Trabalho 1:
Cifra de Vigenère

Pedro Lísias Viana Arcoverde Alves, 19/0036559
Marcelo Piano Patusco Santiago, 20/0049496

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0201 - SEGURANÇA COMPUTACIONAL - T02 (2022.1 - 24N12)
Prof. João José Costa Gondim

1. Introdução

Este projeto é composto por duas partes na primeira parte iremos programar um cifrador que recebe uma senha e uma mensagem que é cifrada segundo a cifra de Vigenère, gerando um criptograma. E um decifrador que recebe uma senha e um criptograma que é decifrado segundo a cifra de Vigenère, recuperando uma mensagem que foi cifrada. Na segunda parte, iremos programar uma ferraenta capaz de receber duas mensagens cifradas (uma em português e outra em inglês) com senhas diferentes e somente a partir delas e de tabelas de frequência das línguas portuguesa e inglesa, recuperar a senha geradora do keystream usado na cifração e então decifra-las.

2. Descritivo do Código

2.1. Primeira parte

Na linha 14 definimos o alfabeto como sendo uma string com todas os caracteres inseridos nele e logo após esse alfabeto(na linha 17 e 18), criamos 2 dicionários `(letra_para_numero e numero_para_letra)` que tem como chave caracteres e números, respectivamente. Para facilitar nossa implementação, estudamos maneiras de tratar a mensagem a ser cifrada (msg) a fim de tornar a aplicação do código mais simples, assim, da linha 23 até 26 fizemos a string mensagem ficar mais enxuta, sem espaços, apenas com caracteres minúsculos, sem acentuação, sem pontuação e retiramos os números da mensagem.

Dando continuidade, a primeira parte referente a cifração e decifração do algoritmo se inicia na linha 35 onde iteramos a lista de listas, e em seguida, iteramos novamente letra por letra das listas com lenght = lenght da chave. Dentro dessa segunda iteração criamos a variável `msg_cifrada` aos poucos, concatenando na variável a letra referente a operação de cifração de Vigenère (Pi + Ki) % 26, em que Pi é o valor numérico de uma letra da mensagem a ser cifrada, e Ki é o valor numérico da chave com índice correspondente. Após as iterações mencionadas, iremos gerar uma `msg_cifrada`

```
# separa a mensagem em partes do tamanho da chave
    msg_cifrada = ''
    split_msg = [msg[i:i + len(chave)] for i in range(0, len(msg), len(chave))]

# converte a mensagem e a chave em numeros, cifra por Vigenère a mensagem
    for each_split in split_msg:
        i = 0
        for letra in each_split:
            numero = (letra_para_numero[letra] + letra_para_numero[chave[i]]) % len(alfabeto)
            msg_cifrada += numero_para_letra[numero]
            i += 1

# retorna a mensagem cifrada
    return msg_cifrada
```

Figure 1. Função cifradora.

A seguir, realizamos a parte da decifração da mensagem cifrada que é representada pela função decifra(). Essa função realiza etapas semelhantes a e cifração, porém, na hora de fazer a operação a formula fica (Pi - Ki) % 26, o que significa que estamos realizando a operação inversa e obtendo a decifração da mensagem.

```
# converte a mensagem e a chave em numeros
    for each_split in split_msg_cifrada:
        i = 0
        for letra in each_split:
            numero = (letra_para_numero[letra] - letra_para_numero[chave[i]]) % len(alfabeto)
            msg += numero_para_letra[numero]
            i += 1

# retorna a mensagem decifrada
    return msg
```

Figure 2. Função decifradora.

