|  |  |
| --- | --- |
| Ex.No.1 | DEVELOPMENT OF SECURE CLIENT SERVER COMMUNICATION USING SYMMETRIC ALGORITHMS |

|  |
| --- |
| AIM: |

To implement

* Encryption
* Decryption
* Brute force Attack
* Frequency Analysis attack

in Caesar cipher using Java.

|  |
| --- |
| **THEORY:** |

**Encryption and Decryption in Caesar Cipher:** The Caesar Cipher is a type of substitution cipher where each letter in the plaintext is shifted a certain number of places down or up the alphabet.

Encryption:

To encrypt a message, shift each letter by a fixed number of positions.

*Formula: E(x) = (x + n) mod 26*

Where E(x) is the encrypted letter, x is the position of the letter in the alphabet 0 for A, 1 for B, ..., 25 for Z, and n is the number of positions to shift.

Decryption:

To decrypt a message, shift each letter by the same number of positions in the opposite direction.

*Formula: D(x) = (x - n) mod 26*

**Example:**

Encryption:

Plaintext: "HELLO" (7 4 11 11 14), key = 3 (in range [1,26])

Ciphertext: "KHOOR" (10 7 14 14 17)

Decryption:

Ciphertext: "KHOOR"( 10 7 14 14 17), key = 3(in range [1-26])

Plaintext: "HELLO"

**Brute Force Attack in Caesar Cipher**

A brute force attack involves trying all possible shifts until the correct one is found. The Caesar Cipher has only 25 possible keys (since a shift by 26 would return the original text). To perform a brute force attack, decrypt the ciphertext with all possible keys until a readable plaintext is found.

**Example**

Ciphertext: "KHOOR"

Steps:

1. Shift by 1: "JGNNQ"

2. Shift by 2: "IFMMP"

3. Shift by 3: "HELLO" (Readable plaintext)

**Frequency Analysis Attack in Caesar Cipher**

Frequency analysis exploits the fact that certain letters appear more frequently in a language. By comparing the frequency distribution of the letters in the ciphertext to the typical frequency distribution of letters in the target language, the shift can be deduced.

**Example**

Ciphertext: "ZOLSS"

Frequency of letters in English (approximate):

E, T, A, O, I, N, S, H, R, D, L, C, U, M, W, F, G, Y, P, B, V, K, J, X, Q, Z

If "ZOLSS" has high frequency of "S" and "S" is likely to be "E" in English, then the shift

might be 4 (since S- E = 18 - 4 = 14).

Shift by 4:

Z (25)- 4 = V (21)

O (14) - 4 = K (10)

L (11) - 4 = H (7)

S (18) - 4 = O (14)

S (18) - 4 = O (14)

Plaintext: "VKHOO" (Not readable, but further shifts can be tried)

By refining the frequency analysis, the correct shift can be determined. (Correct Plain text is SHELL)

|  |
| --- |
| **ALGORITHM:** |

1. Initialization:

* Initialize the `CaesarCipher` class with `alpha\_list` and `alpha\_size`.
* Create a dictionary mapping each character in `alpha\_list` to its corresponding index.

2. Encoding:

* Convert each character in `plain\_text`:
* If the character is in `dictionary`:
  + Calculate the new position by adding the `key` to the current position and taking the modulo with `alpha\_size`.
  + Append the character at the new position in `alpha\_list` to the result.
* If the character is not in `dictionary`, append it directly.
* Return the resulting encrypted text.

3. Decoding:

* Convert each character in `cipher\_text`:
* If the character is in `dictionary`:
  + Calculate the new position by subtracting the `key` from the current position and taking the modulo with `alpha\_size`.
  + Append the character at the new position in `alpha\_list` to the result.
* If the character is not in `dictionary`, append it directly.
* Return the resulting decrypted text.

4. Brute Force:

* Initialize an empty list result to store all possible plaintexts along with the time taken for decryption.
* For each possible key from 1 to alpha\_size - 1:
  + Use the decode method to decrypt the cipher\_text with the current key
  + Append the decrypted text to the results list.
* Return the results list containing all possible plaintexts along with their respective decryption times.

