How Mathematics Will Break Cybersecurity: Quantum Computing

Quantum Computing is an emerging field of Computer Science where the bits (now qubits) are modelled by subatomic spins rather than transistors, meaning they may be in a **superposition** state of **both** 0 and 1 until measurement. Therefore, multiple possibilities, sometimes referred to as universes by quantum physicists, can be watched at once, where then wave interference is utilised to isolate only the most likely answer.
By contrast, iterations and functions are run one at a time in classical computing.
Hence, quantum computers are significantly faster at **particular** problems; for example, Google's latest quantum chip "Willow" was found to solve a specific problem in 5 minutes that "the most powerful conventional [super] computers would take 10 septillion years to manage" (Neven, 2024).

However, what is less noticed is how mathematics plays a role in this. Researchers are fundamentally using calculations and new notation to evaluate methods to improve error correction and algorithms in new quantum computers, so it is worthwhile exploring how topics like matrices and graphs apply, which are used by scientists to enhance or threaten our current encryption.

Fundamental Theory (Preface)

Qubits

The first point is learning how qubits can be mathematically represented as you can't simply write 0 or 1 anymore, as qubits are in **superposition**.

Instead, we can use column **vectors** to denote how much of one state (0 or 1) the qubit's superposition contains.
Importantly, when measuring a qubit, it will collapse to either 0 or 1 - you can not measure the superposition itself.
Note: a similar phenomenon occurs in Young's double-slit experiment where light appears to shine and diffract through both slits, forming an interference pattern. However, measuring which slit the photons pass through causes the interference pattern to disappear. This almost magic of 'photons knowing when you are watching' is partly elucidated by the Heisenberg Uncertainty Principle**.**

The general form of a qubit can be represented as follows:

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

where |α|^2 is the probability of measuring the qubit as 0 and |β|^2 is the probability of measuring the qubit as 1 (an axiom known as the Born Rule).

For example, the 0 state would be given as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

as the probability of measuring it as 0 is 100% whereas the probability of measuring it as 1 is 0%.

Therefore, it's useful to know that $\alpha^2 + \beta^2 \equiv 1$, and if this is not satisfied then the vector does not represent a valid qubit.

Dirac Notation

Conventionally, mathematicians use Dirac / bra-ket notation instead of vectors when referring to qubits, because it simplifies representing many entangled qubits. I've already used this above, encasing the ψ and 0 states, with the symbols around them called **kets**.
Dirac notation expresses a qubit in terms of the 0 and 1 states - here's how:

Remember the 0 and 1 states are:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Manipulating a general qubit vector can let it take the form:

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

$$|\psi\rangle = \begin{pmatrix} \alpha \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \beta \end{pmatrix}$$

Therefore, factoring out α and β from their respective vectors gives an arbitrary qubit that can be represented as:

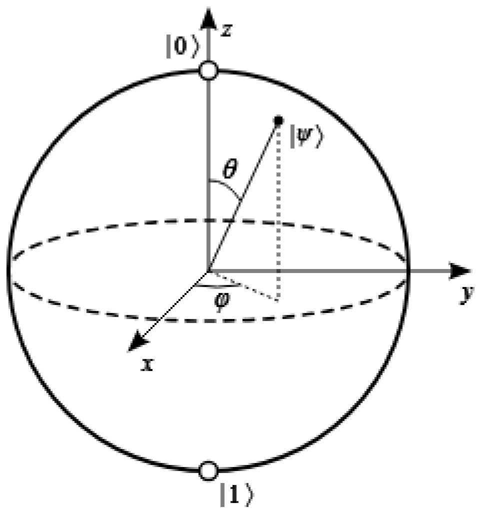
$$|\psi\rangle = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

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

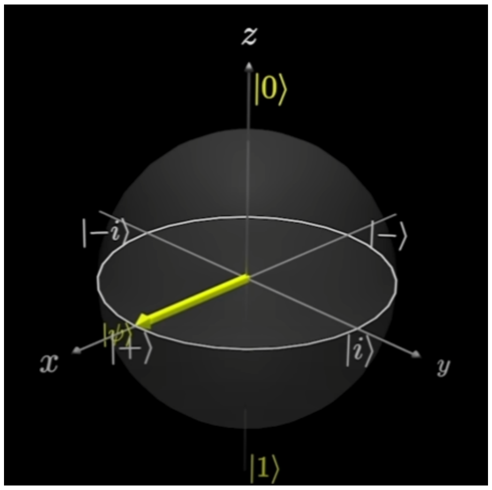
Graphical Representation

Mathematicians love to represent everything on graphs, from your Oxford interview to real-world statistics. However, did you know that they are also useful in quantum mechanics and computer science?

Qubits may be represented on a **Bloch Sphere**, which geometrically displays all of the possible states a qubit can take on a 3-dimensional plane.



The z-axis denotes the probabilities of measuring a |0⟩ state or |1⟩ state; the closer the qubit is to the top, the greater the probability of measuring it as 0.
Therefore, a qubit with an even chance of being measured as 0 or 1 could look like:



(freeCodeCamp.org, 2024)

as the vector lies in the middle of the z-axis.

