way, to describe quantum computational system, we'll define quantum state, quantum gates and quantum measurement.

1.1.1 Quantum state

In classical computers, a bit is either 0 or 1 at any moment. In quantum computers, a quantum bit (aka qubit) can exist at both states 0 and 1 at the same time. In fact, a qubit is a superposition of states $|0\rangle$ and $|1\rangle$: $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ where the amplitudes α_0, α_1 are complex numbers. On the one hand, the *ket* $|\rangle$ notation $|q\rangle, |0\rangle, |1\rangle$ just means quantum states, instead of classical ones. On the other hand, $|q\rangle$ denotes the state vector $|q\rangle = \begin{pmatrix}\alpha_0 \\ \alpha_1\end{pmatrix} = \alpha_0 |0\rangle + \alpha_1 |1\rangle$.

Is it strange that α_0, α_1 are complex numbers instead of real numbers? It's even stranger to learn that we never have access to α_0, α_1. As we'll see in the later section, when we measure $|q\rangle$, we'll get $|0\rangle$ with probability $|\alpha_0|^2$ and $|1\rangle$ with probability $|\alpha_1|^2$. I.e., we can observe these complex numbers' magnitudes (which are real numbers), but not the numbers themselves. Nature is mysterious!

Generalize the previous paragraphs, 2 qubits is a superposition of states $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, for instance, $1/2(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$. n qubits is a superposition of states $|00 \cdots 0\rangle$, $\cdots$,

$|11\cdots1\rangle$, i.e., $|q\rangle = \sum_{x=0}^{2^n-1} \alpha_x |x\rangle$. It's amazing that n qubits hold information of 2^n states at the same time. This makes quantum computers more powerful than classical computers.

1.1.2 Quantum gates

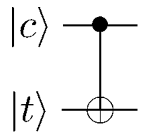
In classical computers, we use classical logic gates such as AND, NOT, OR, NAND to change bits' values. In quantum computers, we use quantum gates to change qubits. Recall that $|q\rangle$ denotes state vector $|q\rangle = \begin{pmatrix}\alpha_0\\ \alpha_1\end{pmatrix} = \alpha_0|0\rangle + \alpha_1|1\rangle$. To transform 2×1 column vectors, we will use 2×2 matrix. Therefore, we can describe quantum gates in the form of matrices.

Bit flip gate X Bit flip gate $X = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$. We have $X|q\rangle = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}\alpha_0\\ \alpha_1\end{pmatrix} = \begin{pmatrix}\alpha_1\\ \alpha_0\end{pmatrix}$, i.e., it transforms $|0\rangle$ into $|1\rangle$ and $|1\rangle$ into $|0\rangle$.

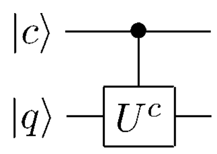
Phase flip gate Z Phase flip gate $Z = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$. We have $Z|q\rangle = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}\alpha_0\\ \alpha_1\end{pmatrix} = \begin{pmatrix}\alpha_0\\ -\alpha_1\end{pmatrix}$. If we denote $|+\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle)$ and $|-\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle)$ then we have $Z|+\rangle = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}\\ 1/\sqrt{2}\end{pmatrix} = \begin{pmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{pmatrix} = |-\rangle$ and $Z|-\rangle = |+\rangle$.

The bit flip gate X and phase flip gate Z are basic and boring as they just transform back and forth between standard states $|0\rangle \leftrightarrow |1\rangle$, $|+\rangle \leftrightarrow |-\rangle$. The following 2 quantum gates are more interesting.

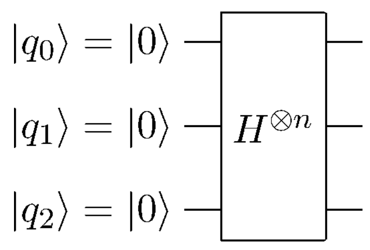
CNOT (Controlled-NOT) gate



CNOT (Controlled-NOT) gate: $|c\rangle|t\rangle \rightarrow |c\rangle|c\oplus t\rangle$ where c is the control bit, t is the target bit. What it means is that if c is 1 then it flips the target bit, otherwise it keeps the target bit as is. This can be generalized to implement if/else: if c is 1, execute gate $U^c = U$, else don't execute U (i.e. execute $U^c = U^0 = I$ identity).



