The Qubit and Shor's Algorithm:

A step-by-step exploration.

A research project

By

AHMED MAGHRI

Bachelor of Computer Science.

Class of 2027

Constructor University Bremen

March 2025

Major subjects: Mathematics & Computer science.

Abstract

Shor's algorithm presents a polynomial-time quantum solution to the integer factorization problem, threatening classical encryption protocols like RSA. This article introduces the mathematical and computational foundations required to understand the algorithm, including modular arithmetic, continued fractions, and quantum principles. We provide a step-by-step breakdown of Shor's algorithm and its quantum subroutine, emphasizing accessibility and conceptual clarity, while also addressing practical constraints in current quantum hardware. This article aims to make the concepts of Shor's algorithm accessible to readers with limited background in mathematics and quantum computing, while maintaining mathematical rigor and conceptual depth throughout.

Acknowledgements

I would like to sincerely thank Omar Elshinawy, head of the Mathematics Society at Constructor University, for his valuable guidance and support throughout the research and writing of this article. I also wish to thank Muaz Hussein, a first-year ECE major, for providing the first reviews and constructive feedback on the manuscript.

A heartfelt thanks goes to my mathematics and computer science professors and fellow Math Society members, whose enthusiasm and insight made this research journey both enriching and enjoyable.

Finally, and most importantly, I express my deepest gratitude to my parents, without whom none of this would have been possible.

Contents

0 Preliminaries.

This article has been made such that almost any person is able to understand the whole content, however, a minimum of reasoning, computer science, and mathematics concept will be required, and to ensure that the reader has these requirements, here are some definitions and concepts that we are going to use during our journey (and even some advanced definitions). The reader is not obliged to read it now, but whenever he needs to.

1- Number's Theory:

- **Modular arithmetic:** We say $A \equiv B \pmod{N}$, when the division of A by N gives a remainder of B. Quick example: $\frac{5}{2}$ gives a remainder of 1 ($5 = 2 \times 2 + 1$). Thus we say: $5 \equiv 1 \pmod{2}$. If this is still a bit unclear for you, let's say that you take the number A and subtract N until you can't. In our previous example:

$$5 - 2 = 3, \quad 3 - 2 = 1$$

  The last number is the remainder.

- **A is a divisor of B:** A is able to divide B, A/B gives an integer.

- **The gcd of A and B (gcd(A, B)):** It's the greatest common divisor of A and B. For instance: 24 and 18 have as common divisors: 1, 2, 3, 6. So $\gcd(24, 18) = 6$.

- **A and B are coprime:** $\gcd(A, B) = 1$. Meaning that A and B have no common divisors (not counting the trivial 1, which is a divisor of all whole numbers). For example, $\gcd(15, 8) = 1$, so 15 and 18 are coprime.

- **The order of A modulo N:** is the smallest number r such that $A^r \equiv 1 \pmod{N}$.

- **A is semiprime:** A just has two divisors (not counting 1 and A itself). For example, 15 is semiprime. Another way to see it is that A is the product of two prime numbers.

2– Computer Science:

- **Polynomial time:** We say a problem can be solved in polynomial time if: Given a problem of size n, the problem is solvable in a time of n^r, with r a real number.

- **P problems:** Are the problems for which we can find solutions in polynomial time.

- **NP problems:** Problems for which we can't find solutions in polynomial time. Also explained as problems where we can easily check if a given solution is correct, but finding that solution is very hard.

- **An algorithm:** A set of instructions that is designed to accomplish a task.

- **A private key:** A key that is known only by its owner.

- **A public key:** A key that is known by everyone.

- **Converting binary to decimal:** Let's say 1101 is our binary number. You compute:

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$$

  This helps illustrate the pattern. If not clear, a quick Google search will explain it very well.

- **Number of possible states:** Something we will use often: if we have n bits, then we have 2^n possible states.

