|  |  |
| --- | --- |
| **Academic Year: 2024-25** | **Programme: BTECH CSE Cybersecurity** |
| **Year: 2nd** | **Semester: IV** |
| **Student Name: Mishitha** | **Batch : K2** |
| **Roll No: K073** | **Date of experiment: 5/1/25** |
| **Faculty: Prof. Rejo Mathew** | **Signature with Date:** |

Experiment 1: Caesar Cipher

**Aim:** To implement shift ciphers and to study various terms related to cryptography.

# Learning Outcomes:

After completion of this experiment, student should be able to

1. Describe basic symmetric key encryption and decryption process.
2. Understand the working of substitution ciphers.
3. State limitations of shift ciphers.
4. Implement brute force attack

# Theory:

Following are the basic terms related to cryptography

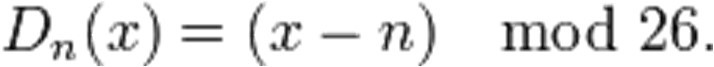
* Plain Text: the original information.
* Cipher text: unintelligible gibberish data. It is the output of encryption process
* Encryption is the process of converting ordinary information (plaintext) into unintelligible gibberish (i.e., cipher text).
* Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext.
* A cipher (or cipher) is a pair of algorithms which create the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a key.
* Key: This is a secret parameter (ideally known only to the communicants) for a specific message exchange context. It is a discreet data set that control the operation of cryptographic algorithm

In cryptography, a **Caesar cipher**, also known as **Caesar's cipher**, the **shift cipher**, **Caesar's code** or **Caesar shift**, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. The method is named after Julius Caesar, who used it in his private correspondence.

The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,..., Z = 25. Encryption of a letter by a shift *n* can be described mathematically as,



Decryption is performed similarly,



In the above, the result is in the range 0...25. If *x+n* or *x-n* are not in the range 0...25, we have to subtract or add 26. The replacement remains the same throughout the message, so the cipher is classed as a type of *monoalphabetic substitution.*

*Example:*

Plaintext is HELLO WORLD

Key is 3. Change each letter to the third letter following it (X goes to A, Y to B, Z to C) Ciphertext is KHOOR ZRUOG

# Procedure:

**Part A: Encryption**

1. Accept key and plain text from the user.
2. Encrypt plaintext using substitution cipher
3. Output cipher text

# Part B: Decryption

1. Accept key and cipher text from the user.
2. Decrypt cipher text using substitution cipher
3. Output Plain text

# Part C: Brute force attack

1. Accept cipher text from the user.
2. Decrypt cipher text using substitution cipher
3. Output Plain text for all possible key

**Algorithm for Caesar Cipher:**

**Encryption**

1. Input:

- Plaintext message.

- A shift value (key) (k).

2. Process:

- For each character in the plaintext:

1. Check if the character is a letter.

2. Shift the character by (k) positions forward in the alphabet.

3. Wrap around to the beginning of the alphabet if the shift exceeds 'Z' (for uppercase) or 'z' (for lowercase).

4. Leave non-alphabet characters unchanged.

3. Output:

- Ciphertext message.

**Decryption**

1. Input:

- Ciphertext message.

- The same shift value (key) (k) used for encryption.

2. Process:

- For each character in the ciphertext:

1. Check if the character is a letter.

2. Shift the character by (k) positions backward in the alphabet.

3. Wrap around to the end of the alphabet if the shift goes below 'A' (for uppercase) or 'a' (for lowercase).

4. Leave non-alphabet characters unchanged.

3. Output:

- Decrypted plaintext message.

**Code: *type or copy your completed working code here***

**Part A: Encryption**

# encryption for caesar cipher

# taking input from user for plain text

p = input("Enter Plain Text: ")

# taking input from user for key

k = int(input("Enter Key: "))

# function to encrypt the text

shiftedp = ""

for char in p:

    if char.isalpha():

        # wrap around logic for lowercase letters

        if char.islower():

            newchar = chr(((ord(char) - ord('a') + k) % 26) + ord('a'))

        # wrap around logic for uppercase letters

        elif char.isupper():

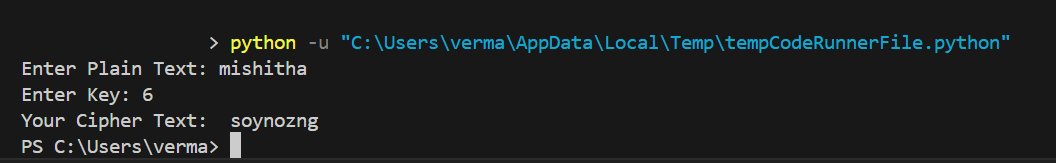
            newchar = chr(((ord(char) - ord('A') + k) % 26) + ord('A'))

        shiftedp += newchar  #store the new character to the result string

    else:

        shiftedp += char

print("Your Cipher Text: ", shiftedp)



# Part B: Decryption

# # decryption for caesar cipher

# # taking input from user for cipher text

# c = input("Enter Cipher Text: ")

# # taking input from user for key

# k = int(input("Enter Key: "))

# # function to decrypt the text

# shiftedc = ""

# for char in c:

# if char.isalpha():

# # wrap around logic for lowercase letters

# if char.islower():

# newchar = chr(((ord(char) - ord('a') - k) % 26) + ord('a'))

# # wrap around logic for uppercase letters

# elif char.isupper():

# newchar = chr(((ord(char) - ord('A') - k) % 26) + ord('A'))

# 

# shiftedc += newchar  # store the new character to the result string

# else:

# shiftedc += char

# print("Your Decrypted Text: ", shiftedc)

# 

# Part C: Brute force attack

# # function to decrypt a caesar cipher with a given key

# def bruteforce(ciphertext, key):

# plaintext = ""  # to store the decrypted message

# for char in ciphertext:

# if char.isalpha():

# # decrypt for lowercase letters

# if char.islower():

# decrypted\_char = chr(((ord(char) - ord('a') - key) % 26) + ord('a'))

# # decrypt for uppercase letters

# elif char.isupper():

# decrypted\_char = chr(((ord(char) - ord('A') - key) % 26) + ord('A'))

# plaintext += decrypted\_char

# else:

# # keep non-alphabet characters unchanged

# plaintext += char

# return plaintext

# # accept cipher text from the user

# ciphertext = input("Enter Cipher Text: ")

# # try all keys from 0 to 25 and print the decrypted result

# for key in range(26):

# decryptedtext = bruteforce(ciphertext, key)

# print("Key " + str(key) + ": " + decryptedtext)

# 

**Questions:**

1. Describe the characteristics of a secure cryptographic algorithm?

Ans: A cryptographic algorithm is a way to protect information by making it unreadable (called ciphertext) so only the right person can understand it. A secure cryptographic algorithm has several important features:

Keeps data private: Only the person with the right key (password or code) can read the data. No one else can access it.

Prevents changes: It makes sure no one can change the data while it’s being sent or stored. If someone tries to change it, the system will notice.

Proves identity: It helps confirm that the person you are talking to is who they say they are.

Stops denial: It ensures that once something is sent or received, the sender can’t deny it. You’ll know who sent the data.

Hard to break: The encryption is strong and difficult for hackers to crack, even with powerful computers.

Safe key handling: Keys (codes or passwords used for encryption) must be stored and used safely.

Works fast: The algorithm should not slow down the system too much while protecting the data.

Stays secure over time: Even as computers get faster, the encryption should remain hard to break.

1. What are some common tools used for brute force attacks, and how do they function?

Ans: A brute force attack is when someone tries all possible password combinations to guess the right one. This method takes time, but it works. Here are some tools used for brute force attacks:

Hashcat: This tool uses powerful computers (like graphics cards) to try many password combinations very quickly. It’s one of the fastest tools for this kind of attack.

John the Ripper: This tool tries many ways to guess a password, including brute force. It works on many different kinds of encrypted data.

Aircrack-ng: This tool is used to break Wi-Fi passwords. It tries many password combinations on a Wi-Fi network until it finds the right one.

Ophcrack: This tool is used to crack Windows passwords by using "rainbow tables" or trying many different password combinations

1. How can frequency distribution analysis be used to identify patterns in ciphertext?

Ans: Frequency analysis is a way to break encrypted messages (ciphers). In most languages, some letters or symbols appear more often than others. For example, in English, the letter "e" is the most common letter.

In an encrypted message, if you see one letter or symbol showing up a lot, it might represent a common letter like "e." By counting how often each symbol appears in the message, you can guess what it represents. For example:

If one symbol appears 10 times and another only once, the symbol that appears more often is likely the letter "e" or another common letter in the language.

This method helps break the encryption and figure out the original message

1. What are the key characteristics of a brute force attack, and how does it differ from a dictionary attack?

Ans: Both brute force attacks and dictionary attacks are ways to guess passwords, but they work in different ways:

Brute force attack: In this attack, the attacker tries every possible combination of letters, numbers, and symbols until they find the correct password. It’s like trying every key on a keychain to unlock a door. It’s slow, but it will eventually work because it checks every possibility.

Dictionary attack: This attack uses a list of common passwords or words (called a dictionary) to guess the password. It’s faster than brute force because it tests only common passwords that people often use, like "123456" or "password." But it only works if the password is simple or common.

So, a brute force attack tries all combinations, but it’s slower. A dictionary attack is faster but only works with common passwords.

1. How can statistical analysis be used to detect anomalies in network traffic that may indicate a brute force attack?

Ans: Brute force attacks can be detected by looking at strange patterns in network activity. Here’s how you can spot them:

Too many login attempts: If there are many login attempts in a short period, especially from the same computer or IP address, it could be a brute force attack.

Many failed login attempts: If there are lots of failed login attempts, it might mean someone is trying to guess the correct password.

Repeating patterns: If the same username or password is being tried over and over again, it’s likely that an automated tool is trying to guess the password.

Timing of attempts: If the login attempts happen in a quick and regular pattern, it might mean a computer is making the attempts instead of a person trying manually.

By watching for these signs, you can detect and stop brute force attacks before they cause harm

**Conclusion:** In this program, we showed how encryption, decryption, and brute force attacks work using the Caesar Cipher. The encryption shifts letters in the plain text by a key value, making the message unreadable without the correct key. Decryption does the opposite, changing the cipher text back to the original message. We also tested a brute force attack by trying all possible keys (0 to 25) to break the cipher. This showed that the Caesar Cipher is weak because it has only 26 possible keys, making it easy to crack. While it's good for learning about cryptography, the Caesar Cipher isn’t safe for protecting important information.