Hadamard gate H Hadamard transform $H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$. We have $H|0\rangle = 1/\sqrt{2}(|0\rangle+|1\rangle)$, $H|1\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle)$. If we apply $H^{\bigotimes 2}$ (this notation just means we apply H to 1st and 2nd qubit independently) to 2 qubits $|00\rangle$, we have $H^{\bigotimes 2}|00\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle)1/\sqrt{2}(|0\rangle + |1\rangle) = 1/2(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$. Observe that Hadamard gate transforms basis state into superposition of states, for instance, H transforms $|0\rangle$ into superposition of states $|0\rangle$, $|1\rangle$ and $H^{\bigotimes 2}$ transforms $|00\rangle$ into superposition of states $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. We have $H^{\bigotimes n}|00\cdots0\rangle = \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle$, i.e., we can create superposition of all basis states $|0...0\rangle, \cdots, |1...1\rangle$ by applying $H^{\bigotimes n}$ gate to $|0...0\rangle$. We'll use this trick over and over again in designing quantum algorithms.


$|q_0\rangle = |0\rangle$
$|q_1\rangle = |0\rangle$
$|q_2\rangle = |0\rangle$
$H^{\otimes n}$

If we take a closer look at Hadamard gate $H|0\rangle = 1/\sqrt{2}(|0\rangle + \sqrt{2}|1\rangle)$, $H|1\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle)$, we'll see that $H|u\rangle = 1/\sqrt{2}(|0\rangle + (-1)^u|1\rangle)$. Therefore

$$\begin{aligned}
H^{\bigotimes 2}|u_1u_2\rangle &= 1/\sqrt{2}(|0\rangle + (-1)^{u_1}|1\rangle)1/\sqrt{2}(|0\rangle + (-1)^{u_2}|1\rangle) \\
&= 1/2(|00\rangle + (-1)^{u_2}|01\rangle + (-1)^{u_1}|10\rangle + (-1)^{u_1u_2}|11\rangle) \\
&= 1/2((-1)^{(u_1,u_2).(0,0)}|00\rangle + (-1)^{(u_1,u_2).(0,1)}|01\rangle + (-1)^{(u_1,u_2).(1,0)}|10\rangle + (-1)^{(u_1,u_2).(1,1)}|11\rangle) \\
&= 1/2\sum_x(-1)^{u.x}|x\rangle
\end{aligned}$$

For n qubits, we have $H^{\bigotimes n}|u_1u_2\cdots u_n\rangle = \frac{1}{2^{n/2}}\sum_x(-1)^{u.x}|x\rangle$ where $u.x = u_1x_1 + u_2x_2 + \cdots + u_nx_n$ is the inner product of u, x.

Unitary transformation If we take a closer look at X, Z, H we see that $I = X^2 = Z^2 = H^2$. This is not an accident. For every quantum transformation U, if we denote U^* as conjugate transpose of U then $UU^* = U^*U = I$. A matrix U satisfying the above equation is called a unitary matrix. In the above examples, it just happens that $X^* = X$, $Z^* = Z$, $H^* = H$ and hence $X^2 = XX^* = I$, $Z^2 = ZZ^* = I$, $H^2 = HH^* = I$.

1.1.3 Quantum measurement

In classical computers, measurement is a trivial operation because what you see is what you get. In quantum computers, we never have access to the qubit q's complex amplitudes α_0, α_1. When we measure $|q\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, we'll see $|0\rangle$ with probability $|\alpha_0|^2$ and see $|1\rangle$ with probability $|\alpha_1|^2$. Furthermore, the measured qubit forever collapses to state $|0\rangle$ or $|1\rangle$, i.e., you can't never put the measured qubit back to superposition of states $|0\rangle$, $|1\rangle$. To a certain extent, quantum measurement is a destructive operation. Finally, to make sure that the probabilities of measurement outcomes summing up to 1, α_0 and α_1 must satisfy the equation $|\alpha_0|^2 + |\alpha_1|^2 = 1$. This is the reason why you see $1/\sqrt{2}$ in state like $|+\rangle = 1/\sqrt{2}|0\rangle + 1/\sqrt{2}|1\rangle)$ because $(1/\sqrt{2})^2 + (1/\sqrt{2})^2 = 1$. For n qubits $|q\rangle = \sum_{x=0}^{2^n-1}\alpha_x|x\rangle$, we have $\sum_{x=0}^{2^n-1}|\alpha_x|^2 = 1$.

