UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

FCT – FACULDADE DE CIÊNCIAS E TECNOLOGIA
DMC – DEPARTAMENTO DE MATEMÁTICA E COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JUAN CARDOSO DA SILVA - 171257138
GUILHERME DE AGUIAR PACIANOTTO - 181251019

MATÉRIA SEGURANÇA DA INFORMAÇÃO

ATIVIDADE 4

Presidente Prudente, 25/08/2022

INTRODUÇÃO

Nas aulas apresentadas, foram ensinados os algoritmos de criptografia e compartilhamento de chaves, sendo eles o RSA(Rivest-Shamir-Adleman) e o Diffie-Hellmam. O RSA é um algoritmo poderoso de criptografia para compartilhar mensagens e palavras secretas pela internet, seja utilizando SSH, validações por emails e dentre outros. O Diffie-Hellmam é um algoritmo de compartilhamento de chaves criptográficas, utilizando geração de chaves simétricas em ambos lados para chegar no resultado final, sendo apenas o resultado do valor calculado intermediário como valor trocado entre as entidades A e B do Diffie-Hellmam.

A ideia deste projeto é ajudar os alunos a entenderem como esses dois mecanismos trabalhando junto colaboram para criptografia e segurança de troca de mensagens através de um socket, sendo uma simulação de como os algoritmos funcionam em um ambiente profissional em funcionamento, seja em uma rede local ou na internet.

OBJETIVO

O objetivo deste trabalho é ajudar o programador-aluno a entender como os mecanismos de criptografia assimétrica tal como o RSA, podem funcionar em conjunto com outros mecanismos distribuidores de chaves, neste caso, escolhido o Diffie-Hellmam.

Dentro do escopo do RSA, não existe compartilhamento de chaves, sejam públicas ou privadas, no meio de comunicação entre usuários, logo a utilização de um método seguro foi implementada para fazer o trabalho funcionar.

DISCUSSÃO

O problema inicial no desenvolvimento foi a construção do RSA e pensar em um meio de fazer com que ambos lados dos usuários, A e B, possam desencriptar com sucesso as chaves. No algoritmo RSA, a escolha do coprimo para execução do programa pode ser feita de N maneiras diferentes, uma delas seria utilizar Diffie-Hellmam para compartilhar uma chave que é parte dos números primos geradores de P e Q. Considerando que P seja um número primo quadrado de Q, essa geração não permite a aleatoriedade na geração de um coprimo.

Uma solução foi utilizar a chave final no cálculo do Diffie-Hellman como um índice para escolher N números coprimos aleatórios de 2 até a Φ calculado pelo programa, assim dando um fator de segurança a mais ao projeto. Desta maneira foi possível encontrar um meio de compartilhar as chaves do Diffie-Hellmam e utilizá-las no RSA para poder realizar as criptografias entre usuários.

Na construção do projeto foi utilizado o protocolo UDP para realizar a comunicação entre as pessoas A e B. Por ser um protocolo não voltado a conexão, utilizar os mecanismos de criptografia desenvolvidos nesse projeto é importante para adicionar uma camada de segurança. Caso as pessoas A e B façam o uso desse protocolo para se comunicarem, todo dado enviado está criptografado e qualquer terceira parte obtendo essas informações não seria capaz de compreender o que está sendo compartilhado entre elas.

Para gerar as encriptações, foi utilizado uma tabela de tuplas em Python, onde o caractere representa a chave (k) e o valor numérico correspondente a essa chave, sendo um valor (v). Com essas tuplas é possível realizar o algoritmo de criptografia do RSA do qual:

$$valor_{numérico}^{public-key} mod\ n \rightarrow valor_{criptografado}$$

O valor numérico é escolhido da seguinte maneira: compara-se o char da string com o valor da chave, este valor é convertido para inteiro e realiza-se a conta. No cálculo da volta, o mesmo processo é feito, mas se inicia com o valor numérico direto e é realizado o cálculo do reverso, comparando o resultado com o valor da chave e adicionando a chave a um construtor de string. Assim é estabelecido o processo de troca de mensagens criptografadas entre os usuários A e B.

CONCLUSÃO

Com isso pode-se concluir que o RSA, como um poderoso mecanismo de criptografia, necessita de um método para definir os coprimos entre usuários, já que a natureza assimétrica do algoritmo impede o compartilhamento de chaves. Assim, a necessidade de utilizar algo como o Diffie-Hellmam é necessária para conseguir com que ambos os lados possam encriptar e desencriptar as mensagens trocadas.

