RSA Algorithm for Cryptography

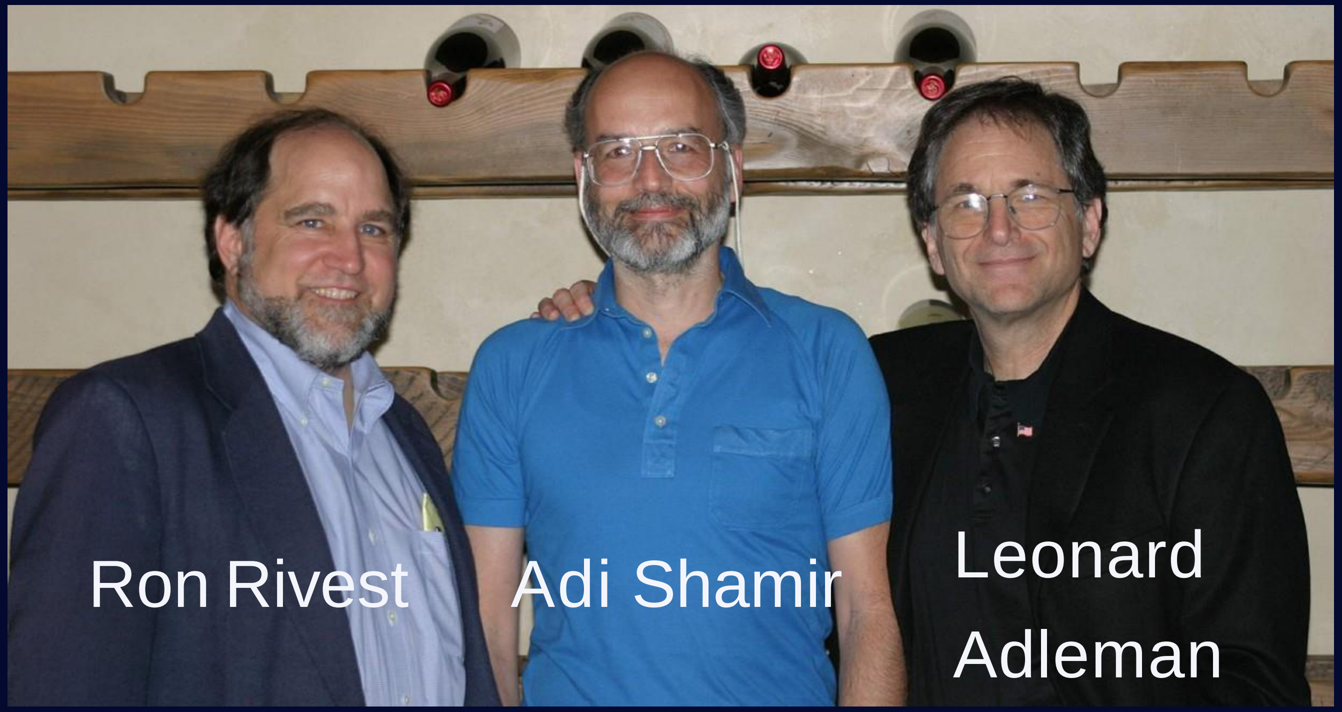
Introduction To Discrete Mathematics
Lecturer: Dr. Veng Sotheara
Class: Data Science & Engineering(DSE)

Table of contents

Invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977

RSA is a popular algorithm used to securely transmit data over the internet. It works by using a public key to encrypt messages, which can only be decrypted using a private key

Inventors



Key Generation

01

Key Generation

02

Public Key

03

Private Key

Key Generation

- Choose two large prime numbers (p) and (q).
- Calculate their product (n = p*q). And the value of (n) is part of the public key.
- Calculate the Euler's totient function of (n), ϕ(n) or T = (p-1) * (q-1).
- Choose an integer (e), such that 1< e < T and [gcd(e, T)=1]. The value of (e) is part of a public key.
- Calculate the modular multiplicative inverse of (e) modulo T, which is another integer (d). The value of d is a private key.

Public Key (n,e)

- n: The modulus, which is the product of the two large prime numbers.
- e: The public exponent, used for encryption.

