Published in Cantor's Paradise

Varun Bansal Follow

Sep 27, 2021 · 8 min read · Listen

RSA Algorithm: In-Depth Mathematical Walk-through

Understand the concepts to generate your own key pair!

Photo by Matt Artz on Unsplash

RSA is a *Public Key Cryptography* Algorithm. It has been securing your communications

about how this could be implemented practically.

The RSA algorithm was a ***practical, concrete implementation*** of Diffie-Hellman's idea.

What is Public Key Cryptography?

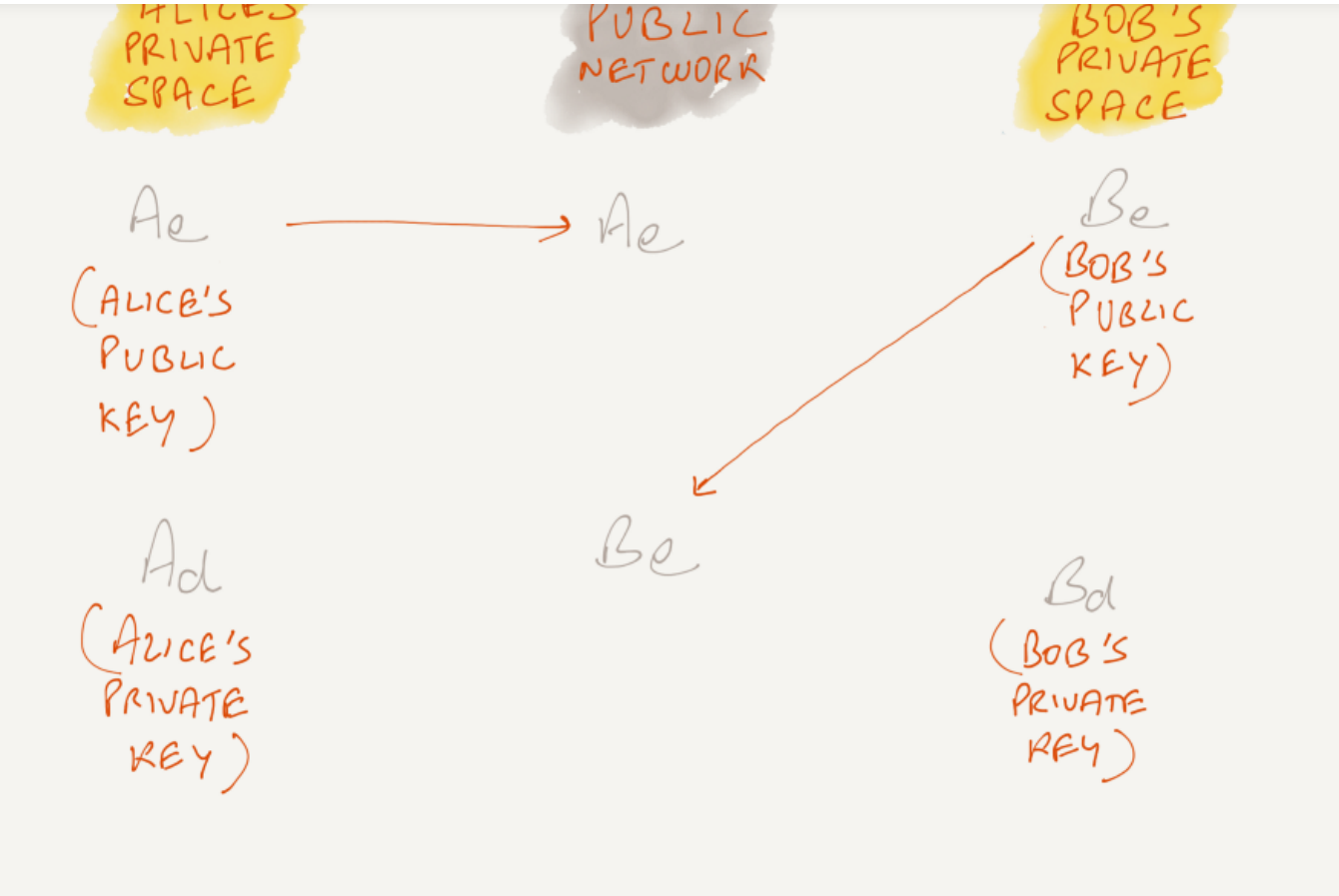
Public Key Cryptography is an Algorithm, in which, the key that is used to encrypt a message is not the same one used to decrypt it.

