Poly-alphabetic Substitution: Vigenère Cipher

Implementation and Brute-Force Attack

Arav Amawate (2022091)
Angadjeet Singh (2022071)

January 31, 2025

Abstract

This report presents the implementation of the Vigenère cipher, a poly-alphabetic substitution cipher, focusing on its encryption and decryption mechanisms. Additionally, a brute-force attack method is developed to recover the encryption key by exhaustively searching through all possible 4-letter keys. The system incorporates a hash validation technique to ensure the accuracy of decrypted texts. Experimental results demonstrate the effectiveness of the cipher and the brute-force approach in key recovery.

1 Introduction

The Vigenère cipher is a classic method of encrypting alphabetic text by using a simple form of poly-alphabetic substitution. Unlike mono-alphabetic ciphers, which use a single substitution rule, poly-alphabetic ciphers utilize multiple substitution rules, significantly enhancing security by mitigating frequency analysis attacks. This paper details the implementation of the Vigenère cipher, including encryption and decryption processes, and explores a brute-force attack strategy to recover the encryption key. A hash validation mechanism is also introduced to verify the correctness of decrypted texts.

2 System Overview

The implemented system comprises four primary components:

- **Encryption:** Transforms plaintext into ciphertext using a repeating key based on the Vigenère cipher.
- **Decryption:** Reverts ciphertext back to plaintext using the same key.
- **Hash Validation:** Utilizes a custom hash function to validate the integrity of decrypted texts.
- **Brute-Force Attack:** Systematically attempts all possible 4-letter keys to uncover the correct encryption key.

3 Encryption & Decryption System

3.1 Encryption Process

The encryption mechanism adheres to the standard Vigenère cipher formula:

$$C_i = (P_i + K_{i \bmod m}) \mod 26 \tag{1}$$

where:

- P_i represents the i^{th} plaintext character converted to its corresponding numeric value (0-25).
- $K_{i \bmod m}$ is the i^{th} key character, with m being the key length.
- C_i is the resulting ciphertext character.

Example:

Plaintext: `angadjeetsingh`
Key: `nscx`
Ciphertext: `nfixqbgbgkkktzi`

3.2 Decryption Process

Decryption reverses the encryption formula:

$$P_i = (C_i - K_{i \bmod m} + 26) \mod 26 \tag{2}$$

where:

- C_i is the i^{th} ciphertext character.

- $K_{i \bmod m}$ is the i^{th} key character.
- P_i is the recovered plaintext character.

Example:

Ciphertext: `nfixqbgbgkkktzi`
Key: `nscx`
Decrypted Text: `angadjeetsingh`

4 Hash Validation

To ensure the integrity and correctness of decrypted texts, a simple hash function is employed:

$$\text{Hash(text)} = \left(\sum(\text{ord(char)} - 97)\right) \mod 26 \tag{3}$$

This function calculates the sum of the numerical values of all characters in the text (with 'a' as 0) and then takes the modulo 26 of the result. This hash is used to verify whether the decrypted text is likely to be valid.

Example:

Plaintext: `networksecurity`
Hash: `d`
Recognizability Check: `True`

5 Brute-Force Attack

5.1 Motivation

Given that the key length is known to be 4 characters, a brute-force attack involves trying all possible combinations of 4-letter keys (totaling $26^4 = 456,976$ possibilities) to decrypt the ciphertext and identify the correct key based on the hash validation.

5.2 Algorithm

The brute-force attack follows these steps:

1. **Key Generation:** Generate all possible 4-letter key combinations from 'a' to 'z'.
2. **Decryption:** For each key, decrypt the ciphertext using the Vigenère decryption process.
3. **Validation:** Compute the hash of the decrypted text and compare it against expected hash values.
4. **Verification:** If a decrypted text passes the hash validation, verify its correctness. If multiple keys pass, continue the search to ensure uniqueness.
5. **Key Identification:** Upon successful validation, identify the key as the correct encryption key.

Example Output:

Attempting to break the key, try number: `450000` out of `456976`
Key found: `nscx`

6 Results

6.1 Sample Inputs & Outputs

The following table showcases sample encryption and decryption results using the implemented Vigenère cipher with the key `nscx`.

Plaintext	Key	Ciphertext
angadjeetsingh	nscx	nfixqbgbgkkktz
networksecurity	nscx	awvtbjmpruwovla
aravamawate	nscx	njcsnectnlg

Table 1: Sample Encryption and Decryption Outputs

6.2 Brute-Force Attack Performance

The brute-force attack successfully identified the correct key `nscx` after attempting all 456,976 possible keys. The decryption was validated using the hash function, ensuring that the decrypted texts matched expected values.

7 Discussion

The implementation demonstrates that the Vigenère cipher provides a reasonable level of security against simple frequency analysis due to its poly-alphabetic nature. However, when the key length is known and relatively short, as in this project, a brute-force attack remains feasible. The hash validation technique effectively filters out incorrect keys, streamlining the search process.

8 Conclusion

This project successfully implemented the Vigenère cipher's encryption and decryption mechanisms and demonstrated the viability of a brute-force attack in recovering a short encryption key. The incorporation of a hash validation function ensures the accuracy of decrypted texts, providing a reliable method for key verification. While effective for short keys, the brute-force approach highlights the importance of using sufficiently long and complex keys to enhance cryptographic security.

9 References

- Vigenère, B. (1586). *Traicté des chiffres*.
- Kasiski, H. (1863). *Investigations about the ciphers used by the Arabians*.
- Shannon, C. E. (1949). *Communication Theory of Secrecy Systems*. Bell System Technical Journal.
- Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice*. Pearson.
- Neso Academy. *Vigenère Cipher Tutorial*. Retrieved from `https://www.nesoacademy.org`