The way quantum measurement works restricts quantum computers's computation capability. While n qubits $|q\rangle = \sum_{x=0}^{2^n-1}\alpha_x|x\rangle$ holds information of 2^n basis states at the same time, every time we measure q, we only observe a tiny bit of information: a specific state $|i\rangle$ with probability $|\alpha_i|^2$ and even worst, $|q\rangle$ forever collapses to $|i\rangle$. In other words, most q's information is hidden inside inaccessible $\alpha_j, j \in \{0,1\}^n$. This is the reason why quantum computers is not 2^n more powerful than classical computers.

Partial measurement If we measure 2 qubits like $1/2|00\rangle + 1/2|01\rangle + 1/\sqrt{2}|10\rangle$, we'll see one of states $|00\rangle$, $|01\rangle$, $|10\rangle$. What would happen if we only measure the 1st qubit? If the measured 1st qubit is $|0\rangle$ then the 2 qubits become $1/\sqrt{2}(|00\rangle + |01\rangle))$, i.e., superposition of states where the 1st qubit is $|0\rangle$. Note that the amplitudes change from $1/2$ to $1/\sqrt{2}$ to make sure that the sum of new states' probabilities is 1. Similarly, if we measure the 1st qubit and we see $|1\rangle$ then the 2 qubits become $|10\rangle$ because among three states $|00\rangle, |01\rangle, |10\rangle$ only $|10\rangle$ has the 1st qubit as 1.

1.2 Reversible computation

The fact that any quantum transformation U is unitary has a profound impact on quantum computation. If we start with quantum state x and we apply quantum computation U to it then we can always get x back by appling U^* to $U|x\rangle$. The reason is that $U^*(U|x\rangle) = (U^*U)|x\rangle = I|x\rangle = |x\rangle$, i.e., all quantum computation is reversible. This contrasts with classical computers where there are many one way functions.

The question is if we are given a classical function $f(x)$, can we implement it using reversible quantum computation? The trick is to carry x to the output as well so that we have enough information to reverse the computation. There is quantum circuit U_f that implements $U_f|x\rangle|b\rangle = |x\rangle|b \oplus f(x)\rangle$.

1.3 Bernstein-Vazirani's algorithm

To demonstrate the power of quantum computation, we'll take a look at Bernstein-Vazirani's algorithm [5] that solves parity problems faster than any classical algorithm.

Given a function $f : \{0,1\}^n \to \{0,1\}$ as a "black box" [1] and $f(x) = u.x \mod 2$ for some hidden $u \in \{0,1\}^n$, find u.

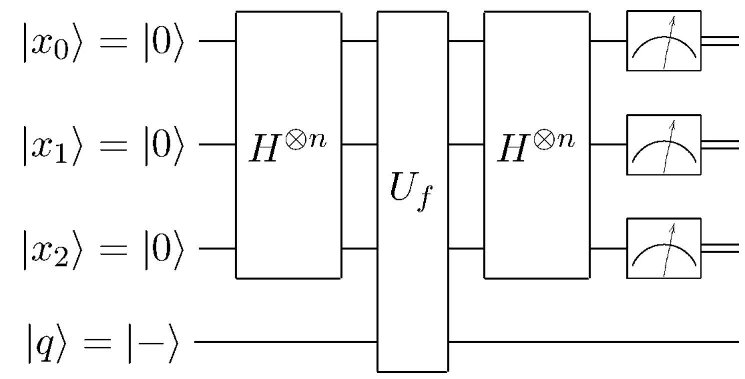
In classical computers, whenever we query the black box f we only get 1 bit of information $f(x)$. As $u \in \{0,1\}^n$ has n bits information, we need at least n queries to the black box f to find n-bit u. Furthermore, if we query f n times using the following x values: $10\cdots0$, $010\cdots0$, $\cdots$, $00\cdots1$ then we'll find u because i^{th} query reveals i^{th} bit of u. Therefore, the optimal classical algorithm requires n queries to f.

