Segurança Computacional
Geração e Verificação de Assinatura Digital

Pedro Henrique de Brito Agnes, 18/0026305
Pedro Pessoa Ramos, 18/0026488

Dep. Ciência da Computação - Universidade de Brasília (UnB)

1 Implementação

Foi desenvolvido um algoritmo em python na versão 3.8 ou acima que recebe um arquivo e gera a assinatura digital para ele ou verifica a assinatura. Para executar passando o documento `msg.txt` e produzindo o resultado em `sample/cifra.txt`, pode-se usar o comando abaixo:

```
1 python src/signature.py sample/msg.txt -o sample/cifra.txt
```

Vão ser geradas as chaves pública e privada do RSA que foram usadas para a cifração do hash (assinatura) e da chave de sessão (AES), que será gerada também, contendo a original em um arquivo e a sua cifra em outro. O programa vai avisar o local e nome de cada uma das chaves geradas pra que os arquivos possam ser passados como parâmetro na hora da verificação da assinatura. Pode ser passado o argumento `-h` para o programa para a listagem de opções disponíveis conforme mostrado abaixo:

- **-o** - Onde será feito o output. Obrigatório
- **-k** - Arquivo com a chave pública ou privada, a depender da operação. Obrigatório na decifração.
- **-s** - Arquivo com a chave de sessão ou o criptograma da chave de sessão. Obrigatório na decifração.
- **-d** - Argumento que indica que o programa vai descriptografar. Deve ser passado no final do comando sem parâmetros adicionais.

Por exemplo, para cifrar o arquivo `sample/msg.txt` com a chave de sessão encontrada no arquivo `keys/session_sample` gerar a sua assinatura digital com a chave privada localizada em `keys/key_sample` e armazenar o criptograma da mensagem no arquivo `sample/cifra.txt`, pode ser usado o comando abaixo:

```
1 python src/signature.py sample/msg.txt -s keys/session_sample -k
    keys/key_sample -o sample/cifra.txt
```

Além do argumento passado para o `-o`, será gerado um outro arquivo de mesmo nome concatenado com `_sign` que representa a assinatura. Este arquivo não precisará ser passado ao programa, ele será encontrado automaticamente ao passar o `-o` ao usar a opção de decifrar.

Da mesma forma, para decifrar o criptograma gerado com a chave de sessão cifrada `keys/session_sample.pub` e armazenar o resultado em um arquivo `sample/out.txt`, pode-se usar o comando abaixo que utiliza a chave privada:

```
1 python src/signature.py sample/cifra.txt -s keys/session_sample.pub -k
    keys/key_sample.pub -o sample/out.txt -d
```

Após a finalização da execução, o programa vai informar se o hash gerado após as operações é igual ao hash obtido anteriormente, ou seja, vai verificar se a assinatura é válida.

2 RSA

2.1 Geração de Chaves

Inicialmente são gerados 2 números primos aleatórios `p` e `q` de 1024 bits cada. Para obter estes números, foi inicialmente gerado um número qualquer do tamanho especificado, em seguida, sua primalidade foi testada para cada um dos primos até 1000, que foram dispostos em um array após serem computados uma única vez pelo crivo de Erastótenes. Em seguida, é realizado o teste de primalidade probabilístico de Miller Rabin, executado 20 vezes. Se ambos os testes passarem, o número é considerado primo e selecionado. Caso contrário, um novo número aleatório é gerado para realizar o teste novamente. Os códigos relacionados à geração e verificação dos primos estão localizados na pasta `src/util/primes.py`, incluindo a implementação do algoritmo de Miller Rabin.

Com `p` e `q` definidos, a chave pública é obtida usando 2 valores, o `n` e o `e`. O número `n` é o resultado da multiplicação de `p` e `q`, logo sendo de 2048 bits e já o `e` será um primo bem menor que `p` e `q` sendo coprimo com `n`. Já a chave privada é obtida pela função totiente de Euler em `n`, onde temos a multiplicação de 2 números primos, logo, podendo ser calculada pela fórmula abaixo:

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

Com o resultado da função acima, é calculado o seu inverso multiplicativo com `e` para assim obter o valor de `d` que é o único valor necessário para a chave privada. Para fins de performance, também são adicionados os valores de `p` e `q` na chave privada. Após a geração, as chaves são guardadas na pasta `keys` com a formatação base64 sendo a pública com a extensão `.pub`. O conteúdo dos arquivos é basicamente os números que compõem cada uma das chaves fixos com tamanho de 2048 bits concatenados em um só, de forma a facilitar o *parsing* depois.

2.2 Cifragem e Decifragem

Como trata-se da assinatura digital, é usada a chave privada para cifrar e a pública para decifrar, sendo o contrário do RSA "convencional". Para cifrar, é feita simplesmente uma potenciação entre a mensagem (bytes concatenados em um número gigante) e o valor de `d`,

onde é pego o resto da divisão por `n`. Já para decifrar, é feita a potênciação da mensagem com `e`, também pegando o resto da divisão por `n`.

No programa, o que é cifrado pelo RSA é o hash do criptograma da mensagem gerado pelo AES, resultando na assinatura digital e em seguida, a chave de sessão que foi usada no AES. Para a verificação, a assinatura é decifrada para assim obter o hash, e em seguida, esse hash é comparado com o hash gerado pelo criptograma que foi recebido e o programa informa se os hashs são iguais. Também é decifrada a chave de sessão para que a mensagem original seja recuperada. Os arquivos relacionados à implementação RSA podem ser encontrados na pasta `src/rsa/`.

