Relatorio: Cifra de Vigenère

Igor Bastos de Oliveira, 18/0122177

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

1. Introdução

Este projeto foi desenvolvido com o propósito de explorar e implementar a codificação utilizando o método de Vigenère, uma técnica clássica de criptografia que tem sido fundamental para a segurança de comunicações ao longo dos anos. Além disso, também nos dedicaremos a investigar as estratégias de ataque a essa cifra, fazendo uso do método de Friedman, com o objetivo de compreender suas vulnerabilidades e possíveis contramedidas.

A cifra de Vigenère é reconhecida por sua capacidade de proteger mensagens por meio da substituição de letras em um texto original, utilizando uma chave. Sua versatilidade e eficácia tornam-na um tópico de estudo fundamental na área de criptografia clássica, oferecendo uma compreensão valiosa dos princípios subjacentes.[4]

O método de Friedman, por outro lado, é uma ferramenta crucial para a análise da segurança da cifra de Vigenère. Ao simular ataques por meio de análises de frequência de letras e estatísticas, pretendemos explorar diferentes estratégias para quebrar a cifra e avaliar seu grau de segurança sob diversas circunstâncias.[1]

Este relatório abordará a implementação da cifra de Vigenère, sua aplicação em processos de codificação e decodificação de mensagens, bem como a simulação de ataques por meio do método de Friedman. O resultado dessa investigação nos proporcionará conslusões sobre esse metodo classico de criptografia.

2. Criptografia

A cifra de Vigenère, um importante método de criptografia, opera com uma chave, uma sequência de caracteres. Em seu processo, cada letra da mensagem original é combinada, uma a uma, com caracteres correspondentes na chave, resultando em um texto cifrado[2]. É fundamental notar que a chave deve ser repetida para abranger todo o comprimento da mensagem, garantindo que cada parte da mensagem seja cifrada de maneira única.[3]

Em nossa implementação em Java, Criamos uma classse de criptografia e descriptografia que adota os conceito essencial da cifra de Vigenère e aplica no contexto do projeto. Inicialmente, criamos um objeto do tipo StringBuilder, nomeado de encryptedText, que desempenha o papel de armazenar o texto à medida que ele é criptografado. Além disso, introduzimos duas variáveis: message, responsável por conter a mensagem ou arquivo a ser criptografado, e key, que servirá como a base para o processo de cifragem.

O processo de criptografia é conduzido por um loop que percorre minuciosamente cada caractere da mensagem. Durante esse ciclo, o deslocamento (shift) é calculado com base nos caracteres da chave. Esse cálculo implica a subtração do valor ASCII de 'A' do caractere correspondente na chave. A seguir, aplicamos a fórmula da cifra de Vigenère, que subtrai 'A' do caractere original, acrescenta o deslocamento e garante que o resultado

permaneça dentro do alfabeto. O caractere resultante é prontamente anexado ao StringBuilder encryptedText, e o loop persiste até que todos os caracteres da mensagem tenham sido processados.

O resultado final é um texto criptografado, pronto para ser armazenado em um arquivo TXT ou exibido na tela, conforme a preferência do usuário. Antes do processo de criptografia, a mensagem também passa por uma etapa de tratamento que remove espaços em branco e caracteres especiais ou pontuações, aprimorando a precisão do processo.

```
public class EncryptionDecryption {
  public static String encrypt(String message, String key) {
      StringBuilder encryptedText = new StringBuilder();
      message = message.toUpperCase();
      key = key.toUpperCase();

      for (int i = 0, j = 0; i < message.length(); i++) {
          char c = message.charAt(i);
          if (Character.isLetter(c)) {
              int shift = key.charAt(j % key.length()) - 'A';
              char encryptedChar = (char)
              ((c - 'A' + shift) % 26 + 'A');
              encryptedText.append(encryptedChar);
              j++;
          } else {
              encryptedText.append(c);
          }
      }

      return encryptedText.toString();
  }
```

3. Descriptografia

Após termos implementado o processo de criptografia usando os princípios da cifra de Vigenère, é crucial também possuir um método que desfaça essa criptografia e revele a mensagem original. Em essência, esse processo é o inverso da cifragem[**?**].