- **BQP problems:** BQP stands for Bounded-Error Quantum Polynomial Time. These are decision problems that can be solved by a quantum computer in polynomial time with a probability of at least $\frac{2}{3}$ of being correct. The error probability is at most $\frac{1}{3}$ for all instances.

3– Computer Architecture:

- **A bit:** A bit (short for binary digit) is the smallest unit of information in classical computing. It can take on one of two possible values: 0 or 1.

- **A register:** A register is a small, fast storage location inside a computer's processor that holds data temporarily. In classical computing, it typically consists of several bits and is used for arithmetic or logical operations.

4– Linear Algebra:

- i**:** Represents the elementary complex number, where $i^2 = -1$.

- **Transpose of a matrix:** Given a matrix M, its transpose is the matrix with rows and columns swapped. For example, here is a matrix and its transpose:

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 3i \end{bmatrix}, \quad M^T = \begin{bmatrix} 1 & 3 \\ 2 & 3i \end{bmatrix}$$

- **Complex conjugate of a matrix:** Given a matrix M with complex entries, its complex conjugate is basically the same matrix but negating each element that has i in it:

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 3i \end{bmatrix}, \quad \overline{M} = \begin{bmatrix} 1 & 2 \\ 3 & -3i \end{bmatrix}$$

5– Quantum Computing:

- **Physical qubits:** Physical qubits are the actual hardware qubits built using physical systems (like trapped ions or superconducting circuits).

- **Logical qubits:** Idealized qubits, error-corrected, essentially used in theory. Approximately 1 logical qubit $\approx$ 10,000 physical qubits.

- **Transformation to matrix:** Each quantum transformation can be represented as a matrix, or as its direct application on a qubit (for linear algebra adepts, you can see it as when you can represent a transformation in its matrix form or directly how it acts on the vectors).

6– Set's Theory:

- $\mathbb{N}$**:** Natural numbers — counting numbers like 1, 2, 3, ...

- $\mathbb{Z}$**:** Integers — all whole numbers, positive and negative, including zero: $..., -2, -1, 0, 1, 2, ...$

- $\mathbb{Q}$**:** Rational numbers — numbers that can be written as a fraction $\frac{a}{b}$ where $a, b \in \mathbb{Z}$ and $b \neq 0$

- $\mathbb{R}$**:** Real numbers — all rational and irrational numbers (e.g., π, $\sqrt{2}$)

- $\mathbb{C}$**:** Complex numbers — numbers of the form $a + bi$, where $a, b \in \mathbb{R}$ and $i^2 = -1$

1 Introduction to Cryptography.

Cryptography is the study of mathematical techniques for securing communication and data in the presence of adversaries. Its primary goal is to ensure the confidentiality, integrity, authenticity, and non-repudiation of information. We are mainly dealing with two subjects in it: Encryption & Decryption.

Definition 0:

In cryptography, **encryption** (more specifically, **encoding**) is the process of transforming information in a way that, ideally, only authorized parties can decode.

1.1 RSA.

The RSA cryptosystem was introduced in 1977 by **Rivest**, **Shamir**, and **Adleman**, from whom it takes its name.I t is a public-key encryption scheme designed to securely encode messages over insecure communication channels by relying on the computational difficulty of factoring large composite integers. Let's discuss how it works and what is the algorithm behind it.

1.2 Algorithm of RSA.

Take Alice and Bob. Alice and Bob both have two big prime numbers (in general around 313 digits). Let's name the prime numbers of Alice P1 and P2, and Bob's P3 and P4.

Both Alice and Bob are going to multiply their two big prime numbers to get a new big semiprime number. Let's call them respectively SP1 and SP2.

P1 and P2 are then called the private keys of Alice, SR1 is her public key. And similarly for Bob.

