



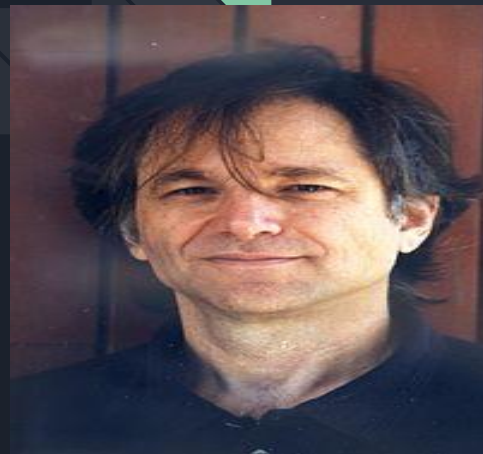
Algoritmo de RSA

Matemática Discreta

Equipe: Ascânio Sávio, Jorge Firmo, João Ayalla, Gabriel Souza, Danilo Vasconcelos

Histórico

O acrônimo RSA deriva das iniciais do sobrenome de seus criadores (Rivest, Shamir e Adleman) os quais descreveram o algoritmo em 1978, sendo revelado em 1997.





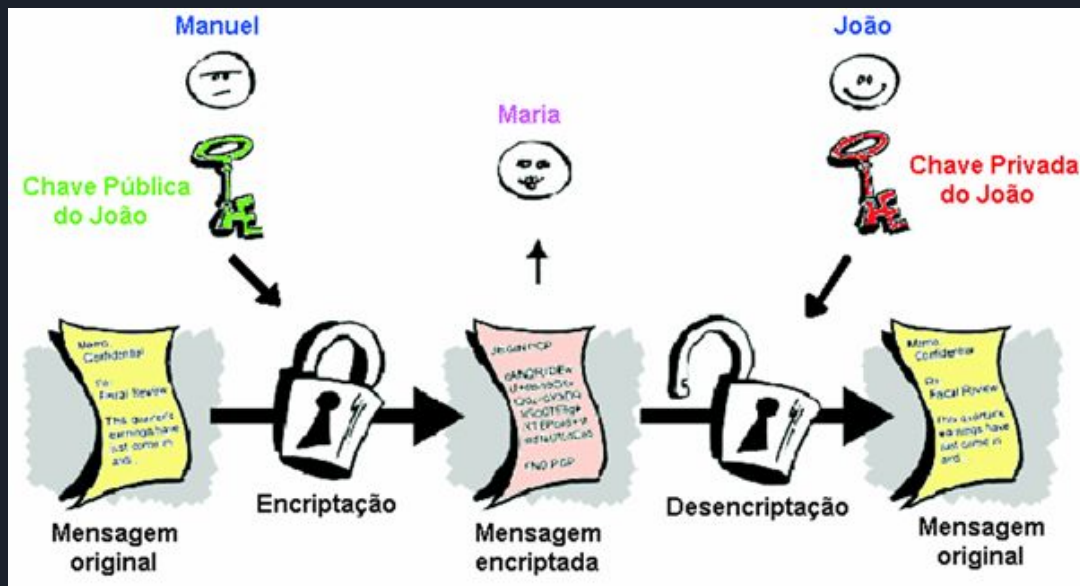
Funcionamento

O algoritmo é usado para criptografar mensagens utilizando conceitos de **aritmética modular**. A partir de uma chave (**chave pública**) gerada com dois números **primos** (**p** e **q**), é possível **criptografar uma mensagem**.

Para decodificar essa mensagem é necessário achar os dois números primos “p” e “q” utilizados na criação da **chave pública**.

A depender da chave pública, o processo de **decodificação**, para quem não contém os números “p” e “q”, se torna lento à medida que a chave contém números maiores.

Funcionamento



Fonte: UFScar



Etapas do algoritmo

1 - Criação da chave pública

2 - Criptografia

3 - Descriptografia

Criação da chave pública


Nesse processo, recebe-se (ou gera-se) do usuário dois números primos “p” e “q”, e um número “E”, tal que “E” seja co-primo do produto: $(p-1).(q-1)$ conhecido como função totiente de N, sendo $N = p . q$

A partir disso, é gerado um arquivo “PublicKey.txt”, o qual contém o “N” e “E”.

```
n = p * q
FiN = int((p - 1) * (q - 1))
if not check_prime(p):
    print('p não é um numero primo')
elif not check_prime(q):
    print('q não é um numero primo')
elif n <= 26:
    print('O produto p.q deve ser maior que 26')
elif gcd(e, FiN) != 1 :
    print('não é co-primo com o produto (p-1).(q-1)')
else :
    create_public_key_file(n, e)
    print('Chave publica gerada com sucesso!')
```


```
def create_public_key_file(n, e):
    try:
        file = open("PublicKey.txt", "w")
        file.write(str(n))
        file.write(' ')
        file.write(str(e))
        file.close()
    except FileNotFoundError:
        print("Houve um erro ao criar o arquivo, tente novamente")
```

Criptografia


$$m^e \equiv c \pmod n$$
$$c = \text{cod}(m) = m^e \pmod n$$

```
def crypt(read_file, msg, e, n):  
    end = len(msg)  
    cryptMsg = ""  
    for i in range(end):  
        m = msg[i]  
        cryptMsg += str(pow((alfa.index(m) + 2), e, n))  
        if(i + 1 < end):  
            cryptMsg += ' '
```

Descriptografia


$$c^d \equiv m \pmod n$$
$$m = \text{decod}(c) = c^d \pmod n$$

```
def decrypt(cryptMsg, d, n):  
    decryptMsg = ""  
    i = 0  
    end = len(cryptMsg)  
    while i < end:  
        current = ""  
        while i < end and cryptMsg[i] != ' ':  
            current += cryptMsg[i]  
            i += 1  
        i += 1  
        current = int(current)  
        decryptMsg += alfa[pow(current, d, n) - 2]
```




Referências

[Geeks for Geeks: RSA Algorithm in Cryptography](#)
[Khan Academy: RSA encryption](#)