

## Trabalho 2 – Gerador e Verificador de Assinaturas RSA com OAEP

**Autores:**

**Pedro Brum Tristão de Castro - 202067470**

**Mateus Oliveira Santos - 221029150**

**Link para o repositório:**

<https://github.com/mateusap1/rsa>

### 1. Introdução

Este trabalho tem como objetivo implementar um sistema de assinatura digital baseado no algoritmo RSA, com uso de OAEP como esquema de preenchimento (padding), além de um algoritmo de assinatura digital. A assinatura digital garante a integridade e autenticidade de arquivos, sendo uma das aplicações mais importantes da criptografia assimétrica. Além disso, utiliza-se a função de hash SHA-3 para obter o resumo da mensagem a ser assinada.

### 2. Fundamentos Teóricos

#### 2.1 Testes de Miller-Rabin

O algoritmo de Miller-Rabin é um teste de primalidade probabilístico eficiente, usado para verificar se um número é primo. Ele reescreve  $n-1$  como  $2^s \cdot d$ , escolhe bases aleatórias  $a$ , e realiza testes modulares. Se um teste falha, o número é composto; se passa em várias rodadas, é "provavelmente primo" com alta confiança. Miller-Rabin nunca erra ao identificar um número composto.

#### 2.2 RSA e Assinaturas Digitais

O algoritmo RSA é um sistema de criptografia de chave pública que utiliza a dificuldade de fatorar números primos grandes para garantir a segurança. Ele emprega aritmética modular para gerar mensagens criptografadas e permite verificar a autoria da criptografia, sendo amplamente utilizado para assinaturas digitais. O RSA funciona através da geração de duas chaves a partir de dois números primos: a chave pública  $(n, e)$  e a chave privada  $(n, d)$ .

A criptografia de uma mensagem  $(m)$  é feita com a chave pública, resultando em  $c = m^e \bmod n$ . A descryptografia da mensagem cifrada  $(c)$  é realizada com a chave privada, através de  $m = c^d \bmod n$ . Para assinar um arquivo, um resumo dele (hash) é criado e criptografado usando a chave privada.

## 2.3 OAEP (Optimal Asymmetric Encryption Padding)

O OAEP (Optimal Asymmetric Encryption Padding) é um esquema de preenchimento que aumenta a segurança do RSA, protegendo contra ataques que exploram a estrutura do texto original e assegurando que entradas similares gerem saídas distintas. Seu funcionamento envolve três passos principais:

1. **Preparação da Mensagem:** A mensagem original recebe um preenchimento e é combinada com uma máscara criada por uma função de geração de máscara (MGF) usando uma "semente" (seed) aleatória. Isso forma uma mensagem embaralhada (ME).
2. **Aplicação de Máscaras:** A semente é então mascarada utilizando a MGF e a própria mensagem embaralhada (ME) como semente. Isso garante que pequenas modificações na mensagem ou na semente produzam resultados completamente diferentes.
3. **Codificação Final:** O resultado é concatenado e formatado para ser cifrado pelo algoritmo RSA.

A principal vantagem do OAEP é transformar o RSA de um algoritmo determinístico em um probabilístico, dificultando ataques por análise de padrões. Além disso, ele incorpora verificações internas que detectam alterações ou corrupções nos dados antes da decifração.

## 3. Estrutura da Implementação

### 3.1 Geração de Chaves

- Escolha de dois primos grandes ( $p$  e  $q$ ), com pelo menos 1024 bits.
- Teste de primalidade com Miller-Rabin.
- Cálculo de  $n = p * q$ ,  $z = (p-1)(q-1)$ .
- Escolha do expoente público  $e$  e cálculo do inverso modular  $d$ .

### 3.2 Assinatura

- Geração do hash SHA-3 da mensagem original.
- Cifração do hash com a chave privada usando RSA (**sign**).
- Concatenação **Hash\_algorithm||n||e||sign**.
- Codificação do resultado em BASE64 para armazenamento e transmissão.

### 3.3 Verificação

- Parsing da mensagem assinada e extração da assinatura BASE64.
- Decodificação e decifração com a chave pública.

- Recomputação do hash da mensagem original e comparação com o hash decifrado.

## 4. Testes e Resultados

### 4.1 RSA e OAEP

A implementação do RSA foi feita, em primeiro lugar, com números inteiros. Como exemplo, utilizando o número 42, a forma mais básica do código RSA ficou:

```
(3.11.8) mateusoliveirasantos@192 rsa % python3 rsa.py
p gerado: 1533764118734417322801531405071762397368948221220879875024271193427675632254102727392875745283557527499590631555
23124824513716679975783338937824209870248237815563879728474094387923284709634573092472929487044819335062155458932749193172
5005935785783110727213151736426158074964222451715034499881913475779650754229
q gerado: 1715887446482190613637483495818843333887624250488021882077109477767317880815261783565071760443721418005073714621
25604594244977207336049701568204476143798724701752796326490637813548605451425157911160210106095674729908013428543731232674
843504528598333961437744301240710796503614397619047278723354692507260660171
Valor criptografado: 13583636407793232451735154789268507836869584196957554947484817423996348234246871794424231527260913295
13350371667943439764202855732368256908495965286450292719095063803872923759390414463795886698542212968723465729854910797514
7700399406382599831557891146710135809480849181580878421925760527215213353368295932987136253362589245342619609297494646072
22341086110516444398759026462321443671140918962270003563902261035847998935639437567442554455971333676609919949540187211383
4804949770701849078958867692408313618816397651294057886860901255231216099940605913473831440931893820812218082496645047645
4007962577589655170716068941
Valor descriptografado: 42
```