If Alice wants to send something to Bob, she is going to hash her message in a way such that she uses his public key SP2, and the only way to decrypt her message, must be that her receptor knows the two big prime numbers P3 and P4, that created the public key SP2. In our case, Bob knows them, but no one else knows does. Both the public keys of Alice and Bob can be viewed by anyone, as long as her hashed message. However, due to the fact that only Bob knows what P3 and P4 are, he is the only one able to decode the hashed message.

Note 0:

That we are not covering how the message is hashed, and we won't. This part is not relevant for the rest of the article and thus is out of scope.

Note 1:

We call this an asymmetric key system because two different keys are used to send and to receive the message.

1.3 How to make RSA irrelevant?

The whole RSA is standing on one assumption: Finding the prime factors of a semiprime number is a NP problem. Meaning that if you have a big semiprime Number N, I can check if two prime numbers are its two prime factors (divisors). However, finding the two prime factors takes way more time, it is possible, but for example for a 313 digits semiprime number, it may take 24000 years to find them, even for a supercomputer. Thus, to be able to make RSA breakable, it is enough to show that the problem of finding the prime factors of a semiprime number is actually a P problem, so if we find a way to find efficiently factor a semiprime number, we broke the RSA system.

And this is where Shor's algorithm comes.

1.4 Introduction to Shor's Algorithm.

Shor's algorithm, developed by **Peter Shor in 1994**, is a quantum algorithm for efficiently factoring large integers. It runs in polynomial time on a quantum computer and poses a fundamental threat to classical cryptographic systems such as RSA, whose security relies on the hardness of integer factorization.

But before explaining how it works, you may have noticed that there is a quite scary word in its definition, "quantum". Thus, before explaining the algorithm and how it isg implemented, for the sake of the reader's comfort, I will have to give you a minimum knowledge about the world of Quantum Computing.

2 Introduction to the world of quantum computing.

Note 2:

Our goal in this article is not to focus on quantum computing, there are multiple lectures about it, and some really advanced topics will be needed during section 3.3 to explain the implementation of Shor's algorithm, and these concepts need their own article. If you feel a bit lost during the quantum computing parts, this is totally normal, but I am going to do my best to keep you on board.

Note 3:

While reading this article, you may encounter some intermediary algorithms, while the article will still give a short explanation of them, again our goal won't be to go deeply into them. If you would like to know more about them, a quick google search will serve you.

2.1 Theoritcal part.

Briefly: Quantum computers use quantum bits (qubits) instead of normal bits.

We all heard this, without really getting what is happening behind it. We also heard the famous sentence: While bits must be either in a 0 or 1 state, a qubit can be in a superposition of states $|0\rangle$ and $|1\rangle$. Some people also heard that when you measure a quantum state, you can't measure the superposition, you just randomly measure one of the possible states. At measurement, you can say that the quantum states collapse into exactly one. Let's try to discover what is really happening behind the scenes.

Formally, a quantum state (so a state "between" the $|0\rangle$ and the $|1\rangle$) is described by amplitudes α and β, with probability $|\alpha|^2$ of measuring 0, and probability $|\beta|^2$ of measuring 1. Thus, we have:

$$|\alpha|^2 + |\beta|^2 = 1$$

And we write for a state X:

$$|X\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

For a state with 2 qubits, instead of 1, we get 2^2 possibilities:

$$|X\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle$$

$$|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$$

As an example, consider the state Y where:

$$|Y\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$$

If we try to measure the state Y, we have equally $\frac{1}{4}$ probability of measuring either 00, 01, 10, or 11.

2.2 Physical part.

But how do we create such a computer? How do we create a qubit? Usually in classical computer, we used to create a bit of 0 or 1 by using the "presence "or "absence "of electricity, but we cannot do it for quantum computers, there is no "electricity a bit present or a bit absent".

We had to find a new way to describe the 0 and the 1 state.