In quantum computers, if we denote U_f the quantum circuit that implements f then Bernstein-Vazirani's algorithm only uses U_f once. We'll describe the algorithm by working backwards. Let's recall what we've learned in the previous sections.

- When we measure n qubits $|q\rangle = \sum\limits_{i=0}^{2^n-1} \alpha_i |i\rangle$, we'll see state $|i\rangle, i \in \{0,1\}^n$ with probability $|\alpha_i|^2$. In a special case when $\alpha_u = 1$, $\alpha_{i\neq u} = 0$ or $|q\rangle = |u\rangle$, measuring $|q\rangle$ gives us $|u\rangle$.
- $H^{\bigotimes n}|u\rangle = H^{\bigotimes n}|u_1u_2\cdots u_n\rangle = \frac{1}{2^{n/2}}\sum_x(-1)^{u.x}|x\rangle$.
- $H^2 = I$. This implies $H^{\bigotimes n}H^{\bigotimes n}|x\rangle = I^{\bigotimes n}|x\rangle = |x\rangle$.

Based on the above facts, if we can set up quantum state $\frac{1}{2^{n/2}}\sum_x(-1)^{u.x}|x\rangle = H^{\bigotimes n}u$ then applying $H^{\bigotimes n}$ to it gives $|u\rangle$, so the final quantum measurement results in $|u\rangle$. Now, the question is how to set up the quantum state $\frac{1}{2^{n/2}}\sum_x(-1)^{u.x}|x\rangle$. Note that we don't know u. However, we know $u.x = f(x)$, which means that we can rewrite the above state as $\frac{1}{2^{n/2}}\sum_x(-1)^{f(x)}|x\rangle$. We make progress, but we're not done yet. Even though we don't know how to create state $\frac{1}{2^{n/2}}\sum_x(-1)^{f(x)}|x\rangle$ yet, we know how to create the state that looks similar to it using Hadamard transform $H^{\bigotimes n}|00\cdots0\rangle = \frac{1}{2^{n/2}}\sum_x|x\rangle$. The last bit of the puzzle is how to create $(-1)^{f(x)}$. From "Reversible computation" section, we have $U_f|x\rangle|b\rangle = |x\rangle|b\oplus f(x)\rangle$. Let's see what happens to $U_f|x\rangle|-\rangle = U_f|x\rangle(|0\rangle - |1\rangle) = U_f|x\rangle|0\rangle - U_f|x\rangle|1\rangle = |x\rangle|f(x)\rangle - |x\rangle|1\oplus f(x)\rangle = |x\rangle(|f(x)\rangle - |1\oplus f(x)\rangle) = |x\rangle(-1)^{f(x)}(|0\rangle - |1\rangle) = |x\rangle(-1)^{f(x)}|-\rangle$. Now, we have $(-1)^{f(x)}$.

[1]We can input x to the "black box" and ask it to compute $f(x)$, but we don't have access to its internal computation process.


$|x_0\rangle = |0\rangle$
$|x_1\rangle = |0\rangle$
$|x_2\rangle = |0\rangle$
$|q\rangle = |-\rangle$
$H^{\otimes n}$
U_f
$H^{\otimes n}$

Putting together in the forward order, we have the algorithm as shown in the above figure:

1. $H^{\bigotimes n} |00...0\rangle$ gives us $\frac{1}{2^{n/2}} \sum_x |x\rangle$.
2. Applying U_f to $\frac{1}{2^{n/2}} \sum_x |x\rangle |-\rangle$ gives us $\frac{1}{2^{n/2}} \sum_x (-1)^{f(x)} |x\rangle |-\rangle = \frac{1}{2^{n/2}} \sum_x (-1)^{u.x} |x\rangle |-\rangle$
3. Apply $H^{\bigotimes n}$ to the first n qubits of $\frac{1}{2^{n/2}} \sum_x (-1)^{u.x} |x\rangle |-\rangle$ gives us $|u\rangle |-\rangle$.
4. Measure the first n qubits of $|u\rangle |-\rangle$ gives us $|u\rangle$.

