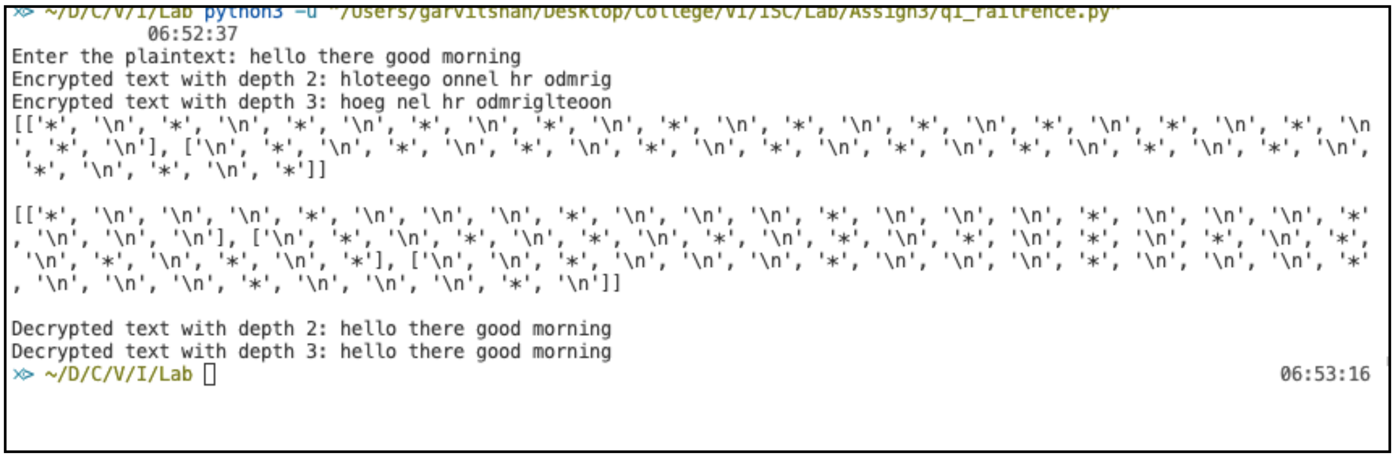
Lab Assignment 3

U21CS089 | Garvit Shah

1) Write a program to implement the Rail Fence Cipher to perform encryption and decryption. Take plain text from the user and generate an encrypted text using the Rail Fence Cipher (Consider depth = 2 and depth = 3).

```
def rail_fence_encrypt(text, depth):
    encrypted_text = [''] * depth
    row = 0
    direction = -1

    for char in text:
        encrypted_text[row] += char
        if row == 0 or row == depth - 1:
            direction *= -1
        row += direction

    return ''.join(encrypted_text)

def rail_fence_decrypt(cipher, key):
# Create the matrix to cipher plain text
    # key = rows, length(text) = columns
    rail = [['\n' for _ in range(len(cipher))] for _ in
range(key)]
    row = 0
    col = 0
    direction = -1

    for i in range(len(cipher)):
        rail[row][col] = '*'
        if row == 0 or row == key - 1:
            direction *= -1
        row += direction
        col+=1

    print(rail)
    print()

    # Now we can construct the fill the rail matrix
    index = 0
    for i in range(key):
        for j in range(len(cipher)):
            if rail[i][j] == '*' and index < len(cipher):
                rail[i][j] = cipher[index]
                index += 1

    # Now read the matrix in zig-zag manner to construct
    # the resultant text
    result = ''
```