If you are curious, this qubit would be:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

as squaring the coefficients gives a ½ = 50% probability of measuring each state.

You may wonder what the x and y axes are for if they don't affect probability, but they are just as important. They represent the phase of a qubit, which is the backbone of quantum algorithms and mathematics. Stay tuned…

Representing Multiple Qubits

Lastly, we have only considered a system of 1 qubit so far, so here's how to represent multiple. You will notice how dirac notation makes this much easier to display, rather than showing a multiplication of vectors.

Example:

$$|\psi\rangle = \frac{1}{2}|00\rangle + \frac{1}{4}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle + \frac{\sqrt{3}}{4}|11\rangle$$

The first number of each bra-ket is for the first qubit, so the second number of each bra-ket is for the second qubit, etc.
The equation shows the 4 possible states the 2 qubits can be in: 00, 01, 10, 11 - which I hope makes intuitive sense. Hence, if you measure the system, you will only get one of these states (e.g. 01, meaning the first qubit is 0 and the second qubit is 1).

This represents a two-qubit system, but how did we get to this expression?

The **tensor product** is the operator used to add one quantum state to another, denoted by the symbol $\otimes$.

For example, you can represent two zero states as follows:

$$|0\rangle \otimes |0\rangle = |00\rangle$$

If you want to represent two qubits in superposition:

$$(\alpha|0\rangle+\beta|1\rangle)\otimes(\gamma|0\rangle+\delta|1\rangle) = \alpha|0\rangle\otimes\gamma|0\rangle+\alpha|0\rangle\otimes\delta|1\rangle+\beta|1\rangle\otimes\gamma|0\rangle+\beta|1\rangle\otimes\delta|1\rangle$$

The tensor product follows similar rules to binomial expansion when multiplying, giving the above.

Evaluating the RHS,

$$\alpha|0\rangle\otimes\gamma|0\rangle+\alpha|0\rangle\otimes\delta|1\rangle+\beta|1\rangle\otimes\gamma|0\rangle+\beta|1\rangle\otimes\delta|1\rangle = \alpha\gamma|00\rangle+\alpha\delta|01\rangle+\beta\gamma|10\rangle+\beta\delta|11\rangle$$

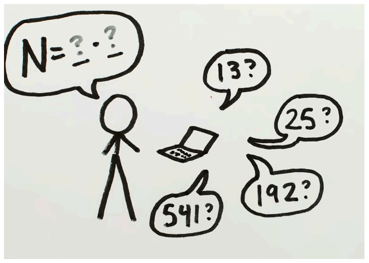
Uses in Encryption

Now that I've given you some basic theory, the real essay can begin (well… not "real", complex numbers are quite common here). The backbone of RSA encryption, which is commonly used internationally, involves factoring a colossal number into 2 prime factors, which provide the access key for any data holder. The numbers are purposefully too large for any hacker to bother computing, as the number of iterations would consume too much time and energy. However, **Shor's algorithm may actually factorise the number**, N, because a quantum computer can reduce the time complexity of factorising large integers from exponential to polynomial time, which is essential for larger problems. So, let's explore how mathematics plays into our quantum journey.

How does Shor's algorithm work?

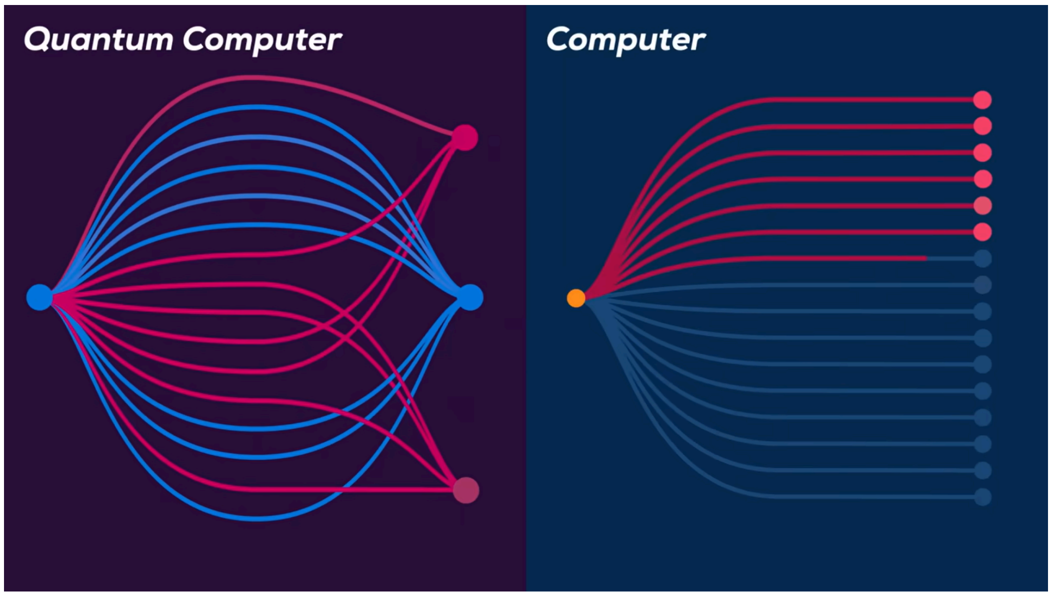
N = pq, where p and q are prime numbers.
Let's imagine how a classical computer could solve this for a very large N. A common approach is to loop through many guesses until one of them is a prime factor of N,

eventually cracking the encryption. Wait… what did you say? This takes supercomputers billions of years? Even with optimised programs? Oh…



