Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL504 **DATE:** 06 – 10 – 24

COURSE NAME: Cryptography and Network Security Laboratory **CLASS:** TYBTech

SAPID: 60003220250

EXPERIMENT NO. 3

CO/LO: Design secure system using appropriate security mechanism

AIM / OBJECTIVE:

Design and Implementation of a Hill cipher on Gray Scale Image / Colour Image.

DESCRIPTION OF EXPERIMENT:

In this lab, we have to implement the Hill Cipher technique. Use two different images, one is covering image which act as key image which is shared by both sender and receiver and other is Informative image. As a first step, we add a cover image and informative image to obtained resultant image. The gray scale image is passed to the Hill Cipher algorithm to form encrypted image. The encrypted image is communicated over an unsecured channel. The encrypted image after receiving by receiver passed to Hill Cipher technique. Receiver first obtained inverse of Key image, K-1. The resulted image which is encrypted is passed to the Hill Cipher to obtain Informative Image. The cover image is subtracted from merged image to obtained informative image. The detail process is summarized in figure 1.

QUESTIONS:

Show encryption and decryption using 3*3 key

Discuss the cryptanalysis of Proposed Hill Cipher Technique.

SOURCE CODE:

HILL CIPHER ON PLAINTEXT:

```
import numpy as np
import math
from string import ascii_uppercase, digits, punctuation

def mod_inverse(a, m):
    a = a % m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def is_invertible(matrix):
    det = int(round(np.linalg.det(matrix))) % 256
    return det != 0 and math.gcd(det, 256) == 1

def generate_random_invertible_matrix(size=2):
    while True:
        matrix = np.random.randint(0, 256, (size, size))
        if is_invertible(matrix):
            return matrix

def preprocess_text(text):
```

```
    return ''.join(filter(lambda x: ord(x) < 256, text))

def encrypt(plaintext, key_matrix):
    plaintext = preprocess_text(plaintext)
    n = key_matrix.shape[0]
    padded_length = (len(plaintext) + (n - 1)) // n * n
    plaintext = plaintext.ljust(padded_length, 'X')

    ciphertext = ""
    for i in range(0, len(plaintext), n):
        block = np.array([ord(char) for char in plaintext[i:i+n]])
        encrypted_block = np.dot(key_matrix, block) % 256
        ciphertext += ''.join(chr(num) for num in encrypted_block)
    return ciphertext

def decrypt(ciphertext, key_matrix):
    n = key_matrix.shape[0]
    det = int(round(np.linalg.det(key_matrix))) % 256
    det_inv = mod_inverse(det, 256)

    if det_inv is None:
        raise ValueError("The key matrix is not invertible.")
    key_matrix_inv = np.round(np.linalg.inv(key_matrix) * np.linalg.det(key_matrix)).astype(int) % 256
    key_matrix_inv = (det_inv * key_matrix_inv) % 256

    decrypted_text = ""
    for i in range(0, len(ciphertext), n):
```

```
        block = np.array([ord(char) for char in ciphertext[i:i+n]])
        decrypted_block = np.dot(key_matrix_inv, block) % 256
        decrypted_text += ''.join(chr(num) for num in decrypted_block)
    return decrypted_text.rstrip('X')

if __name__ == "__main__":
    key_matrix = generate_random_invertible_matrix(size=2)
    print("Invertible Key Matrix :\n", key_matrix)
    plaintext = input("Enter Text Message : ")
    print("Plaintext :", plaintext)
    encrypted_message = encrypt(plaintext, key_matrix)
    print("Encrypted Text:", encrypted_message)
    decrypted_message = decrypt(encrypted_message, key_matrix)
    print("Decrypted Text:", decrypted_message)
```

OUTPUT:

```
Invertible Key Matrix :
 [[159 210]
 [213 225]]
Enter Text Message : Hello1234
Plaintext : Hello1234
Encrypted Text:  ¬È#läm
Decrypted Text: Hello1234
```

