Segurança Computacional: Cifra de Vigenère e Ataque Por Análise de Frequências

Gabriel Fontenele, 15/0126760

[1] Dep. Ciência da Computação – Universidade de Brasília (UnB)
Maio de 2023

CIC0201 - Segurança Computacional
Turma 01 - Prof. João José da Costa Gondim

1. Introdução

O presente projeto visa implementar os métodos de cifração, decifração e quebra da cifra de Vigenère. A linguagem C++ foi utilizada para escrever um programa que cifra e decifra mensagens de texto através de uma chave fornecida via interface de usuário ou faz um ataque de recuperação da chave de um texto cifrado utilizando comparação de histogramas de frequência de letras do texto original e do idioma do texto, que também é escolhido previamente via interface de usuário.

1.1. Entrada e Saída

O programa oferece as várias opções no início da execução, cabendo ao usuário inserir a opção desejada e em seguida apertar *Enter*. Subsequentemente, o usuário deve inserir corretamente os dados que forem pedidos.

Ao final da execução de cada opção, o programa pergunta se o usuário deseja que o texto resultante seja salvo em um novo arquivo de texto. Os arquivos de texto inseridos na entrada devem ser encontrados na pasta raiz do programa. Mensagem cifradas aparecem em maiúsculas e decifradas em minúsculas.

2. Inplementação do método de Vigenère

2.1. Cifrador

O cifrador recebe uma mensagem de texto e uma chave que é usada para cifrar a mensagem utilizando o método de Vigenère. Veja a Figure 1, onde cada letra do texto (coluna) é combinada com uma letra da chave (linha), que se repete quando acaba, obtendo assim uma nova letra (célula) que irá compor um caractere do texto cifrado. Inicialmente são desconsiderados todos os caracteres não alfabéticos e toda a acentuação. Em seguida, os demais caracteres são convertidos em letras maiúsculas.

Inicia-se, então, o processo de cifração - veja o Listing 1. Os caracteres são lidos como valores inteiros e como, a partir da segunda linha, cada linha da Figure 1 faz um *shift* à esquerda da ordem dos caracteres da linha acima, reiniciando o alfabeto na posição necessária da tabela, é possível calcular a posição no alfabeto de uma letra cifrada a partir da soma da posição no alfabeto de uma letra da mensagem original com a posição no alfabeto de uma letra da chave. Tal cálculo se dá escolhendo, a partir do primeiro caractere da mensagem e do primeiro caractere da chave e obtendo em sequência cada par de caracteres, posição i do texto e na posição $i \mod n$ enquanto o texto da mensagem não chegar ao final, com n sendo o tamanho da chave.

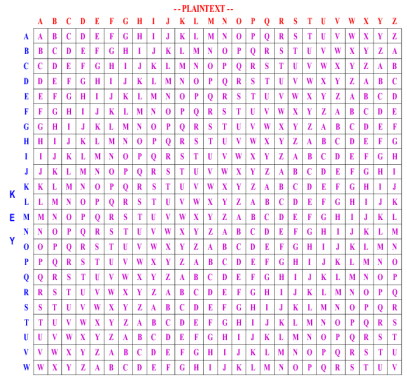

Figure 1. Tabela de cifração de Vigenère Clique aqui para visualizar em tamanho maior.

O alfabeto com as letras maiúsculas na tabela ASCII tem início a partir de 65 em sistema decimal, porém o alfabeto tem 26 letras. Assim para reiniciar o alfabeto do início caso o valor da soma ultrapasse 25, é preciso subtrair 65 para cada termo da soma, ou seja 130 do valor total da soma, e calcular a soma congruente a módulo 26. Assim é obtida a posição no alfabeto de cada letra cifrada, bastanto somar 65 para equalizar os caracteres com os correspondentes na tabela ASCII. Cada vez que a chave acaba, o valor de incremento da chave para cifração também deve ser congruente módulo o tamanho da chave.

```
while (i < message.size()) {
    cipherChar = ((message[i]+key[j%key.size()]-130)%26)+65;
    cipherText.append(1, cipherChar); j++; i++;
}
```

Listing 1. Trecho de código da cifração

Por fim cada caractere cifrado é adicionado à mensagem de saída do programa.

2.2. Decifrador

Se com o cifrador obtém-se cada caractere cifrado a partir da soma da posição no alfabeto de um caractere da mensagem original com a posição no alfbateo de um caractere da chave, para decifrar o código reutiliza-se a mesma chave para calcular a distancia da posição de um caractere da chave para um caractere do texto cifrado, recuperando assim um caractere do texto decifrado. Veja o Listing 2.

```
while (i < cipherText.size()) {
    decipherChar = ((cipherText[i]-key[j%key.size()]+26)%26)+97;
    message.append(1, decipherChar); j++; i++;
}
```

Listing 2. Trecho de código da decifração

Nesse caso, a distância de uma posição a outra deve estar no intervalo de inteiros $[0, 26)$. Porém, como a cada shift o alfabeto se repete, essa distância pode assumir valores negativos. A solução é somar 26 e calcular o valor congruente a módulo 26, que fornece um valor positivo no intervalo de posições das letras do alfabeto. Por fim, soma-se 97 a cada caractere para igualar aos caracteres correspondentes e minúsculos da tabela ASCII. Cada letra é, por fim, adicionada à mensagem de saída do programa.