(minutephysics, 2020)

Don’t worry, I’ve been teaching you the solution. Unlike a classical computer that runs iterations one at a time, since qubits can be both 0 and 1 at the same time (in that superposition state until measured), multiple outcomes are watched at once in quantum computing.



(Kurzgesagt, 2016) Depicts computers simply running iterations one at a time but quantum computers more complicatedly but efficiently reaching all paths quickly.

As companies like IBM, Google and Microsoft are racing to build bigger and better quantum computers, mathematicians are exploring how we can apply such a powerful algorithm to solve this notorious problem.

The first step of Shor’s algorithm, similar to a regular computer, is to take a random guess for a prime factor of N, let’s call it ‘a’, which comes with two assumptions:

1 < a < N, or else how would ‘a’ possibly be a factor of N?

gcd(a, N) = 1, meaning the greatest common divisor of a and N is 1, so they share no factors - or else you've gotten very lucky and can evaluate N's prime factors without the quantum algorithm, so why are you still here?

We need somehow to turn this random guess into a better guess so that it has a much higher chance of sharing a prime factor of N. We don't necessarily have to guess the factors of N because if we have shared factors with N, i.e. $gcd(a,\ N)\ \neq\ 1$, we can use **Euclid's algorithm** to find N, which is very efficient.

What is this "better guess"?

We need to find expressions in terms of a that are likely to share factors with N, as our current guess 'a' isn't very useful.

Let $a,\ N\ \in\ \mathbb{Z}$.

We can use the axiom that if you multiply a by itself enough times, you will eventually get a number 1 greater than b, given gcd(a, b) = 1 (which is another reason this assumption is important). In other words, a and b are **coprime**.
Hence, $a^p\ =\ kN\ +\ 1$, for some integers p and k.

For example, if a = 3 and N = 10,

a^3 = 27 = 2N + 7
a^4 = 81 = 8N + 1

Therefore, for this example p = 4 and k = 8, giving a remainder of 1.

Hence, this can be represented with **modular arithmetic** as follows:

$3^1\ =\ 3mod(10)$
Note: the remainder above is 3, as 3mod(10) = 3 % 10, which is the remainder of 3/10.
$3^2\ =\ 9mod(10)$
$3^3\ =\ 7mod(10)$
$3^4\ =\ 1mod(10)$
Remainder = 1, nice.

Okay, okay. I give in. I'll prove this result for you (read ahead if you wish to skip to more on Shor's algorithm).

To prove the result $a^p = kN + 1$ is true for all $a, N \in \mathbb{Z}$, given that a and N are coprime, for some $p, k \in \mathbb{Z}$.

Using **Euler's Theorem**, which states that if gcd(a, N) = 1, a raised to the power of the totient function of N is congruent to 1mod(N). The totient function counts how many numbers less than N are coprime to N.

$a^{\varphi(N)} \equiv 1mod(N)$

However, 1mod(N) means the remainder when dividing by N is 1, which is essentially what $kN + 1$ states.
Hence, we can deduce:
$a^{\varphi(N)} = kN + 1$

Let p = $\varphi(N)$

$a^p = kN + 1$, as required.

Q.E.D

If you want to learn more about this proof, I suggest reading about Euler's theorem and the totient function, an essential cornerstone of number theory.

I couldn't find this proof online but came across Fermat's Little Theorem and Bezout's identity, however, I don't think they specifically apply to this proof, but please contact me if you feel this proof could be improved.

$a^p = kN + 1$

Let's rearrange this to make N the subject:
$a^p - 1 = kN$
$(a^{p/2} + 1)(a^{p/2} - 1) = kN$, taking the difference of two squares.

Therefore, the two expressions in brackets share factors with N, so those are our "better guesses", and these are what Shor's algorithm aims to find. Importantly, introducing an integer p is why we can't use this method today, as iterating over all of these possible integers is the time-consuming part.

Finding our "better guess" - 1

To find this better guess, we only need to find p, which is what Shor's algorithm actually does.

Let's consider our modular exponentiation of 3 again:

$3^{1} = 3mod(10)$
$3^{2} = 9mod(10)$
$3^{3} = 7mod(10)$
$3^{4} = 1mod(10)$
$3^{5} = 3mod(10)$
$3^{6} = 9mod(10)$
$3^{7} = 7mod(10)$
$3^{8} = 1mod(10)$
$3^{9} = 3mod(10)$
$3^{10} = 9mod(10)$
$3^{11} = 7mod(10)$
$3^{12} = 1mod(10)$

The remainder when dividing by 10 forms a pattern:
<u>3, 9, 7, 1</u>, 3, 9, 7, 1, 3, 9, 7, 1 [...]

