Segurança Computacional
Alexandre Souza Costa Oliveira
170098168
Universidade de Brasília - UnB

Trabalho 03:

1. Compilação e utilização:

A implementação do código para RSA foi feita em **Python** 3.10.0 em ambiente Windows ( versão 11 ).

Para execução do código, é apenas necessário entrar na pasta dele com o cmd ou powershell e executar o comando ***py rsa.py***

Você deve colocar o texto cifrado ou a mensagem que deseja cifrar no arquivo **texto.txt**. Feito isso, após executar o programa, deve escolher sobre se quer cifrar ou decifrar. Se escolher cifrar, guarde com cuidado as chaves N (pública) e D (privada) que aparecerão na tela, você precisará delas para decifrar depois.

2. Sobre o trabalho

Neste trabalho foi implementado um gerador e verificador de assinaturas RSA para um texto contido num arquivo. O RSA é um algoritmo de segurança computacional responsável por cifrar mensagens e no qual é extremamente difícil de quebrar, pois utiliza de números primos enormes, fazendo com que ele fique imprevisível, além disso ele possibilita a assinatura digital, onde é possível garantir a autenticidade da mensagem no quesito de se ela foi modificada ou se partes da mesma foi perdida. O objetivo deste trabalho é implementar o RSA com chaves de 1024 bits, em conjunto de hash SHA3 para cifrar, decifrar e realizar a assinatura digital e garantir a autenticidade da mensagem.

3. Implementação

Para este trabalho, foi utilizado um código já feito para a função de Miller Rabin, na qual é responsável por garantir probabilisticamente se um número é primo. A seguir estão listadas todas as funções implementadas para o funcionamento do algoritmo.

```
# Funcao responsável por garantir probabilisticamente que o número é primo
def miller_rabin(n, k):
# Função geradora de números primos
def GerarNumeroPrimo():
# Função geradora de Hash
def GerarHash(mensagem):
# Função de Parsing para Base64
def EncodeBase64(mensagem):
# Função inverso de Parsing para Base64
def DecodeBase64(objeto):
```

```
# Função que converte string em int
def StringBytesToInt(mensagem):
# Função que converte Int para Bytes
def IntToBytes(mensagem):
# Função que cifra uma mensagem
def Cifrar(e, n, mensagem):
# Função que verifica a hash
def VerificarHash(mensagem, hash):
# Função que decifra uma mensagem
def Decifrar(d, n, cifrado):
# Função que lê o arquivo texto.txt
def LerArquivo():
# Função main
def main():
```

Para gerar os números primos foi utilizado a biblioteca random do python, com a utilização do método getrandbits, no qual é possível gerar um número aleatório de N bits. Foi então gerado com 512 bits, para que o resultado final das chaves fossem de 1024 bits. Os cálculos na main para geração das chaves, ficaram assim:

```
        p = GerarNumeroPrimo()
        q = GerarNumeroPrimo()
        n = p*q
        phi = (p-1)*(q - 1)
        e = 339225029003687
        d = libnum.invmod(e ,phi)
```

E a função geradora de números primos, assim:

```
# Funcao geradora de numeros primos
def GerarNumeroPrimo():
    isPrimo = False

    while(not isPrimo):
        numero = random.getrandbits(512)

        isPrimo = miller_rabin(numero, 40)

    return numero
```

Dessa forma, teremos as chaves privadas e públicas geradas. O único que não foi gerado aleatoriamente, foi o ‘e’, no qual foi colocado um número primo grande suficiente que satisfaz a regra de geração do mesmo 1 < e < phi.