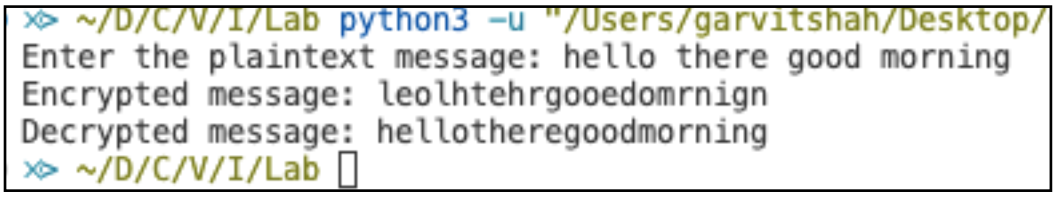
```python
    row = 0
    col = 0
    for i in range(len(cipher)):
        # Check the direction of flow
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False

        # Place the marker
        if rail[row][col] != '*':
            result += rail[row][col]
            col += 1

        # Find the next row using direction flag
        if dir_down:
            row += 1
        else:
            row -= 1

    return result


def main():
    text = input("Enter the plaintext: ")

    depth_2_cipher = rail_fence_encrypt(text, 2)
    print("Encrypted text with depth 2:", depth_2_cipher)

    depth_3_cipher = rail_fence_encrypt(text, 3)
    print("Encrypted text with depth 3:", depth_3_cipher)

    decrypted_depth_2 = rail_fence_decrypt(depth_2_cipher, 2)
    decrypted_depth_3 = rail_fence_decrypt(depth_3_cipher, 3)

    print("Decrypted text with depth 2:", decrypted_depth_2)
    print("Decrypted text with depth 3:", decrypted_depth_3)

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



```
⋊> ~/D/C/V/I/Lab python3 -u "/Users/garvitshah/Desktop/College/VI/ISC/Lab/Assign3/q1_railfence.py"
         06:52:37
Enter the plaintext: hello there good morning
Encrypted text with depth 2: hloteego onnel hr odmrig
Encrypted text with depth 3: hoeg nel hr odmriglteoon
[['*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n
', '*', '\n'], ['\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n',
 '*', '\n', '*', '\n', '*']]

[['*', '\n', '\n', '\n', '*', '\n', '\n', '\n', '*', '\n', '\n', '\n', '*', '\n', '\n', '\n', '*', '\n', '\n', '\n', '*'
, '\n', '\n', '\n'], ['\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*', '\n', '*',
 '\n', '*', '\n', '*', '\n', '*'], ['\n', '\n', '*', '\n', '\n', '\n', '*', '\n', '\n', '\n', '*', '\n', '\n', '\n', '*'
, '\n', '\n', '\n', '*', '\n', '\n', '\n', '*', '\n']]

Decrypted text with depth 2: hello there good morning
Decrypted text with depth 3: hello there good morning
⋊> ~/D/C/V/I/Lab                                                                                              06:53:16
```

2) Implement a program to perform encryption and decryption using a Permutation Cipher with a block size up to 10 characters. Also show how to compute the reverse permutation to decrypt the cipher text and get the plain text.

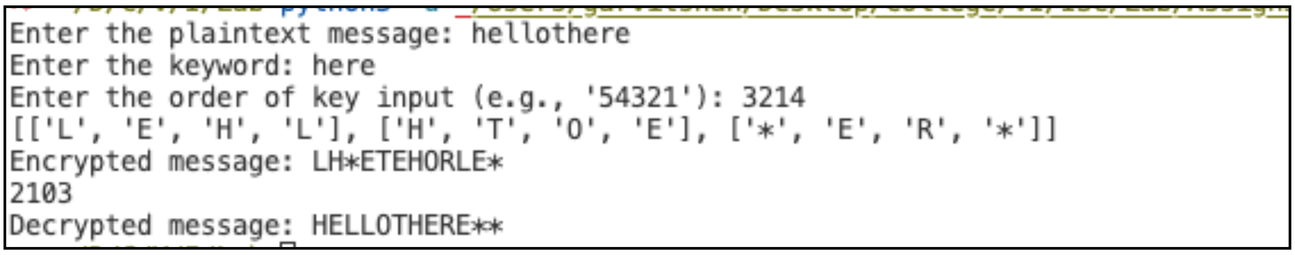
```
def encrypt_permutation_cipher(plaintext, block_size,
permutation_key):
    # Pad the plaintext if its length is not a multiple of the
block size
    plaintext += ' ' * ((block_size - len(plaintext) % block_size)
% block_size)


    # Encrypt the plaintext
    ciphertext = ''
    for i in range(0, len(plaintext), block_size):
        block = plaintext[i:i+block_size]
        for j in permutation_key:
            ciphertext += block[j-1]

    return ciphertext

def decrypt_permutation_cipher(ciphertext, block_size,
permutation_key):
    # Invert the permutation key for decryption
    inverse_permutation_key = [0] * len(permutation_key)
    i = 0