This can be defined by the observation:
$a^{x} = kN + r \Rightarrow a^{x+p} = k_{2}N + r$, where $x \in \mathbb{Z}$ (but is not necessarily p).
This shows that after a period of p, the remainder is still the same, 'r'.

Therefore, p has a repeating or periodic property as adding/subtracting the power x by p (or 2p, or any multiply of p) will keep the remainder r constant. The reason p happens to have this property is because the modular exponentiation actually begins at $3^{0} = 1mod(10)$, so the first term always has a remainder of 1. Thus, the next time a remainder of 1 appears (at p), it will be a period of p after the first.

But of course, we’re here to represent this process mathematically…
Our process of raising 3, or generally ‘a’, to a list of powers, ‘x’, and then finding the remainder can be defined by a matrix.

Let $U_{a,N}$ be a transformation matrix such that
$U_{a,N}|x\rangle = |xamod(N)\rangle$

Matrices are used as a mathematical representation of quantum gates, but all it is saying is “Let U be how we create this modular exponentiation list”.
We apply the U gate to the 1 state because it's a multiplicative identity.

Applying the U gate multiple times (for all values of x, i.e all of the powers of a):

$U^0_{a,N}|1\rangle = |1mod(N)\rangle$
$U^1_{a,N}|1\rangle = |amod(N)\rangle$
$U^2_{a,N}|1\rangle = |a^2mod(N)\rangle$
$U^3_{a,N}|1\rangle = |a^3mod(N)\rangle$
$U^4_{a,N}|1\rangle = |a^4mod(N)\rangle$
$U^5_{a,N}|1\rangle = |a^5mod(N)\rangle$
These new equations are a general form of the list we have already been exploring.

To show that a^p has a remainder of 1:
$U^p_{a,N}|1\rangle = |a^pmod(N)\rangle = |1mod(N)\rangle$

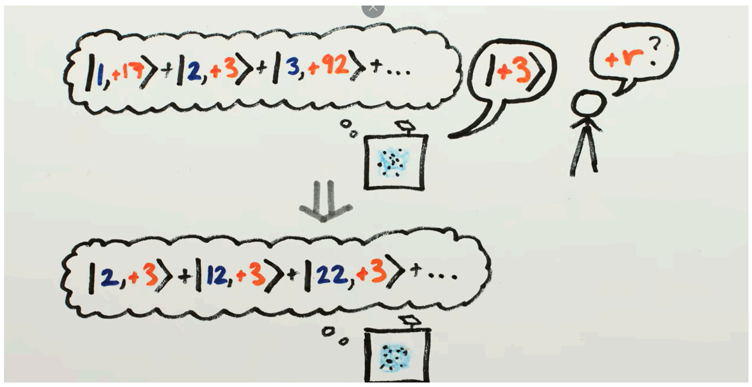
To summarise, at this point, we have taken a superposition of x and transformed it into a superposition of the remainder of a^x / N.

$|1\rangle + |2\rangle + |3\rangle + \ldots \rightarrow |1, 1mod(N)\rangle + |2, 2mod(N)\rangle + |3, 3mod(N)\rangle + \ldots$
Note, these aren’t qubits but rather the information they make up, so aren’t represented the same way as discussed in the preface.

Finding our “better guess” - 2

Since p has this periodic property, if we can find how often we get the same remainder, we would have p.
Suppose **we measure one of these remainders**, in the picture below it’s 3. In that case, **the quantum system will collapse** and will only contain qubits with 3 as the

remainder (the specific number 3 isn't important but the period is). Then, we can find how often the remainder repeats, the period (in the picture it's every 10 integers, from 2 to 12 to 22).



(minutephysics, 2020)

So, in this example p = 10. However, this is a conceptual analysis of Shor's algorithm - unfortunately, a quantum computer can't look at these numbers like we can.
You may be familiar with the Fourier Transform, which fundamentally finds the frequencies of a superposed wave function.

It'd be nice if we could add the word 'quantum' before this and… Oh, wait.

The Quantum Fourier Transform

A more rigorous mathematical explanation would require us to return to our superposition expression.
$|1\rangle + |2\rangle + |3\rangle + ... \rightarrow |1, 1mod(N)\rangle + |2, 2mod(N)\rangle + |3, 3mod(N)\rangle + ...$
The usage of commas within the qubits isn't how they're mathematically represented, this is showing 2 registers in the computer together (the x value and the associated remainder), to help you understand.

Let $a^s mod(N)$ be the remainder we have measured.
Therefore, the superposition has collapsed to:
$|s, a^s mod(N)\rangle + |s + p, a^s mod(N)\rangle + |s + 2p, a^s mod(N)\rangle + ...$

I've purposefully written the remainder of every state as $a^s mod(N)$ to demonstrate how the other remainders have collapsed from the superposition.

