LISTA 01

TREINAMENTO PARA NOVOS BOLSISTAS 2019

Autor: JOÃO VICTOR FERRO

LCCV/UFAL
Maceió, ABRIL DE 2019

1 Introdução

Este tipo de algoritmo é chamado de criptografia assimétrica ou de chave pública, pois existem duas chaves que são usadas. A primeira chave é a chave pública, utilizada para criptografar ou "cifrar" os dados que vão ser enviados, por exemplo: Logins e senhas. A segunda chave é utilizada para descriptografar os dados, é chamada de chave privada, apenas ela consegue retornar o texto "cifrado" no texto original. Esta estrutura composta por duas chaves difere da criptografia simétrica, onde apenas uma chave é utilizada para criptografar e descriptografar o texto, tornando-se inviável o seu uso na internet, pois a chave precisaria ser transferida junto com a mensagem criptografada, sendo facilmente descoberta por um atacante, que conseguiria ler os dados enviados. Para calculá-lo pode ser feito da seguinte maneira. Escolhem-se dois números primos distintos e multiplica-os entre si

$$n = n.m \times B) \tag{1}$$

Em seguida usando a função totiente de Euler, temos que:

$$Phi(N) = Phi(P) * Phi(Q) \tag{2}$$

Agora teremos que achar um outro número Aleatório E, que tem que satisfazer as condições: ser maior que 1 e menor que Phi(N), e também ser primo entre Phi(N).

$$1 < E < Phi(N) = 1 < E < 640$$

mdc(640, E) == 1
E = 13
Pois 1 ¡ 13 ¡ 640 e mdc(640, 13) == 1 (3)

A chave pública é composta pelo N e o E, 697 e 13 respectivamente. Agora teremos que criptografar o nosso texto que será enviado para o servidor. Transformaremos os caracteres, podendo assim fazer operações aritméticas com os valores das letras.Agora devemos achar a chave privada que será usada para descriptografar a Mensagem cifrada enviada pelo cliente. A chave privada é chamada de D e é encontrada seguindo o algoritmo:

$$D * EmodPhi(N) == 1$$

D * 13 mod 640 == 1
D = 197 (4)

Esse Algoritmo RSA é seguro pois para um atacante saber a chave privada D, ele teria que saber quais foram os primos P e Q que foram utilizados para achar o N, para assim calcular o Phi(N). Como o RSA utiliza números de 2048 bits, significa então 2 possibilidades, ou seja, 10 sendo extremamente maior do que a quantidade de átomos no universo observável, que estima-se ser na ordem de grandeza de 10. Fazer a multiplicação de dois números primos de 2048 bits é facil (computacionalmente falando), porém para achar os números primos originários de um N de 4096 bits é preciso fatorar, algo que é computacionalmente inviável, levando milhares de anos para achar os dois números primos que foram multiplicados para dar o N. Sendo assim um algoritmo seguro para comunicação na internet.

2 Resultados e discussões

Primeiramente foi deselvolvido a main, que é basicamente a Classe cliente. De modo que, seja respoonsável por chamar todas as outras funções que são responsáveis pela criptografia ou recursos adicionais no programa.

```
import Gerador_de_chave
import MDC
import encrypt
import decrypt

class Cliente (flag, chavepub,chavepriv):
          dados = []
          contador = 0
         def __init__ (self):
         while True:
                  user = input('')
                  if user == flag:
                           break
                  else:
                           password = input('')
                           dados.append(user)
                  except:
                           print('Sinto muito, n o foi dessa
                              vez, amigo, tenta de novo
                              amanha!')

         def procurar(nome):
                  posicao = filter(nome)

         def apagar(procurar,ordem)
                  if ordem == 1:
                           posicao.remove(nome)
                  else:
```

```
            ordem   = 0
    if chavepriv == 0 or chavepub == 0
    Gerador_de_chave_pub()
    Gerador_de_chave_priv()
    encrypt()
    decrypt()
```

Em seguida o código está disposto em suas funções que são responsáveis de desenvoler a criptografia RSA. Primeiramente temos a função que calcula o MDC entre dois números. Sem nenhuma dúvida é uma das funções mais importantes para a criptografia RSA. Ela segue o teorema de Euclides.

```
def MDC(a,b):

    anterior = a
    atual    = b
    resto    = anterior % atual

    while resto != 0:
        anterior = atual
        atual    = resto
        resto    = anterior % atual
```

Em seguida temos a função que calcula a chave privada. Ela segue o algoritmo supracitado. Nela se teve como objetivo gerar dois numeros primos aleatórios, em tese. Pois, fora solicitado que para evitar futuros problemas os numeros fossem pequenos. Então a criação de uma lista com uma quantidade pequena de numeros finitos. Estes eram selecionados aleatoriamente. Caso fossem iguais, o próprio código era responsável por qualificá-los.

```
from random import randint
import MDC

def Gerador_de_chave_pub:

    def __init__(self,primeiro_primo,segundo_primo,value,n)
        :
        lista = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
        primeiro = lista[random.randint(0,10)]
        while Primeiro_primo == Segundo_primo:
            Segundo_primo = lista[random.randint(0,10)]

        n = Segundo_primo * Primeiro_primo
'''
######### C lculo da fun   o totiente #################
```

```
'''
totiente = (Segundo_primo - 1)*(Primeiro_primo - 1)
'''
######### GERACAO DA CHAVE PUBLICA #################
'''
for i in range (0,totiente,1):
    if  MDC(totiente,i) == 1:
        if i != totiente:
            value =  i
key_pub = [value, totiente]
'''
#########         FIM DA FUNCAO      #################
'''
```

No entanto, para calcular a chave privada, um algoritmo mais simples foi implementado, ao custo de ser perigoso, pois, o comando while pode cair num retorno infinito de modo que o número não possa ser encontrado.

```
def gerador_chave_priv(totiente,e):

    d=0
    while(mod(d*e,totiente)!=1):
        d=d+1
    return d
    key_priv[d,n]
```

Por fim temos a função encrypt que também vai dempenhar o mesmo papel da função decrypt, ao passo que seus atributos mudarão quando uma entrada diferente for disposta. Pois ela vai utilizar-se das chaves para traduzir o texto com sua nova "escala".

```
def Encrypt (totiente, expoente, mensagem):

'''
############## fun   o deve receber dois numeros chave  =
   [totiente, i]  e tamb m uma entrada com uma string em
   seguida
'''
        mensagem = mensagem.split('')
        tamanho = len(mensagem)
        for i in range (0, tamanho, 1):
                '''
                ####### neste caso o comprimento vai variar
                    de acordo com cada palavra adicionada
                '''
                comprimento = len(mensagem[i])
```

```
                for  j range (0, comprimento, 1):
                        valor = ord(mensagem[i][j])
                        mensagem[i][j] = chr((valor**
                            expoente)%totiente)
```

3 Conclusão

Esse tipo de criptografia é muito eficiente sua implementação é simples, seu manuseio também é. No entanto deve se conter a alguns cuidados. Primeiramente, o código tem que gerar números primos aleatórios. E a depender de como isso for feito o processo pode ficar mais extendido. Ao passo que, se for gerada uma chave pública proveniente de um numero primo muito grande, vai levar muito tempo tempo para ser descoberta a chave privada. Por fim concluí-se que essa implementação possui alguns erros a serem corrigidos e que ainda é muito ineficiente.