O método ”decrypt” tem como missão principal reverter o processo de criptografia, convertendo o texto cifrado de volta à sua forma original. Ele requer como entrada o texto cifrado e a chave conhecida, e durante o processo de descriptografia, ele acumula o texto descriptografado em um objeto StringBuilder chamado ”decryptedMessage”. Além disso, realiza um tratamento de texto para padronizar a mensagem e a chave, removendo caracteres especiais, espaços em branco, e outros elementos que não fazem parte do processo de descriptografia.

No loop de descriptografia, o texto cifrado é minuciosamente percorrido, e a cada iteração, verifica-se se o caractere ”c” é uma letra. Se for o caso, é executado um processo de deslocamento semelhante ao utilizado na cifragem, com base na chave. No final desse processo, é aplicada a operação inversa da fórmula de Vigenère para obter a mensagem descriptografada.

O resultado descriptografado é armazenado no objeto "decryptedMessage" e, uma vez que todo o texto cifrado tenha sido processado, o método "decrypt" retorna o texto descriptografado, pronto para ser exibido ao usuário ou armazenado em um arquivo TXT.

O processo de descriptografia desempenha um papel fundamental em restaurar mensagens criptografadas à sua forma original, permitindo a recuperação de informações importantes em uma comunicação segura.

```
public static String decrypt(String encryptedText, String key) {
        StringBuilder decryptedMessage = new StringBuilder();
        encryptedText = encryptedText.toUpperCase();
        key = key.toUpperCase();

        for (int i = 0, j = 0; i < encryptedText.length(); i++) {
            char c = encryptedText.charAt(i);
            if (Character.isLetter(c)) {
                int shift = key.charAt(j % key.length()) - 'A';
                char decryptedChar = (char)
                ((c - 'A' - shift + 26) % 26 + 'A');
                decryptedMessage.append(decryptedChar);
                j++;
            } else {
                decryptedMessage.append(c);
            }
        }

        return decryptedMessage.toString();
    }
```

4. Descobrindo a chave para um ataque

A segunda parte do projeto envolve a análise de uma mensagem criptografada, onde o objetivo é descobrir a chave que foi usada para criptografá-la e, assim, simular um ataque para revelar a mensagem original. Para essa tarefa, o método de Friedman foi empregado.

A classe FindKey contém vários métodos que, juntos, realizam o processo de determinação da chave de criptografia. O primeiro deles é o findKeySize, que é responsável por iniciar o processo.

O método findKeySize opera em três etapas essenciais:

- Identificação de Sequências Repetitivas: O método inicia procurando todas as sequências de três caracteres no texto cifrado. Essas sequências repetitivas são indicadores-chave de que uma cifra polialfabética, como a cifra de Vigenère, pode ter sido utilizada.
- Cálculo da Distância entre Sequências: Após encontrar as primeiras ocorrências das sequências repetitivas, o método calcula a distância entre elas. Essa distância é crucial porque reflete o período da chave de criptografia, que está diretamente relacionado ao seu tamanho.
- Estimativa do Tamanho da Chave: O método findKeySize então examina quais fatores dividem a distância entre as sequências repetitivas encontradas. Isso é

feito por meio de um loop que verifica a divisibilidade por números de 2 a 20. Os tamanhos possíveis da chave são registrados em uma lista, e o tamanho com mais ocorrências é considerado o tamanho estimado da chave.

```
public static int findKeySize(String encryptedText) {
        List<Integer> factors = new ArrayList<>();

        for (int i = 0; i < encryptedText.length() - 2; i++) {
            String seq = encryptedText.substring(i, i + 3);

            for (int j = i + 3; j < encryptedText.length() - 2; j++) {
                if (encryptedText.substring(j, j + 3).equals(seq)) {
                    int dist = j - i;


                    for (int k = 2; k <= 20; k++) {
                        if (dist % k == 0) {
                            factors.add(k);
                            break;
                        }
                    }
                    break;
                }
            }
        }
        int maxSize = 0;
        int maxCount = 0;

        for (int i = 5; i <= 10; i++) {
            int count = countOccurrences(factors, i);

            if (count > maxCount) {
                maxCount = count;
                maxSize = i;
            }
        }

        if (maxCount >= 1) {
            return maxSize;
        } else {

            return 0;
        }
    }
```

Após a determinação do tamanho da chave, o método findKey é invocado com o objetivo de encontrar a chave real para o processo de descriptografia.

