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 $e \ mod \ \phi(n)$, 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: $c(m) = m^e \ mod \ n$. 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: $m(c) = c^d \ mod \ n$. 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.

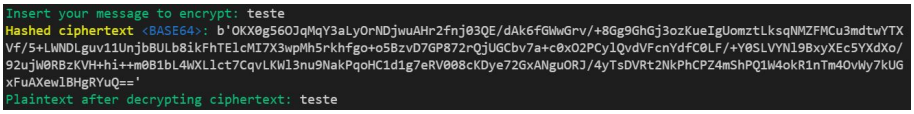

Figura 1. Processo de cifração e decifração de uma mesma mensagem.

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 $sign(h) = h^d \ mod \ n$, 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, $h = sign(h)^e \ mod \ n$. 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.

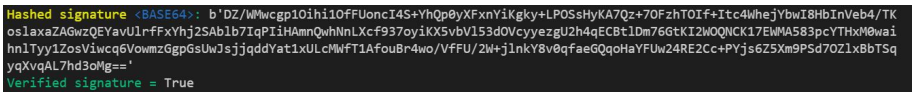

Figura 2. Processo de geração de um *hash* para cifração e posterior verificação se o retorno da decifração condiz com o texto puro.

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.

6 Resultado dos testes

Destarte, é apresentado o resultado da cifração de uma mensagem "Hello world", isto é, saída em terminal e fragmento do arquivo *out.md* gerado, no mesmo diretório, de modo a conter a chave pública, a título de exemplo:

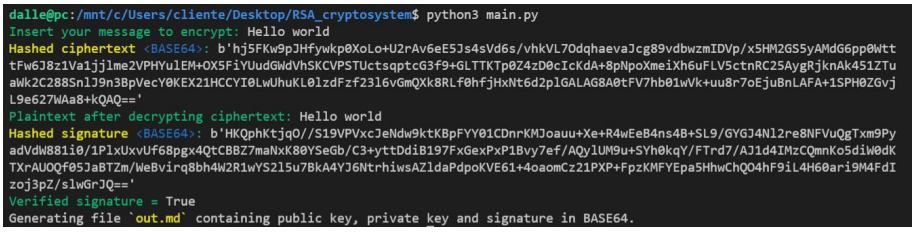

Figura 3. Saída em terminal.

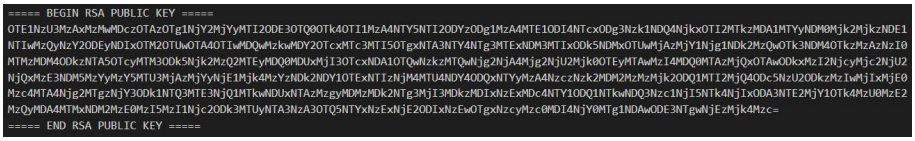

Figura 4. Fragmento do arquivo *out.md* (além disso, contém dados acerca da chave privada e assinatura).

Referências

1. Wikipedia contributors, Wikipedia, The Free Encyclopedia., 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, 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, 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, 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, Last accessed 27 Oct 2021