A widely used method to build qubits is by using the **spin of an electron**, often the electron of a phosphorus atom embedded in silicon. The two possible spin orientations — **spin-up** and **spin-down** — are used to represent the states $|1\rangle$ and $|0\rangle$ respectively. But in contrast to classical bits, the electron's spin can also be in a superposition of both up and down at once, meaning it's partly $|0\rangle$, partly $|1\rangle$. This is what allows the qubit to behave fundamentally differently from a classical bit and makes quantum computing possible.

Now that we have our foundations, we can now move the interesting part I would say.

3 Literally, Shor's algorithm.

Let's first state clearly the goal of the algorithm: **Given a semiprime number$N$, find the two prime numbers $P1$ and $P2$ such that: $P1 \times P2 = N$.**

The classical way to find these numbers is just by guessing randomly, also known as the brute force method. The essence or the main idea of Shor's algorithm will be that, given a random guess g, transform this bad guess into a better guess.

3.1 Initialization.

We start by guessing randomly a number g such that $1 < g < N$. This lead to two cases:

3.1.1 We are lucky.

g is a factor of N, or shares common factors with N. In this case, we are done thanks to Euclid's algorithm.

Definition 1:

Euclid's Algorithm is a classical method for efficiently computing the **greatest common divisor** (GCD) of two integers. It works by repeatedly replacing the larger number with its remainder when divided by the smaller, until the remainder is zero. The last non-zero remainder is the GCD.

Note 4:

Using that, to find a factor of N, we don't need then to guess a factor, just to find a number that shares factors with N.

After finding one of the prime numbers, let's assume $P1$, compute $N/P1$ and this will give you $P2$.

3.1.2 Most probably, we won't be lucky.

And this will make us start our process of making our bad a guess a better guess. To achieve that, we will borrow a quite helpful theorem.

Theorem 0:

For any two whole numbers A and B that are coprime, there exists a power r and a multiple m such that:

$$A^r = m \cdot B + 1$$

Equivalently:

$$A^r \equiv 1 \pmod{B}$$

A good example to visualize it will be $A = 2$ and $P = 5$, we find that $r = 1$:

$$2^1 \equiv 2 \pmod 5 \qquad 2^2 \equiv 4 \pmod 5 \qquad 2^3 \equiv 3 \pmod 5 \qquad 2^4 \equiv 1 \pmod 5$$

In our case, our A is our bad guess g, and B is our semiprime number N that we want to factor, and ultimately, we know the existence of our r.

3.2 Making the bad guess a better guess.

Let's assume that we found r, such that $g^r \equiv 1 \pmod N$. Bring 1 to the left and you get:

$$g^r - 1 \equiv 0 \pmod N$$

And using our famous identity: $(a+b)(a-b) = a^2 - b^2$:

Replace a by g, and b by 1, and going from the right side of the equality to the left, we get:

$$\underbrace{(g^{r/2} - 1)}_{\text{P1'}} \cdot \underbrace{(g^{r/2} + 1)}_{\text{P2'}} \equiv 0 \pmod N$$

Notice that r has to be even, otherwise, r/2 won't be a whole number. If we found an r odd, we repeat our algorithm from the beginning.

What we found now is actually really close to what we want, instead of finding the two numbers P1 and P2 such that $P1 \times P2 = N$, we found that N divides the product of two other numbers P1' and P2'. Which can be written as:

$$P1' \times P2' = mN \quad , m \in \mathbb{N}$$

Even more precisely, we found two numbers P1' and P2' that share factors with N. Thus:

$\gcd(\text{P1'}, N) = \text{P1 or P2.}$ (We can replace P1' by P2', the outcome stays the same).

And remember that thanks to our beautiful Euclide's algorithm, we are able to compute

$$\gcd(\text{P1'}, N) = P1$$

easily. Compute then $\frac{N}{\text{P1}} = \text{P2}$, (and vice versa if we got that gcd(P1', N) = P2). And we have successfully found our P1 and P2.

End of our algorithm.

Or maybe not?

You may feel that this is too easy, in the sense where, using this algorithm we could directly break the RSA, why are we using it? Remember also that I gave you an introduction about quantum computing, but I didn't talk at all about it at any point.

