

# Trabalho de Implementação 3

## Segurança Computacional - Turma B

Leonardo Rodrigues de Souza - 17/0060543  
Lucas Dalle Rocha - 17/0016641

Universidade de Brasília - UnB {170060543,170016641}@aluno.unb.br

### 1 Descrição do sistema criptográfico RSA

O terceiro trabalho da disciplina de Segurança Computacional, da Universidade de Brasília, consistiu na implementação do sistema criptográfico RSA [1], isto é, um gerador e verificador de assinaturas RSA. O algoritmo utilizado pelo RSA é de chave assimétrica, ou seja, existe a manipulação de duas chaves diferentes: pública, que é utilizada para a cifragem de mensagens; e privada, que é utilizada para decifragem de mensagens e é mantida em segredo. Dessa forma, como as chaves são de, no mínimo, 1024 *bits*, faz-se também a utilização de um esquema de preenchimento de *bytes* que, em nosso caso, é o OAEP [2].

### 2 Geração de chaves

Por utilizar um algoritmo de chave assimétrica, as chaves devem ser diferentes e, desse modo, cada chave é composta por um par de números, de modo que a chave pública equivale a  $(n, e)$  e a chave privada equivale a  $(n, d)$ . Para a geração desse par de números, há uma sequência de passos a ser seguida:

1. Geração de dois números primos aleatórios  $p$  e  $q$ , cada qual com o tamanho mínimo de 1024 *bits*. Vale lembrar que nessa etapa de geração, foram utilizadas algumas heurísticas, como a divisão do número gerado pelos primeiros 100 primos, a fim de que se reduza o tempo de processamento das demais etapas e, além disso, o teste de primalidade Miller-Rabin [4] com 40 rodadas.
2. Computação de  $n$ , que equivale a  $p \times q$ .
3. Computação da função totiente de Euler [3]  $\phi(n)$ , que foi feita pela multiplicação de  $p - 1$  e  $q - 1$ , dividido pelo máximo divisor comum dos mesmos (isso nos retorna o mínimo múltiplo comum).
4. Geração de um número aleatório  $e$ , de modo que  $1 < e < \phi(n)$  e, ademais,  $e$  deve ser coprimo de  $\phi(n)$ .
5. Computação de  $d$ , que equivale ao inverso multiplicativo modular de  $e \bmod \phi(n)$ , isto é,  $d$  equivale ao número que, multiplicado pelo nosso fator, é igual a 1.
6. Assim, a chave pública é composta pelo par  $(n, e)$  e, a chave privada, por sua vez, é composta pelo par  $(n, d)$ .

### 3 Cifração e Decifração RSA

#### 3.1 Cifração RSA

O processo de cifração utiliza o esquema de chave pública e, para a cifragem  $c(m)$  de uma mensagem  $m$ , segue a operação:  $c(m) = m^e \bmod n$ . Dessa forma, em nossa implementação, inicialmente a mensagem passa pelo esquema de preenchimento OAEP, de modo a converter o esquema para probabilístico, além da adoção de *bytes* de preenchimento (zeros) para a mensagem. Ademais, a função *hash* adotada foi pela aplicação do SHA-3 [5] em blocos de *bytes* da mensagem. Por fim, realiza-se a potenciação dada por  $c(m)$  e é obtido o bloco cifrado, através da concatenação das duas entradas do OAEP, com a garantia de segurança probabilística, isto é, não há vazamento de informação alguma quanto ao texto cifrado.