Now, we solved the problem but it's not clear why using U_f once gives us n bit information about u. If we look closer at the 2nd step, even though we use U_f once, we see that the sum $\frac{1}{2^{n/2}} \sum_x (-1)^{f(x)} |x\rangle |-\rangle$ have $f(x)$ for all $x \in \{0,1\}^n$. How come? The reason is that U_f's input can be a superposition of all basis states $\sum_x |x\rangle$ (this is in contrast with classical computers where the input is only 1 specific $|i\rangle$) and so the output contains all $f(x), x \in \{0,1\}^n$.

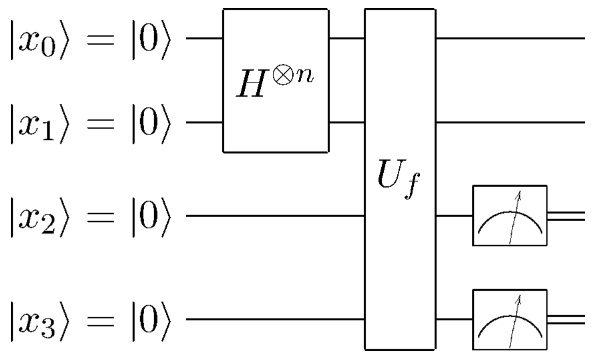
1.4 Simon's algorithm

Given a 2-to-1 function $f : \{0,1\}^n \to \{0,1\}^n$ and $f(x) = f(x \oplus s)$ for some secret $s \in \{0,1\}^n$, find s.

Designing classical algorithms is hard, let alone quantum algorithms. In this problem, the condition $f(x) = f(x \oplus s)$ is not even natural, so it's tough to even start the thinking process on how to solve the problem. We have no clue. One method that I found helpful is to play with our existing knowledge, analyze them and/or try to extend them with the hope that somehow it will lead us closer to the solution.

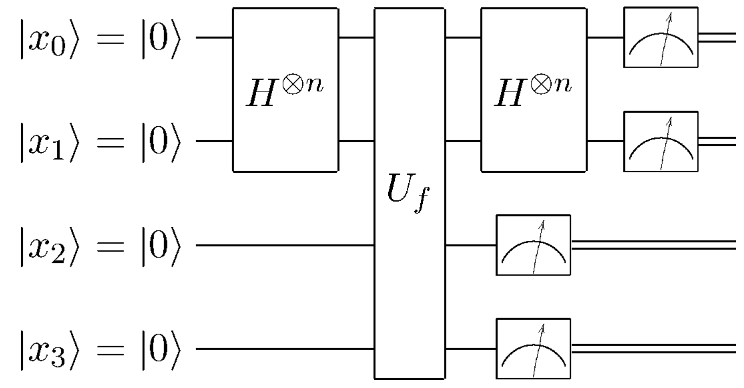
In the previous section, we've seen how to access all $f(x), x \in \{0,1\}^n$ using $H^{\bigotimes n}$, followed by U_f. Recall that $H^{\bigotimes n}$ gives us $\frac{1}{2^{n/2}} \sum_x |x\rangle$ and applying U_f to $\frac{1}{2^{n/2}} \sum_x |x\rangle |00...0\rangle$ gives us $\frac{1}{2^{n/2}} \sum_x |x\rangle |f(x)\rangle$. Note that $\frac{1}{2^{n/2}} \sum_x |x\rangle |f(x)\rangle$ are 2n qubits as $f(x)$ is n-bit.

If we measure the 1st n qubits of $\frac{1}{2^{n/2}} \sum_x |x\rangle |f(x)\rangle$ and the 1st n qubits collapses to $|r\rangle$ for some r, then the 2n qubits collapses to $|r\rangle |f(r)\rangle$. We've made progress as we've just learned that this path leads nowhere :)


$|x_0\rangle = |0\rangle$
$|x_1\rangle = |0\rangle$
$|x_2\rangle = |0\rangle$
$|x_3\rangle = |0\rangle$
$H^{\otimes n}$
U_f

If we measure the 2nd n qubits of $\frac{1}{2^{n/2}}\sum_x |x\rangle |f(x)\rangle$ and we see $|f(r)\rangle$ for some r then the 2n qubit collapses to $|r\rangle |f(r)\rangle$. Wait a second, this wasn't correct. From the problem statement, we know $|f(r)\rangle = |f(r \oplus s)\rangle$, i.e., if we see $|f(r)\rangle$, then we see $|f(r \oplus s)\rangle$ as well because they both equal each other. In other words, the 2n qubits state collapses to $1/\sqrt{2}(|r\rangle |f(r)\rangle + |r \oplus s\rangle |f(r \oplus s)\rangle) = 1/\sqrt{2}((|r\rangle + |r \oplus s\rangle) |f(r)\rangle)$. This is pretty cool as we've discovered a new state $1/\sqrt{2}(|r\rangle + |r \oplus s\rangle)$ that we've never seen before.