Let's recall the algorithm's main steps.

1 – Make a guess, $g < N$ that shares no factors with N.

2 – Find r such that $g^r = mN + 1$.

3 – If r is even, calculate $(g^{r/2} + 1)$ and $(g^{r/2} - 1)$. If r is odd, go back to step 1.

4 – Use Euclid's algorithm to find the greatest common divisor.

Where is the **quantum** here?

3.3 The quantum subroutine of Shor's algorithm.

Notice that we didn't discuss before how we are finding r, which is in fact that hardest (probably due to the quantum concepts) part of the whole algorithm.

Note 5:

Through this part, the quantum part of the explanation (which is still the main thing) is going to be a bit simplified, essentially when we come to some deep notions such as Quantum Fourier Transform, etc. Such quantum concepts as said before need even their own whole article. And as again mentioned before, one of the primary goals of the article is to keep the reader on board. There will be some analogies, to the classical architecture of a computer, for the sake of your understanding, please go back to section 0 and give a quick overview to the preliminaries again.

3.3.1 Initialization.

To achieve this quantum subroutine, we will need two quantum registers, as normal registers that we have in normal computers, these ones are just going to contain qubits instead of normal bits.
First register: is going to be 2n qubits, initialized as:

$$|0\rangle^{\otimes 2n}$$

The number n just determines how accurate we want our calculations to be, however, the 2nd register size depends on the first one.

Second register is going to be n qubits, initialized to: $|1\rangle$
Note that $\otimes$ in context of quantum computing doesn't mean the XOR gate as a lot of people would think. It is a tensor product. Just consider it as a normal multiplication but between the quantum states.

Note also that for the first register, we mean that all the bits are set to 0, meaning that the state of that register is: $|000\ldots000\rangle$.

But for the second register, we mean that the value of that register is one, meaning that the state of that register is: $|000\ldots001\rangle$. It is just a story of notations.
What does it do? Applying the Hadamard Transform to a $|1\rangle$ or $|0\rangle$ gives this:

Example 1. Applying this to $0\rangle$ and $1\rangle$ gives

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

3.3.2 Hadamard Transform.

Let's first define how the Hadamard Transform works. The Hadamard gate is formally defined as this:

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

What does it do? Applying the Hadamard Transform to a $|1\rangle$ or $|0\rangle$ gives this:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

During this part, we will be interested essentially in its application on the 0 state, since our first register is just full of 0's.

Notice that on 0, it created exactly a state such that the probability of getting one of the states x such that x in $\{0, (2^1 - 1)\}$ (converting the decimal to binary) is exactly equal.

Let's try to apply it to 0 states with 2 qubits.

$$H^{\otimes 2}|00\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

And on 3 qubits?

$$H^{\otimes 3}|000\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

Did you notice the pattern? If not let me show it to you:

Whenever we apply the H gate to a register of size n, such that all its qubits are 0, we get a state such that the probability of measuring a state in $\{0, 1, 2, \ldots, 2^n - 1\}$ is exactly equal for all of them.

You may wonder now why we are doing that, we will see why soon.

To end this part, we will formally say that the state of our first register now is:

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

3.3.3 U.

We will apply another transformation, but to the 2nd register. Let's call this transformation U. U's definition is part of the algorithm, meaning that it is not a famous transformation created from before. Formally, the definition of U is:

$$U|k\rangle = |g \cdot k \bmod N\rangle$$

With g the random guess that we picked before.

Even more formally:

$$U|k\rangle = \begin{cases} |k\rangle & \text{if } 0 \leq k < N \\ |g \cdot k \bmod N\rangle & \text{if } N \leq k < 2^n \end{cases}$$

One good property of U is that U is a unitary transformation: which means that if you take the transpose then the complex conjugate of its matrix representation, it gives you U back.

How are we going to use U? First let's look at our state now.

