Descritivo do trabalho de Segurança Computacional 2021.1

Mateus Luis Oliveira - 180112490

Agosto 2021

1 Introdução

A cifra de Vigenère é um exemplo de cifra polialfabética, que é uma cifra baseada na substituição, usando vários alfabetos de substituição. Nestre trabalho iremos tratar da implementação de um cifrador/decifrador/quebrador de cifra de Vigenère.

2 Implementação

2.1 Arquitetura do Projeto

Para definir a forma que o projeto foi estruturado decidi tomar como base uma interface de usuário criada pelo RAD tool (Rapid-application development) wxFormBuilder, que gera uma inteface de usuário usando como base os objetos de inteface wxPython. Um arquivo de interface é gerado dentro da ferramenta contendo todas as classes de telas (frames) e outros objetos que compõem a interação com usuário, como botões e caixas de texto, conforme descrito na figura 1.

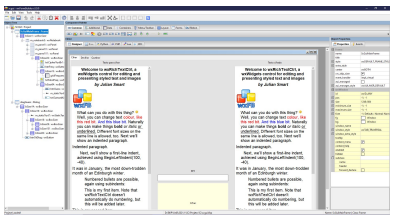

Figura 1: Projeto criado no wxFormBuilder.

O arquivo gerado pelo wxFormBuilder usado como base no projeto é o arquivo ../scgui.py, que é importado dentro do arquivo principal do projeto, ../main.py, para que seus componentes sejam consumidos.

2.2 Desenvolvimento

Dentro da implementação do projeto foi usado como base uma estrutura de dados criada de acordo com o código descrito na figura 2.

```
12  #criação da matrix de Vigenere:
13  def createMatrix():
14      a = []
15      vigenereMatrix = list(string.ascii_lowercase)
16      alphabets = list(string.ascii_lowercase)
17      start_index = 0
18      length = len(alphabets)
19
20      for i in range(length):
21          for i in range(length):
22              element_index = start_index % length
23              a.append(alphabets[element_index])
24              start_index += 1
25          vigenereMatrix = np.vstack((vigenereMatrix,a))
26          a=[]
27          start_index += 1
28      vigenereMatrix = np.delete(vigenereMatrix, 0,0)
29      return vigenereMatrix
```

Figura 2: Implementação da matriz de Vigenère.

A estrutura consiste em uma lista de listas que contém todas as 26 letras do alfabeto. Onde cada lista varia apenas a letra de início. Usando essa matriz a cifração e decifração se torna mais simples, pois conseguimos acessar a letra cifrada ou decifrada com índices.
Também foi criado uma estrutura mais básica para consulta do índice de acordo com a posição da letra do alfabeto, como demonstrado na figura 3 .

```
#criação do dicionário do alfabeto para consulta.
def createDicAlpha():
    dicAlpha ={}
    for i in list(string.ascii_lowercase):
        number = ord(i) - 97
        dicAlpha[i] = number
    return dicAlpha
```

Figura 3: Implementação de um dicionário contendo as letras do alfabeto.

2.2.1 Cifração

A cifração foi implementada de acordo com o descrito no código da figura 4. Onde podemos observar o uso da indexação da matriz criada na figura 2 em vermelho, usando como base o dicionário do alfabeto criado na figura 3 em azul.

```
#Cifrador
def cipher(self, event):
    plainText = handleText(self.txtPlainTextCi.GetValue())
    key = self.txtKeyCi.GetValue().lower().strip()
    keyStream = createKeyStream(plainText, key)
    keyStream = cycle(keyStream)
    cipherText = ''
    for char in plainText:
        if char == ' ':
            cipherText += ' '
        elif char.isnumeric():
            cipherText += char
        else:
            cipherText += vigenereMatrix[dicAlpha[char]][dicAlpha[next(keyStream)]]

    self.txtCipherTextCi.SetValue(cipherText)
```

Figura 4: Implementação do cifrador Vigenère.

Também podemos observar nos pontos marcados em rosa na figura 4 o uso de um keystream que é criado de acordo com a função createKeyStream, que usa como base o texto original não cifrado de tamanho N para criar um texto de repetição de key também de tamanho N.