3 AES

O AES foi usado no modo CTR com o nonce fixo no valor 0 para a cifração simétrica da mensagem, que é usada para gerar o hash, sendo usada a chave de sessão para cifrar e decifrar. A implementação da cifra de blocos foi aproveitada do trabalho anterior da disciplina de segurança computacional, podendo ser econtrada na pasta `src/aes/`.

A geração da chave é feita na pasta `keys/`, sendo nomeada `session` e concatenada com um id incremental. Consequentemente, será gerado o arquivo de mesmo nome porém com a extensão `.pub` que representa a mesma chave porém cifrada pelo RSA e é a usada na verificação da assinatura, porém deve ser decifrada antes de ser utilizada.

4 Assinatura

Para a assinatura, foi criado um hash do criptograma gerado pelo AES usando a função `sha3_256` da `hashlib` diponibilizada pela linguagem python. Para a formatação dos valores ao escrevê-los em arquivos, foi usado o `base64`, que é disponibilizado pelo python também, dispondo de operações para codificar e decodificar no formato. A assinatura do documento é gerada em um arquivo separado que tem o mesmo nome do output (criptograma da mensagem) concatenado com um `_sign`. A implementação completa pode ser resumida às imagens abaixo:

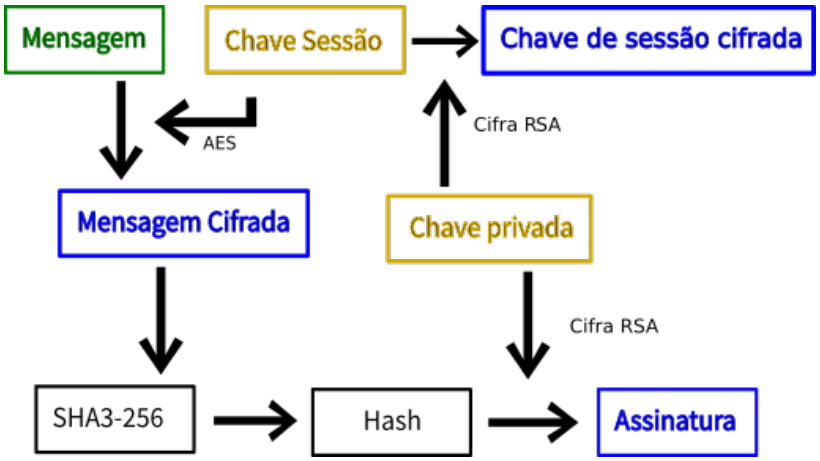

Figura 1: Geração da assinatura

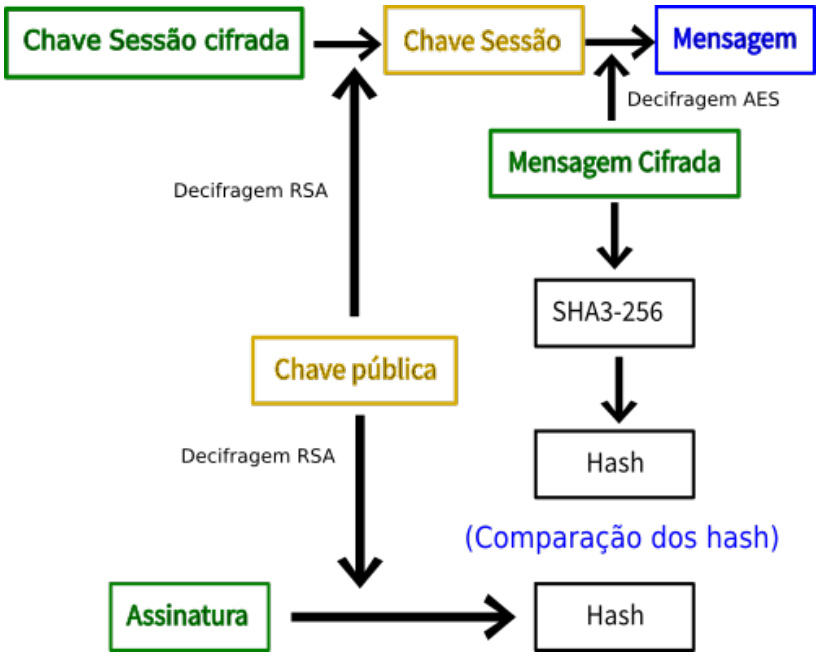

Figura 2: Verificação da assinatura

O script que realiza a geração e verificação das assinaturas pode ser encontrado no arquivo `src/hash/sign.py`. Nele, são recebidos os valores necessários para a operação a ser utilizada, onde o criptograma é cifrado pelo RSA para produzir a assinatura que é formatada em base64 no caso da geração. Já na verificação, o arquivo com a assinatura em base64 é decodificado e em seguida, decifrado com a chave pública para encontrar o hash que havia sido produzido anteriormente, que em seguida, é comparado com o novo hash gerado pelo criptograma. Se ambos forem iguais, a assinatura é válida.

5 Conclusões

Com a assinatura do documento, ao cifrá-lo e enviar a outra pessoa com as chaves necessárias, fica mais fácil de se verificar a integridade, pois qualquer mínima alteração no documento gerará um hash diferente que será identificado na verificação. O RSA é um grande nome no ramo das assinaturas digitais, sendo muito usado para a cifração do hash, já que é um algoritmo assimétrico e permite o envio apenas da chave pública para o recipiente, o que impede de que seja gerada uma assinatura falsa.