5. Frequency Analysis:

* Use the Counter from the collections module to count the frequency of each character in the cipher\_text.
* Sort the frequencies in descending order using most\_common().
* Initialize an empty list results to store all possible plaintexts along with the time taken for decryption.
* Analyse the frequency and search for the possible key until you get a meaningful word
* Append the decrypted text and the time taken as a tuple to the results list.

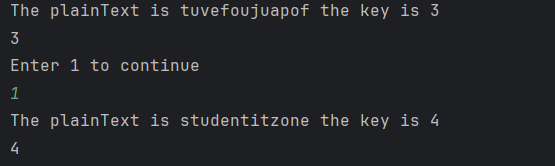
|  |
| --- |
| CODING: |

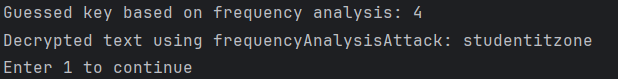
1) Extend the code to perform bruteforce attack and frequency analysis attack.

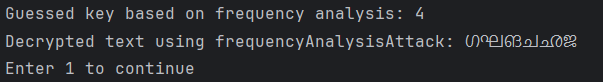
3) Modify the code to perform encryption/ decryption for any language of your choice apart from English. Report on the modifications done

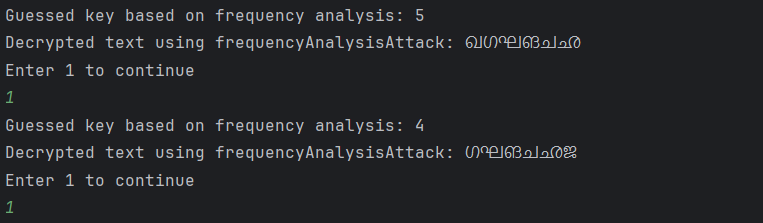
import java.util.HashMap;  
import java.util.Map;  
import java.util.Scanner;  
  
class CaesarCipher {  
 public int k = 0;  
 public int l = 0;  
 public int fl = 0;  
 private final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";  
 private final String Frenchalp = "അആഇഈഉഊഋഎഏഐഒഓഔകഖഗഘങചഛജഝഞടഠഡഢണതഥദധനപഫബഭമയരലവശഷസഹളക്ഷഴറ";  
 private final String Alpha = "etaoinshrdlcumwfgypbvkjxqz";  
  
 public String encrypt(String plainText, int shiftKey) {  
 plainText = plainText.toLowerCase();  
 String cipherText = "";  
 for (int i = 0; i < plainText.length(); i++) {  
 int charPosition = ALPHABET.indexOf(plainText.charAt(i));  
 int keyVal = (shiftKey + charPosition) % 26;  
 char replaceVal = this.ALPHABET.charAt(keyVal);  
 cipherText += replaceVal; }  
 return cipherText; }  
  
 public String decrypt(String cipherText, int shiftKey) {  
 cipherText = cipherText.toLowerCase();  
 String plainText = "";  
 for (int i = 0; i < cipherText.length(); i++) {  
 int charPosition = this.ALPHABET.indexOf(cipherText.charAt(i));  
 int keyVal = (charPosition - shiftKey) % 26;  
 if (keyVal < 0) {  
 keyVal = this.ALPHABET.length() + keyVal; }  
 char replaceVal = this.ALPHABET.charAt(keyVal);  
 plainText += replaceVal; }  
 return plainText; }  
  
 public void bruteforce(String cipherText) {  
 String plainText = decrypt(cipherText, k);  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.println("The plainText is " + plainText + " the key is " + k);  
 System.*out*.println(k);  
 System.*out*.println("Enter 1 to continue");  
 int choice = sc.nextInt();  
 if (choice == 1) {  
 k++;  
 bruteforce(cipherText); } }  
  
 public String Fncencrypt(String plainText, int shiftKey) {  
 plainText = plainText.toLowerCase();  
 String cipherText = "";  
 for (int i = 0; i < plainText.length(); i++) {  
 int charPosition = Frenchalp.indexOf(plainText.charAt(i));  
 int keyVal = (shiftKey + charPosition) % 56;  
 char replaceVal = this.Frenchalp.charAt(keyVal);  
 cipherText += replaceVal; }  
 return cipherText; }  
  
