# Projeto Segurança Computacional

Igor Bispo de Moraes - 170050432

Guilherme Oliveira Loiola - 170050335

## Descrição da Implementação

RSA é utilizado para criptografia assimétrica e para assinaturas digitais.

No processo da criptografia assimétrica utiliza-se as chaves como parâmetros do algortimo. Dessa forma, as chaves variam entre os usuários e o par de chaves contém chave pública e privada. A chava pública é usada para criptografar os dados e a chave privada é a chave que utiliza-se para descriptografar os dados.

Por se tratar de uma criptografia assimétrica as chaves do processo de criptogradia e descriptografia são distintas. As chaves são ligadas, porém é inviável computacionalmente descobri-lás. A partir desse funcionamento podemos dividir as chaves em pública e privada.

Pela criptografia assimétricaa distribuição de chaves ocorre de maneira mais simples, bastando assim o segredo da chave privada ser mantido e as chaves públicas podem ser distribuidas.

SHA é a função de hash utilizada para mesclar os dados, tornando assim ainda mais difícil a quebra da criptografia. Afinal, ainda que conhecido o algoritmo de hash é inviável recalcular uma sáida da função hash. Tornando assim, a junção dos algoritmos um acréscimo na segurança.

Geração das chaves No RSA as chaves são geradas da seguinte maneira: 1. Escolha de forma aleatória dois números primos grandes p e q, da ordem de 10 100 no mínimo 2. Calcule n=pq 3. Calcule a função totiente em n: $\Phi(n) = (p-1)(q-1)$ 4. Escolha um inteiro e tal que $1 < e < \Phi(n)$, de forma que e e $\Phi$(n) sejam co-primos (ou seja, o único divisor comum seja 1) 5. Calcule d de forma que de $\equiv 1 \pmod{\Phi(n)}$, ou seja, d seja o inverso multiplicativo de e em (mod $\Phi$(n)). Por fim, temos: A chave pública: o par (n, e,), e a chave privada: a tripla (p, q, d).

- Cifração Para transformar uma mensagem m, onde $1 < m < n-1$ numa mensagem cifrada usando a chave pública do destinatário n e e basta fazer uma potenciação modular: $m \equiv c \pmod{n}$ e A mensagem então pode ser transmitida em canal inseguro para o receptor (há um algoritmo para realizar esta potência rapidamente).
- Decifração Para recuperar a mensagem m da mensagem cifrada c usando a respectiva chave privada do receptor n e d, basta fazer outra potenciação modular: $c \equiv m \pmod{n}$.

**SHA-3**

Um algoritmo de hash criptográfico (alternativamente, "função" de hash) é projetado para fornecer um mapeamento aleatório de uma sequência de dados binários para um "resumo de mensagem" de tamanho fixo e atingir certas propriedades de segurança. Algoritmos de hash podem ser usados para assinaturas digitais, códigos de autenticação de mensagem, funções de derivação de chave, funções pseudo-aleatórias e muitos outros aplicativos de segurança.

O método SHA-3 tem base no algoritmo de hash Keccak.

Pseudocódigo Keccak :

```
Keccak-f[b](A) {

  for i in 0…n-1

  A = Round[b](A, RC[i])

  return A
}

Round[b](A,RC) {

  //Theta step

  C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4], - for x in 0…4

  D[x] = C[x-1] xor rot(C[x+1],1), - for x in 0…4

  A[x,y] = A[x,y] xor D[x], - for (x,y) in (0…4,0…4)

  //Rho and Pi steps

  B[y,2*x+3*y] = rot(A[x,y], r[x,y]), - for (x,y) in (0…4,0…4)

  //Chi step

  A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]), - for (x,y) in (0…4,0…4)

  //Iota step
  A[0,0] = A[0,0] xor RC

  return A

}
```

Diferenciais SHA-3:

- Resistência a ataques do tipo “length extension attacks”.
- SHA-3 usa a estrutura de esponja com a permutação Keccak.
- Completely different internal design.

Design SHA-3:

Dada uma string de bits de entrada N, uma função de preenchimento pad, uma função de permutação f que opera em blocos de bits de largura b, uma taxa r e um comprimento de saída d, nós temos capacidade c = b - re a construção de esponja Z = esponja[f, pad, r] , rendendo uma string de bits Z de comprimento d, funciona da seguinte forma:

- Preencha a entrada N usando a função pad, produzindo uma string de bits preenchida P com um comprimento divisível por r (de tal modo que n = P / r é um inteiro)
- Dividir P em n pedaços consecutivos de r bits P0 , …, P−1
- Inicializar o estado S para uma string de b bits zero
- Absorver a entrada no estado: para cada bloco Pi :
  - Estender P i no final por uma string de c bits zero, resultando em uma de comprimento b
  - XOR isso com S
  - Aplique a permutação de bloco f ao resultado, produzindo um novo estado S
- Inicializar Z para ser a string vazia
- Enquanto o comprimento de Z é menor que d :
  - Acrescente os primeiros r bits de S a Z
  - Se Z ainda tiver menos que d bits, aplique f a S , produzindo um novo estado S
- Truncar Z para d bits

A linguagem utilizada para implementação dos algoritmos foi o python-3.

Todas as funções do algoritmo RSA, SHA-3 e geração de chaves foram implementações próprias, utilizando apenas bibliotecas de suporte para os algoritmos. Assim, cobrindo as competências soliciadas pelo trabalho.

## Resultados

O trabalho de implementação alcançou um bom funcionamento. Utilizando RSA + SHA para realizar a assinatura. Os processos de criptografia e descriptografia funcionam bem caracterizando uma implementação concisa dos métodos de criptografia RSA e SHA.

Dos requisitos solicitados para o trabalho tem a falta do método OEAP.

## Como Executar

Vá até o diretório raiz e execute:

“python3 main.py <input_file> <s/v> <public_key_file>”

em que:

- <input_file> é o arquivo que será assinado
- <s/v> ‘s’ para assinar e ‘v’ para verificar
- <public_key_file> é o arquivo que contém a chave pública gerada pelo processo de assinatura, necessário para verificação

Ao executar com opção ‘s’ para assinar, será gerado :

- um arquivo com nome “signature_<input_file>” contendo a assinatura RSA em hexadecimal
- um arquivo com nome “public.key” contendo a chave pública RSA
- um arquivo com nome “private.key” contendo a chave privada RSA

Ao executar com opção ‘v’ para verificar, será impressa na tela uma mensagem de confirmação caso a assinatura seja válida.