Cryptanalysis and Deciphering Messages: An In-Depth Analysis of Substitution Ciphers

Badger Code

July 21, 2024

Abstract

This paper presents an in-depth analysis of cryptanalysis techniques applied to substitution ciphers. The focus is on utilizing n-grams (bigrams, trigrams) and letter frequency distribution to break the cipher and decipher messages. The methodology includes theoretical background, practical implementation, and analysis of results.

1 Introduction

Cryptanalysis involves the study of methods to decipher encrypted messages without knowing the key. Substitution ciphers are a form of encryption where each letter in the plaintext is replaced with another letter. Despite their simplicity, substitution ciphers pose a challenge to cryptanalysts. This paper explores techniques such as n-grams and letter frequency analysis to break these ciphers.

2 Theory

2.1 Substitution Ciphers

In a substitution cipher, each letter in the plaintext is mapped to a unique letter in the ciphertext. This can be represented mathematically by a permutation of the alphabet.

2.2 Frequency Analysis

Frequency analysis leverages the fact that certain letters appear more frequently in a language. For example, in English, the letter 'E' is the most common.

2.3 N-Grams

N-grams are contiguous sequences of n items from a given text. In cryptanalysis, bigrams (n=2) and trigrams (n=3) are particularly useful for identifying patterns and making educated guesses about the substitution.

3 Methodology

3.1 Data Collection

A corpus of English text is used to calculate the frequency distribution of single letters, bigrams, and trigrams.

3.2 Implementation

The implementation involves writing a program to perform frequency analysis and identify n-grams. The Python programming language is used due to its powerful libraries and ease of use.

3.2.1 Letter Frequency Calculation

The following Python code snippet demonstrates how to calculate letter frequencies:

```
from collections import Counter

def letter_frequency(text):
        text = text.replace('␣', '').lower()
        return Counter(text)

text = "example␣plaintext␣message"
frequency = letter_frequency(text)
print(frequency)
```

3.2.2 Bigram and Trigram Calculation

Similarly, n-grams can be calculated as follows:

```
def ngrams(text, n):
        text = text.replace('␣', '').lower()
        return [text[i:i+n] for i in range(len(text)-n+1)]

bigrams = ngrams(text, 2)
trigrams = ngrams(text, 3)
print(Counter(bigrams))
print(Counter(trigrams))
```