#### 3.2 Decifração RSA

Já o processo de decifração utiliza o esquema de chave privada e, para a decifragem  $m(c)$ , dado o criptograma  $c$ , segue a operação:  $m(c) = c^d \bmod n$ . Assim, é inteligível notar que o processo é realizado através da primitiva reversa do processo de cifração. Portanto, em nossa implementação, inicialmente o criptograma realiza a potenciação  $m(c)$ , e em seguida aplica-se o processo inverso do OAEP, ou seja, ocorre a divisão de  $m(c)$  em dois blocos e, após aplicado o processo inverso, obtemos a mensagem original adicionada dos *bytes* de preenchimento. Dessa forma, basta removê-los e temos a mensagem original.

```
Insert your message to encrypt: teste
Hashed ciphertext <BASE64>: b'OKX8g560JqMqY3aLyOrNDjwuAhr2fnj03QE/dAk6fGwWGrv/+8Gg9GhGj3ozKueIgUomztLksqNMZFMCu3mdtwVTX
Vf/5+LWNDLguv11UnjbBULb8ikFhTE1cMI7X3wpMh5rkhfgo+oSBzvD7GP872rQjUGCbv7a+c0x02Pcy1QvdVfCnYdfC0LF/+Y0SLVYN19BxyXec5YXdXo/
92ujW0RBzKVH+hi++m0B1bL4wXL1ct7CqvLKW13nu9NakPqoHC1dig7eRV008cKDye72GxANgu0Rj/4yTsDVRt2NkPhCPZ4mShPQ1w4okR1nTm4Ovwy7kUG
xFuAXewl8HgRYuQ=='
Plaintext after decrypting ciphertext: teste
```

Figura 1. Processo de cifração e decifração de uma mesma mensagem.

### 4 Assinatura e Verificação

#### 4.1 Assinatura

Para o processo de assinatura da mensagem, ocorre a cifração do nosso *hash* com a utilização da chave privada, isto é, suponha  $h = sha3\_256(m)$ , uma vez que foi o algoritmo utilizado para realizar os processos de cifragem e decifragem. Assim, através da chave privada, temos  $sign(h) = h^d \bmod n$ , que deve ser verificado na próxima etapa e comparado após o processo de decifração do *hash* cifrado.

## 4.2 Verificação

Para o processo de verificação da assinatura, utiliza-se a chave pública, e o processo aplicado é o inverso, ou seja,  $h = \text{sign}(h)^e \bmod n$ . Dessa forma, verifica-se se o texto puro inicial, após o *hash*, é igual ao *h* obtido pelo processo inverso. Em caso positivo, tem-se a garantia de que a mensagem foi realmente enviada pelo usuário desejado e, ademais, não sofreu alterações em seu processo.

```
Hashed signature (BASE64): b'DZ/WWwcp10ih10ffUoncI4S+YhQp8yXFnYiKgky+LP0SsHyKA7Qz+70FzhTOIf+Itc4WhejYbwI8HbInVeb4/TK
os1axaZAGwzQEFYavU1rffxYhj2SAb1b7IqPIiHAmnQwhNnLXcf937oyiKX5vbV153dOVcyyezgU2h4qECBt1Dm76GtKI2WOQNCK17EhWA583pcYTHxM0wai
hnlTyy1ZosVwicz6VowmzGgpGsUwJsjjddYat1xULcMwFT1AfouBr4wo/VffU/2W+jlnkY8v8qfaeGQqoHaYFUw24RE2Cc+PYjs6Z5Xm9P5d7OZ1xBbTSq
yqXvqAL7hd3oMg==
Verified signature = True
```

**Figura 2.** Processo de geração de um *hash* para cifração e posterior verificação se o retorno da decifração condiz com o texto puro.

## 4.3 Formatação BASE64

Por fim, as chaves pública e privada, além da assinatura são formatadas em BASE64, para impressão em arquivo. Desse modo, cada *byte* é convertido para o seu valor ASCII correspondente, para cada uma das estruturas e, como a representação em BASE64 tem 6 *bits* (0 a  $2^6-1$ ), realiza-se a divisão do bloco completo de 6 em 6 *bits*, e a representação equivale, em ordem, aos caracteres [A-Z], [a-z], [0-9], "/" e "+".