The first register of the quantum computer (containing s, s+p, s+2p, etc.), has the quantum Fourier transform (QFT) applied to it. The QFT analyses the states that existed between, say, s and s + p (one cycle), to find p (unfortunately you can't just subtract s from s+p).

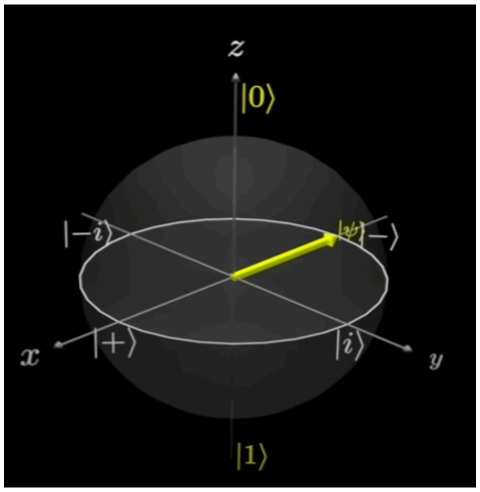
$$\sum_{k=0}^{p-1} e^{-2\pi isk/p}|a^k mod(N)\rangle$$

This may look daunting, but I hope it makes some intuitive sense, especially if you have looked at discrete Fourier transforms before. We sum over 1 to p - 1 as we are only interested in *one cycle* (summing over to p would enter the next cycle). The $e^{-2\pi isk/p}$ term is part of a concept we have not explored much yet, but it is the mathematical representation of **phase**.

Phase

The phase of a qubit is its position along an oscillation of its wave function.
Phase determines whether the wavefunction of qubits interfere constructively or destructively (examples will be demonstrated soon), a concept explored with waves in A-Level physics already.

Contrasting the Bloch sphere shown in the preface, here's another one:



(freeCodeCamp.org, 2024)

This also denotes an equal probability of measuring the qubit as a |0⟩ or |1⟩ state, so why is this special?
Well, this state is written as:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

The -1 coefficient in front of the |1⟩ state shows this qubit in antiphase (180°/π radians, i.e. completely out of phase) to the first one, which means they would destructively interfere.

In other words, the qubit is rotated π radians around the z-axis. To represent this rotation within the equation, we conventionally use Euler's identity, $-1 \equiv e^{i\pi}$.

For context, the purpose of applying a different phase to each value within a cycle is to allow for constructive interference at the frequency of these cycles, but I won't delve too much into the physics or Fourier transforms or else the essay would be too long, just treat the $e^{-2\pi isk/p}$ term as a phase applied to each value by the quantum Fourier transform.

Otherwise, we have a list of the remainders in one cycle, e.g. $a^{0}mod(N)$, $a^{1}mod(N)$, $a^{2}mod(N)$ etc, which is what the Quantum Fourier Transform analyses.

Alright, back to the maths.

$$|u_s\rangle = \sum_{k=0}^{p-1} e^{-2\pi isk/p}|a^k mod(N)\rangle$$

Evaluating the sum gives

$$|u_s\rangle = (e^{-2\pi is(0)/p}|a^0 mod(N)\rangle + e^{-2\pi is(1)/p}|a^1 mod(N)\rangle + \ldots + e^{-2\pi is(p-2)/p}|a^{p-2} mod(N)\rangle + e^{-2\pi is(p-1)/p}|a^{p-1} mod(N)\rangle)$$

There's a small problem with this equation though. Remember how the sum of probabilities of every possible state you can measure has to equal 1? Let's find that sum for this equation.

Finding the probability of measuring one single state within the expression,

$$|e^{-2\pi isk/p}|^2 = |(e^{i\pi})^{-2sk/p}|^2 = 1^2 = 1$$

This sum is from 0 to p-1, so there are p amount of values being summed. Hence, the sum of probabilities = p, not 1.
To counter this, we can insert a **normalisation constant**, and let it be denoted by c.
We need each state to have a 1/p chance of being measured, so that the sum over p states is 1.

$$|c * e^{-2\pi isk/p}|^2 = |c * (e^{i\pi})^{-2sk/p}|^2 = c^2 = 1/p$$

$c = 1/\sqrt{p}$

Hence, we have our normalisation constant. Let's correct our u_s state.

$$|u_s\rangle = \frac{1}{\sqrt{p}}(e^{-2\pi is(0)/p}|a^0 mod(N)\rangle + e^{-2\pi is(1)/p}|a^1 mod(N)\rangle + \ldots + e^{-2\pi is(p-2)/p}|a^{p-2} mod(N)\rangle + e^{-2\pi is(p-1)/p}|a^{p-1} mod(N)\rangle)$$

Quantum Phase Estimation

Next, you can apply the U matrix to both sides,

$$U|u_s\rangle = U\frac{1}{\sqrt{p}}(e^{-2\pi is(0)/p}|a^0 mod(N)\rangle + e^{-2\pi is(1)/p}|a^1 mod(N)\rangle + \ldots + e^{-2\pi is(p-2)/p}|a^{p-2} mod(N)\rangle + e^{-2\pi is(p-1)/p}|a^{p-1} mod(N)\rangle)$$