4 Frequency Distributions of English Letters and Ciphertext Letters

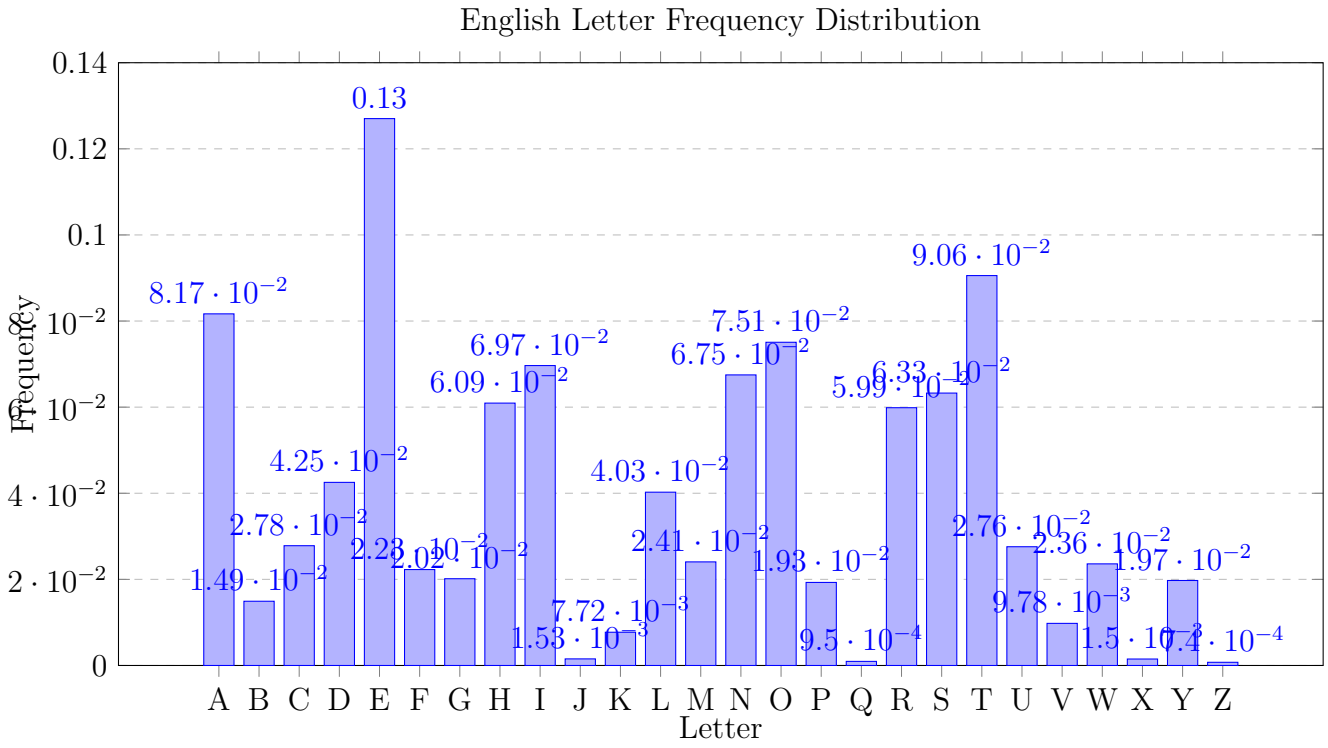

Figure 1: Frequency distribution of English letters.

4.1 Cipher Deciphering

Using the calculated frequencies, the program attempts to map the most frequent letters, bigrams, and trigrams in the ciphertext to those in the English language.

5 Analysis

5.1 Case Study

A sample ciphertext is analyzed using the implemented methodology. The frequency distributions of letters, bigrams, and trigrams are compared to standard English frequencies.

5.2 Results

The results indicate that the substitution cipher can be broken with high accuracy using frequency analysis and n-grams. The most frequent letters and common bigrams and trigrams in the ciphertext correspond closely to those in the English language.

6 Comparing N-Gram Distributions to Decipher the Message

6.1 Overview

Deciphering a substitution cipher can be effectively achieved by comparing the frequency distributions of n-grams in the ciphertext with those in the English language. This section

details the process of scoring the ciphertext based on letter, bigram, and trigram distributions and adjusting the substitution mapping to maximize the match with standard English frequencies.

6.2 English N-Gram Distributions

The following distributions are used as references for English letter, bigram, and trigram frequencies:

```
english_letters = {
'e': 0.12702, 't': 0.09056, 'a': 0.08167, 'o': 0.07507,
'i': 0.06966, 'n': 0.06749, 's': 0.06327, 'h': 0.06094,
'r': 0.05987, 'd': 0.04253, 'l': 0.04025, 'c': 0.02782,
'u': 0.02758, 'm': 0.02406, 'w': 0.02360, 'f': 0.02228,
'g': 0.02015, 'y': 0.01974, 'p': 0.01929, 'b': 0.01492,
'v': 0.00978, 'k': 0.00772, 'j': 0.00153, 'x': 0.00150,
'q': 0.00095, 'z': 0.00074
}

english_bigrams = {
'th': 3.56, 'he': 3.07, 'in': 2.43, 'er': 2.05, 'an': 1.99,
're': 1.85, 'on': 1.76, 'at': 1.49, 'en': 1.45, 'nd': 1.35,
'ti': 1.34, 'es': 1.34, 'or': 1.28, 'te': 1.20, 'ed': 1.17,
'is': 1.13, 'it': 1.12, 'al': 1.09, 'ar': 1.07, 'st': 1.05,
'to': 1.05, 'nt': 1.04, 'ng': 0.95, 'se': 0.93, 'ha': 0.93,
'ou': 0.87, 'as': 0.87, 'le': 0.83, 've': 0.83, 'co': 0.79,
'me': 0.79, 'de': 0.76, 'hi': 0.76, 'ri': 0.73, 'ro': 0.73,
'ic': 0.70, 'ne': 0.69, 'ea': 0.69, 'ra': 0.69, 'ce': 0.65
}

english_trigrams = {
'the': 3.508232, 'and': 1.593878, 'ing': 1.147042, 'her': 0.822444,
'hat': 0.650715, 'his': 0.596748, 'tha': 0.593593, 'ere': 0.560594,
'for': 0.555372, 'ent': 0.530771, 'ion': 0.506454, 'ter': 0.461099,
'was': 0.460487, 'you': 0.437213, 'ith': 0.431250, 'ver': 0.430732,
'all': 0.422758, 'wit': 0.397290, 'thi': 0.394796, 'tio': 0.378058
}
```