Each individual on the network, has their own key-set, comprising of — **Public(encryption) and Private(decryption) Key**. Anyone who wants to receive encrypted messages makes their public key available to everyone else on the network. While the private key is kept secret. Only the individual who generated it, knows it.

For example, if Alice and Bob want to communicate securely on a public network using a Public Key Cryptography Algorithm, they would follow the following steps:

1. Alice generates her own public key and a private key — *Ae* and *Ad* respectively
   Bob generates his own public key and a private key — *Be* and *Bd* respectively

2. Alice broadcasts the public key *Ae* to the network.
   Bob broadcasts the public key *Be* to the network.

3. Now, Alice wants to send a message to Bob. Let's assume the message is ***M***. Alice knows Bob's public key *Be*, Since its available on the network publicly.

4. Alice uses Bob's public key to encrypt ***M*** into a *cypher-text* ***C***.

```
C = Be(M)
```

5. Alice send **C** over the public network to Bob. Other people can see **C** being transmitted over the network, but can't deduce **M** from it.

6. Bob gets C. He uses his private key *Bd* to *decipher* **M** from **C**.

```
M = Bd(C)
```

This is what Diffie-Hellman proposed in their paper → A basic flow of a public key

What implementation did RSA Algorithm give?

RSA Algorithm relies on the *Factoring Problem* — it's difficult to find prime factors of a *large integers*.

- Checkout my previous post on prime numbers, if you too found them interesting all of a sudden!

This is how the Algorithm goes —

Part 1 — Calculate the public key

- Choose two large prime numbers **p** and **q.** Don't share these with anyone.
- Calculate the product of p and q. Let's call it **N.**

```
N = p * q
```

- Compute **Φ(N)**. It is called *Euler's Totient Function*. In our case:

```
Φ(N) = (p-1) * (q-1)

We'll talk more about this in the next section.
```

- Choose another prime number ***'e'*** such that:

```
1 < e < Φ(N)

and,

e, Φ(N) are co-primes
```

- Compute **d**(*Private Key)*, such that :

```
d > max(p, q)

and,

de ≡ 1 mod Φ(N) — This is a common way to write expressions in the
modulo world. What this means is, when we divide the product of d
and e with Φ(N), we would get the remainder 1. We'll use this later
in the proof!
```

Part 3 — Sending the message. Securely!

- To send a message **M**, you calculate the *cypher-text* **C** of *message* M, using the public key provided to you by the recipient. C will be sent over the public network. Every one will be able to *sniff* C but wont be able to get M from it.

```
C = M^e mod N
```

Part 4 — *Receiving the message. Securely!*

- The intended recipient(person whose public key was used to encrypt the message) receives **C** and extracts **M** from it, using his private key — **d**.

```
M = C^d mod N
```

The communication was successful, and secure!

This can be a little overwhelming at first. All the new expressions and functions. But after we workout the math in the next section, it'll start to make sense.

> *At this point, the part that amazes me is → How can a value — **M**, reduced by modulo → **C = M^e mod N**, be retrieved again **M = C^d mod N**.*
> *Modular Arithmetic isn't just for reducing values, its also for securing them!*

The "pretty looking" mathematical expressions behind RSA

Concepts used: Modular Arithmetic, Euler's Theorem, Euler's Totient Function

At the heart of the proof for RSA Algorithm, lies **Euler's Theorem:**

```
M^Φ(N) ≡ 1 mod N

Given that:

* M and N are co-primes. 2 numbers are co-primes, when they only
have 1 as their Greatest Common Divisor.

* Φ(N) is Euler's Totient Function - it counts the no of integers
b/w 0 and N which are co prime to N.
```

Euler's Totient Function:

```
1. Φ for non prime number(N) with prime factors p and q = Φ(p) *
Φ(q)

2. Φ for prime number p and q is (p - 1) and (q - 1) respectively.

So,

Φ(N) = (p-1) * (q-1)
```

Lets park the *Euler's Theorem* for a minute, we will come back to it in a minute.

