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`, pode-se usar o comando abaixo:

```
python src/signature.py sample/msg.txt
```

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.
- **-d** - Argumento que indica que o programa vai descriptografar. Deve ser passado no final do comando sem parâmetros adicionais.

Por exemplo, para gerar a assinatura digital do arquivo `sample/msg.txt` com a chave privada `keys/key_sample` e armazenar o criptograma da mensagem no arquivo `sample/cifra.txt`, pode ser usado o comando abaixo:

```
python src/signature.py sample/msg.txt -k keys/key_sample.pub -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.

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

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

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`. 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`.

2.2