No método findKey, o texto cifrado é convertido em letras maiúsculas para normalizar o processamento. Ele então entra em um loop que itera por todas as letras da chave. Dentro desse loop, todas as possíveis rotações da cifra de César são testadas. Para cada rotação, o texto é descriptografado e uma pontuação de frequência é calculada com base nas frequências esperadas das letras na língua de destino, utilizando a função calculateFrequencyScore e o conjunto de frequências fornecido pela classe LoadFrequenciesFile. A rotação com a menor pontuação é considerada a melhor opção, e a letra correspondente é adicionada à chave.

Portanto, o processo envolve a identificação do tamanho da chave usando o método de Friedman e, em seguida, a descoberta da chave real usando a análise de frequência das letras na língua de destino, permitindo a descriptografia bem-sucedida do texto cifrado.

```
public static int countOccurrences(List<Integer> list, int target) {
        int count = 0;
        for (int num : list) {
            if (num == target) {
                count++;
            }
        }
        return count;
    }
public static double calculateFrequencyScore(String text, double[]
    text = text.toUpperCase();
    int[] letterCounts = new int[26];
    int totalLetters = 0;

    for (char c : text.toCharArray()) {
        if (Character.isLetter(c)) {
            letterCounts[c - 'A']++;
            totalLetters++;
        }
    }

    double score = 0.0;
    for (int i = 0; i < 26; i++) {
        double expectedFrequency = languageFrequencies[i]
        * totalLetters;
        double observedFrequency = letterCounts[i];
        score += Math.pow(expectedFrequency - observedFrequency, 2)
    }

    return score;
}
```

5. Conclusão

O estudo e a construção deste projeto proporcionou através da implementação da cifra de Vigenère revelou a riqueza e a complexidade da criptografia. A compreensão da importância da chave, a repetição estratégica dela e a aplicação de deslocamentos precisos durante o processo de criptografia e descriptografia foram lições valiosas. Este projeto não apenas ilustrou os princípios fundamentais da segurança da informação, mas também destacou a necessidade contínua de proteger a confidencialidade e a integridade das comunicações em um mundo digital em constante evolução.

Ao explorar a cifra de Vigenère, aprendemos que a criptografia é uma ferramenta vital na salvaguarda de informações sensíveis e na garantia da privacidade digital. Essas lições nos inspiram a continuar aprofundando nosso entendimento da segurança cibernética e a buscar maneiras mais avançadas de proteger os dados em um cenário cada vez mais interconectado e suscetível a ameaças.

References

[1] CHRISTENSEN, C. Cryptanalysis of the vigenère cipher: The friedman test. *a a 1*, 1 (2015), 1.

[2] SERMENO, J. P., SECUGAL, K. A. S., MISTIO, N. E., ET AL. Modified vigenere cryptosystem: An integrated data encryption module for learning management system. *International Journal of Applied Science and Engineering 18*, 4 (2021), 1–10.

[3] SOOFI, A. A., RIAZ, I., AND RASHEED, U. An enhanced vigenere cipher for data security. *Int. J. Sci. Technol. Res 5*, 3 (2016), 141–145.

[4] SOOFI, A. A., RIAZ, I., AND RASHEED, U. An enhanced vigenere cipher for data security. *Int. J. Sci. Technol. Res 5*, 3 (2016), 141–145.

[5] WIKIPEDIA. Systemverilog — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher`. [Online; accessed 9-July-2016].