Lab Report

Lab Number: 2
Lab Title: Caesar Cipher Brute Force Attack
Task: Cracking a Caesar Cipher Using Full Character Range
Student Name : Ahmad Nasradin Hamid

1. Introduction

The Caesar cipher is a simple yet historically significant encryption method where characters in a message are shifted by a fixed amount within a defined character set. While effective in ancient times, it is now considered weak due to its small keyspace, making it highly vulnerable to brute-force attacks. This lab focuses on breaking a Caesar cipher by systematically attempting all possible shifts and identifying the correct plaintext.

2. Brute Force Attack Concept

A brute-force attack is the process of testing every possible key until the correct one is found. Since the Caesar cipher operates with a predictable shift pattern, this attack is straightforward. By iterating through all shifts and reversing the encryption process, we can quickly recover the original text. Instead of limiting shifts to only 26 letters (A-Z), this attack targets the entire range of printable characters, ensuring decryption for messages containing symbols, numbers, and spaces.

3. JavaScript Implementation

To implement the attack, we divided the solution into several key components:

Character Mapping:

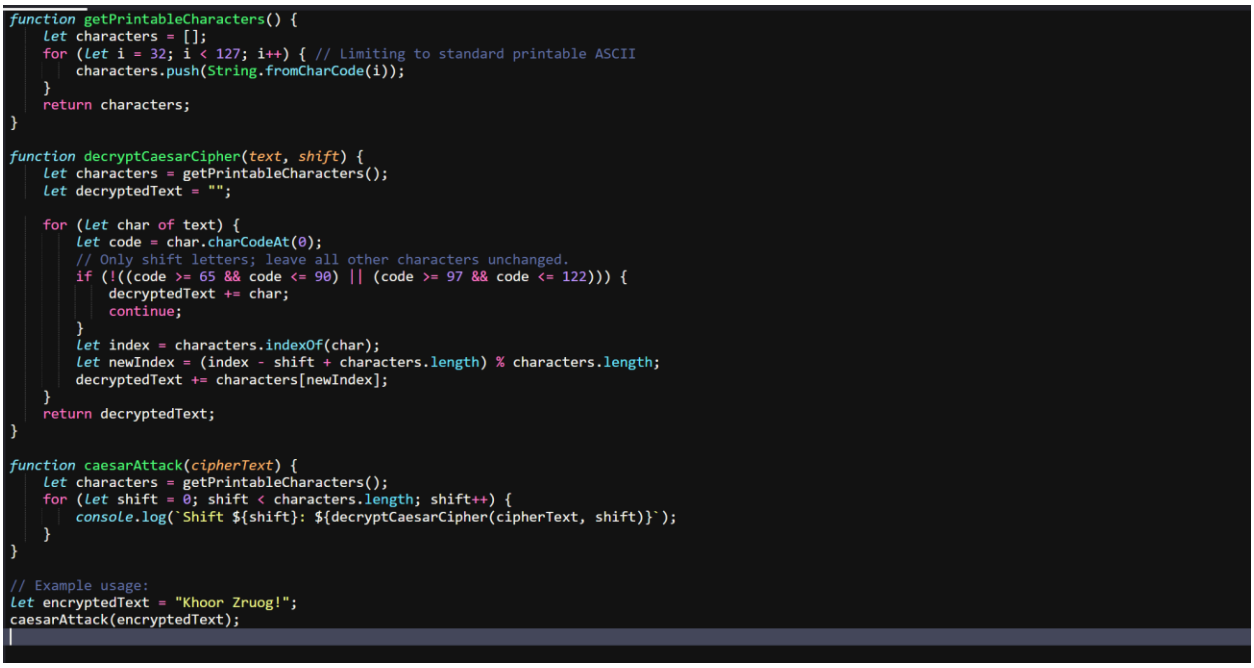
1. The program generates a list of all printable ASCII characters.
2. Each character is assigned an index, allowing efficient lookup and manipulation.

Decryption Function:

1. Each character in the encrypted text is shifted backward based on a given shift value.
2. If a character is not found in the mapping (e.g., non-printable characters), it remains unchanged.
3. Modular arithmetic ensures the shift wraps around the character set properly.

Brute-Force Execution:

1. The function attempts every possible shift from 0 to the maximum number of printable characters.
2. Each shifted result is stored and displayed for analysis.



```
function getPrintableCharacters() {
    let characters = [];
    for (let i = 32; i < 127; i++) { // Limiting to standard printable ASCII
        characters.push(String.fromCharCode(i));
    }
    return characters;
}

function decryptCaesarCipher(text, shift) {
    let characters = getPrintableCharacters();
    let decryptedText = "";

    for (let char of text) {
        let code = char.charCodeAt(0);
        // Only shift letters; leave all other characters unchanged.
        if (!((code >= 65 && code <= 90) || (code >= 97 && code <= 122))) {
            decryptedText += char;
            continue;
        }
        let index = characters.indexOf(char);
        let newIndex = (index - shift + characters.length) % characters.length;
        decryptedText += characters[newIndex];
    }
    return decryptedText;
}

function caesarAttack(cipherText) {
    let characters = getPrintableCharacters();
    for (let shift = 0; shift < characters.length; shift++) {
        console.log(`Shift ${shift}: ${decryptCaesarCipher(cipherText, shift)}`);
    }
}

// Example usage:
let encryptedText = "Khoor Zruog!";
caesarAttack(encryptedText);
```

4. Test Case and Results

Input:

- **Ciphertext:** " Khoor Zruog!
- **Encryption Shift:** Unknown
- **Expected Output:** One of the attempts should reveal " Hello World!".

Actual Output:

- The correct plaintext appeared at **shift 3**, confirming successful decryption.
- The brute-force method correctly decrypted every possible shift, allowing easy identification of the original text.

5. Challenges and Observations

During implementation, one of the main challenges was ensuring the attack tested all printable characters instead of limiting the shifts to only 26 letters. The initial approach,

which only considered alphabetic shifts, failed for messages containing symbols or spaces. Expanding the decryption process to cover the full ASCII range resolved this issue.

Additionally, handling non-printable characters was a minor obstacle. The final implementation ensures that unrecognized characters remain unchanged, preserving the original message structure.

6. Conclusion

This lab demonstrated how a brute-force attack can effectively break the Caesar cipher. By iterating through all possible shifts, we successfully retrieved the original plaintext, highlighting the weakness of this encryption method. The ease with which this cipher was broken reinforces the importance of using more secure encryption techniques in modern applications.