 public String Fncdecrypt(String cipherText, int shiftKey) {  
 cipherText = cipherText.toLowerCase();  
 String plainText = "";  
 for (int i = 0; i < cipherText.length(); i++) {  
 int charPosition = this.Frenchalp.indexOf(cipherText.charAt(i));  
 int keyVal = (charPosition - shiftKey) % Frenchalp.length();  
 if (keyVal < 0) {  
 keyVal = this.Frenchalp.length() + keyVal; }  
 char replaceVal = this.Frenchalp.charAt(keyVal);  
 plainText += replaceVal; }  
 return plainText; }  
  
 public void frequencyAnalysisAttack(String cipherText) {  
 Map<Character, Integer> frequencyMap = new HashMap<>();  
 for (char c : cipherText.toCharArray()) {  
 frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1); }  
  
 char mostFrequentChar = ' ';  
 int maxFrequency = -1;  
 for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {  
 if (entry.getValue() > maxFrequency) {  
 maxFrequency = entry.getValue();  
 mostFrequentChar = entry.getKey(); }}  
 int guessedKey = (Alpha.indexOf(mostFrequentChar) - Alpha.indexOf(Alpha.charAt(l)) + 26) %26;  
 System.out.println("Guessed key based on frequency analysis: " + guessedKey);  
 String guessedPlainText = decrypt(cipherText, guessedKey);  
 System.out.println("Decrypted text using frequencyAnalysisAttack: " + guessedPlainText);  
 Scanner sc = new Scanner(System.in);  
 System.out.println("Enter 1 to continue");  
 int choice = sc.nextInt();  
 if (choice == 1) {  
 l++;  
 frequencyAnalysisAttack(cipherText);}}  
 public void fnfrequencyAnalysisAttack(String cipherText) {  
 Map<Character, Integer> frequencyMap = new HashMap<>();  
 for (char c : cipherText.toCharArray()) {  
 frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1); }  
 char mostFrequentChar = ' ';  
 int maxFrequency = -1;  
 for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {  
 if (entry.getValue() > maxFrequency) {  
 maxFrequency = entry.getValue();  
 mostFrequentChar = entry.getKey(); }}  
 int guessedKey = (Alpha.indexOf(mostFrequentChar) - Alpha.indexOf(Alpha.charAt(fl)) + 56)%56;  
 System.out.println("Guessed key based on frequency analysis: " + guessedKey);  
 String guessedPlainText = Fncdecrypt(cipherText, guessedKey);  
 System.out.println("Decrypted text using frequencyAnalysisAttack: " + guessedPlainText);  
 Scanner sc = new Scanner(System.in);  
 System.out.println("Enter 1 to continue");  
 int choice = sc.nextInt();  
 if (choice == 1) {  
 fl++;  
 fnfrequencyAnalysisAttack(cipherText); }}}  
class CaesarDemo {  
 public static void main(String args[]) {  
 String plainText = "studentitzone";  
 String fnplainText = "ഗഘങചഛജ";  
 int shiftKey = 4;  
 CaesarCipher cc = new CaesarCipher();  
 String cipherText = cc.encrypt(plainText, shiftKey);  
 System.out.println("Your Plain Text :" + plainText);  
 System.out.println("Your Cipher Text :" + cipherText);  
 String cPlainText = cc.decrypt(cipherText, shiftKey);  
 System.out.println("Your Plain Text :" + cPlainText);  
 cc.bruteforce(cipherText);  
 System.out.println("\nFrequency Analysis Attack:");  
 cc.frequencyAnalysisAttack(cipherText);  
 String FncipherText = cc.Fncencrypt(fnplainText, shiftKey);  
 System.out.println("Your Plain Text :" + fnplainText);  
 System.out.println("Your Cipher Text :" + FncipherText);  
 String FncPlainText = cc.Fncdecrypt(FncipherText, shiftKey);  
 System.out.println("Your Plain Text :" + FncPlainText);  
 cc.fnfrequencyAnalysisAttack(FncipherText); }}





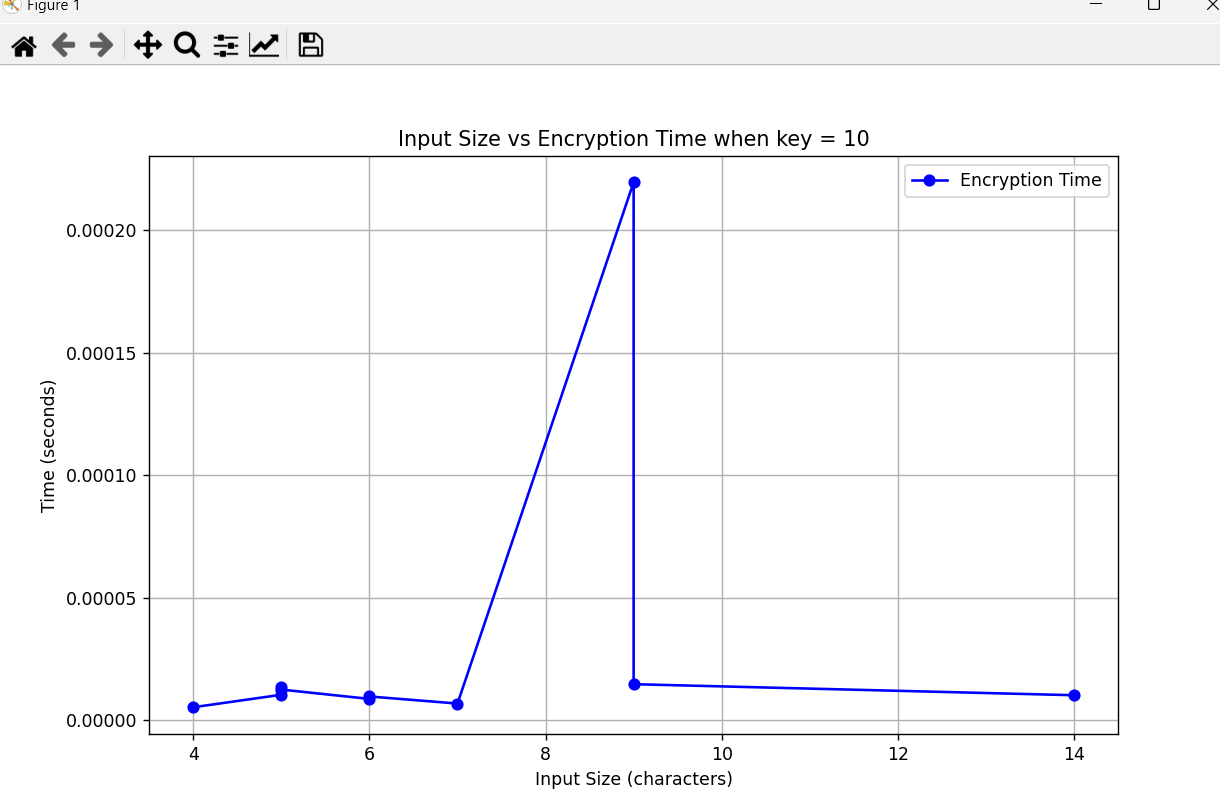






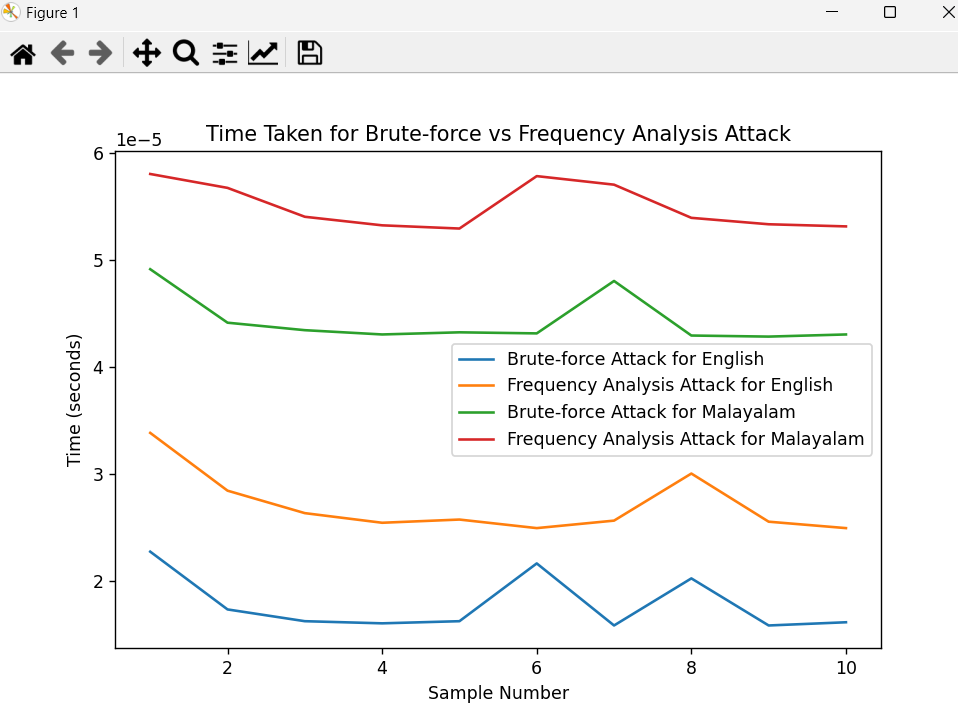
2) Perform encryption of files of different sizes (10 samples) and plot a graph of File size Vs. Encryption time. What is your inference?

import time  
import matplotlib.pyplot as plt  
  
  
class CaesarCipher:  
 ALPHABET = "abcdefghijklmnopqrstuvwxyz"  
  
 def encrypt(self, plain\_text, shift\_key):  
 plain\_text = plain\_text.lower()  
 cipher\_text = ""  
 for char in plain\_text:  
 if char in self.ALPHABET:  
 char\_position = self.ALPHABET.index(char)  
 key\_val = (shift\_key + char\_position) % 26  
 replace\_val = self.ALPHABET[key\_val]  
 cipher\_text += replace\_val  
 else:  
 cipher\_text += char  
 return cipher\_text  
  
 def decrypt(self, cipher\_text, shift\_key):  
 cipher\_text = cipher\_text.lower()  
 plain\_text = ""  
 for char in cipher\_text:  
 if char in self.ALPHABET:  
 char\_position = self.ALPHABET.index(char)  
 key\_val = (char\_position - shift\_key) % 26  
 plain\_text += self.ALPHABET[key\_val]  
 else:  
 plain\_text += char  
 return plain\_text  
  
def read\_from\_file(file\_path):  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 return file.read().strip()  
  
def write\_to\_file(file\_path, content):  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(content)  
  
cc = CaesarCipher()  
sizes = []  
durations = []  
print("Encryption time on Size for Caesar Cipher ")  
shift\_key = int(input("Enter shift key :"))  
for i in range(1, 11):  
 plain\_text = read\_from\_file(f"PlainEngSample\_{i}.txt")  
 start = time.perf\_counter()  
 cipher\_text = cc.encrypt(plain\_text, shift\_key)  
 end = time.perf\_counter()  
 duration = end - start  
 encrypted\_file\_size = len(plain\_text.encode('utf-8'))  
 sizes.append(encrypted\_file\_size)  
 durations.append(duration)  
print(sizes, "\n", durations)  
sorted\_indices = sorted(range(len(sizes)), key=lambda k: sizes[k])  
sizes = [sizes[i] for i in sorted\_indices]  
durations = [durations[i] for i in sorted\_indices]  
print(sizes, "\n", durations)  
plt.figure(figsize=(10, 6))  
plt.plot(sizes, durations, marker='o', color='blue', label='Encryption Time')  
plt.title(f'Input Size vs Encryption Time when key = {shift\_key}')  
plt.xlabel('Input Size (characters)')  
plt.ylabel('Time (seconds)')  
plt.grid(True)  
plt.legend()  
plt.show()



4) Compute the time taken to launch bruteforce attack and frequency analysis attack for different languages for 10 samples. Draw a graph and write your inference

import time  
import string  
import random  
import matplotlib.pyplot as plt  
  
mal = "അആഇഈഉഊഋഎഏഐഒഓഔകഖഗഘങചഛജഝഞടഠഡഢണതഥദധനപഫബഭമയരലവശഷസഹളക്ഷഴറ"  
  
def generate\_ciphertext(length=100):  
 letters = string.ascii\_lowercase  
 return ''.join(random.choice(letters) for i in range(length))  
  
def generate\_malciphertext(length=100):  
 return ''.join(random.choice(mal) for i in range(length))  
  
def brute\_force\_attack(ciphertext):  
 start\_time = time.perf\_counter()  
 for key in string.ascii\_lowercase:  
 decrypted\_text = ''.join(chr((ord(char) - ord(key)) % 26 + ord('a')) for char in ciphertext)  
 if decrypted\_text == ciphertext:  
 break  
 end\_time = time.perf\_counter()  
 return end\_time - start\_time  
  
def mallu\_brute\_force\_attack(ciphertext):  
 start\_time = time.perf\_counter()  
 for key in mal:  
 decrypted\_text = ''.join(mal[(mal.index(char) - mal.index(key)) % len(mal)] for char in ciphertext)  
 if decrypted\_text == ciphertext:  
 break  
 end\_time = time.perf\_counter()  
 return end\_time - start\_time  
  
def frequency\_analysis\_attack(ciphertext):  
 start\_time = time.perf\_counter()  
 frequency = {char: ciphertext.count(char) for char in set(ciphertext)}  
 most\_frequent = max(frequency, key=frequency.get)  
 key = (ord(most\_frequent) - ord('e')) % 26  
 decrypted\_text = ''.join(chr((ord(char) - key) % 26 + ord('a')) for char in ciphertext)  
 end\_time = time.perf\_counter()  
 return end\_time - start\_time  
  
def mallu\_frequency\_analysis\_attack(ciphertext):  
 start\_time = time.perf\_counter()  
 frequency = {char: ciphertext.count(char) for char in set(ciphertext)}  
 most\_frequent = max(frequency, key=frequency.get)  
 key = (mal.index(most\_frequent) - mal.index('ജ')) % len(mal)  
 decrypted\_text = ''.join(mal[(mal.index(char) - key) % len(mal)] for char in ciphertext)  
 end\_time = time.perf\_counter()  
 return end\_time - start\_time  
  
# Generate samples  
samples = [generate\_ciphertext() for \_ in range(10)]  
mallu\_samples = [generate\_malciphertext() for \_ in range(10)]  
  
# Perform attacks  
bruteforce = [brute\_force\_attack(sample) for sample in samples]  
mallu\_bruteforce = [mallu\_brute\_force\_attack(mallu\_sample) for mallu\_sample in mallu\_samples]  
  
frequencyanalysis = [frequency\_analysis\_attack(sample) for sample in samples]  
mallu\_frequencyanalysis = [mallu\_frequency\_analysis\_attack(mallu\_sample) for mallu\_sample in mallu\_samples]  
  
# Plot results  
plt.plot(range(1, 11), bruteforce, label='Brute-force Attack for English')  
plt.plot(range(1, 11), frequencyanalysis, label='Frequency Analysis Attack for English')  
plt.plot(range(1, 11), mallu\_bruteforce, label='Brute-force Attack for Malayalam')  
plt.plot(range(1, 11), mallu\_frequencyanalysis, label='Frequency Analysis Attack for Malayalam')  
plt.xlabel('Sample Number')  
plt.ylabel('Time (seconds)')  
plt.title('Time Taken for Brute-force vs Frequency Analysis Attack')  
plt.legend()  
plt.show()



|  |
| --- |
| RESULT: |

Thus the programs for

* Encryption
* Decryption
* Bruteforce Attack
* Frequency Analysis attack

in Caesar cipher are implemented in Java and the results are verified.

Evaluation:

|  |  |  |
| --- | --- | --- |
| Parameter | Max Marks | Marks Obtained |
| Uniqueness of the Code | 7 |  |
| Use of Comment lines and standard coding practices | 3 |  |
| Viva | 10 |  |
| Sub Total | 20 |  |
| Completion of experiment on time | 3 |  |
| Documentation | 7 |  |
| Sub Total | 10 |  |
| Signature of the faculty with Date |  |  |