Distributing U,

$$U|u_s\rangle = \frac{1}{\sqrt{p}}(e^{-2\pi is(0)/p}U|a^0 mod(N)\rangle + e^{-2\pi is(1)/p}U|a^1 mod(N)\rangle + \ldots + e^{-2\pi is(p-2)/p}U|a^{p-2} mod(N)\rangle + e^{-2\pi is(p-1)/p}U|a^{p-1} mod(N)\rangle)$$

Remember the U gate/matrix's function,

$$U^0_{a,N}|1\rangle = |1 mod(N)\rangle$$
$$U^1_{a,N}|1\rangle = |a mod(N)\rangle$$
$$U^2_{a,N}|1\rangle = |a^2 mod(N)\rangle$$

Applying the U gate essentially multiplies the state by a.
Thus,

$$U|u_s\rangle = \frac{1}{\sqrt{p}}(e^{-2\pi is(0)/p}|a^1 mod(N)\rangle + e^{-2\pi is(1)/p}|a^2 mod(N)\rangle + \ldots + e^{-2\pi is(p-2)/p}|a^{p-1} mod(N)\rangle + e^{-2\pi is(p-1)/p}|a^p mod(N)\rangle)$$

Remember,

$$|a^p mod(N)\rangle = |1 mod(N)\rangle = |a^0 mod(N)\rangle$$

We can make this change in our big expression too.

$$U|u_s\rangle = \frac{1}{\sqrt{p}}(e^{-2\pi is(0)/p}|a^1 mod(N)\rangle + e^{-2\pi is(1)/p}|a^2 mod(N)\rangle + \ldots + e^{-2\pi is(p-2)/p}|a^{p-1} mod(N)\rangle + e^{-2\pi is(p-1)/p}|a^0 mod(N)\rangle)$$

If we multiply the RHS by 1, or $e^{2\pi is/p} * e^{-2\pi is/p}$ (which equals 1), we get:

$$U|u_s\rangle = e^{2\pi is/p}e^{-2\pi is/p}\frac{1}{\sqrt{p}}(e^{-2\pi is(0)/p}|a^1 mod(N)\rangle + e^{-2\pi is(1)/p}|a^2 mod(N)\rangle + \ldots + e^{-2\pi is(p-2)/p}|a^{p-1} mod(N)\rangle + e^{-2\pi is(p-1)/p}|a^0 mod(N)\rangle)$$

Distributing $e^{-2\pi is/p}$ into the expression in brackets,

$$U|u_s\rangle = e^{2\pi is/p}\frac{1}{\sqrt{p}}(e^{-2\pi is(1)/p}|a^1 mod(N)\rangle + e^{-2\pi is(2)/p}|a^2 mod(N)\rangle + ... + e^{-2\pi is(p-1)/p}|a^{p-1} mod(N)\rangle + e^{-2\pi is(p)/p}|a^0 mod(N)\rangle)$$

Since $e^{-2\pi is(p)/p} = e^{-2\pi is * 1} = 1 = e^0 = e^{-2\pi is(0)/p}$,

$$U|u_s\rangle = e^{2\pi is/p}\frac{1}{\sqrt{p}}(e^{-2\pi is(1)/p}|a^1 mod(N)\rangle + e^{-2\pi is(2)/p}|a^2 mod(N)\rangle + ... + e^{-2\pi is(p-1)/p}|a^{p-1} mod(N)\rangle + e^{-2\pi is(0)/p}|a^0 mod(N)\rangle)$$

As you can see, we have just manipulated this equation to form our u_s state again.

$$|u_s\rangle = \frac{1}{\sqrt{p}}(e^{-2\pi is(0)/p}|a^0 mod(N)\rangle + e^{-2\pi is(1)/p}|a^1 mod(N)\rangle + ... + e^{-2\pi is(p-2)/p}|a^{p-2} mod(N)\rangle + e^{-2\pi is(p-1)/p}|a^{p-1} mod(N)\rangle)$$

(though the $a^0 mod(N)$ state is at the end in the other expression)

Hence,

$$U|u_s\rangle = e^{2\pi is/p}|u_s\rangle$$

Therefore, we can say that the u_s state is an **eigenvector** of the U matrix, with an **eigenvalue** of $e^{2\pi is/p}$.

The reason I'm showing this is to prove we can use the **Quantum Phase Estimation** (QPE) algorithm to find p, which approximates $e^{2\pi is/p}$ given its eigenvector and the matrix U, which I will briefly explain.

In actual quantum computers, although we could do this on a single u_s state, we realise it's much easier to construct the equal superposition of all u_s states. This is because each u_s state has a slightly different eigenvalue (as s is different), so accounting for them all in the algorithm allows for a better approximation of p (though this is a simplification).

$$\frac{1}{\sqrt{p}}\sum_{s=0}^{p-1}|u_s\rangle = |1 mod(N)\rangle$$

I'll attach a proof of this in the further reading section, but I'm worried the essay is getting too long.