We'll go through the Algorithm steps one by one and understand the math:

- When someone is generating their key-set of Public-Private Key, the first step is to choose 2 prime numbers **p and q**.
- Then you compute **N and Φ(N)**

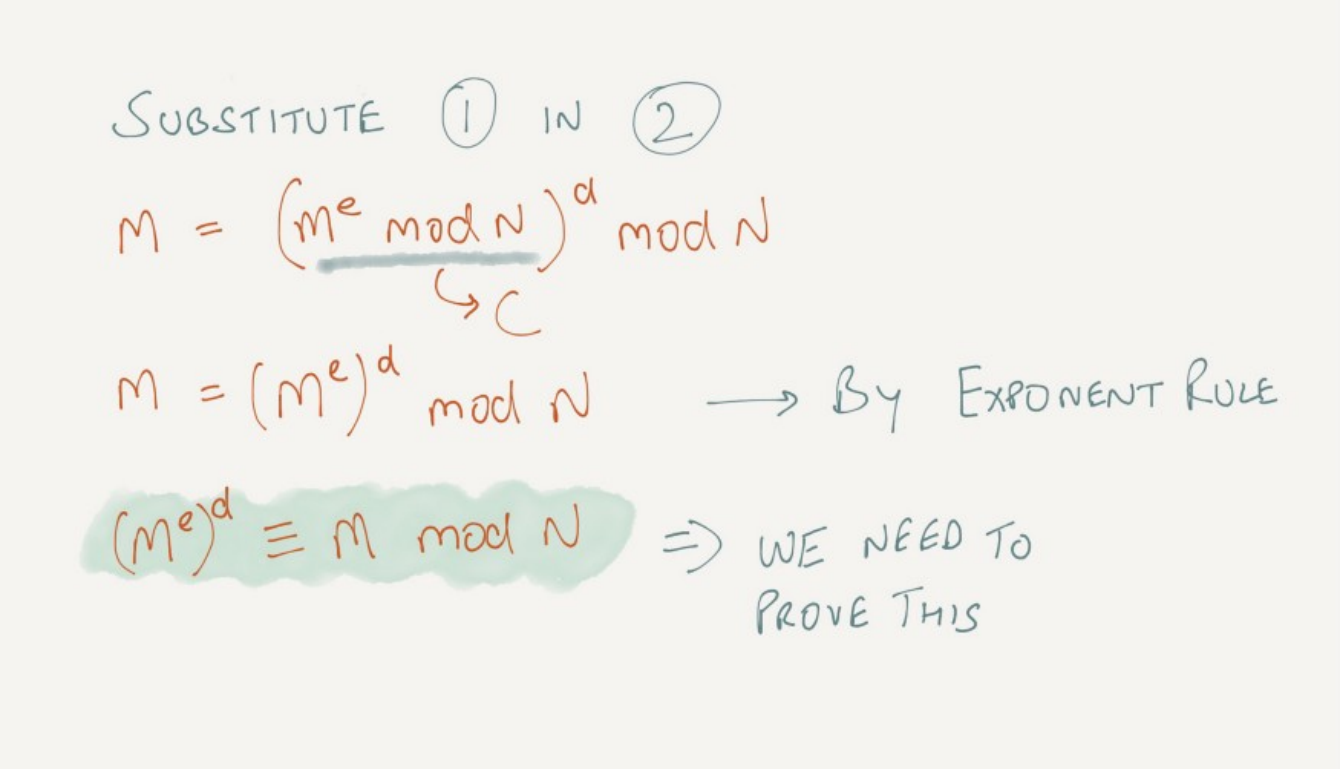
- Now that we have computed N and Φ(N) — we can compute the values **e** and **d**.
- **'e'** needs to be some minimum value to ensure that the mod N is exceeded at-least once when encrypting the message **M —** *M^e mod N*. We can ignore M=0 and M=1 as any power of 0 or 1 is again 1. Next minimum value of M=2. So, if we can ensure that *2^e > N* we can be sure that any message *M≥2* will always give *M^e > N*. Hence, there will be at-least one cycle around N when we do *M^e mod N*. We need this cycle because if after encryption, the cypher-text C is less than N, then we are making it easy for the attacker to guess M.
- Then **'d'** is computed to satisfy the following condition:

```
1. d > max(p,q)

2. de ≡ 1 mod Φ(N)

This implies d and e are Modular Inverse of each other. This
property enables -
This enables:

C = M^e mod N

and then,

M = C^d mod N
```