    # Fill in the inverse permutation key based on the positions
in the permutation key
    for pos in permutation_key:
        inverse_permutation_key[pos-1] = i
        i+=1

    # Decrypt the ciphertext
    plaintext = ''
    for i in range(0, len(ciphertext), block_size):
        block = ciphertext[i:i+block_size]
        for j in inverse_permutation_key:
            plaintext += block[j]

    return plaintext.strip()

def main():
    plaintext = input("Enter the plaintext message: ")
    block_size = 10
    key = [3,2,5,4,1,6,7,9,8,0]

    # Encrypt the plaintext
    ciphertext = encrypt_permutation_cipher(plaintext, block_size,
key)
    print("Encrypted message:", ''.join(ciphertext.split(' ')))
    # Decrypt the ciphertext
    decrypted_text = decrypt_permutation_cipher(ciphertext,
block_size, key)
    print("Decrypted message:", ''.join(decrypted_text.split('
')))

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



```
 ~/D/C/V/I/Lab python3 -u "/Users/garvitshah/Desktop/
Enter the plaintext message: hello there good morning
Encrypted message: leolhtehrgooedomrnign
Decrypted message: hellotheregoodmorning
 ~/D/C/V/I/Lab
```

3) Implement a Columnar Transposition Cipher (5x5) with column ordering, to perform both encryption and decryption processes.

```
import math
def encrypt_columnar_transposition(plaintext, keyword, order):
    keyword = keyword.replace(" ", "").upper()
    num_rows = math.ceil(len(plaintext) / len(keyword))

    # Pad the plaintext with spaces to fit the grid
    plaintext += '*' * (num_rows * len(keyword) - len(plaintext))

    # Create the grid
    grid = [[' ' for _ in range(len(keyword))] for _ in
range(num_rows)]

    # Fill the grid with the plaintext characters
    index = 0
    for row in range(num_rows):
        for char, col in enumerate(order):
            grid[row][int(col)-1] = plaintext[index]
            index += 1
    print(grid)
    # Extract the ciphertext by reading the grid column by column
    ciphertext = ''
    for col in range(len(keyword)):
        for row in range(num_rows):
            ciphertext += grid[row][col]

    return ciphertext

def decrypt_columnar_transposition(ciphertext, keyword, order):
    # Remove spaces from the keyword and convert to uppercase
    keyword = keyword.replace(" ", "").upper()
    inv_order = ["" for i in range(len(order))]
    for ind, val in enumerate(order):
        inv_order[int(val)-1] = str(ind)
    inv_order = "".join(inv_order)
    print(inv_order)
    # Determine the number of rows required for the grid
    num_rows = math.ceil(len(ciphertext) / len(keyword))

    # Create the grid
    grid = [[' ' for _ in range(len(keyword))] for _ in
range(num_rows)]

    # Fill the grid with the ciphertext characters according to
the order
    index = 0
    for char, col in enumerate(inv_order):
        for row in range(num_rows):
            grid[row][int(col)] = ciphertext[index]
            index += 1

    # Extract the plaintext by reading the grid row by row
    plaintext = ''
    for row in range(num_rows):
        for col in range(len(keyword)):
            plaintext += grid[row][col]

    return plaintext.strip()

def main():
    # Input plaintext, keyword, and order of key input from the
user
    plaintext = input("Enter the plaintext message: ").upper()
    keyword = input("Enter the keyword: ").upper()
    order = input("Enter the order of key input (e.g., '54321'):
")

    # Encrypt the plaintext using columnar transposition cipher
    ciphertext = encrypt_columnar_transposition(plaintext,
keyword, order)
    print("Encrypted message:", ciphertext)

    # Decrypt the ciphertext using columnar transposition cipher
    decrypted_text = decrypt_columnar_transposition(ciphertext,
keyword, order)
    print("Decrypted message:", decrypted_text)

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



```
Enter the plaintext message: hellothere
Enter the keyword: here
Enter the order of key input (e.g., '54321'): 3214
[['L', 'E', 'H', 'L'], ['H', 'T', 'O', 'E'], ['*', 'E', 'R', '*']]
Encrypted message: LH*ETEHORLE*
2103
Decrypted message: HELLOTHERE**
```