**Aim:** Implementation of Hill Cipher encryption and decryption algorithm.

**Program:**

# Function to generate the 3x3 key matrix from the key string

def getKeyMatrix(key):

return [[ord(key[i \* 3 + j]) % 65 for j in range(3)] for i in range(3)]

# Function to multiply matrices (3x3 \* 3x1)

def matrixMultiply(A, B):

return [[sum(A[i][k] \* B[k][0] for k in range(3)) % 26] for i in range(3)]

# Function to get the determinant of a 3x3 matrix

def determinant(matrix):

return (matrix[0][0] \* (matrix[1][1] \* matrix[2][2] - matrix[1][2] \* matrix[2][1])

- matrix[0][1] \* (matrix[1][0] \* matrix[2][2] - matrix[1][2] \* matrix[2][0])

+ matrix[0][2] \* (matrix[1][0] \* matrix[2][1] - matrix[1][1] \* matrix[2][0])) % 26

# Function to find modular inverse of a number

def modInverse(a, m):

for x in range(1, m):

if (a \* x) % m == 1:

return x

return -1

# Function to get the adjugate matrix

def adjugate(matrix):

adj = [[0]\*3 for \_ in range(3)]

adj[0][0] = (matrix[1][1] \* matrix[2][2] - matrix[1][2] \* matrix[2][1]) % 26

adj[0][1] = (matrix[0][2] \* matrix[2][1] - matrix[0][1] \* matrix[2][2]) % 26

adj[0][2] = (matrix[0][1] \* matrix[1][2] - matrix[0][2] \* matrix[1][1]) % 26

adj[1][0] = (matrix[1][2] \* matrix[2][0] - matrix[1][0] \* matrix[2][2]) % 26

adj[1][1] = (matrix[0][0] \* matrix[2][2] - matrix[0][2] \* matrix[2][0]) % 26

adj[1][2] = (matrix[0][2] \* matrix[1][0] - matrix[0][0] \* matrix[1][2]) % 26

adj[2][0] = (matrix[1][0] \* matrix[2][1] - matrix[1][1] \* matrix[2][0]) % 26

adj[2][1] = (matrix[0][1] \* matrix[2][0] - matrix[0][0] \* matrix[2][1]) % 26

adj[2][2] = (matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]) % 26

return adj

# Function to get the inverse key matrix

def inverseMatrix(matrix):

det = determinant(matrix)

invDet = modInverse(det, 26)

adj = adjugate(matrix)

return [[(invDet \* adj[i][j]) % 26 for j in range(3)] for i in range(3)]

# Function to encrypt message

def encrypt(message, keyMatrix):

messageVector = [[ord(char) % 65] for char in message]

cipherVector = matrixMultiply(keyMatrix, messageVector)

return ''.join([chr(cipherVector[i][0] + 65) for i in range(3)])

# Function to decrypt message

def decrypt(cipherText, keyMatrix):

cipherVector = [[ord(char) % 65] for char in cipherText]

invKeyMatrix = inverseMatrix(keyMatrix)

decryptedVector = matrixMultiply(invKeyMatrix, cipherVector)

return ''.join([chr(decryptedVector[i][0] + 65) for i in range(3)])

# Example usage

message = "ACT"

key = "GYBNQKURP"

print("Key: ",key)

print("Plaintext: ",message)

keyMatrix = getKeyMatrix(key)

cipherText = encrypt(message, keyMatrix)

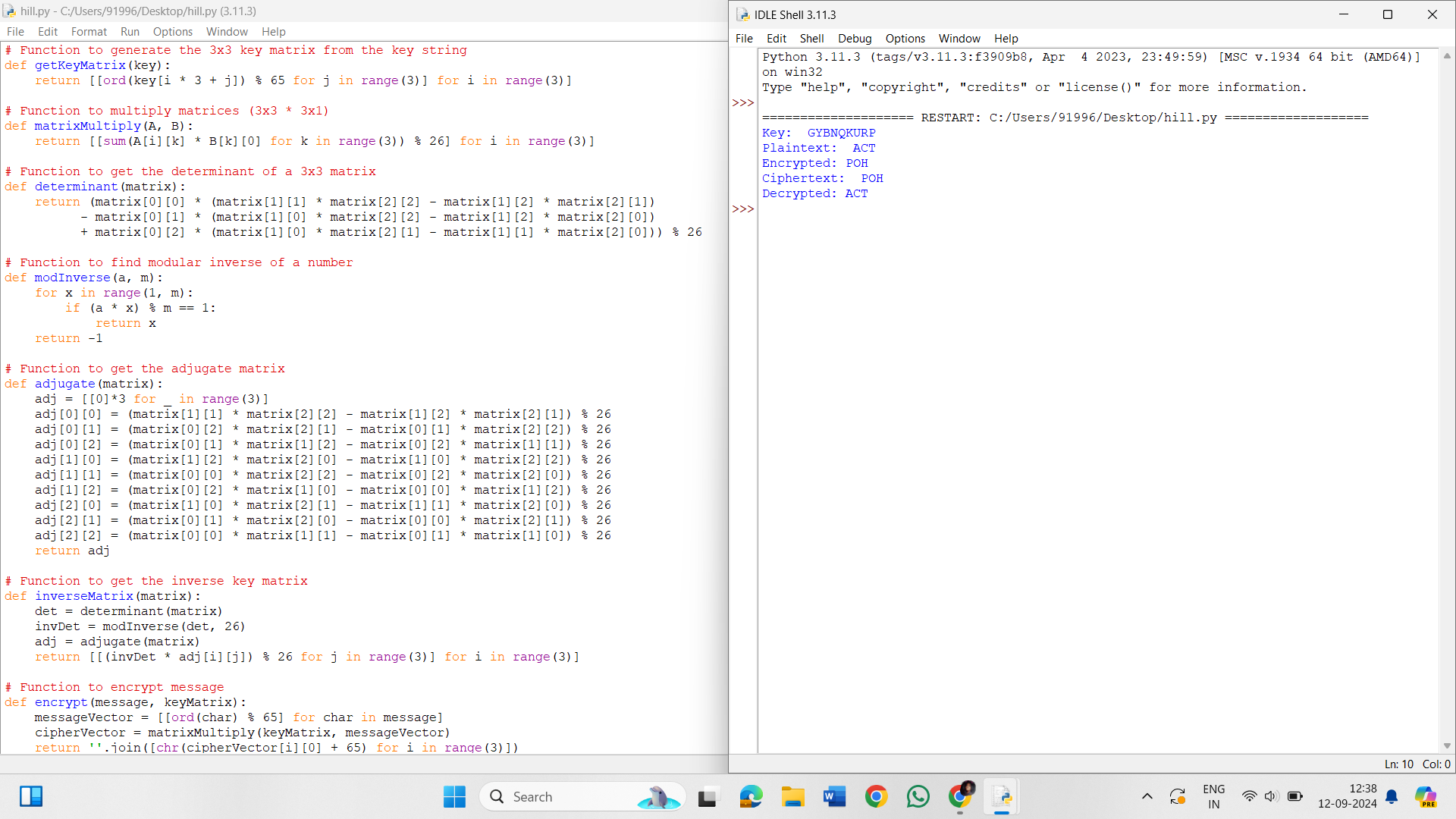
print("Encrypted:", cipherText)

decryptedText = decrypt(cipherText, keyMatrix)

print("Ciphertext: ",cipherText)

print("Decrypted:", decryptedText)

**Output:**

****