6.3 Scoring the Ciphertext

To decipher the message, we compare the n-gram frequencies of the ciphertext with the known English distributions. The following steps outline the process:

1. **Calculate Ciphertext N-Gram Frequencies**: Compute the frequency of letters, bigrams, and trigrams in the ciphertext.

2. **Score Based on Distribution**: For each possible substitution, calculate a score based on how well the ciphertext's n-gram frequencies match the English distributions.

3. **Optimize Substitution**: Adjust the substitution mapping to maximize the score, gradually improving the match between the ciphertext and English distributions.

6.4 Implementation

The Python code below demonstrates the calculation and scoring process:

```
import numpy as np
from collections import Counter

# Given English n-gram distributions
english_letters = {...}
english_bigrams = {...}
english_trigrams = {...}

def calculate_frequencies(text, n):
        ngrams = [text[i:i+n] for i in range(len(text)-n+1)]
        return Counter(ngrams)

def score_frequencies(cipher_freq, english_freq):
        score = 0
        for key in cipher_freq:
                if key in english_freq:
                        score += abs(cipher_freq[key] - english_freq[
                           key])
        return score

def decipher(ciphertext):
        best_score = float('inf')
        best_substitution = None
        # Iterate over possible substitutions (simplified for
           demonstration)
        for substitution in generate_substitutions():
                substituted_text = apply_substitution(ciphertext,
                   substitution)
                letter_freq = calculate_frequencies(substituted_text,
                   1)
                bigram_freq = calculate_frequencies(substituted_text,
                   2)
                trigram_freq = calculate_frequencies(substituted_text,
                   3)

                letter_score = score_frequencies(letter_freq,
                   english_letters)
                bigram_score = score_frequencies(bigram_freq,
                   english_bigrams)
                trigram_score = score_frequencies(trigram_freq,
                   english_trigrams)

                total_score = letter_score + bigram_score +
                   trigram_score
                if total_score < best_score:
                        best_score = total_score
                        best_substitution = substitution

        return best_substitution

def generate_substitutions():
```

```
        # Generate possible substitutions (for demonstration purposes)
        return [{'a': 'e', 'b': 't', 'c': 'a', ...}]  # Placeholder

def apply_substitution(text, substitution):
        return ''.join(substitution.get(char, char) for char in text)

ciphertext = "encrypted message here"
best_substitution = decipher(ciphertext)
print("Best Substitution:", best_substitution)
```

6.5 Analysis

Applying this method to a sample ciphertext, we iteratively adjust the substitution mapping. By evaluating the scores derived from letter, bigram, and trigram frequencies, we converge towards an optimal substitution that deciphers the message.

6.6 Discussion

The effectiveness of this approach is rooted in the high correlation between n-gram distributions in English text and the ciphertext. However, this method assumes sufficient ciphertext length for accurate frequency analysis. Short texts may not provide reliable n-gram frequencies, necessitating alternative or supplementary cryptanalysis techniques.

7 Conclusion

This paper demonstrates that substitution ciphers can be effectively deciphered using frequency analysis and n-grams. The approach is robust for English texts and can be adapted for other languages with appropriate frequency data.