Trabalho 1 - Segurança computacional

Universidade de Brasília

Aluno: Gabriel de Sousa vieira

Matrícula: 16/0006350

Email: gsvieira98@gmail.com

1 Descrição da Implementação

1.1 Cifrador / Decifrador

Na realização deste trabalho foi feito um estudo sobre criptografia na qual a partir da cifra de césar, na qual há o uso de um número para a senha, foi criada a cifra de vigenère, que por sua vez utiliza de uma palavra utilizando-se de quaisquer letras do alfabeto.

No primeiro arquivo chamado “vegenere_cipher.py” temos a criação das funções cipher e decipher que fazem a cifragem e decifragem da mensagem, respectivamente.

vegenere_cypher.py > ...

```
1   import string, re
2   characters = list(string.ascii_lowercase) #+ list(string.digits) + [" "]
3
4   map_int_to_char = {}
5   map_char_to_int = {}
6
7   i=0
8   for char in characters:
9     map_int_to_char[i] = char
10    map_char_to_int[char] = i
11    i+=1
12
13  last_char_position = len(map_int_to_char)-1
```

Fig 1. Imagem do código implementado

Primeiramente, é criado dois mapas que são utilizados para transformar um caracter para inteiro onde ‘A’ = 0, ‘B’ = 1... ‘Z’ = 25 e o outro que volta de inteiro para caracter.

Na função apresentada na figura 2, é feita a cifragem da mensagem para um criptograma. Primeiramente é feita a retirada de todos os caracteres não alfabéticos, após isso a senha, também conhecida como chave (key) é recebe um index que volta ao inicio da chave sempre em que chegar ao final garantindo que a senha será repetida para toda a mensagem. Logo em seguida a chave e o caracter da mensagem são transformados em inteiros e somados de tal forma que ao voltarem a ser caracter a mensagem foi cifrada.

```
15  def cipher(message, key):
16      #message = message.replace("\n", " ").lower()
17      message = re.sub("[^a-zA-Z]+","", message).lower()
18      cipher_message = []
19      key_index = 0
20      list_key = list(key.strip(" "))
21      for character in message:
22          if key_index > len(list_key)-1:
23              key_index = 0
24          key_char = list_key[key_index]
25
26          key_char_value = map_char_to_int[key_char]
27          message_char_value = map_char_to_int[character]
28          new_char_value = message_char_value + key_char_value
29
30          if new_char_value > last_char_position:
31              new_char_value = new_char_value - last_char_position - 1
32          new_char = map_int_to_char[new_char_value]
33          cipher_message.append(new_char)
34          key_index+=1
35      return "".join(cipher_message)
```

Fig 2. Imagem do código implementado

Na próxima imagem é feito o processo de decifragem no qual o processo é muito semelhante a cifragem. As etapas de retirada de caracteres não alfabéticos permanecem, juntamente com a transformação da chave e do carácter do criptograma em inteiros. Porém, nesse passo é feita a subtração do criptograma pela senha, tal que o criptograma volta a ser a mensagem cifrada anteriormente.

```python
def decipher(message, key):
  message = re.sub("[^a-zA-Z]+","", message).lower()
  deciphered_message = []
  key_index = 0
  list_key = list(key.strip(" "))
  for character in message:
    if key_index > len(list_key)-1:
      key_index = 0
    key_char = list_key[key_index]

    key_char_value = map_char_to_int[key_char]
    message_char_value = map_char_to_int[character]
    new_char_value = message_char_value - key_char_value

    if new_char_value < 0:
      new_char_value = new_char_value + last_char_position + 1
    new_char = map_int_to_char[new_char_value]
    deciphered_message.append(new_char)
    key_index+=1
  return "".join(deciphered_message)
```

Fig 3. Imagem do código implementado

1.2 Ataque à cifra

No outro arquivo “breakcipher.py” são denominada as funções que realizaram o ataque a cifra de vigenere nomeadas "get_key_length" e “get_key”. A primeira função trabalha com o método chamado index of coincidence no qual determina se a distribuição dos caracteres é aleatória ou pertencem à alguma linguagem de acordo com a figura 4. Com o resultado da equação os cosets (divisões da mensagem) que tiverem a média do index of coincidence maiores são os mais prováveis de que a senha seja pertencente a aquela divisão.

$$\mathrm{IC} = \frac{1}{N(N-1)} \sum_{i=1}^{n} F_i(F_i - 1)$$

Fig 3.Índice de coincidência

Na outra função ele usa a frequência de letras do inglês de tal forma que os cosets que foram cifrados pelo mesmo caráter da chave possuem uma frequência semelhante à do inglês caso a mensagem original seja em inglês, porém deslocada. Então ao deslocar o histograma das frequências para esquerda aquele que possuir o menor chi² será o offset da que caracteriza a letra que cifrou aquela divisão do criptograma.

$$\chi^2 = \sum_{i=1}^{n} \frac{(f_i - F_i)^2}{F_i}$$

Fig 4.Fórmula de Chi²

```python
def get_key_length(ciphertext):
  possible_keys = []
  #corrige espaços e outros caracteres não alphanumericos
  ciphertext_treated = re.sub("[^a-zA-Z]+","", ciphertext).lower()
  file = open("intermediario.txt","w")
  file.write(ciphertext_treated)
  #divide a messagem em cosets de tamanho 2 a 10 para calculo de IC Bug chave = 8
  for i in range(2,11):
    cipher_list = list(ciphertext_treated)
    cipher_list_size = len(cipher_list)
    sublists = []
    IC_list = [] #testar media de IC
    IC_sum, IC = 0.0, 0.0
    #gera matrix de coset com tamanho entre 2 e 10
    for _ in range(i):
      sublists.append([])
      IC_list.append([])
    #separa a string nos cosets removendo o primeiro elemento da lista e adicionando ao sublist
    for j in range(cipher_list_size):
      char = cipher_list.pop(0)
      (sublists[j%i]).append(char)
    #test
    """ for p in range(i):
      print(sublists[p]) """
    #calcula o IC
    for k in range(i):
      N = len(sublists[k])
      freq = collections.Counter(sublists[k])
      freq_sum = 0.0
      for elem in string.ascii_lowercase:
        freq_sum += freq[elem] * (freq[elem]-1)
      if N==0:
```

Fig 5. Imagem do código implementado

2 Descrição da execução

2.1 Cifragem e decifragem

Para fazer a cifragem e decifragem é preciso escrever na linha de comando ~$ python vegenere_cipher.py nome_da_chave

Ele usará o texto no arquivo “input.txt” e retornará o cifrado e decifrado em "ciphered.txt" e "deciphered.txt"

2.2 Ataque

Para fazer o ataque é preciso escrever na linha de comando ~$ python breakcipher.py nome_do_arquivo.txt (tamanho da chave-opcional)

O retorno da chave será no terminal