> *Now, when we say that* ***de ≡ 1 mod Φ(N)****, we are actually saying that for some value k —*
>
> ***1 + k * Φ(N) = de***

For a message M, that is transmitted over the network it actually goes through two '*mod N*' operations, one during encryption, another during decryption.

We are getting started with the *math!*



IS TRUE

$$de \equiv 1 \mod(\phi(N))$$

WE ARE SAYING THAT

$$de = 1 + k * \phi(N) \quad \text{—— (4)}$$

Let's bring back **Euler's Theorem**

$$M^{\phi(N)} \equiv 1 \mod N \rightarrow \text{EULERS THEOREM}$$

$$M^{\phi(N)*k} \equiv 1^k \mod N$$

$$M^{\phi(N)*k} * M \equiv 1^k * M \mod N$$

$$M^{(\phi(N)*k+1)} \equiv M \mod N$$

SUBSTITUTE de FROM (4)

$$M^{de} \equiv M \mod N$$

HENCE PROVED

We just proved the legitimacy of RSA!

with, but you knew what you were getting into.

You are encouraged to explore more about Modular Arithmetic, Euler's Theorem — as per your ***curiosity appetite***.

The Missing Piece and an Example

1. We know that we have to choose p and q.
2. We know how to calculate N and Φ(N).
3. We know how to choose an 'e'

The only thing left is — *How to calculate 'd'?*

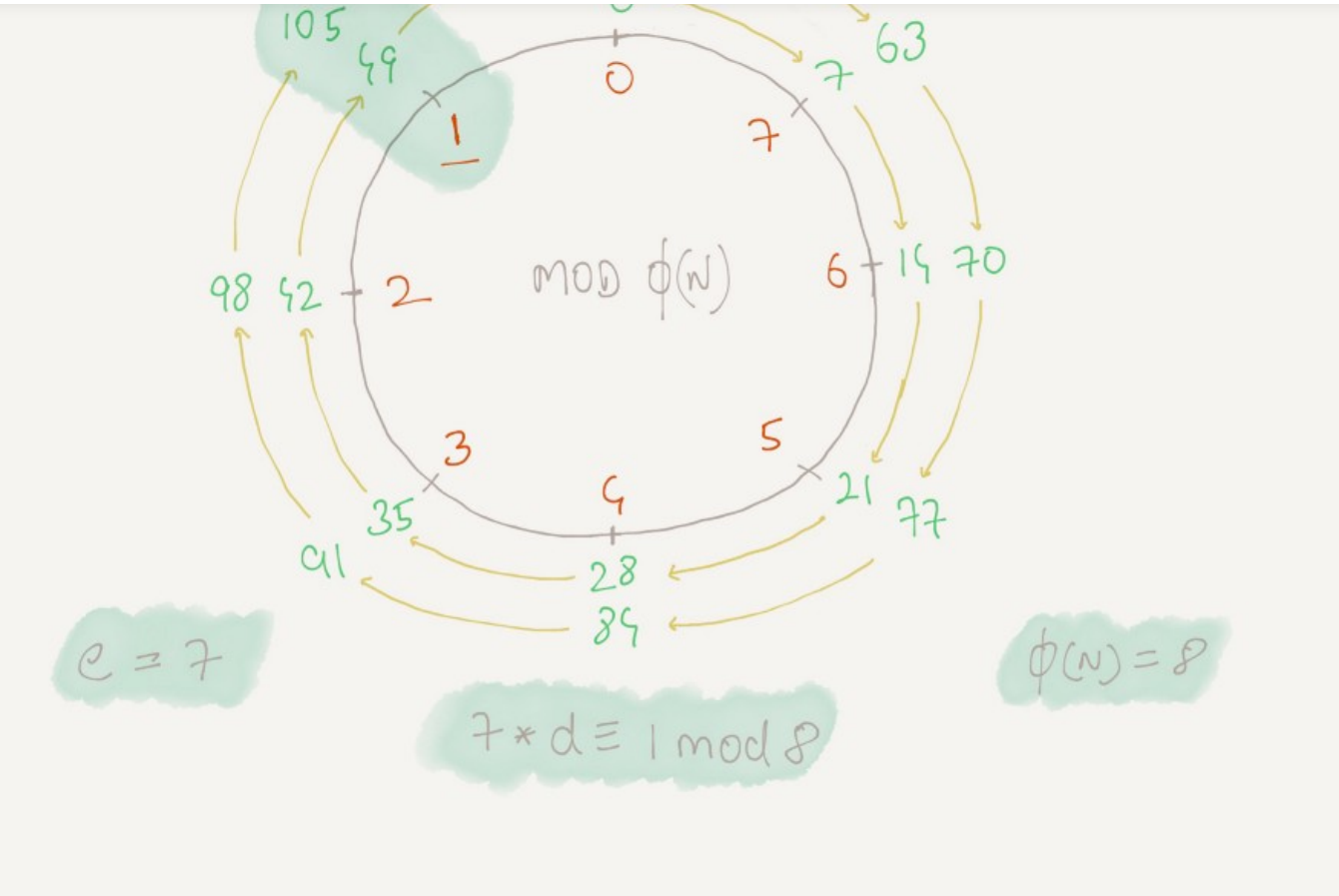
After **e** is chosen, we need to find the **Multiplicative Inverse** of 'e' for Φ(N).

```
de ≡ 1 mod Φ(N)

d ≡ e^(-1) mod Φ(N)
```

Lets consider an example:

```
p = 3
q = 5

N = p * q = 3 * 5 = 15


Φ(N) = (p - 1) * (q - 1) = 2 * 4 = 8

e = 7          1 < e < Φ(N)
```

Now, let's see how to compute **'d'** with a visual method which will also explain a bit of Modular Arithmetic:

Following the above method, we can see that *d*e* could be —

```
49    =   7    *   7

105   =   15   *   7

.
.
.

de    =   d    *   e
```

and if we follow the trail —

```
49    =   7    *   7

105   =   15   *   7
```

.
.

de = d * e

So **d** can be chosen from any one of — *7, 15, 23, 31*…. Since, all of them are greater than *max(p=3, q=5)*.

> *When the prime numbers p and q are chosen for real world implementation, they are quite large(more than 200 digits), an easier way to calculate* ***d*** *in those scenarios is by leveraging the equation* ***de = k*Φ(N) + 1****. We know Φ(N), e and k is chosen so that value of d > max(p,q)*

Interesting Facts about RSA

- RSA's security relies on Factoring Problem
- 'e' and 'd' are modular inverses of each other
- RSA is also used to sign the message by the sender so that the receiver knows the identity of the sender and the legitimacy of the message. In, our example of Alice sending message **M** to Bob, the signed cypher-text C would be computed by *Ad*(*Be*(M)). Again, Bob would decipher C by *Bd*(*Ae*(C)).
- RSA is a *Trap Door One-Way Function* — as it is difficult to compute 'd' from *'e' and N* because its difficult to break N into p and q

Conclusion

Next time you have successfully visited a website without being attacked in the internet, you can thank *RSA, Euler, Modular Arithmetic, Prime Numbers*.

But only until the Quantum Computers rise up!

Diffi-Hellman 1976

RSA 1977

Euler's Theorem

Euler's Totient Function

Exponent Rule-Modular Arithmetic

Shor's Algorithm for Quantum Computers