HILL CIPHER ON IMAGE:

```
import numpy as np
from PIL import Image
```

```
# Function to check if a matrix is invertible in mod 256
def is_invertible(matrix, mod=256):
    det = int(round(np.linalg.det(matrix))) % mod
    return det != 0 and np.gcd(det, mod) == 1

# Function to find modular inverse of a matrix in mod 256
def mod_matrix_inverse(matrix, mod=256):
    det = int(round(np.linalg.det(matrix))) % mod
    det_inv = pow(det, -1, mod)  # Inverse of determinant mod 256
    matrix_mod_inv = (
        np.round(det_inv * np.linalg.inv(matrix) * np.linalg.det(matrix))
        .astype(int) % mod
    )
    return matrix_mod_inv

# Function to generate a random 5x5 invertible matrix in mod 256
def generate_random_invertible_matrix(size=5, mod=256):
    while True:
        matrix = np.random.randint(0, mod, (size, size))
        if is_invertible(matrix, mod):
            return matrix

# Encrypt function using Hill cipher
def hill_cipher_encrypt(image, key_matrix):
    # Convert image to numpy array and get its shape
    pixel_values = np.array(image)
```

```
    original_shape = pixel_values.shape
    pixel_values = pixel_values.flatten()

    # Padding to make the length a multiple of 5
    padding_length = (5 - len(pixel_values) % 5) % 5
    pixel_values = np.pad(pixel_values, (0, padding_length), mode='constant')

    # Split into groups of 5 pixels
    pixel_groups = np.split(pixel_values, len(pixel_values) // 5)

    # Encrypt each group
    encrypted_pixels = []
    for group in pixel_groups:
        encrypted_group = np.dot(key_matrix, group) % 256
        encrypted_pixels.extend(encrypted_group)

    # Reshape back to original image shape
    encrypted_image = np.array(encrypted_pixels[: len(pixel_values)]).reshape(original_shape)
    return Image.fromarray(encrypted_image.astype(np.uint8))

# Decrypt function using Hill cipher
def hill_cipher_decrypt(encrypted_image, key_matrix):
    # Get inverse of key matrix in mod 256
    key_matrix_inv = mod_matrix_inverse(key_matrix)

    # Convert image to numpy array
    pixel_values = np.array(encrypted_image)
```

```
    original_shape = pixel_values.shape
    pixel_values = pixel_values.flatten()

    # Split into groups of 5 pixels
    pixel_groups = np.split(pixel_values, len(pixel_values) // 5)

    # Decrypt each group
    decrypted_pixels = []
    for group in pixel_groups:
        decrypted_group = np.dot(key_matrix_inv, group) % 256
        decrypted_pixels.extend(decrypted_group)

    # Reshape back to original image shape
    decrypted_image = np.array(decrypted_pixels[: len(pixel_values)]).reshape(original_shape)
    return Image.fromarray(decrypted_image.astype(np.uint8))

# Example usage:
if __name__ == "__main__":
    # Open the grayscale image
    image = Image.open("frog.jpg").convert("L")  # Convert to grayscale

    # Generate a random invertible 5x5 key matrix
    key_matrix = generate_random_invertible_matrix()

    print("Random Invertible Key Matrix:\n", key_matrix)

    # Encrypt the image
```

```
encrypted_image = hill_cipher_encrypt(image, key_matrix)
encrypted_image.save("encrypted_image.png")

# Decrypt the image
decrypted_image = hill_cipher_decrypt(encrypted_image, key_matrix)
decrypted_image.save("decrypted_image.png")
```

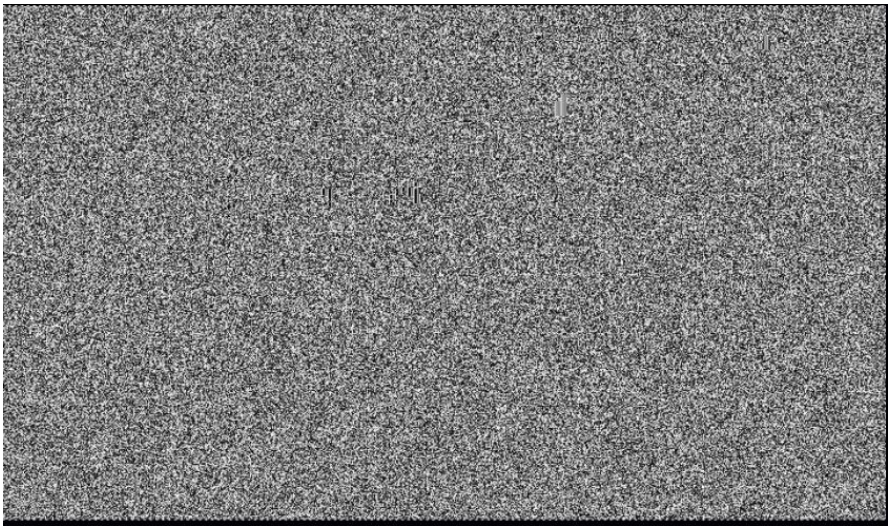
Invertible Key Matrix:

```
[[136 248  51   9 243]
 [212  61  71  17 105]
 [160 170 225 117 136]
 [ 13 233  10  83 101]
 [128  57 195 222 235]]
```

OUTPUT:

Original Image:

Encrpyted Image:



Decrypted Image:

CONCLUSION: In this experiment, We have successfully implemented Hill cipher encryption and decryption algorithm on text message and on color image and also discussed the cryptanalysis of hill cipher algorithm.

REFERENCES:

1. Bruce Schneier, “Applied Cryptography, Protocols Algorithms and Source Code in C”, 2nd Edition, Wiley, 2006.
2. Mark Stamp, “Information Security Principles and Practice”, 2nd Edition, Wiley, 2011.
3. Behrouz A. Ferouzan, “Cryptography & Network Security”, 3rd Edition, Tata McGraw Hill, Nov 2015.
4. William Stallings, “Cryptography and Network Security, Principles and Practice”, 7th Edition, Pearson Education, 2017.