Ultimately, the phase estimation circuit can then approximate the eigenvalue $e^{2\pi i s/p}$ for one s, where $0 \leq\ s\ \leq p - 1$.
This is because

$$\frac{1}{\sqrt{p}}\sum_{s=0}^{p-1}|u_s\rangle = \frac{1}{\sqrt{p}}(|u_0\rangle + |u_0\rangle + \ldots + |u_{p-1}\rangle)$$

so when we measure the remainder ‘s’, we will get the eigenvalue for one of the u_s states.

From the quantum phase estimation algorithm, we get a decimal number, j, which is equal to s/p (from measuring the eigenvalue’s exponent).

The algorithm uses a technique called continued fractions to convert j into a fraction (of certain precision). Since $j = s/p$, we find p by choosing estimates where the denominator is less than N (as p < N) and checking if $a^p = kN + 1$.

And viola, after a series of steps and algorithms, some more complicated than the others, we have finished our method of finding p, and thus $(a^{p/2} + 1)(a^{p/2} - 1) = kN$, allowing us to easily find N through Euclid’s algorithm.
I don’t know about you, but I need a nap after that. We went from exploring what a qubit is to investigating how to mathematically apply this to famous quantum algorithms. Trust me, there are barely any resources on these topics online, so understanding this crazy concept is impressive. Congratulations.

I will very quickly show an example of this.

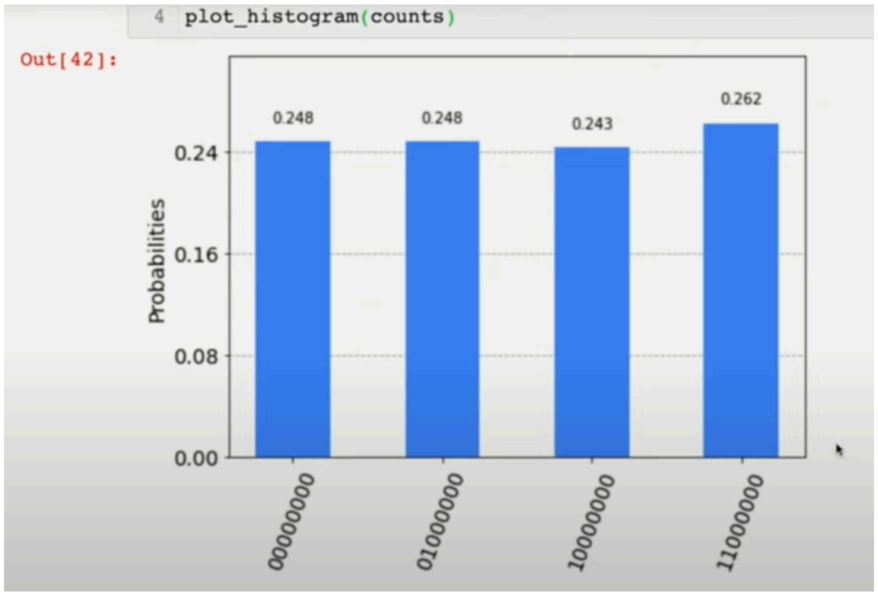
Example

Let’s assume I’m trying to factorise 15, and my random guess at a prime factor is 7 (this is a tad bit concerning for a Year 12 student, I must admit, but oh well).

Modular exponentiation list:
$7^0 = 1 mod(15)$
$7^1 = 7 mod(15)$
$7^2 = 4 mod(15)$
$7^3 = 13 mod(15)$
$7^4 = 1 mod(15)$

I'm going to skip over the QFT and QPE parts, which is what occurs under the hood of a quantum computer, because we can see that p = 4.

This can be implemented through code as well, such as on IBM's Qiskit.



```
plot_histogram(counts)
```

```
Out[42]:
```

(From a Qiskit course, 2021)

This shows, in binary, that the possible values of p are 1, 2 and 4, where r = 4 has the greatest probability.

$(a^{p/2} + 1)(a^{p/2} - 1) = kN$

$(7^{4/2} + 1) = 50$

$(7^{4/2} - 1)$ = 48

Therefore, 50 and 48 share factors with 15.

The common prime factors of 50 and 48 are 5 and 3, so **5 and 3** are the prime factors of 15.

Epic! We can factorise 15 into 5 and 3. But c'mon, you're staring at me right now thinking that you could already do that. Why did you read all of this just to factorise 15 with some crazy complicated quantum mathematics?

Well, if you are thinking that…

Factorise
“35794234179725868774991807832568455403003778024228226193532908190484670252364677411513516111204504060317568667”

Yeah, who’s laughing now? This is the RSA-110 and is nowhere close to the numbers that are used to encrypt our data today (phew). However, anybody with a working quantum computer today could theoretically breach highly sensitive data, such as bank details or IDs, meaning our security protocols may become redundant. This is highlighted by the National Security Administration (NSA,2024), which states that “a sufficiently large quantum computer, if built, would be capable of undermining all widely-deployed public key algorithms used for key establishment and digital signatures”, exposing the fragility of current protection and how the outcomes of Shor’s algorithm may be “devastating”.

