Trabalho de Implementação 1
Cifra de Vigènere

Gabriel Martins de Almeida, 190013371
Pedro de Tôrres Maschio, 190018763

[1]CIC0201 - Segurança Computacional - T01

1. Introdução

A Cifra de Vigènere foi descrita pela primeira vez em 1553 pelo criptólogo italiano Giovan Battista Bellaso. Apesar de razoavelmente simples, a cifra somente foi quebrada pela primeira vez em 1863. A cifra recebeu esse nome por ter sido incorretamente atribuída ao criptógrafo francês Blaise de Vigenère.

De modo geral, a ideia segue da seguinte forma, primeiro é associado um valor a cada letra do alfabeto que será utilizado, devemos considerar o módulo do tamanho desse conjunto para qualquer valor que ultrapasse o alfabeto. Então, será definida uma chave de tamanho inferior ao texto original, a partir do texto inicial devemos deslocar o índice de cada letra i vezes para encontrar o texto cifrado, no qual i é o valor da letra da chave na posição i. A cifra de Vigènere funciona como múltiplas cifras de César, no qual cada letra do texto original é cifrado por uma letra diferente da chave.

Por exemplo, na Figura 1, temos um exemplo do processo de cifração, com uma mensagem e uma chave, tendo também os índices das letras representados. Observamos que a mensagem é ”MENSAGEM EXEMPLO” e que a chave é ”ABC”. Como a chave tem tamanho inferior à mensagem, ela se repete. Então, cada índice da mensagem original é somado aos índices da chave, e depois retirado o módulo do tamanho do alfabeto. Ao final, esse índice obtido é o caractere da mensagem cifrada.

	12	4	13	18	0	6	4	12		4	23	4	12	15	11	14
Mensagem	M	E	N	S	A	G	E	M		E	X	E	M	P	L	O
	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0
Chave	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A
	12	5	15	18	1	8	4	13		6	23	5	14	15	12	16
Mensagem cifrada	M	F	P	S	B	I	E	N		G	X	F	O	P	M	Q

Figure 1. Exemplo de processo de cifração de mensagem

2. Cifrador e Decifrador

Na Figura 2 a função *cipher_or_decipher* recebe a mensagem e realiza algumas operações, como verificar o tamanho da chave e mensagem, transformar os caracteres de ambas para minúsculo e garantir que a mensagem não está vazia. Esta função é utilizada tanto para cifração quanto para decifração, a depender do parâmetro *operation* recebido.

```
def cipher_or_decipher(self, message, key, operation):
    message_length = len(message)
    key_length = len(key)
    message = ''.join(character.lower() for character in message if character.isascii())
    key = key.lower()

    if message_length <= 0 or key_length <= 0 or message_length < key_length:
        return "NULL_MSG"

    cipher_text = ""
    num_special = 0
```

Figure 2. Código Cipher e Decipher - parte 1

Na Figura 3 temos um laço de repetição que percorrerá todos os caracteres da mensagem. Sendo assim, o primeiro passo é realizar uma normalização, para que a letra ”a” seja o índice 0, em seguida verifica se estamos trabalhando com uma operação de cifração ou decifração para então ser realizada uma análise cujo a função é verificar se existe algum caractere com um valor fora do intervalo do alfabeto utilizado. Então, os caracteres válidos serão inseridos na mensagem de forma cifrada ou decifrada de acordo com a chave. Caso algum caractere esteja fora da faixa numérica especificada, será utilizada a variável *special_character* como uma *flag* para indicar que o caractere atual não é valido, está condição é utilizada para informar que os caracteres inválidos devem ser inseridos de forma literal na mensagem e que a chave atual deve ser reutilizada para o próximo símbolo.

```
    for i in range(len(message)):
        caracter_ord = ord(message[i]) - ord('a')
        special_character = False

        if operation == 'C':
            if message[i].isalpha():
                ciphered_character = chr((caracter_ord + ord(key[(i-num_special)%len(key)]) - ord('a'))%26 + ord('a'))
            else:
                special_character = True
        else:
            if message[i].isalpha():
                ciphered_character = chr((caracter_ord - (ord(key[(i-num_special)%len(key)]) - ord('a')))%26 + ord('a'))
            else:
                special_character = True

        if not special_character:
            cipher_text += ciphered_character
        else:
            num_special += 1
            cipher_text += message[i]

    return cipher_text
```

Figure 3. Código Cipher e Decipher - parte 2

3. Ataque por análise de frequência

Na Figura 4 é possível ver a definição da função *find_matchings*, que tem como parâmetro a mensagem base, esta operação faz comparações entre os caracteres da mensagem original e a mesma deslocada N vezes, com N variando de 1 até o tamanho da

mensagem. Sendo assim, esta operação tem como objetivo encontrar a quantidade de combinações entre as letras a cada deslocamento, a função retornará um vetor contendo a quantidade de combinações a cada deslocamento, este vetor será utilizado para encontrar um possível tamanho para a chave.

```python
def find_matchings(self, cipherText):
    matching_numbers = []

    for i in range(1, len(cipherText)):
        match_count = 0
        temp = i
        for j in range(len(cipherText)):
            if cipherText[j] == cipherText[temp] and cipherText[j].isalpha() and cipherText[temp].isalpha():
                match_count += 1
            temp += 1
            if temp >= len(cipherText):
                break
        matching_numbers.append(match_count)

    return matching_numbers
```