Private Key (n,d)

- n: The modulus, The same as in the public key.
- d: The private exponent, used for decryption.

Encryption

To encrypt a message (M) represented as an integer into a cipher text: (C)
$C = M^e \pmod{n}$

01

Choose a Message to Send

The message can be any text or data that you want to transmit securely

02

Convert Message to Number

Each character in the message is converted to a number, based on its ASCII value

03

Apply Public Key

Using the public key, each number in the message is raised to the power of the public key e, and then taken modulo the public key n

Decryption

To decrypt the cipher-text (C) back to the original message (M)
$M = C^d \pmod{n}$

01

Apply Private Key

Using the private key, each number in the encrypted message is raised to the power of the private key, and then taken modulo the public key n

02

Convert Number to Message

The resulting numbers are converted back to characters, based on their ASCII values, to reveal the original message

Example of Key Generation

✓ Key generation:

- Choose two prime numbers (p)= 2 and (q)=7
- Calculate (n: p*q) = 2 *7 = 14
- Calculate T = (p-1) * (q-1) = 1*6= 6
- Choose {(e, d) mod 6 =1} [e: encryption, d: decryption]; [Note: e and T should be relatively prime, and d is the inverse of e in mod T]
- Choose e =5, d=11

So, the public key is (14, 5) And The private key is (14, 11)

How to encrypt a message

- The published key is (14, 5)

So, let’s send a one-letter secret message “B”, Instead of (B) Let’s use the number 2.

Suppose: A=1, B=2, C=3 and etc.

We get Encryption value = 2^5 mod 14 is 4.

Therefore, the encrypted value is 4. The encrypted message of 4 will be translated to letter ‘D’ if we were to translate directly.

How to decrypt a message

- The private key is (14, 11)

So, we know that the secret value is 4

Supposed: A=1, B=2, C=3…. Etc.

We get the decrypted value = 4^{11} mod 14

→ 4^{11} = 4,194,304 mod 14

→ 4,194,304/14 We get remainder 2

Therefore: 2 is a decrypted message which is letter "B"

```python
import random

#function to the gcd of 2 numbers
def gcd(a, b):
    remainder = 0
    while(1):
        remainder = a % b
        if (remainder == 0):
            return b
        a = b
        b = remainder

#function to do extended ecuclidean for gcd, x and y which are coefficients for linear combination
def extended_euclidean(a, b):
    if a == 0:
        return b, 0, 1
    else:
        gcd, x1, y1 = extended_euclidean(b % a, a)
        x = y1 - (b // a) * x1
        y = x1
        return gcd, x, y

#function to find the modular inverse
def mod_inverse(e, t):
    gcd, x, y = extended_euclidean(e, t)
    if gcd != 1:
        raise ValueError("Modular inverse does not exist since 'e' and 't' are not coprime.")
    return x % t
```

```python
if __name__ == '__main__':
    #intialize values for prime numbers p, q, calculate n and t and set initial value for e
    p = 124070563
    q = 334003807
    n = p * q
    e = 2
    t = (p - 1) * (q - 1)

    #find e which is co-prime with t
    while(1):
        if gcd(e, t) == 1:
            break
        else:
            e = random.randint(2, t-1)

    #calculate d as the modular inverse of e modulo t
    d = mod_inverse(e, t)

    #message to be encrypted
    message = 12
    print("")
    print("Original message data = ", message, end="\n\n")

    #encrypt the message
    ciphertext = pow(message, e, n)
    print("+ Encryption:")
    print("Encrypted message data (ciphertext) = ", ciphertext)
    print(f"Using: n = {n}, e = {e}", end = "\n\n")

    #decrypt the message
    decrypted_message = pow(ciphertext, d, n)
    print("+ Decryption:")
    print("Decrypted message data (orginal) = ", decrypted_message)
    print(f"Using: n = {n}, d = {d}", end = "\n\n")
```

Output

```
Original message data =  12

+ Encryption:
Encrypted message data (ciphertext) =  14352020436792370
Using: n = 41440040378633341, e = 4811942912821487

+ Decryption:
Decrypted message data (orginal) =  12
Using: n = 41440040378633341, d = 24038365170580919
```

Thank You for Paying Attention!