Images haven’t loaded yet. Please exit printing, wait for images to load, and try to print again.

Algorithm

Jonathan Hui Follow

Dec 12, 2018 · 7 min read

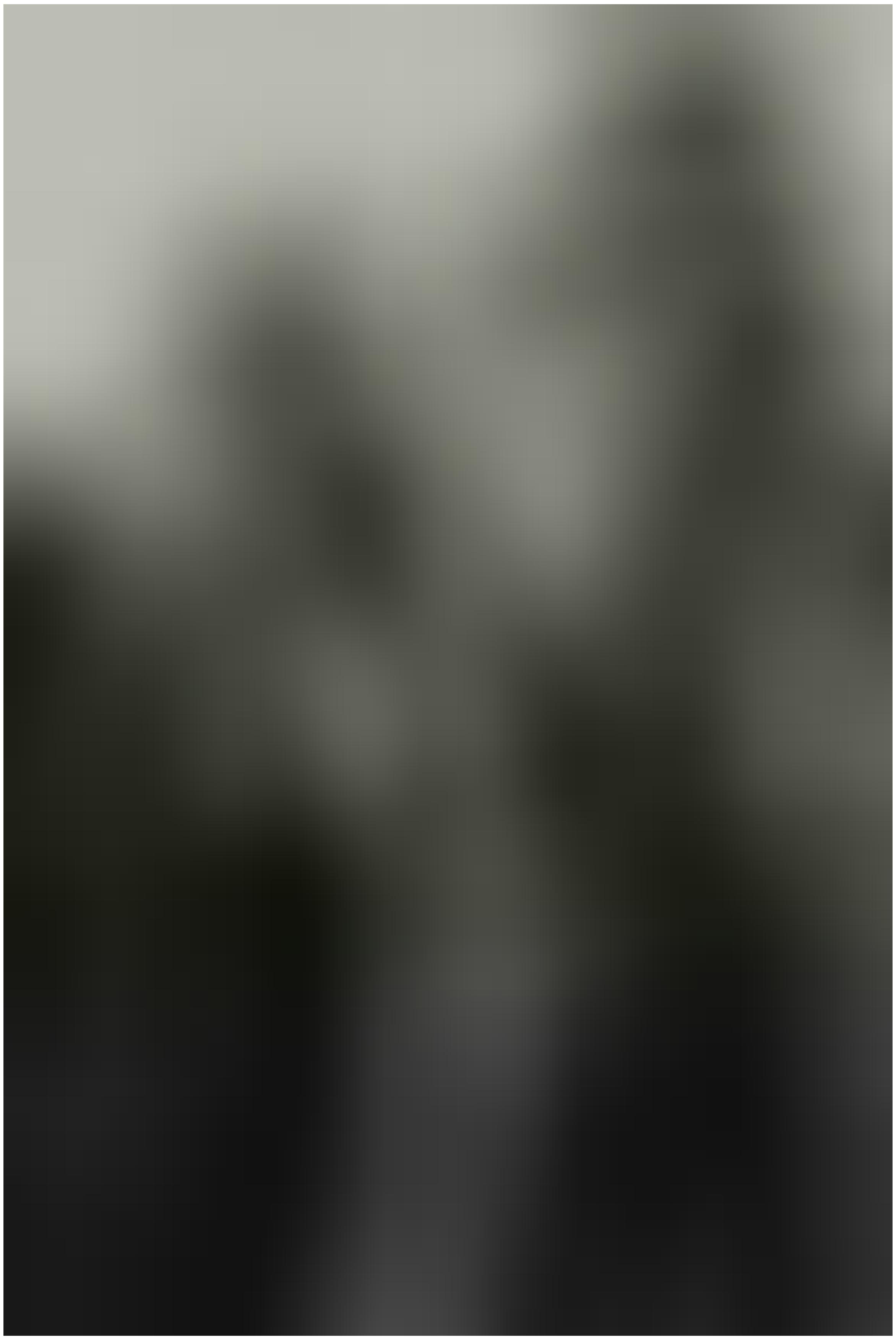


Photo by Ali Yahya

RSA is the standard cryptographic algorithm on the Internet. The method is publicly known but extremely hard to crack. It uses two keys for encryption. The public key is open and the client uses it to encrypt a random session key. Anyone intercepts the encrypted key must use the second key, the private key, to decrypt it. Otherwise, it is just garbage. Once the session key is decrypted, the server uses it to encrypt and decrypt further messages with a faster algorithm. So, as long as we keep the private key safe, the communication will be secure.

RSA encryption is based on a simple idea: prime factorization. Multiplying two prime numbers is pretty simple, but it is hard to factorize its result. For example, what are the factors for 507,906,452,803? Answer: 566,557 × 896,479.

Based on this asymmetry in complexity, we can distribute a public key based on the product of two prime numbers to encrypt a message. But without knowing the prime factors, we cannot decrypt the message to its original intention. In 2014, WraithX used a budget of $7,600 on Amazon EC2 and his/her own resources to factorize a 696-bit number. We can break a 1024-bit key with a sizeable budget within months or a year. This is devasting because SSL certificates holding the public key last for 28 months. Fortunately, the complexity of the prime factorization problem grows exponentially with the key length. So, we are pretty safe since we switch to 2048-bit keys already.