What are we going to do with $1/\sqrt{2}(|r\rangle + |r \oplus s\rangle)$? This is a specific state for some random r, so let's apply $H^{\bigotimes n}$ to make it in superposition of all basis states again. Are you sick of $H^{\bigotimes n}$ yet? :) I'm sorry that I have to make you real sick because we'll continue using $H^{\bigotimes n}$ over and over again. Applying $H^{\bigotimes n}$ to $|r\rangle$ gives us $\frac{1}{2^{n/2}}\sum_x (-1)^{r.x} |x\rangle$ while applying $H^{\bigotimes n}$ to $|r \oplus s\rangle$ gives us $\frac{1}{2^{n/2}}\sum_x (-1)^{(r\oplus s).x} |x\rangle$, therefore applying $H^{\bigotimes n}$ to $1/\sqrt{2}(|r\rangle + |r \oplus s\rangle)$ gives us $\frac{1}{2^{(n+1)/2}}\sum_x ((-1)^{r.x} + (-1)^{(r\oplus s).x}) |x\rangle$. The value $(-1)^{r.x} + (-1)^{r.x \oplus s.x}$ is pretty special because if $s.x = 1 \mod 2$ then $(-1)^{r.x} + (-1)^{r.x\oplus 1} = (-1)^{r.x}(1 + (-1)) = 0$ (i.e. we never see the corresponding $|x\rangle$), otherwise if $s.x = 0$ then $(-1)^{r.x} + (-1)^{r.x\oplus 0} = 2(-1)^{r.x} \neq 0$. What it means is after measurement, we only see $|x\rangle$ such that $s.x = 0$ for some random x. The complete Simon algorithm [6] is shown in the below figure


$|x_0\rangle = |0\rangle$
$|x_1\rangle = |0\rangle$
$|x_2\rangle = |0\rangle$
$|x_3\rangle = |0\rangle$
$H^{\otimes n}$
U_f
$H^{\otimes n}$

All right, we get $s.x = 0$ for some random x. This isn't enough to solve for s. What we can do is to repeat the algorithm so that we will receive a system of linear equations where each equation has the form $s.x = 0$ with probably different random x. Roughly we need n equations of the form $s.x = 0$ to find s using Gauss elimination algorithm.

1.5 Quantum Fourier Transform (QFT)

Denote ω N-rooth of unity: $\omega^N = 1$. Classical Fourier transform is defined by the following matrix:

$$F_N = 1/\sqrt{N}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)^2} \end{pmatrix}$$

Quantum Fourier Transform (QFT) is simply applying the same matrix to a quantum state: $F_N \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{N-1} \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{N-1} \end{pmatrix}$ or $F_N(\sum_j \alpha_j |j\rangle) = \sum_k \beta_k |k\rangle$. In a special case, when $\alpha_0 = 1$ and $\alpha_i = 0, i \neq 0$ we have $F_N |0\rangle = 1/\sqrt{N}\sum_k |k\rangle$, i.e., it transform $|0\rangle$ into superposition of all basis states. Did you notice that it looks similar to Hadamard transform $H^{\bigotimes n}$? The reason is that in fact, Hamamard transform is a special case of QFT.

I don't know about you but F_N looks scary as hell to me :) Fourier transform is beyond my depth, let alone QFT. However, QFT plays an important role in Shor's algorithms, so we have to study it. One method to get away from dealing with deep math is to learn its properties without proving them. We can dig deeper into the proof once our math muscle is stronger. For now, let's see QFT's nice properties.

Efficient implementation QFT can be implemented in $O((logN)^2)$. In other words, QFT can be efficiently implemented in polynomial time of N's bit length.