CÓDIGO FONTE

```
class DiffieHellman:
  def __init__(self, p, alpha, secret) -> None:
    self.p = p             # Valor p
    self.alpha = alpha     # Valor Alpha
    self.secret = secret   # Secret único de cada cliente, não deve ser
compartilhado
    self.x = 0           # Valor x calculado na geração da chave pública.
    self.incoming_x = 0    # Valor que é enviado para fazer o calculo da
chave privada.
    self.key = 0           # Valor da chave.

  # Calcular o X
  def calc_x(self):
    self.x = int(pow(self.alpha, self.secret, self.p))

  # Gera a chave PSK para os usuários.
  def generate_psk(self):
    self.key = int(pow(self.incoming_x, self.secret, self.p))
    return self.key

  # set do x que chega, aqui apelidado de y
  def set_incoming_x(self, y):
    self.incoming_x = y

  # get do x que chega ao cliente.
  def get_incoming_x(self):
    return self.incoming_x

  # retornar o x para usar no socket.
  def get_x(self):
    return self.x
```

```
class RSA:
  def __init__(self, p, q):
    self.p = p  # p do rsa
    self.q = q  # q do rsa
    self.n = 0  # p * q
    self.euler_totient = 0  # euler totient
    self.public_key = []    # chave pública, [n, e]
    self.private_key = []   # chave privada, [n, d]
    self.e = 0               # co-primo escolhido da lista de co-primos
que serão gerados.
    self.table = {'a': "101", 'b': "102", 'c': "103", 'd': "104", 'e':
"105", 'f': "106", 'g': "107", 'h': "108",
      'i': "109", 'j': "110", 'k': "111", 'l': "112", 'm': "113", 'n':
"114", 'o': "115", 'p': "116",
      'q': "117", 'r': "118", 's': "119", 't': "120", 'u': "121", 'v':
"122", 'w': "123", 'x': "124",
      'y': "125", 'z': "126", " ": "127", 'A': "201", 'B': "202", 'C':
"203", 'D': "204", 'E': "205",
      'F': "206", 'G': "207", 'H': "208", 'I': "209", 'J': "210", 'K':
"211", 'L': "212", 'M': "213",
      'N': "214", 'O': "215", 'P': "216", 'Q': "217", 'R': "218", 'S':
"219", 'T': "220", 'U': "221",
      'V': "222", 'W': "223", 'X': "224", 'Y': "225", 'Z': "226", ",":
"301", ".": "302", '[':"303", ']':"304"
    } # dicionário de strings para gerar a mensagem criptografada.

  # calcula p * q
  def calc_n(self):
    self.n = self.p * self.q

  # não utilizado, verificar remoção depois.
  def set_n(self, number):
    self.n = number * self.p * self.q

  # calcula euler totient
  def calc_euler_totient(self):
    self.euler_totient = (self.p - 1) * (self. q - 1)

  def calc_public_key(self, index):
    # Geração de co-primos, para cada x dentro do intervalo de 2 até euler
totient, verificar se ele é co primo do euler totient
    coprimes = [
```

```
      x   for   x   in   range(2,   self.euler_totient)   if(math.gcd(x,
self.euler_totient) == 1)
    ]
    # Escolhendo um co-primo baseado na chave resultante do DH.
    self.e = coprimes[index]
    self.public_key = [self.n, self.e]
    return self.public_key

  def calc_private_key(self):
    d, text = 0, []
    # Geração da chave privada.
    for k in range(1, self.e):
      if (k * self.euler_totient + 1) % self.e == 0:   # verifica se k
pode pertenceer ao intervalo que queremos para calcular nosso d.
        d = (k * self.euler_totient + 1) // self.e    # se ele pertencer,
calcule D e saia do loop.
        break
    self.private_key = [self.n, d]
    return self.private_key

  def encrypt_message(self, message):
    # retornando um vetor onde cada posição é um char encriptado.
   # para cada char na mensagem procurar o valor correspondente da tupla
e calcular valor^public_key % n
    return [
      int(pow(int(self.table[i]),                    self.public_key[1],
self.public_key[0]))
      for i in message
    ]

  def decrypt_message(self, message):
    # Mostrando a chave privada do usuario
    print(f"Chave privada e n: {self.private_key} {self.n}")
    # Fazendo o calculo do reverso com a chave privada.
    # para cada valor da string message convertido para inteiro, calcular
valor^private_key % n
    array = [
      int(pow(int(i), self.private_key[1], self.n))
      for i in message
    ]
    decrypted_message = ""
    #  para  o  tamanho  da  nossa  mensagem,  procurar  os  simbolos
correspondentes na tabela de simbolos.
```

```
    for i in array:
      for k, v in self.table.items(): # para todos items da tupla, pegar
uma chave e valor em um loop de 0 até n-1.
        if(int(v) == i):  # valor da tabela equivale ao o i.
          decrypted_message += str(k) # adiciona a chave da tabela de
tuplas, construindo nossa string.
          break
    return decrypted_message
```