2.2. Segunda parte

Nesta parte do trabalho, que é a de uma ferramenta de ataque à cifra de Vigenère, representada pela função ataque na linha 70, iniciamos definindo dois dicionários de frequência `(en_freq, pt_freq)`, que tem como chave um caractere ligado a um valor que representa sua frequência no inglês e português, respectivamente. Em seguida, começamos o processo de encontrar um tamanho provável para a chave em questão (linha 93). O processo inicia com a definição de uma lista de espaçamento, uma limitação do tamanho da possível chave igual a 20 (para evitar possíveis erros) e uma tolerância também definida para que não assumamos um novo valor para a chave sem um novo MMC consideravelmente maior q o anterior e ocorram erros. Dando continuidade a implementação, na primeira iteração dessa etapa, temos como objetivo agrupar o código cifrado em grupinhos de 3 caracteres, e com esses grupinhos iterar novamente para cada próximo grupinho de 3 caracteres, e a cada igualdade dos dois grupos de 3 caracteres, fomos preenchendo a lista de espaçamento apenas com o valor de suas distancias, criando uma lista o seguinte modelo por exemplo: espaçamento = [3,6,3,6,6]. Em seguida (linha 102), demos inicio ao processo final dessa etapa, onde com base na lista de espaçamento fizemos um cálculo de qual seria o valor de maior incidência na lista e o colocamos como sendo o `tam_chave`, ou seja, o possível tamanho da chave. Vale ressaltar que as variáveis descritas acima (`max_chave e tolerância`) foram utilizadas aqui para ajustar e refinar o tamanho da chave provável de forma estatística.

```
# frequencias em cada idioma para análise
    en_freq = [
            ('a', 8.167), ('b', 1.492), ('c', 2.782), ('d', 4.253),
            ('e', 12.702), ('f', 2.228), ('g', 2.015), ('h', 6.094),
            ('i', 6.966), ('j', 0.153), ('k', 0.772), ('l', 4.025),
            ('m', 2.406), ('n', 6.749), ('o', 7.507), ('p', 1.929),
            ('q', 0.095), ('r', 5.987), ('s', 6.327), ('t', 9.056),
            ('u', 2.758), ('v', 0.978), ('w', 2.360), ('x', 0.150),
            ('y', 1.974), ('z', 0.074)]

    pt_freq = [
            ('a', 14.63), ('b', 1.04), ('c', 3.88), ('d', 4.99),
            ('e', 12.57), ('f', 1.02), ('g', 1.30), ('h', 1.28),
            ('i', 6.18), ('j', 0.40), ('k', 0.02), ('l', 2.78),
            ('m', 4.74), ('n', 5.05), ('o',10.73), ('p', 2.52),
            ('q', 1.20), ('r', 6.53), ('s', 7.81), ('t', 4.34),
            ('u', 4.63), ('v', 1.67), ('w', 0.01), ('x', 0.47),
            ('y', 0.01), ('z', 0.47)]
```

Figure 3. Tabela de frequências das letras em português e em inglês.

```python
# encontrar o comprimento provavel da chave
    espacamento = []
    max_chave = 20
    tolerancia = 10

    for i in range(len(msg_cifrada) - 2):
        tmp = msg_cifrada[i] + msg_cifrada[i+1] + msg_cifrada[i+2]
        for j in range(3, len(msg_cifrada) - 2 - i):
            if tmp == msg_cifrada[i+j] + msg_cifrada[i+j+1] + msg_cifrada[i+j+2]:
                espacamento.append(j)
                break

    if max_chave > len(msg_cifrada): max_chave = len(msg_cifrada)
    max_mmc = 0
    tam_chave = 0
    for i in range(2, max_chave + 1):
        counter = 0
        for n in espacamento:
            if n % i == 0:
                counter += 1
        if counter + tolerancia > max_mmc:
            tam_chave = i
            max_mmc = counter

    print('\nTAMANHO PROVAVEL DA CHAVE: ' + str(tam_chave))
```

Figure 4. Descobrindo o tamanho provável da chave.

Dando continuidade ao ataque, na linha 125, definimos uma tabela que é preenchida, com listas de caracteres que serão utilizadas para realizar uma analise de frequência de para cada caractere da chave. Isso é possível graças a etapa anterior que nós deu um estimativa do tamanho a chave. Em seguida, iteramos entre as listas/grupos de caracteres da lista mencionada, e, para cada grupo, realizamos 26 iterações, sendo que em cada uma iteramos para fazer o grupo 'pular' i casas para o lado e depois mais uma iteração para armazenar na variável dif a diferença entre a probabilidade da letra e a probabilidade referente a lista de probabilidades da língua. Dito isso, calculamos qual é a menor diferença e assim conseguimos chegar a um possível numero de pulos para alcançar o possível caractere da chave naquela posição.

```
# análise de frequencia de cada grupo utilizando as rfequecias fornecidas pela wikipedia
    freq = en_freq if idioma == 0 else pt_freq
    chave_provavel = []

    for grupo in tabela:
        chave_atual = ''
        min_dif = 10000
        for i in range(26):
            aux = []
            for c in grupo:
                if ord(c) - i >= ord('a'):
                    aux.append(chr(ord(c) - i))
                else:
                    aux.append(chr(ord('a') + 26 - i + (ord(c) % ord('a'))))
            dif = 0
            for c in aux:
                prob = [letra for letra in freq if letra[0] == c][0][1] / 100
                dif += abs(prob - (aux.count(c) / len(aux)))
            dif = dif / len(aux)
            if dif < min_dif:
                min_dif = dif
                chave_atual = chr(ord('a')+i)
        chave_provavel.append(chave_atual)
```

Figure 5. Análise das frequências.

Logo apos isso, definimos uma parte para a iteração com o terminal, possibilitando o usuário a realizar todas as opções requisitadas pelo professor. No terminal é possível para o usuário, realizar a cifração de um texto, a decifração e o ataque a uma cifra, sendo ele de forma padrão ou ainda podendo ser editados os valores do tamanho máximo da chave e da tolerância de comparação para a mesma.

3. Referências

[1] Wikipédia da Cifra de Vigenère: wikipedia.org/Vigenère
[2] Guia para o ataque a cifra:crypto.interactive-maths.com/kasiski-analysis