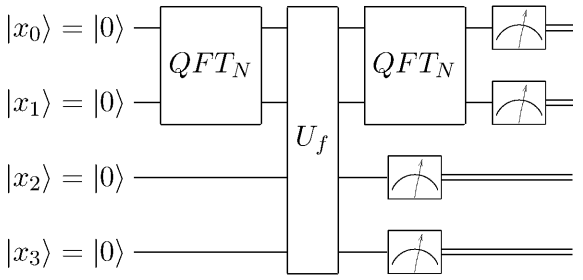
Shift property $F_N \begin{pmatrix} \alpha_{N-1} \\ \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{N-2} \end{pmatrix} = \begin{pmatrix} 1\beta_0 \\ \omega\beta_1 \\ \omega^2\beta_2 \\ \vdots \\ \omega^{N-1}\beta_{N-1} \end{pmatrix}$ or $F_N(\alpha_{N_1}|0\rangle + \alpha_0|1\rangle + \alpha_1|2\rangle + \cdots + \alpha_{N-2}|N-1\rangle = \sum_k \omega^k\beta_k|k\rangle$, i.e., when we move the 1st amplitude α_0 to the 2nd position, the 2nd amplitude α_1 to the 3rd position, etc then applying QFT to the result corresponds to multiplying β_k with ω^k. When we measure $\sum_k \omega^k\beta_k|k\rangle$, we'll see specific state $|k\rangle$ with probability $|\omega^k\beta_k|^2 = |\omega^k|^2|\beta_k|^2$. As the magnitude of ω^k is 1, we'll see specific state $|k\rangle$ with probability $|\beta_k|^2$. This is exactly the same as if we measure $\sum_k \beta_k|k\rangle$. What it means is that if we shift the input α_i, it doesn't change the measurement of the QFT's output.

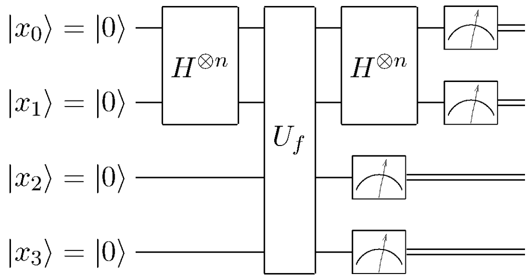
Periodic property If $f(x)$ is a periodic function with period r (i.e., $f(x) = f(x+r \mod N)$) then $F_N f(x)$ is a periodic function with period N/r. Let's consider a special case $f(x) = 1/\sqrt{N/r}\sum_{j=0}^{N/r-1}|jr\rangle$, i.e., the amplitudes are $1/\sqrt{N/r}$ at $0, r, 2r, \cdots, (N/r-1)r$ and are zeros everywhere else, hence $f(x)$ is a period function with period r. In this special case, $F_N(1/\sqrt{N/r}\sum_{j=0}^{N/r-1}|jr\rangle) = 1/\sqrt{r}\sum_{l=0}^{r-1}|lN/r\rangle$. Note that the right hand side's amplitudes are $1/\sqrt{r}$ at $0, N/r, 2(N/r), \cdots,$ $(r-1)(N/r)$ and are zeros everywhere else and hence it's a period function with period N/r.

1.6 Period finding

Assume $f : \{0, 1, \cdots, N-1\} \to S$ (S is just some set) is a periodic function with hidden period r ($r|N$): $f(x) = f(x + r(\mod N))$. Find r.

The algorithm is shown in the left figure.


$|x_0\rangle = |0\rangle$
$|x_1\rangle = |0\rangle$
$|x_2\rangle = |0\rangle$
$|x_3\rangle = |0\rangle$
QFT_N
U_f
QFT_N


$|x_0\rangle = |0\rangle$
$|x_1\rangle = |0\rangle$
$|x_2\rangle = |0\rangle$
$|x_3\rangle = |0\rangle$
$H^{\otimes n}$
U_f
$H^{\otimes n}$

I guess you're disappointed with me because I didn't explain the thinking process in designing the solution, instead I jumped directly to the final algorithm as shown above. The reason is that if you look back Simon's algorithm (as shown in the right figure), you'll notice that it looks exactly the same as period finding's algorithm, except we replace $H^{\bigotimes n}$ in Simon's algorithm with QFT_N in period finding algorithm. Surprise!

The function f in period finding problem is different from Simon's algorithm and QFT_N is similar but different from $H^{\bigotimes n}$. Therefore, we have to analyze the period finding algorithm closely.

