Trabalho 1 - Segurança Computacional
Cifra de Vigenere

Maria Eduarda M. de Holanda
Dep. de Ciência da Computação
Universidade de Brasília
Brasília, DF, Brazil
Matrícula 19/0043725
eduarda.holanda@aluno.unb.br

Guilherme Silva Souza
Dep. de Ciência da Computação
Universidade de Brasília
Brasília, DF, Brazil
Matrícula 19/0014059
guilherme-silva.gs@aluno.unb.br

***Resumo*—A criptografia é a arte da comunicação segura na presença de partes não autorizadas. Uma da mais famosa técnicas de criptografia é a cifra de Vigenère, que envolve o uso de uma palavra chave para codificar a mensagem por meio de um processo de substituição polialfabética. Embora já tenha sido considerada inquebrável, a cifra de Vigenère se mostrou vulnerável a certos ataques. Nesse trabalho vamos falar como a cifra funciona e como quebra-lá.**

***Palavras chaves*—criptografia, cifra de Vigenère, segurança computacional**

I. Introdução

A criptografia é a prática e o estudo de técnicas de comunicação segura na presença de terceiros. Envolve a criação e análise de protocolos que impossibilita terceiros de ler mensagens privadas. Uma das técnicas utilizadas é a cifra de Vigenère, que é um método de criptografar um texto alfabético. Ela usa uma forma bem simples da substituição polialfabética.

Na cifra de Vigenère, uma mensagem é criptografada usando uma palavra chave secreta para determinar a substituição das letras. A palavra chave é repetida várias vezes para gerar uma palavra chave tão longa quanto a mensagem a ser criptografada. As letras da mensagem são deslocadas por um certo número de espaços com base na letra correspondente na palavra chave. A cifra de Vigenère utiliza várias cifras de César para determinar esse deslocamento.

A cifra de Vigenère foi considerada inquebrável por séculos, mas no século 19, Charles Babbage e Friedrich Wilhelm [1] desenvolveram técnicas para decifrá-la. Apesar disso, a cifra continua sendo utilizada em alguns sistemas de comunicação devido a sua simplicidade e a facilidade de sua implementação.

Esse presente trabalho esta divido em seções. Na metodologia falaremos quais ferramentas foram utilizadas para a implementação da cifra. Na seção cifra de Vigenére discutimos sobre a implementação proposta, apresentando a criptografia e a descriptografia de uma mensagem, e como é quebrada essa criptografia se a chave para descriptografar for desconhecida. Finalizamos com a conclusão onde diremos algumas vantagens e desvantagens da cifra de Vigenère.

II. Metodologia

Para implementar a cifra utilizamos a linguagem de programação *Python* e para o ataque usamos a técnica da frequência de letras [2], para tal pegamos as frequências relativas das letras em português e em inglês. Foram usados alguns arquivos para teste de criptografar, descriptografar e ataque da cifra. Toda a implementação e arquivos de testes está disponível no Github [3] dos autores.

III. Cifra Vigenère

Nessa seção, detalharemos melhor como foi implementada a nossa versão da Cifra de Vigenère, começando pela criptografia e descriptografia utilizando uma palavra-chave conhecida e em seguida o método utilizado para decifrar uma mensagem encriptada sem conhecer a palavra-chave.

A. Criptografia e Descriptografia

Como foi explicada na seção I, sabemos que a cifra de Vigenère se baseia no deslocamento das letras do texto ditados por uma *palavra-chave*. Sabemos também que o alfabeto possui apenas 26 letras, por isso o deslocamento máximo seria 25, pois um deslocamento de 26 seria o mesmo de não deslocar. Por isso, dado C_i a letra criptografada, P_i a letra original e K_i a letra da palavra-chave, podemos representar a criptografia algebricamente da seguinte forma:

$$C_i \equiv P_i + K_i \mod 26 \tag{1}$$

De forma análoga, a descriptografia é feita inversamente, de modo a retirar o deslocamento feito na criptografia, e podemos representá-la algebricamente como:

$$P_i \equiv C_i - K_i + 26 \mod 26 \tag{2}$$

Em nossa implementação, recebemos como entrada da função a palavra-chave, o texto e a opção (criptografar ou descriptografar) e retornamos o texto conforme a opção desejada. Além disso, é importante notar que a palavra-chave pode ser menor ou maior que o nosso texto, por isso chamamos uma função *key_pattern* para tratar isso e transformar a chave do tamanho do texto, podendo truncá-la ou repeti-la.

```
def crypt_decrypt(self, key, text, option):
    if len(text) <= 0 or len(key) < 2:
        raise ValueError('Tamanho do texto ou da
        ↪  chave invalida')
    text = text.upper()
    new_key = self.key_pattern(key, text)
    text_ans = ""
    i = 0
    for letter in text:
        if letter not in self.alphabet:
            text_ans += letter
        elif option == 'C':
            text_ans += self.alphabet[((ord(letter)
            ↪  + ord(new_key[i])) % 26)]
            i += 1
        else:
            text_ans += self.alphabet[(ord(letter) -
            ↪  ord(new_key[i]) + 26) % 26]
            i += 1
    return text_ans
```

B. *Descriptografia sem a palavra chave*

Para conseguirmos decifrar um texto encriptado com a Cifra de Vigenère sem a palavra chave existem algumas técnicas diferentes, mas optamos por utilizar um método que envolve o índice de coincidência e a frequência das letras.