2.2.2 Decifração

Para a decifração foi implementado um algoritmo que gera também um keystream baseado na key que o usuário passa e usa as mesmas estruturas descritas na figura 4 para fazer o processo de tradução/cifração. O código pode ser conferido na figura 5.

```
#decifrador
def decipher(self, event):
    decipherText = ''
    cipherText = handleText(self.txtCipherTextDe.GetValue())
    key = self.txtKeyDe.GetValue().lower().strip()
    keyStream = createKeyStream(cipherText, key)
    lstSpaces = []
    keyStream = list(keyStream)

    #Lista todos os espaços em branco no ciperText
    for pos,char in enumerate(cipherText):
        if(char == ' ' or char.isnumeric()):
            lstSpaces.append(pos)
    #Aplica todos os espaços em branco do ciperText no keyStream
    for spaceIndex in lstSpaces:
        keyStream.insert(spaceIndex, ' ')
    keyStream = ''.join(keyStream)

    for charKeyStream,i in zip(keyStream, cipherText):
        if charKeyStream != ' ':
            decipherText += vigenereMatrix[0][np.where(vigenereMatrix[dicAlpha[charKeyStream]] == i)[0][0]]
        elif i.isnumeric():
            decipherText += i
        else:
            decipherText += ' '
    self.txtDecipherTextDe.SetValue(decipherText)
```

Figura 5: Implementação do decifrador Vigenère.

2.2.3 Quebra da cifra por análise de frequências

Para conseguirmos decifrar um texto não sabendo sua chave temos que dividir a quebra em algumas etapas. O primeiro passo é inferir o tamanho da chave através do seguinte método: Listamos todas as sequências de 3 caracteres que se repetem pelo menos 2 vezes no texto e traçamos todas as distâncias desses seqências entre elas e se calcula sua divisibilidade das distâncias para chaves de 2 a 20 caracteres. O tamanho com mais repetição de ocorrências costuma ser o tamanho da chave.

Dado uma chave K que é usada para cifrar um texto, tendo conhecimento do tamanho da chave, podemos inferir que para cada caracter de K as frequências relativas do texto se mantêm, por conta das propiedades do keystream, fazendo uma troca do índice do alfabeto. Como explicitado no exemplo da figura 6:

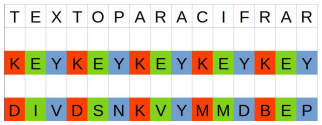

Figura 6: Exemplo de uma implementação do keystream usando uma chave KEY.

Usando como base esta key de tamanho 3, podemos calcular a distribuição de freqência iterando a enésima letra do texto cifrado de acordo com keystream. Então calcularemos a distribuição de todas as frequências que tem como base o caracter 'k' da keystream, depois calculamos para 'E', e em seguida calculamos para 'y'.
Por último analisamos as frequências e trocando os índices conseguimos inferir a letra correta do keystream. Todo o processo de quebra está explicado detalhadamento neste vídeo que pode ser acessado através deste link. A figura 7 contém a sequência a ser seguida dentro da interface de usuário para que se consiga chegar no objetivo de quebra da cifra.

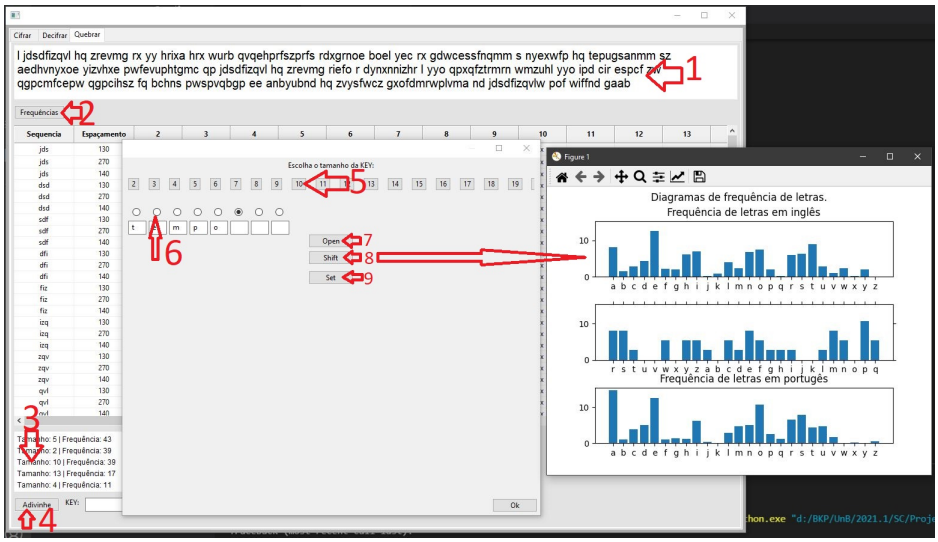

Figura 7: Passos a serem seguidos para o uso do software.