## 5 Instruções para compilação e execução

Para a implementação, foi utilizado o sistema operacional Ubuntu 20.04.2 LTS, além da linguagem de programação *Python* v. 3.9.0. O programa pode ser executado, em seu diretório raiz, pela utilização do seguinte comando em seu terminal:

```
$ python3 main.py
```

Em seguida, você deve inserir apenas a mensagem a ser cifrada, uma vez que o par de chaves será gerado de maneira automática. Dessa forma, ocorre a cifração e decifração da mensagem, a fim de demonstrar o retorno da mensagem original, e a assinatura e verificação logo em seguida. Por fim, é gerado um arquivo nomeado *out.md*, que contém informações a respeito das chaves pública e privada, além da assinatura, em BASE64.

```
dalle@pc:/mnt/c/Users/cliente/Desktop/RS4_cryptosystem$ python3 main.py
Insert your message to encrypt: Hello world
Hashed ciphertext :BASE64=: b'h5fKw9pJHfWkp0xLoU+U2rAv6E53s4sVd6s/vhVkl70dqhaevaJcg89vdbwmzIDVp/XSHM2G55yAMdG6pp0Wtt
rFw83J2v1ajlme2ZYHYUleOMX05fYUudGmDVHskCVPtStsqctG3f9n+GLTKTrpZ4+D8c1cK+d4+8pN0xme1XheFuLV5ctnRC25AygrJknAK4512Uj
xk2c28w8sn13n93bPecyQKEX12HCY012wLhuKl01dzfz2316vmGqXK8rLF0wHk2L1GALAG804ctFV7Hb01wK+uY670eJueJnAFL41SPH0ZG7v
L9e627Waa8+kQAQ==
Please enter decrypting ciphertext: Hello world
Hashed ciphertext :BASE64=: b'HK0phktjQ0j/S19PvCb3C+YnddWmGmKtKBfFYH01CdnrkM3Joau+Xe+r4rEe48n4B+S/Lr/6YG3412rE8NFVQgTxm9Py
aVdWd881i0/1Plxuvf68pxgQcBB27wXmKb188ySe6V/C3ettd018197FgexPx1B3y7f/aJyUJm9u+5YhNk9f/Tr/7/41d412mCmmk0s45WdK
TxRAU0Qf05jaBTzm/WmBvirg8bh4W2R1wYS215u7BKA4Y3NtrhiwsA21dAPdp0KVE61+4oaomCz21PXP+PzKMFYape5HhucQh04f9314h60ar19M4Fd
iozj3p2/Sr8mGrQ==
Verified signature = True
Generating file 'out.md' containing public key, private key and signature in BASE64.
```

[illegible]

## Referências

1. Wikipedia contributors, Wikipedia, The Free Encyclopedia., RSA (cryptosystem), [https://en.wikipedia.org/wiki/RSA\\_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem)), Last accessed 26 Oct 2021
2. Wikipedia contributors, Wikipedia, The Free Encyclopedia., Optimal asymmetric encryption padding, [https://en.wikipedia.org/wiki/Optimal\\_asymmetric\\_encryption\\_padding](https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding), Last accessed 26 Oct 2021
3. Wikipedia contributors, Wikipedia, The Free Encyclopedia., Euler's totient function, [https://en.wikipedia.org/wiki/Euler%27s\\_totient\\_function](https://en.wikipedia.org/wiki/Euler%27s_totient_function), Last accessed 26 Oct 2021
4. Wikipedia contributors, Wikipedia, The Free Encyclopedia., Miller–Rabin primality test, [https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin\\_primality\\_test](https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test), Last accessed 26 Oct 2021
5. Python, hashlib - Secure hashes and message digests, <https://docs.python.org/3/library/hashlib.html>, Last accessed 26 Oct 2021
6. Wikipedia contributors, Wikipedia, The Free Encyclopedia., Digital signature, [https://en.wikipedia.org/wiki/Digital\\_signature](https://en.wikipedia.org/wiki/Digital_signature), Last accessed 27 Oct 2021