However, the important part is the phrase “if built”. Shor’s algorithm is highly theoretical and assumes we have access to millions of qubits that don’t decohere (don’t interfere with surrounding noise or radiation) - whereas the 2nd largest quantum computer today (IBM’s Condor) is just shy of 1125 qubits (Gambetta, 2023). This is because maintaining many entangled qubits is extremely difficult, as they can decohere easily (if one decoheres so do the rest). Moreover, this fear of decryption is catalysing the creation of new encryption, such as lattice-based encryption, in the hopes it will protect us from this newly emerging technology.

Now this is maths, but not as you know it (thanks Tom Rocks Maths).

Conclusion

Nonetheless, we’ve engineered a method to factorise numbers made of 2 prime factors, which is how most of our data is encrypted worldwide. Despite having some limitations, such as the value of p not always being correct (for example, if p is odd, the algorithm is repeated as p/2 wouldn’t be an integer), there is a 99% success rate provided the algorithm is repeated 10 times (minutephysics, 2020), which are good odds for a quantum computer.

In fact, this algorithm is so powerful that when Peter Shor solved the ‘factoring problem’, he received a call 5 days later from Umesh Vazirani (according to Qiskit, 2022) regarding how it works (which I told you - you’re welcome), and rumour of this spread very rapidly despite Shor only telling one or two people.

Recently, after 17 years of research, Microsoft announced their Majorana 1 Chip, which uses Majorana quasi-particles to form a 'topological' qubit array, which requires every single atom to be precisely positioned, spreading quantum information across a topology (holding electrons at the ends of a nanowire) rather than localised particles (Bolgar,2025). This prevents decoherence of the quantum state despite the presence of noise.
Hence, with this improved error correction, the pathway to realising 1 million qubits continues as you read this, and so mathematics may very well be utilised to change the future of cybersecurity sooner than we think.

If the internet breaks, don't blame me! I'm just doing maths…

Everything below here is not included in the word count.

Word count: 4070
(I appreciate the unintentionally large word count; quantum mechanics is such a large topic that isn't covered in school, so I felt there were a lot of fundamentals to cover. Plus, there are a lot of ideas in Shor's algorithm that I wanted to touch on, so I hope this extended essay was enjoyable).

Further reading if interested:

Fourier Transforms: https://www.youtube.com/watch?v=spUNpyF58BY

Hilbert Spaces (for our quantum states):
https://www.youtube.com/watch?v=_kJUUxjJ_FY

Proof of:

$$\frac{1}{\sqrt{p}}\sum_{s=0}^{p-1}|u_s\rangle = |1 mod(N)\rangle$$

by Quantum Soar:
https://drive.google.com/file/d/1h8Gt6BPE72Y43xbYBuPOlE4fgyzAyUvc/view

Euler's Theorem:
https://en.wikipedia.org/wiki/Euler%27s_theorem

https://brilliant.org/wiki/eulers-theorem/

Acknowledgements:

Quantum Soar - their videos on quantum computing are how I first learned about the mathematics behind it. There is basically no other resource on YouTube covering this, so this was a huge help for my understanding. You will notice a lot of my notation in this essay is similar, though I salvaged other resources to complete the essay (as I sometimes found the video a bit tricky personally).

MinutePhysics:
https://www.youtube.com/watch?v=lvTqbM5Dq4Q
This video specifically was very helpful for my learning of Shor's algorithm for the first time, and I have included some pictures from their video in my essay as well.

Bibliography:

(Last double-checked on 06/04/2025)

Google's Willow Chip (Neven,2024):
https://blog.google/technology/research/google-willow-quantum-chip/

Google's Willow Chip (BBC News, 2024):
https://www.bbc.co.uk/news/articles/c791ng0zvl3o

Shor's Algorithm:
https://en.wikipedia.org/wiki/Shor's_algorithm

NSA's Report on Quantum Computing (NSA,2024):
https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/1/CSI_CNSA_2.0_FAQ_.PDF

Microsoft's News on their Majorana 1 Quantum Chip (Bolgar,2025):
https://news.microsoft.com/source/features/innovation/microsofts-majorana-1-chip-carves-new-path-for-quantum-computing/

Pictures from MinutePhysics (which were cited):
https://www.youtube.com/watch?v=lvTqbM5Dq4Q

freeCodeCamp's mathematical course on Quantum Computing (freeCodeCamp,2024), which is where I took the pictures of the Bloch sphere:
https://www.youtube.com/watch?v=tsbCSkvHhMo

Umesh Vazirani's call to Peter Shor (according to Qiskit, 2022) at 2:34:
https://www.youtube.com/watch?v=dONacVnW1Ng

Qiskit's Implementation of Shor's Algorithm is demonstrated here:
https://www.youtube.com/watch?v=EdJ7RoWcU48&t=821s

IBM’s Condor:
https://en.wikipedia.org/wiki/IBM_Condor

(Gambetta, 2023):
https://www.ibm.com/quantum/blog/quantum-roadmap-2033

Author of essay: Mustafa Ali