Title: Attacking Classic Crypto Systems(Lab 2)

Omar faruk
Reg No: 2019831055

May 6, 2024

Subtitle 1:Checkpoint – 1

1 Abstract

This report explores the decryption process of the Caesar cipher, a simple substitution cipher technique. The decryption algorithm iterates through all possible shifts to reveal the original message. We analyze the efficiency and accuracy of the decryption process through a sample ciphertext.

2 Introduction

The Caesar cipher is one of the earliest and simplest encryption techniques in cryptography. It involves shifting each letter in the plaintext by a fixed number of positions down the alphabet. The decryption process requires trying all possible shifts to find the original message

3 Objective

The primary objective of this report is to demonstrate the effectiveness of the Caesar cipher decryption algorithm in revealing the original message from a given ciphertext.

4 METHODOLOGY

The primary objective of this report is to demonstrate the effectiveness of the Caesar cipher decryption algorithm in revealing the original message from a given ciphertext.

1. Caesar Decryption Function:
   a) The decryption function takes a ciphertext and a shift value as inputs.
   b) It iterates through each character in the ciphertext.
   c) For each character, it applies the reverse shift operation to retrieve the original letter.
   d) The decrypted characters are appended to form the plaintext.

Listing 1: Caesar Decryption Function

```
string caesar_decrypt(string ciphertext, int shift) {
     string plaintext;
     for (int i = 0; i < ciphertext.size(); i++) {
          char c = ((ciphertext[i] - 'a' - shift + 26) %
             26) + 'a';
          plaintext.push_back(c);
     }
     return plaintext;
}
```

2. Break Caesar Cipher Function:
   a) This function attempts all 26 possible shifts to decrypt the ciphertext.
   b) For each shift, it calls the Caesar decryption function and prints the decrypted text.

Listing 2: Caesar Decryption Function

```
void break_caesar_cipher(string ciphertext) {
    for (int shift = 0; shift < 26; shift++) {
        string decrypted_text = caesar_decrypt(ciphertext,
            shift);
        cout << "For shift " << shift << " Decrypted text :
            " << decrypted_text << endl;
     }
}
```

3. Sample Input:
   a) A sample ciphertext is provided as input to the decryption algorithm.

5 Results

The decryption algorithm successfully reveals the original message for all 26 possible shifts. Each decrypted text corresponds to a different shift, but only one of them yields a meaningful message. For this specific ciphertext, the original message is "ethereumisthebestsmartcontractplatformoutthere" when decrypted with a shift of 10.

Subtitle 2:Checkpoint – 2

1 Objective

The objective of this C++ program is to implement a basic substitution cipher decoder that takes a ciphered text as input and attempts to decrypt it by mapping the frequency of characters in the cipher text to the typical letter frequency of English text. The assumption here is that the most frequent letter in the English language is 'e', followed by 't', 'a', 'o', etc..

2 Components of the Program

The objective of this C++ program is to implement a basic substitution cipher decoder that takes a ciphered text as input and attempts to decrypt it by mapping the frequency of characters in the cipher text to the typical letter frequency of English text. The assumption here is that the most frequent letter in the English language is 'e', followed by 't', 'a', 'o', etc..

1. Global Variables:
   a) **string cipher_text:** Holds the input ciphered text.
   b) **map<char, int> text_frequency:** Used to store the frequency of each character in the cipher text.
   c) **unordered_map<char, char> m:** Maps cipher text characters to the presumed original English letters based on frequency.
   d) **vector<char> text, english_text:** text stores characters from the cipher text in order of frequency; english_text stores English letters in order of typical frequency.

Listing 3: Global Variables

```
string cipher_text ;
map<char, int>text_frequency;
unordered_map<char, char>m;
vector<char> text, english_text = {'e', 't', 'a', 'o'
    , 'n', 'h', 'i', 's', 'r', 'd', 'l', 'u', 'w', 'm'
    , 'g', 'c', 'f', 'y', 'b', 'p', 'k', 'v', 'j', 'x'
    , 'q', 'z'};
```

2. Function Descriptions:
   a) **void mapping():** Maps characters from the cipher text to English letters based on sorted frequency.

Listing 4: mapping

```
void mapping() {
```

```
        for (int i = 0; i < text.size(); i++) {
                m.insert({text[i], english_text[i]});
        }
}
```

b) **void text_frequency_cont():** Computes the frequency of each character in the cipher text, sorts them by frequency in descending order, and calls the mapping function.

Listing 5: text_frequency_cont

```
void text_frequency_cont() {
        vector<pair<char, int>>p;
        map<int, char> temp;
        for (int i = 0; i < cipher_text.size(); i++) {
                if (isalpha(cipher_text[i])) {
                        text_frequency[cipher_text[i]]++;

                }

        }

        for (auto it : text_frequency) {
                p.push_back({it.first, it.second});
        }
        sort(p.begin(), p.end(), [](const auto & a, const
            auto & b) {
                return a.second > b.second;
        });
        for (auto it : p) {
                text.push_back(it.first);
        }
        mapping();
}
```

c) **void substitution():** Performs the substitution of cipher text characters with mapped English letters and prints the decoded message.

Listing 6: substitution

```
void substitution() {
    text_frequency_cont();
        for (int i = 0; i < cipher_text.size(); i++) {
                if (isalpha(cipher_text[i])) {
                        cout << m[cipher_text[i]];
                }
                else {
                        cout << cipher_text[i];
```

```
                }
            }

    }
```

d) **void solve():** Reads input from the user and triggers the substitution process.

Listing 7: solve

```
void solve() {
        getline(cin, cipher_text);
        substitution();
}
```

3. Algorithm and Logic:
   a) **Frequency Count:** The program first reads the cipher text and calculates the frequency of each alphabetic character.
   b) **Sorting:** Characters are then sorted based on their frequency in descending order.
   c) **Mapping:** Each character from the cipher text is mapped to an English letter starting from 'e' for the most frequent, followed by 't', and so on.
   d) **Decoding:** The program substitutes each character in the cipher text with the corresponding mapped English letter and outputs the decoded text.
4. Key Functions and Methods:
   a) **getline(cin, cipher_text):** Captures the entire line of input text.
   b) **sort() with lambda:** Sorts the vector of pairs (character and frequency) based on the second value (frequency) in descending order.

3 OBSERVATIONS

The accuracy of the decoding depends heavily on the text length and the distribution of characters. Short texts or those with atypical distribution might not be decoded correctly.Small text represent low frequency or less accurate frequency which may increase incorrectness.

From my observation ,the second text was easy to break compare to the first one .The second one was easy as it got lengthy text as compare to the first one.