Primeiramente, é preciso encontrar um possível tamanho para a palavra chave. Para isso, percorremos o texto e procuramos *trigramas* repetidos. Em seguida, checamos com qual frequência esses trigramas se repetem no texto, considerando a operação modular entre 2 e 20, pois somente nos interessamos por tamanhos de chave desse comprimento. Por fim, o tamanho com mais marcações será o melhor candidato para o tamanho de chave, porém, como não é uma técnica tão precisa, deixamos que o usuário possa escolher também o valor de chave que preferir.

```
def key_size(self, text):
    text = text.upper()
    text = self.clean_text(text)
    spacing = []

    for i in range(len(text)-2):
        trigram = text[i] + text[i+1] + text[i+2]
        for j in range(i+1, len(text)-2):
            aux = text[j] + text[j+1] + text[j+2]
            if aux == trigram:
                spacing.append((trigram, j-i))

    spacing = list(set(spacing))
    freq_mod = {}

    for _, space in spacing:
        for i in range(2,21):
            if space % i == 0:
                freq_mod[i] = freq_mod.get(i, 0) + 1

    key_size = (0,0)
    freq_mod = dict(sorted(freq_mod.items(),
    ↪  key=lambda item: item[0]))

    print("Tamanhos de chave possiveis: ")
    for key, value in freq_mod.items():
        if value >= key_size[1]:
            key_size = (key,value)
        print("Tamanho:", key, "-- Quantidade:",
        ↪  value)

    print("Tamanho provavel da chave: ",
    ↪  key_size[0])
    ans = input("Voce deseja continuar com esse
    ↪  tamanho da chave? (S/N)\n>>> ")
    if ans.lower() == 'n':
        aux = int(input("Digite o tamanho da chave
        ↪  desejado (entre 2 e 20).\n>>> "))
        while aux > 20 or aux < 2:
            aux = int(input("Tamanho Invalido.
            ↪  Digite um numero entre 2 e 20.\n>>>
            ↪  "))
        return aux

    return key_size[0]
```

A próxima etapa consiste em descobrir as letras da palavra chave, e utilizamos a frequência das letras relativas das letras de inglês e português. Para isso, para cada letra da palavra-chave checamos a frequência das letras do texto, isto é, se nossa palavra chave possui 5 letras e queremos descobrir a primeira letra, iremos ver a frequências das letras $1, 6, 11, 16...$ do texto. E para de fato encontrar a letra da palavra chave, iremos percorrer o vetor das probabilidades de cada letra aparecer num texto de cada língua e procurar a diferença mínima possível. Vale acrescentar que deixamos que o usuário escolha qual língua ele quer utilizar (Inglês ou Português).

```
def discover_letter(self, probability, language):
    letter = ''
    tot_diff = 1e9

    for i in range(26):
        aux_diff = 0
        for j in range(26):
            if language == 'EN':
                aux_diff += abs(probability[(i+j) %
                ↪  26] -
                ↪  self.eng_probabilities[j]);
            else:
                aux_diff += abs(probability[(i+j) %
                ↪  26] - self.pt_probabilities[j]);
        if aux_diff < tot_diff:
            letter = self.alphabet[i]
            tot_diff = aux_diff

    return letter

def break_keyword(self, key, text, language):
    text = self.clean_text(text)
    keyword = ""
```

```
    for i in range(key):
        total = 0
        frequency_text = {}
        probability_text = []
        for j in range(i, len(text), key):
            frequency_text[text[j]] =
            ↪  frequency_text.get(text[j], 0) + 1
            total += 1
        for letter in self.alphabet:
            aux = frequency_text.get(letter,
            ↪  0)/total*100
            probability_text.append(aux)
        keyword +=
        ↪  self.discover_letter(probability_text,
        ↪  language)

    return keyword
```

IV. Conclusão

A cifra de Vigenère não é totalmente segura e pode ser quebrada facilmente por meio de diferentes técnicas de análise estatística. Também é suscetível a ataques com base em palavras ou padrões repetidos no texto, tornando simples determinar o tamanho correto da chave. Apesar disso, a cifra tem algumas vantagens, como ser fácil de entender e implementar e ser relativamente rápida para criptografar e descriptografar. Além disso a cifra é boa para ser utilizada como estudo para aprender as técnicas de criptografia e quebra de algoritmos.

Referências

[1] T. Schrödel, "Breaking short vigenère ciphers," *Cryptologia*, vol. 32, no. 4, p. 334–347, 2008.

[2] Wikipedia, "Frequência de letras — Wikipedia, the free encyclopedia." http://pt.wikipedia.org/w/index.php?title=Frequ%C3%AAncia%20de%20letras&oldid=64434194, 2022. [Online; accessed 18-December-2022].

[3] G. S. Souza and M. E. M. de Holanda, "Vigenère chiper." https://github.com/gss214/cifra-de-vigenere, 2022.