But as you may guess, the curse of its complexity has been solved by the Shor's algorithm. Shor's algorithm is published in 1994. Before that, quantum computing is more like a curiosity. The introduction of Shor's algorithm really changes the tone. It solves a real problem that cannot be solved by classical computers efficiently. It is the kind of paradigm shift that attracts investments. While we have discussed algorithms with a "to be done" oracle function, Shor's algorithm is a real deal. We are just waiting for a quantum computer with enough qubits.

Shor's algorithm involves many disciplines of knowledge. We try to be comprehensive and wish you can proceed with the speed you like. But we will not cover every implementation details since we have a lot to cover already.

RSA Algorithm

The following is the RSA algorithm. You can take our words for now that if we know how to do prime factorization effectively, RSA will be history.

RSA algorithm

However, we do need to understand a couple terms above. Greatest common divisor (**gcd**) finds the largest divisor between two numbers. For example,

$$\gcd(36, 60) = \gcd(3\times3\times4, 3\times4\times5) = 12$$

Two numbers are **co-prime** if they don’t have any common factors.

$$\gcd(15, 77) = 1$$

“**mod**” is the modulo operator that programmers familiar with (e.g. 14 % 10 = 4). Let’s review some of the modulo calculations.

For example,

$$7^3 \bmod 15 = 7^2 \times 7 \bmod 15 = 49 \times 7 \bmod 15 = 4 \times 7 \bmod 15$$

These tricks become handy to compute

$$f(a) = x^a \bmod N$$

Prime factorization complexity

With classical computing, the best we can solve prime factorization is:

$$O\left(\exp\left(\sqrt[3]{\frac{64}{9}n(\log n)^2}\right)\right)$$

where *n* is the number of bits to represent the product of the prime numbers. Shor's algorithm can do it in

$$O(n^3 \log n)$$

with the number of gates about

$$O(n^2 \log n \log \log n)$$

Prime factorization

But to solve it, we need to formalize the solution in an unusual way. Let's find the prime factors for 21 with the equations below. Magically, these equations end with the number 7 and 3 which is our answer.

But, this is not simple luck. There are many mathematical theories behind them. The basic idea is to find a number (8 in our case) with its square equals the term on the right below.

$$8^2 \equiv \underline{1 \pmod{21}}$$

So, let try our luck again. Start with a random guess *x=2*. First, we want to know whether *x* and *N* are coprime. That can be done with the **Euclid's Algorithm**. Below is an example to find the common factor between 21 and 15.

So 3 is the common factor for 21 and 15 and therefore 21 and 15 are not co-prime. If *x* and *21* are not co-prime, *gcd(x, 21)* will be one of the prime factors and we are done. But don't expect it happen frequently in a real problem. Most likely, *x* is a co-prime with *N*. Now, we compute the following series of powers function.

$$x^a \pmod{21}$$

i.e with *x=2*.

This function has a period of 6 (***r*** = *6*), i.e. the function values repeat itself every other 6 values. If ***r*** is divided by 2, it becomes 3. The number 8 we want is simply *pow(2, 3),* i.e. *pow(x, 3)*.

$$2^6 \equiv 1 \pmod{21}$$
$$(\underbrace{2^3}_{8})^2 \equiv 1 \pmod{21}$$

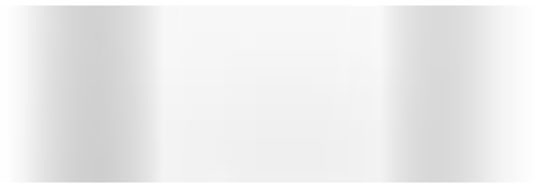
In summary, we start with a guess on ***x*** and verify whether it is co-prime with *N*. If not, we use gcd to find the prime factors. Otherwise, we compute the period of the power function.

$$x^a \bmod N$$

If the period ***r*** is even, the number we are seeking is *pow(x, r/2)*—the one underlined in red below. If ***r*** is not even, we take another guess on ***x*** and try again.

$$(\underline{x^{r/2}})^2 \equiv 1 \pmod{21}$$

In practice, you should have a pretty even chance of getter the period to be even. Consider the case for ***N = 15***. We have



So many choices of *x* will lead us to the right solution. The odds are pretty favorable. The difficult part is finding the period of the modulo function. Shor's algorithm leads us to a class of algorithms called Bounded-error probabilistic polynomial time (BPP). In complexity theory, we calculate the complexity for the worst case scenario, say ***O(n)***. In practice, there are problems that finding the solution in polynomial time is the common norm even though the worst case scenario is still exponential (but the chance is usually extreme small).

Unfortunately, finding the period of the modulo function above cannot be solved easily by classical computing. That is where quantum computing comes in. Once the period of the following function

$$f_x(a) = x^a \bmod N$$

is found by quantum computing and it is even, we compute the *gcd*. This is easily done with Euclid's Algorithm as we discussed before.

But to understand the period finding method, we need to cover a few basic concepts first.

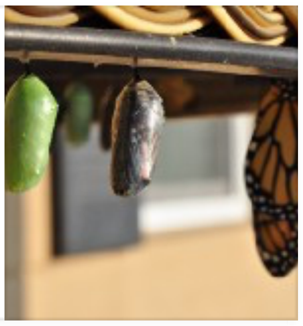
Next

Now, we finish the pre-processing and the post-processing part of Shor's algorithm. Next, we will talk about Quantum Fourier Transform—our first piece of the puzzle.

QC—Quantum Fourier Transform

Quantum Fourier Transform is a very critical part in Shor's Algorithm and many other quantum…

medium.com



Here is the link for the whole series:

QC—Quantum Computing Series

This series covers the basics of quantum computing. The first 2 articles cover the basics…

medium.com