Para a geração de hash, foi utilizado a biblioteca hashlib do python. E com a utilização do método sha3_256, conseguimos gerar uma hash para o texto.

```
# Funcao geradora de Hash
def GerarHash(mensagem):
    hash = hashlib.sha3_256(mensagem.encode()).hexdigest()

    return hash
```

Após a leitura do arquivo e a geração das chaves e da hash, nosso código está pronto agora para realizar a cifragem, assim tanto a hash como o texto são cifrados separadamente na main

```
        # Cifra a mensagem e hash
        textoCifrado = Cifrar(e,n, mensagem)
        hashCifrado = Cifrar(e,n, hash)
```

Com a utilização de N e de E, é realizada a cifração do texto. A função StringBytesToInt converte todo uma string em um int, tornando o processo de cifrar mais fácil e prático. A função pow faz a cifração em si com mod N.

```
# Funcao que cifra uma mensagem
def Cifrar(e, n, mensagem):
    cifrado = pow(StringBytesToInt(mensagem), e, n)

    return cifrado
```

É feita uma concatenação entre a hash e o texto cifrado, realizando assim a assinatura digital. Para fazer isso, é adicionado prefixos de Texto e Hash para separar cada um deles e então é chamada a função de transformar em base64. A função de EncodeBase64, utiliza da biblioteca base64 do python para realizar tal transformação. E assim, nós temos então os dados prontos na base64.

```
# Transforma em Base64
   preparo = "Texto:" + str(textoCifrado) + "Hash:" + str(hashCifrado)
   encoded = EncodeBase64(preparo)
   print("Base64: ", encoded.decode())
```

Para decifrar, será solicitado a chave N e D ao usuário e com base nelas, a decifração será feita. O arquivo texto.txt será lido com a mensagem cifrada e transformado de base64 para string normal. E então a hash e o texto são separados com base nos prefixos antes estabelecidos.

```
        encoded = LerArquivo()

        decoded = DecodeBase64(encoded).decode()
        decoded = decoded.replace("Texto:", "")
        decoded = decoded.split("Hash:")
```

Após termos a chave, a função de decifrar é chamada em conjunto com a função de transformar o resultado inteiro em bytes e em seguida em string. Ao termos o texto decifrado e a hash decifrada, é criada uma hash com esse texto decifrado e é então verificado se a hash do texto e a hash decifrada são iguais, se for, a autenticação é confirmada.

```
# Decifra o Texto
textoDecifrado = IntToBytes(Decifrar(d, n, decoded[0])).decode()
print("Texto Decifrado: ", textoDecifrado)
print("")

#Decifra a Hash
hashDecifrado = IntToBytes(Decifrar(d, n, decoded[1])).decode()
print("Hash Decifrada: ", hashDecifrado)
print("")

# Verificando Assinatura
print("Verificando Hash")
if(VerificarHash(textoDecifrado, hashDecifrado)):
      print("A verificacao foi concluida com sucesso, autenticidade
comprovada.")
else:
      print("A verificacao falhou, a hash nao corresponde a mensagem.")
```

4. Dificuldades e possíveis bugs

A maior dificuldade foi a de entender o OAEP, e por conta de não ter entendido muito bem, ele foi deixado de fora deste trabalho.

Também foi difícil de entender como funcionaria a parte de sessão para cifração simétrica de mensagem. Por mais que eu imaginei que pudesse usar o AES implementado no trabalho anterior, foi um pouco difícil entender a relação dele com a mensagem e a hash, e como isso chegaria até o “receiver”. Então essa parte também foi deixada de fora do trabalho. Então a mensagem é cifrada diretamente com RSA e é considerado que o “receiver” já possui a chave privada D, no caso inserida ao iniciar o programa.

É possível que um bug aconteça se você inserir a chave erroneamente, dando crash no programa. Exemplo, ao cifrar a mensagem, você receberá as chaves N e D, se você alterar propositalmente essas chaves, na hora de decifrar, o programa pode dar erro.

É possível que esse programa seja incompatível com outros programas, por conta dos prefixos Hash e Texto na cifração e decifração da mensagem.

5. Referências


Sobre RSA e seu funcionamento:
https://pt.wikipedia.org/wiki/RSA_(sistema_criptogr%C3%A1fico)
Função Miller Rabin:
https://gist.github.com/Ayrx/5884790
Funcionamento Algoritmo RSA Fácil de Entender
https://www.youtube.com/watch?v=9m8enHN13mw&ab_channel=JeronimoBezerra