Figure 4. Código Ataque por análise - parte 1

Na Figura 5 é possível ver a função $shift\_left$, que realiza um deslocamento à esquerda na lista de acordo com o parâmetro "numShifts". A função $groups$ irá retornar uma matriz com um tamanho de (tamanho da chave $\times$ tamanho do alfabeto). Sendo assim, como descrito anteriormente a cifra de Vigènere é composta por várias cifras de César, logo esta função irá explorar esse comportamento dividindo a mensagem em grupos e se a chave estiver correta, cada grupo terá a frequência das letras da linguagem utilizada com um possível deslocamento.

```python
def shift_left(self, lista, numShifts):
    return lista[numShifts:] + lista[:numShifts]

def groups(self, cipherText, key_length):
    frequencies = []
    for i in range(key_length):
        frequencies.append([0]*26)

    for i in range(len(cipherText)):
        if cipherText[i].isalpha():
            frequencies[i%key_length][ord(cipherText[i])-ord('a')] += 1

    for i in range(key_length):
        for j in range(26):
            frequencies[i][j] /= 26

    return frequencies
```

Figure 5. Código Ataque por análise - parte 2

A função $get\_max\_frequency\_letter$, apresentada na Figura 6, tem como parâmetros a frequência base de um idioma e a frequência das letras de determinado grupo da chave, a função retornará o caractere associado ao índice do deslocamento que contém a maior combinação com a frequência base do idioma. A função $frequency\_analysis$, apresentada na Figura 7, receberá como parâmetro o idioma utilizado para montar uma base de frequência conforme a linguagem especificada no programa, também receberá um grupo, com o tamanho da chave, contendo o índice da frequência de cada letra por grupo. Logo, será chamada a função $get\_max\_frequency\_letter$ para cada um dos grupos e esta função retornará uma letra da chave para cada grupo, gerando uma chave para mensagem.

```
def get_max_frequency_letter(self, baseFrequencies, frequencies):
    letter = 0
    prev_max = 0
    for i in range(26):
        res = 0
        for j in range(26):
            res += baseFrequencies[j]*frequencies[j]

        if res >= prev_max:
            prev_max = res
            letter = i

        frequencies = self.shift_left(frequencies, 1)
    return chr(letter+ord('a'))
```

Figure 6. Código Ataque por análise - parte 3

```
def frequency_analysis(self, groupFrequencies, language):
    key_len = len(groupFrequencies)

    baseFrequencies = []
    if language == 'pt-BR':
        with open('freq_pt_br.txt') as f:
            baseFrequencies = f.readlines()
    else:
        with open('freq_en.txt') as f:
            baseFrequencies = f.readlines()

    baseFrequencies = list(map(float, baseFrequencies))

    key = ""
    for i in range(key_len):
        letter = self.get_max_frequency_letter(baseFrequencies, groupFrequencies[i])
        key += letter
    return key
```

Figure 7. Código Ataque por análise - parte 4

A função $break\_ciphertext$ receberá obrigatoriamente a mensagem cifrada e também opcionalmente um idioma e o tamanho da chave, realizará algumas operações, como transformar o texto em minúsculo para evitar comportamento indesejado na decifração. Então, encontrará um possível tamanho para a chave e a utilizará caso não tenha recebido a chave, então chamará as funções $groups$ para montar os grupos com o tamanho da chave, a $frequency\_analysis$ para encontrar a chave e utilizará a função de decifração descrita na seção dois com a chave e mensagem para encontrar a possível mensagem descriptografada.

```
def break_ciphertext(self, cipherText, language='pt-BR', key_len=None):
    cipherTextTemp = cipherText
    cipherText = ''.join(character.lower() for character in cipherText if (character.isascii() and character.isalpha()))
    matchingsList = self.find_matchings(cipherText)

    if key_len == None:
        key_len =  matchingsList.index(max(matchingsList[:10]))+1 # 10 is the max key length we are considering
    testGroups = self.groups(cipherText, key_len)
    key = self.frequency_analysis(testGroups, language)

    c = Cipher()
    return {'key': key, 'message': c.cipher_or_decipher(cipherTextTemp, key, 'D')}
```

Figure 8. Código Ataque por análise - parte 5

4. Conclusão

A realização deste trabalho foi bastante desafiadora no sentido de compreender bem o funcionamento da cifra de Vigènere e no tratamento de casos excepcionais no código (espaços, acentuação etc.). Também foi bastante engrandecedora pois conseguimos desenvolver nossas habilidades de programação e de implementação de algoritmos.