3. Ataque à cifra de Vigenére

O quebrador de cifra é uma classe que recupera a chave da cifra de Vigenére com base na frequência de caracteres do texto cifrado em comparação com a frequência de caracteres presentes em palavras do idioma utilizado.

Inicialmente, encontram-se todos os trigramas sequenciais (três letras em sequência) no texto cifrado e, para cada um, é salva a distância (em quantidade de caracteres) entre cada repetição presente no texto cifrado. Veja o Listing 3.

```
1 std::string trigram;
2 for (int i=0; i < cipherText.size()-3; i++) {
3     trigram = cipherText.substr(i,3);
4     for (int j=i+3; j < cipherText.size(); j++)
5         if (cipherText.substr(j,3) == trigram)
6             trigrams.push_back(std::make_pair(trigram, j-i));
7 }
```

Listing 3. Função que lê e salva os espaçamentos entre trigramas iguais no texto cifrado

Tomando como convenção tamanhos de chave que variam de 4 a 20 caracteres, obtém-se os divisores, dentro deste intervalo, de cada distância salva e conta-se a quantidade obtida para cada divisor. Os divisores com maior quantidade têm maior peso e descrevem os mais prováveis tamanhos de chave para o texto cifrado. Veja o Listing 4.

```
1 std::map<int, int> keySizesWeights;
2 for (int i=keySizeMin; i<=keySizeMax; i++)
3     keySizesWeights[i] = 0;
4
5 for (int trigramId=0; trigramId < trigrams.size(); trigramId++)
6     for (int i=keySizeMin; i<=keySizeMax; i++)
7         if (trigrams[trigramId].second%i == 0)
8             keySizesWeights[i]++;
```

Listing 4. Função que calcula o peso de cada provável tamanho de chave

A partir de um provável tamanho de chave, sabe-se que a *i*-ésima letra do texto é cifrada pela *k*-ésima letra da chave, com i congruente a k módulo n e n o provável tamanho da chave. Fazendo isso, obtém-se n arranjos distintos de caracteres que são cifrados por uma mesma letra da chave. Em Listing 5, vê-se que no *k*-ésimo arranjo - obtido na linha 5 -, conta-se a quantidade de cada letra do alfabeto presente no texto cifrado - linha 6. O inteiro declarado na linha 2 descreve a quantidade de letras no arranjo k e é usado para calcular a frequência de letras do texto cifrado.

```
1 std::vector<float> cipherLetterFrequencies(26, 0.0f);
2 int keyLetterGroupSize = 0;
3
4 for (int i=0; i < cipherText.size(); i++)
5     if (i%likelyKeySize == k) {
6         cipherLetterFrequencies[cipherText[i]-65]++;
7         keyLetterGroupSize++;
8     }
```

Listing 5. Calculo da ocorrência de cada letra no grupo do digito *k* da chave

Assim, para cada arranjo, compara-se as frequências de cada letra presente no arranjo com as frequências do idioma em que a mensagem foi escrita originalmente. Considerando a ordem alfabética do idioma e posicionando um histograma de frequências das letras de um arranjo de forma que as suas distâncias em relação ao histograma de frequências das letras do idioma sejam as menores possíveis, sabe-se pela distância deste posicionamento qual letra cifrou o arranjo. O algoritmo desenvolvido faz esse cálculo de distâncias de forma automatizada, utilizando uma implementação da fórmula da distância qui-quadrado, que retorna o valor de menor distância obtido. Veja Listing 6 e, em seguida, a fórmula que a função $chiSquareDistance()$ implementa.

```
1 mostLikelyCharacter = chiSquareDistance(cipherLetterFrequencies,
      letterFrequencies[textLanguage]);
2 mostLikelyKey += (char)mostLikelyCharacter;
```

Listing 6. Uso da função de distância qui-quadrado para obtenção do dígito *k* da chave

$$\delta(X,Y) = \sum_{i=1}^{n} \frac{(Xi - Yi)^2}{(Xi + Yi)}$$

Repetindo o processo para cada arranjo, obtém-se as n letras que formam a provável chave de cifração.

3.1. Resultados para o ataque

Através da implementação do código do programa descrito e usando os histogramas de frequência do idioma da chave original - previamente decidido, é possível recuperar de maneira muito simples a chave para diversos textos cifrados.

Vale explicitar, porém, que quanto maior a chave, mais texto cifrado é necessário para obtê-la, uma vez que para chaves maiores a incidência de repetições costuma ser menor. Assim sendo, dado um texto cifrado de tamanho infinito, teoricamente seria possível obter a chave independente de seu tamanho.

Referências

[1] Funcionamento da cifra de Vigenere:
https://youtu.be/SkJcmCaHqS0

[2] Quebrando a cifra de Vigenere:
https://youtu.be/P4z3jAOzT9I

[3] Frequência de letras:
https://pt.wikipedia.org/wiki/Frequência_de_letras

[4] Chi-Square Distance in Python:
https://www.geeksforgeeks.org/chi-square-distance-in-python/