Depois disso, foi implementado também o OAEP com funções para codificar e decodificar:

```
(3.11.8) mateusoliveirasantos@192 rsa % python3 rsa.py
Encoding b"hello"
OAEP Encoded: 005a10f17f8eb5c4f9c0aaebfbfb29d6005886d63505988412e6e94d189f5f78f2b15becd91eb788082ed097a352214c1653f0f7907c
65812cc50b43772efb995ed481a70952028dd288c653e6ce785d3d0f9707a6e7520175a04a8eb79d201b42e5aca0596c6c7586a48d3c46c7b1872f1847
941c3fef33d9e8f62209c19cfb
Decoding: b'hello'
```

Finalmente, foi implementada uma versão do RSA utilizando o OAEP para codificar a mensagem antes de criptografá-la, gerando o seguinte resultado.

```
(3.11.8) mateusoliveirasantos@192 rsa % python3 rsa.py
p gerado: 123519219723999965781352654599890616287025037496134049795865045806631570106102792961886542855088081649195698219
55580407904170095682605567194654430615571769745536110821963841287713887633616032622537173275774915309463423904581152397783
957965629335260205089314112479992142530675743163621000192175435919025016677
q gerado: 1431351271910164326131723117599187002397335129611635074857748110943535798128617272251192417028449216592551018599
18277364321692008924221499310960368315258415884518567872143122834188507950714384445603611463445509564101718946035947088721
71035176143283511132073986414772512928873252205179097519237329902059990743
Codificando mensagem "Attack at Dawn" com chave pública...
Mensagem codificada: 6edf70eb628557180e7da53811ae2a679838c5572fe47914494aa0629c1e3d79a7b1ee0ec16025c17aeb3fdab7352177c3b28
acc6c86be8cd5b2c74f325d9ed2018ad929cb14edfc67e22f0655a505c4ce6d7efa819768f63d185c591e4a4f657ba2deacad1a52a2f556c763ec5514
dcae11698c4d50ad65aeb9df8c5ca943f5bb1fec6c0ad1dcc79a1a634059cb9df29eaf2dcdf3f33f1d12c7ab9932dd3e5cb35920bc0d340ee6254bbca
86e8f32daf5e800541d315be02ed7597afd1de90d398d2bc4ccad840d8fb479f10d6337e150cc485da3380653f67ef19fc998ebab0ec6a7b42f97ea91
adfed20f8ad6e1ebf1238a59c3b42b21479cd8e54af08
Mensagem decodificada com chave privada: Attack at Dawn
```

### 4.2 Assinatura e verificação

Foi usado um arquivo com a seguinte mensagem:

*"Este é um documento de teste para assinatura."*

- **Verificação de sucesso da assinatura.**

Nessas condições a assinatura gerou a seguinte saída:

```
Gerando p (primo de 1024 bits)...
p gerado: 173142109436080084227267035861092431430817221377236854105540834264010098492559745518396719722725680178
6190901539882781508015506342828110281561412218396294359887910612874803088354211895158284537549278469056147294351
01498836258155599486944168022540990988068303541148696897977494602370875447444548504614137672781
Gerando q (primo de 1024 bits)...
q gerado: 162683383896935664480219914287563730213010204261413942143946097140555954251993892374052665190370459266
4024310507704176375855065002438290905220235765792737959269035153166844944211069433731376980148862916588098526932
29902820106744724748954622847428669012571228600572960673195074662640629621497311311491183595081

Hash do documento (32 bytes): 60a6c911dd7a301bb76295f3c29e9b7e2e9824f381a54fb9edf4157e6554b626
```

Enquanto na verificação obtivemos:

```
--- Iniciando Verificação ---
DEBUG: Decrypted hash (bytes): 60a6c911dd7a301bb76295f3c29e9b7e2e9824f381a54fb9edf4157e6554b626
DEBUG: Actual file hash (bytes): 60a6c911dd7a301bb76295f3c29e9b7e2e9824f381a54fb9edf4157e6554b626
Assinatura válida para o arquivo 'documento.txt'!
Verificação concluída com sucesso: Assinatura é autêntica.
```

Já para o caso após modificar o arquivo original, obteve-se a seguinte saída:

```
--- Tentando verificar após adulteração do arquivo original ---
DEBUG: Decrypted hash (bytes): 60a6c911dd7a301bb76295f3c29e9b7e2e9824f381a54fb9edf4157e6554b626
DEBUG: Actual file hash (bytes): 4008a5c5373c0380160b651c993adc87a5ec27d09543da7fa4c250eaa1533aad
Assinatura inválida para o arquivo 'documento.txt'. Os hashes não correspondem.
Adulteração detectada com sucesso (como esperado).
```

Com essas saídas foi observado que os algoritmos para a assinatura e para a verificação funcionam corretamente.

## 5. Considerações Finais

A implementação atendeu todos os requisitos propostos: geração de chaves RSA com teste de primalidade, assinatura de mensagens usando SHA-3 e OAEP, e verificação baseada em parsing e comparação de hashes.

Dificuldades incluíram o tratamento de arquivos grandes e otimização da geração de primos. Como melhoria futura, sugere-se implementar uma interface gráfica ou adaptar para múltiplos formatos de assinatura.

## 6. Referências

- KUROSE, Jim; ROSS, Keith. *Computer networking: a top-down approach*. 8. ed. Boston: Pearson, 2020.

-Wikipedia: [https://en.wikipedia.org/wiki/Optimal\\_asymmetric\\_encryption\\_padding](https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding)