Experiment No. 1
Design and Implementation of a product cipher using Substitution and Transposition ciphers.
Date of Performance:
Date of Submission:

Experiment No. 1

Implementation of a product cipher using Substitution and Transposition

Course Outcome [CSL602.1]: Implement classical encryption techniques

Aim: Design and Implementation of a product cipher using Substitution and Transposition

Objectives:

• To understand the encryption and decryption fundamentals.

• To understand the concepts of the product cipher.

Outcomes: The learner will be able to

• Understand the principles and practices of cryptographic techniques

Hardware / Software Required: C/C++/JAVA/python

Theory:

Substitution cipher is a method of encryption by which units of plaintext are replaced with ciphertext according to a regular system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.

Transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext. That is, the order of the units is changed.

Substitution ciphers can be compared with Transposition ciphers. In a transposition cipher, the units of the plaintext are rearranged in a different and usually quite complex order, but the units themselves are left unchanged. By contrast, in a substitution cipher, the units of the plaintext are retained in the same sequence in the ciphertext, but the units themselves are altered.

1. Caesar Cipher: In cryptography, a Caesar cipher, also known as a 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. For example, with a shift of 3, A would be replaced by D, B would become E, and so on. The method is named after Julius Caesar, who used it to communicate with his generals.

Example:

1.The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. For instance, here is a Caesar cipher using a left rotation of three places (the shift parameter, here 3, is used as the key):

Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC

2.Rail Fence Transposition:

The rail fence cipher (sometimes called zigzag cipher) is a transposition cipher that jumbles up the order of the letters of a message using a basic algorithm.

The rail fence cipher works by writing your message on alternate lines across the page, and then reading off each line in turn.

For example, let's consider the plaintext "This is a secret message".

To encode this message we will first write over two lines (the "rails of the fence") as follows:

Note that all white spaces have been removed from the plain text.

The ciphertext is then read off by writing the top row first, followed by the bottom row:

Implementation:

```
import random


def substitution_cipher_encrypt(plaintext, key):
    encrypted_text = ""
    for char in plaintext:
        if char.isalpha():
            if char.islower():
                encrypted_text += key[0][ord(char) - ord('a')]
            else:
                encrypted_text += key[1][ord(char) - ord('A')]
        else:
            encrypted_text += char
    return encrypted_text

def substitution_cipher_decrypt(ciphertext, key):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():
            if char.islower():
                decrypted_text += chr(key[0].index(char) + ord('a'))
            else:
                decrypted_text += chr(key[1].index(char) + ord('A'))
        else:
            decrypted_text += char
    return decrypted_text

def transposition_cipher_encrypt(plaintext, key):
    num_columns = len(key)
    num_rows = (len(plaintext) + num_columns - 1) // num_columns
    ciphertext = [''] * num_columns
    for col in range(num_columns):
        pointer = col
        for row in range(num_rows):
            if pointer < len(plaintext):
                ciphertext[col] += plaintext[pointer]
            pointer += num_columns
    return ''.join(ciphertext)


def transposition_cipher_decrypt(ciphertext, key):
    num_columns = len(key)
    num_rows = (len(ciphertext) + num_columns - 1) // num_columns
    num_full_columns = len(ciphertext) % num_columns
    plaintext = [''] * num_rows
    col = 0
    row = 0
    for char in ciphertext:
        plaintext[row] += char
        col += 1
        if (col == num_rows) or (col == num_rows - 1 and row >= num_full_columns):
            col = 0
            row += 1
    return ''.join(plaintext)


def generate_substitution_key():
    alphabet = list('abcdefghijklmnopqrstuvwxyz')
    random.shuffle(alphabet)
    substitution_key = [alphabet[:], alphabet[:]]
    return substitution_key


def generate_transposition_key(num_columns):
    return list(range(num_columns))


def main():
    plaintext = "This is a test message for the product cipher"

    # Generate keys
    substitution_key = generate_substitution_key()
    transposition_key = generate_transposition_key(5)  # Example: 5 columns

    # Encryption
    ciphertext = substitution_cipher_encrypt(plaintext, substitution_key)
    ciphertext = transposition_cipher_encrypt(ciphertext, transposition_key)
    print("Encrypted:", ciphertext)

    # Decryption
    decrypted_text = transposition_cipher_decrypt(ciphertext, transposition_key)
    decrypted_text = substitution_cipher_decrypt(decrypted_text, substitution_key)
    print("Decrypted:", decrypted_text)


if __name__ == "__main__":
    main()
```

Output:

```
Encrypted: tsmpTiteisoC a eprmtggeshso   f eeeh isaeractT

Decrypted: This is a test message for the product cipher
```

Conclusion:

The experiment provides valuable insights into the design and implementation of symmetric encryption schemes, highlighting the importance of combining multiple cryptographic techniques to enhance security.