The quantum state of our full system (including the 1st and the 2nd register) is the tensor product $\otimes$ of both first and second register. As I said $\otimes$ is just a normal multiplication of the states. Our first register is now a sum of equal states, and our second register is still just in a 1 state. "Expanding" the expression:

$$\left(\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle\right) \otimes |1\rangle$$

gives:

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (|x\rangle \otimes |1\rangle)$$

Don't overthink about it, it is the exact same way as you would expand $(a+b+c)d = ad+bd+cd$. For each x (the values in each state from 0 to $2^n - 1$), we apply the transformation x times on the qubit at state 1 associated to it. It is a bit hard to understand it like that, we can use a visualization to understand it more:

For each pair $|x\rangle \otimes |1\rangle$, we apply U getting:

$$|x\rangle \otimes |1\rangle \longrightarrow |x\rangle \otimes U^x |1\rangle = |x\rangle \otimes |g^x \bmod N\rangle$$

As an example, applying U to a state $|101\rangle \otimes |1\rangle$ (101 read as $4+0+1=5$, recall the preliminaries) gives:

$$|101\rangle \otimes U^5 |1\rangle = |101\rangle \otimes \left|g^5 \bmod N\right\rangle$$

And again, g is our random guess that we picked before.

Notice that when applying the transformation, we will get a certain periodicity in our quantum system due to modulo properties. To visualize it even better, let us consider that our x is going

from 0 to 7 (using thus 3 qubits), that our N was 26, and our random guess was 3, after applying all what we had before, we will get:

$$\frac{1}{\sqrt{8}}(|000\rangle \left|3^0 \bmod 26\right\rangle + |001\rangle \left|3^1 \bmod 26\right\rangle + |010\rangle \left|3^2 \bmod 26\right\rangle +$$

$$|011\rangle \left|3^3 \bmod 26\right\rangle + |100\rangle \left|3^4 \bmod 26\right\rangle + |101\rangle \left|3^5 \bmod 26\right\rangle +$$

$$|110\rangle \left|3^6 \bmod 26\right\rangle + |111\rangle \left|3^7 \bmod 26\right\rangle)$$

Which exactly gives:

$$\frac{1}{\sqrt{8}}(|000\rangle |1 \bmod 26\rangle + |001\rangle |3 \bmod 26\rangle + |010\rangle |9 \bmod 26\rangle +$$

$$|011\rangle |1 \bmod 26\rangle + |100\rangle |3 \bmod 26\rangle + |101\rangle |9 \bmod 26\rangle +$$

$$|110\rangle |1 \bmod 26\rangle + |111\rangle |3 \bmod 26\rangle)$$

Note 6:

$|x\rangle \otimes |y\rangle$ is the same as $|x\rangle |y\rangle$

Notice also that, the period of the periodicity is 3, which is exactly the number r that we are searching for, $3^3 \equiv 1 \pmod{26}$.

Thus, if we are able to extract the periodicity from the state, we would be done.

3.3.4 The periodicity, aka r.

This is actually a way harder task than expected. When you see the representation of the state in front of you it seems obvious, however, as we said before, if we just measure now our state it is just going to give us a random state since they all have equal probability. Which is not what we want. In fact, we are not able to extract the periodicity. However, we have something called the QFT (for **Q**uantum **F**ourier **T**ransform).

The QFT is formally defined as:

$$\text{QFT}\,|x\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{2\pi i x k}{N}} |k\rangle$$

Where N is the number of qubits that represent x.

