

# Criptografia RSA

## Geração das Chaves

O primeiro passo para a criptografia RSA é a geração das chaves, uma pública e uma privada, que é feito pela parte que irá descifrar as mensagens. Após a geração das chaves, a chave pública é enviada para a outra parte, que faz a cifragem do texto claro, e envia o texto cifrado para a parte que criou as chaves.

Como explicado nas aulas, a chave pública consiste de E e N, a chave privada de D e N, onde N é a multiplicação de P e Q, e P e Q são números primos escolhidos aleatoriamente.

$N = P * Q$

$\phi = (p - 1)(q - 1)$

$E = \text{calculateE}(\phi)$

$D = \text{calculateD}(e, \phi)$

E é um número que é primo de  $\phi$

```
while True:
    e = random.randrange(1, phi) # Número aleatório entre 1 e phi
    g = gcd(e, phi) # Calcular o máximo divisor comum entre E e PHI
    if g == 1: # Se o máximo divisor comum entre E e PHI for 1, quer dizer que eles são primos
        return e
```

Já D pode ser calculado pelo algoritmo euclidiano estendido. Teoricamente, D tem que ser tal que:  $E * D = 1 \bmod \phi(n)$  e  $d < \phi(n)$ .

O algoritmo euclidiano estendido é uma extensão do algoritmo de Euclides que, além de calcular o máximo divisor comum, calcula também os coeficientes x e y tal que:

$ax + by = \text{gcd}(a, b)$

Usando tal algoritmo, é possível calcular D, onde D é x + phi se x < 0, senão, D é igual a x.

```
def calculateD(e, phi):
    g, x, y = xgcd(e, phi)
    if x < 0:
        return x + phi
    else:
        return x
```

## Encrytação

Como exposto na aula, a cifragem do texto claro é feita da seguinte maneira:

Para cada letra do texto claro, primeiro se converte a letra para ASCII, onde ela vira um número:

```
for char in plaintext:
    k = ord(char)
```

esse número é elevado a chave pública:

```
k = k ** int(key)
```

e o resultado é o módulo desse cálculo por n:

```
result = k % int(n)
```

e o resultado é concatenado a string final do texto cifrado, tendo o cuidado de inserir um separador entre cada resultado, pois é necessário essa separação para a decifragem. No caso do programa atual, cada resultado está em uma linha diferente.

## Decriptação

A decifragem é parecida com a cifragem, cada resultado é elevado a chave privada:

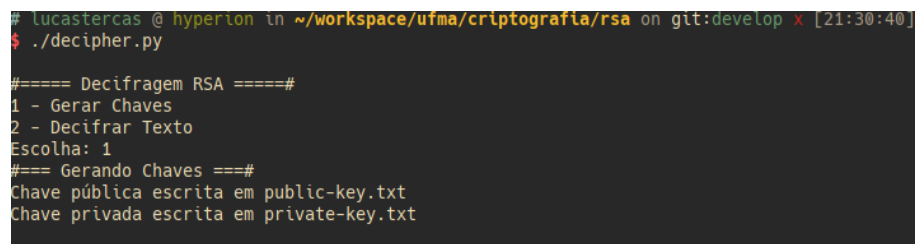
```
for line in ciphertext:
    result = int(line) ** int(key)
```

e o resultado é o resto do cálculo anterior por N, convertido para char:

```
result = result % n
result = chr(result)
```

## Execução do Programa

1. Crie uma chave pública e privada usando o programa decipher.py



```
# lucastercas @ hyperton in ~/workspace/ufma/criptografia/rsa on git:develop x [21:30:40]
$ ./decipher.py

##### Decifragem RSA #####
1 - Gerar Chaves
2 - Decifrar Texto
Escolha: 1
=== Gerando Chaves ===
Chave pública escrita em public-key.txt
Chave privada escrita em private-key.txt
```

Figure 1: Exemplo de Geração de Chave

2. A chave privada será escrita para private-key.txt, e a chave pública será escrita para public-key.txt
3. Usando o programa cipher.py, cifre um texto usando a chave pública
4. O texto cifrado será escrito para textos/texto-cifrado.txt

```
# lucastercas @ hyperion in ~/workspace/ufma/criptografia/rsa on git:develop x [21:34:36] C:130
$ ./cipher.py
#==== Cifragem RSA ====#
Caminho do texto claro: ./textos/texto-claro.txt
Texto cifrado escrito para ./textos/texto-cifrado.txt
```

Figure 2: Exemplo de Cifragem

5. Usando o programa decipher.py novamente, decifre o texto cifrado do passo 4 usando a chave privada.

```
# lucastercas @ hyperion in ~/workspace/ufma/criptografia/rsa on git:develop x [21:33:00]
$ ./decipher.py
#==== Decifragem RSA ====#
1 - Gerar Chaves
2 - Decifrar Texto
Escolha: 2
Caminho do texto cifrado: ./textos/texto-cifrado.txt
```

Figure 3: Exemplo de Decifragem