Relatório - Cifra de Vigenère

Ana Sofia Schweizer Silvestre - 200014382

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)

1. Introdução

Esse trabalho visa implementar a codificação, decodificação e ataque à Cifra de Vigenère. A Cifra de Vigenère é um método de criptografia que aplica várias Cifras de César em que seus deslocamentos são baseados nas diferentes letras das chaves [Wikipedia 2023]. As implementações foram feitas com base nos materiais auxiliares da matéria e em conteúdos online.

2. Codificação

A codificação implementada segue os segue os seguintes passos: descobrir a posição no alfabeto de cada letra do texto e de cada letra na chave, somá-las e aplicar o módulo 26. O código abaixo implementa essa função. A posição resultada determinará a letra cifrada.

```
def table(letter1, letter2, big=False):

    idx1 = alphabet.find(letter1)
    idx2 = alphabet.find(letter2)
    return alphabet[(idx1+idx2)%26] if not big
else ALPHABET[(idx1+idx2)%26]

def get_cipher(text, key):

    cipher = ''

    j = 0
    for i in range(len(text)):
        big = False
        if text[i].lower() in alphabet:
            if text[i] in ALPHABET: big=True
            cipher += table(text[i].lower(), key[j%len(key)]
.lower(), big)
            j += 1
        else:
            cipher += text[i]

    return cipher
```

3. Decodificação

O processo de decodificação é semelhante, dada uma chave, a decodificação é feita subtraindo da posição no alfabeto de cada letra no texto cifrado, a posição no alfabeto da letra da chave na respectiva posição e aplicando o módulo 26.

```
def get_plaintext(text, key):

    plaintext = ''

    j = 0
    for i in range(len(text)):
        if text[i].lower() in alphabet:
            idx2 = alphabet.find(key[j%len(key)])
            if text[i] in ALPHABET:
                idx1 = ALPHABET.find(text[i])
                plaintext += ALPHABET[(idx1-idx2)%26]
            else:
                idx1 = alphabet.find(text[i])
                plaintext += alphabet[(idx1-idx2)%26]
            j += 1
        else:
            plaintext += text[i]

    return plaintext
```

Tando na codificação, quanto na codificação há o suporte para letras maiúsculas, o que torna o código um pouco mais verboso.

4. Ataque

Para a realização do ataque são executados dois passos: o primeiro é descobrir o tamanho da chave e o segundo é descobrir os caracteres dela.

4.1. Descobrindo o tamanho da chave

Para a checagem do tamanho da chave foi feita a checagem de repetição de trigramas para cada trigrama contido no texto. Assim que é encontrada uma repetição, a distância entre os trigramas é medida e então são anotados os fatores dessa distância. Os fatores que foram anotados mais vezes são indicativos dos possíveis tamanhos da chave e o usuário pode testar diferentes tamanhos de chaves [Wikibooks 2022] [Veitch 2014].

```
def find_keysize(text):

    for i in range(len(text)-2):
        seq = text[i:i+3]

        for j in range(i+3, len(text)-2):
            if text[j:j+3] == seq:
                dist = j-i
                print(seq, dist)
                for k in range(2,21):
                    if dist%k==0:
                        factors[k-2] += 1
                break
```

```
    print(f'Tamanhos de chaves e sua quantidade de fatores
encontrada:')

    for i in range(len(factors)):
        print(f'\t{i+2} - {factors[i]}')

    selected = int(input('Selecione o tamanho da chave dese
jada: '))

    return selected
```

4.2. Análise de Frequência

Assim que o tamanho da chave é definido, deve-se fazer a análise de frequência para saber seus caracteres. A análise consiste em comparar a frequência de letras da mensagem com a frequência das letras da língua em que essa mensagem está escrita.

Essas letras são selecionadas por intervalo de acordo com o tamanho da chave. Dada uma chave de tamanho x, para cada posição i dessa chave, deve-se analisar a frequência do conjunto de letras cujas posições são múltiplas de i. Ao final dessa comparação é possível descobrir o deslocamento aplicado nesse conjunto de letra e então, os caracteres da chave.

```
def get_key(cipher, keysize, english=True):

    cipher = ''.join([i for i in cipher.lower() if i in alp
habet])
    key = ''

    if english:
        lang_freq = freq_eng
    else:
        lang_freq = freq_port

    for i in range(keysize):

        distr = ''.join([cipher[j] for j in range(i,len(cip
her),keysize)])
        idx = freq_analysis(distr, lang_freq)
        key += alphabet[idx]

    return key
```

A análise de frequência é feita realizando o produto interno entre as frequências da língua e a frequência da mensagem, o deslocamento que tiver o maior produto interno é o que apresenta a maior similaridade [Stange 2020].

```
def freq_analysis(letters, lang_freq):
```

```
    freq = text_freq(letters)
    dif = []
    for j in range(26):
        dif.append(sum([lang_freq[i]*freq[i] for i in range(26)]))
        freq.append(freq.pop(0))

    return dif.index(max(dif))
```

5. Conclusão

Apesar da Cifra de Vigenère ser demasiada complexa, ela possui muita vulnerabilidade nos casos em que as chaves utilizadas para a codificação são curtas e repetitivas [Wikipedia 2023], por isso deve-se optar por chaves grandes e variadas. Conclui-se que, apesar de ser uma técnica antiga, a Cifra de Vigenère pode ser uma técnica de criptografia segura e eficiente se sua chave for longa e sua implementação for robusta.

Referências

[Stange 2020] Stange, K. (2020). Cryptanalysis of vigenere cipher: not just how, but why it works. https://www.youtube.com/watch?app=desktop&v=QgHnr8-h0xI&ab_channel=ProofofConcept.

[Veitch 2014] Veitch, B. (2014). Cryptography - breaking the vigenere cipher. https://www.youtube.com/watch?v=P4z3jAOzT9I&ab_channel=BrianVeitch.

[Wikibooks 2022] Wikibooks (2022). Cryptography/breaking vigenère cipher. https://en.wikibooks.org/wiki/Cryptography/Breaking_Vigenre_cipher.

[Wikipedia 2023] Wikipedia (2023). Cifra de vigenère. https://pt.wikipedia.org/wiki/Cifra_de_Vigenre.