After applying QFT_N to $|0\rangle$, we get $1/\sqrt{N}\sum_x|x\rangle$. After U_f, we have $1/\sqrt{N}\sum_x|x\rangle|f(x)\rangle$. If we measure $|f(x)\rangle$ and get $f(k)$ for some specific k then as $f(k) = f(k+r) = \cdots = f(k+r(N/r-1))$, we know that $1/\sqrt{N}\sum_x|x\rangle|f(x)\rangle$ collapses to $|k\rangle|f(k)\rangle + |k+r\rangle|f(k+r)\rangle + \cdots + |k+r(N/r-1)\rangle|f(k+r(N/r-1))\rangle$. Rewrite the result as $\sum_{j=0}^{N/r-1}|jr+k\rangle|f(k)\rangle$. However, as shifting doesn't affect QFT's measurement, we can simplify the 1st n qubits to $\sum_{j=0}^{N/r-1} |jr\rangle$. Now, using period property of QFT, we know that the last QFT_N will transform $\sum_{j=0}^{N/r-1} |jr\rangle$ into $\sum_{l=0}^{r-1} |lN/r\rangle$. After measuring the 1st n qubits, we'll get sN/r for some random s.

If we repeat the above algorithm a few times, we get s_1N/r, s_2N/r, s_3N/r, etc. It's easier to see that the greatest common divisor of them gives us N/r or r.

1.7 Shor's algorithms

In the previous section, we have built a general framework to find the period of an arbitrary periodic function. In this section, we'll describe Shor's algorithms [3] by transforming factoring problems and discrete log problems into period finding problems.

1.7.1 Shor's factoring algorithm

Given $n = p_1 * p_2$ where p_1, p_2 are prime numbers, find p_1, p_2.

Shor's algorithm is based on the following classical observation. For a random number x, if we can find an even number r such that $x^r = 1 \mod n$ then $gcd(x^{r/2} - 1, n)$ or $gcd(x^{r/2} + 1, n)$ gives us the factor of n. Let's see why. $x^r - 1 = (x^{r/2} - 1)(x^{r/2} + 1) = 0 \mod n$ implies $(x^{r/2} - 1)(x^{r/2} + 1) = 0 \mod p_1$, i.e., p_1 divides either $(x^{r/2} - 1)$ or $(x^{r/2} + 1)$. In other words, p_1 divides $gcd(x^{r/2} - 1, n)$ or $gcd(x^{r/2} + 1, n)$.

What does $x^r = 1 \mod n$ mean? It means that if we fix the random number x and define $f(a) = x^a \mod n$ then $f(a + r) = x^{a+r} = x^a x^r = x^a * 1 = f(a) \mod n$, i.e., $f(a)$ is a periodic function with period r. Therefore, the algorithm is to generate random x, use the period finding algorithm to find period r of $f(a) = x^a \mod n$ and finally compute $gcd(x^{r/2} - 1, n)$ or $gcd(x^{r/2} + 1, n)$. What if we find odd r, instead of even r? We just repeat the algorithm with a different x.

1.7.2 Shor's discrete algorithm

Assuming p is a prime number and g is the generator of multiplicative group $\mod p$, i.e., the set $\{g^0, g^1, \cdots, g^{p-2}\}$ is the same as $\{1, 2, \cdots, p-1\}$. Given $y = g^x \mod p$, find x.

Let's take a look at the function $f(a, b) = g^a y^{-b} \mod p$, we have $f(a + rx, b + r) = g^{a+rx} y^{-b-r} = g^{a+rx}(g^x)^{-b-r} = g^{a+rx-xb-rx} = g^{a-xb} = f(a, b) \mod p$. I.e., $f(a, b)$ is a periodic function with tuple period (rx, r). Using a period finding algorithm to find the period (r_1, r_2) of $f(a, b)$ and compute discrete log $x = r_1/r_2$.

References

[1] Michael Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information.*

[2] Umesh Vazirani (UC Berkeley). Quantum mechanics and quantum computation.

[3] Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.

[4] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Report on the first round of the nist post-quantum cryptography standardization process.

[5] Ethan Berstein and Umesh Vazirani. Quantum complexity theory.

[6] Daniel Simon. On the power of quantum computation.