The QFT doesn't extract periodicity, however, it is able to extract the frequency. And remember that

$$\text{Period} = \left(\frac{1}{\text{frequency}}\right)$$

Without entering too much into the details, believe me, the QFT would need a whole article for itself, briefly: The QFT in Shor's algorithm transforms the hidden period r of $g^x \bmod N$ into measurable frequency information, allowing us to extract r from a single quantum measurement.
Basically, the amplitudes (that represent the probability of being measured) C are **peaked** near j/r, where j is just the state that we randomly measured, since a frequency function just works with adding $1/r$ each time.
Thus, since the highest probability is near to j/r, we most probably are going to measure j/r.
But now yes we have j/r, how do we get r? If you measure $j/r = 0.312$, how can we extract r?
Luckily we already have a classical (runnable on a classical computer) algorithm that can extract r.

3.3.5 Continued Fraction Algorithm (CFA).

The best way to explain the Continued Fractions Algorithm is to give an example.
Assume that we found that j/r = 0.312
We start by writing it as a fraction:

$$0 + \frac{312}{1000}$$

We then reverse the fraction and get this:

$$= 0 + \frac{1}{\frac{1000}{312}} = 0 + \frac{1}{3 + \frac{8}{39}}$$

And we repeat the same process until we get a 1 in the numerator.

$$0.312 = 0 + \frac{1}{3 + \frac{1}{4 + \frac{1}{1 + \frac{1}{7}}}}$$

For each fraction, we get a better approximation for 0.312:

$$0.312 \approx \frac{1}{3}, \quad j = 1, \quad r = 3$$

$$0.312 \approx \frac{1}{3 + \frac{1}{4}} = \frac{4}{13}, \quad j = 4, \quad r = 13$$

$$0.312 \approx \frac{1}{3 + \frac{1}{4 + \frac{1}{1}}} = \frac{5}{16}, \quad j = 5, \quad r = 16$$

We choose an approximation of r such that it follows two conditions:

1- It should trivially be less than N

2- It should be an even number.

If none of our approximations gives a valid r, we repeat the same process from the beginning.

And this gives an end to our quantum subroutine.

And an official end of our Shor's algorithm.

4 Final thoughts: does Shor's algorithm really break RSA?

During your lecture, you went through all the Shor's algorithm, you had a whole journey with it, and I hope that I was able to make you enjoy the journey. You probably have a lot of questions in your mind now, and one of them would be that you saw with your own eyes that we have an algorithm that breaks our usual encryption. So how is our data safe?

Actually, we not really did. We are assuming that we have a quantum computer that is able to run Shor's algorithm, and we don't. Shor's algorithm would need 4100 logical qubits, or 20 000 000 physical qubits to run.

Currently, the main actor in this field is IBM, and they are aiming for 1121 physical qubits soon, far from our 20 000 000 ones. But the day we will be able to make a quantum computer with 20 000 000 physical qubits, we will definitely be able to break RSA, so the question is not if we are going to be able to, but **when**?

References

1. **P. W. Shor** – Algorithms for Quantum Computation: Discrete Logarithms and Factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science*, IEEE, 1994.

2. **D. Deutsch and R. Jozsa** – Rapid Solution of Problems by Quantum Computation. *Proceedings of the Royal Society of London A*, 1992.

3. **S. Arora and B. Barak** – Computational Complexity: A Modern Approach. Princeton University Press, 2009.

4. **Kristopher Lonnie Watkins** – An Exposition on Peter Shor's Polynomial-Time Factoring Algorithm and Its Effects on Post-Quantum Cryptography.

5. **Classiq Technologies** – Quantum Cryptography – Shor's Algorithm Explained.

6. **Classiq Technologies** – Quantum Algorithms: Shor's Algorithm.

7. **Wikipedia** – Shor's Algorithm.

8. **Wikipedia** – BQP.

9. **Wikipedia** – Hadamard Gate.

10. **Wikipedia** – Quantum Fourier Transform.

11. **Wikipedia** – Euclidean Algorithm.

12. **Quantum Inspire** – Hadamard Gate.

13. **Veritasium** – How Does a Quantum Computer Work?

14. **Veritasium** – How to Make a Quantum Bit?

15. **Quantum Soar** – Quantum Computing Course: 3.8 Shor's Algorithm.