Practical-2

Q1. Write a program to implement normal playfair cipher and improvised playfair cipher

A1.

Normal Playfair Cipher

The Playfair Cipher is a manual symmetric encryption technique and was the first literal digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but bears the name of Lord Playfair for promoting its use.

Steps for Normal Playfair Cipher:

1. **Key Matrix Generation**: Create a 5x5 matrix using a keyword. Remove duplicate letters from the keyword and fill the matrix with remaining letters of the alphabet. Traditionally, 'I' and 'J' are treated as the same letter.
2. **Prepare Plaintext**: Modify the plaintext to ensure it can be encrypted in pairs. If a pair of identical letters appear, insert an 'X' between them. If the plaintext has an odd number of characters, append an 'X' at the end.
3. **Encryption Rules**:
    - **Same Row**: Replace each letter with the letter immediately to its right (wrap around to the beginning if needed).
    - **Same Column**: Replace each letter with the letter immediately below it (wrap around to the top if needed).
    - **Rectangle**: Replace each letter with the letter in the same row but in the column of the other letter of the pair.

Improved Playfair-Vigenère-Affine Cipher with Shuffling

Introduction

In this lab, we implement an encryption and decryption system that combines the Playfair, Vigenère, and Affine ciphers, followed by a simple character shuffling step. This multi-

layered approach enhances the security of the encryption process. Below, we detail the steps and functions involved in this encryption and decryption scheme.

Playfair Cipher

The Playfair cipher is a manual symmetric encryption technique. It encrypts pairs of letters (digraphs), making it more secure than simple substitution ciphers.

Steps for Playfair Encryption:

1. **Matrix Generation**: A 5x5 matrix is generated using a key, skipping one letter (usually 'J').
2. **Input Modification**: The input message is modified to ensure there are no repeating characters in a pair, and 'X' is added if necessary.
3. **Pairwise Encryption**: Each pair of letters is encrypted based on their positions in the matrix.

Functions:

- `getMat(key)`: Generates the Playfair matrix.
- `modifyInput(input_message)`: Modifies the input message for Playfair encryption.
- `playFairEncrypt(input_message, key)`: Encrypts the message using the Playfair cipher.
- `playFairDecrypt(cypher, key)`: Decrypts the message using the Playfair cipher.

Vigenère Cipher

The Vigenère cipher is a method of encrypting alphabetic text by using a simple form of polyalphabetic substitution.

Steps for Vigenère Encryption:

1. **Key Extension**: The key is extended to match the length of the message.
2. **Character-wise Encryption**: Each character of the message is encrypted using the corresponding character of the key.

Functions:

- `vignereEncrypt(input_message, key)`: Encrypts the message using the Vigenère cipher.
- `vignereDecrypt(cypher, key)`: Decrypts the message using the Vigenère cipher.

Affine Cipher

The Affine cipher is a type of monoalphabetic substitution cipher, where each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter.

Steps for Affine Encryption:

1. **Parameters Selection**: Choose two keys, `a` and `b`, such that `a` is coprime with 26.
2. **Mathematical Transformation**: Apply the Affine transformation `(a * x + b) % 26` for encryption.

Functions:

- `affineEncrypt(plaintext, a, b)`: Encrypts the message using the Affine cipher.
- `affineDecrypt(ciphertext, a, b)`: Decrypts the message using the Affine cipher.
- `mod_inverse(a, m)`: Finds the modular inverse of `a` under modulo `m`.
- `nextCoPrime(a)`: Finds the next coprime of `a`.

Shuffling

A simple character shuffling step to further obfuscate the encrypted message.

Steps for Shuffling:

1. **Character Swap**: Swap every two characters in the string.

Function:

- `shuffleTwo(cipher)`: Swaps every two characters in the string.

Encryption Process

1. **Playfair Encryption**: Encrypt the input message using the Playfair cipher.

2. **Vigenère Encryption**: Encrypt the Playfair encrypted message using the Vigenère cipher.
3. **Affine Encryption**: Encrypt the Vigenère encrypted message using the Affine cipher.
4. **Shuffling**: Shuffle the characters of the Affine encrypted message.

Decryption Process

1. **Unshuffling**: Reverse the shuffling step.
2. **Affine Decryption**: Decrypt the shuffled message using the Affine cipher.
3. **Vigenère Decryption**: Decrypt the Affine decrypted message using the Vigenère cipher.
4. **Playfair Decryption**: Decrypt the Vigenère decrypted message using the Playfair cipher and remove padding characters.

Normal Cipher Code:

```
import string

def getMat(key):
    key = key.upper().replace("J", "I")
    usedAlphas = set()
    matList = []

    # Add key characters to the matrix
    for k in key:
        if k not in usedAlphas and k in string.ascii_uppercase:
            usedAlphas.add(k)
            matList.append(k)

    # Add remaining characters to the matrix
    for a in string.ascii_uppercase:
        if a not in usedAlphas and a != "J":
            usedAlphas.add(a)
            matList.append(a)

    # Generate the 5x5 matrix
    mat = [matList[i:i + 5] for i in range(0, 25, 5)]

    return mat

def modifyInput(input_message):
    input_message = input_message.upper().replace(" ", "").replace("J", "I")
    formatted_message = ""

    i = 0
```

```
    while i < len(input_message):
        formatted_message += input_message[i]
        if i + 1 < len(input_message):
            if input_message[i] == input_message[i + 1]:
                formatted_message += 'X'
                i += 1
            else:
                formatted_message += input_message[i + 1]
                i += 2
        else:
            formatted_message += 'X'
            i += 1

    return formatted_message

def findPosition(char, mat):
    for i, row in enumerate(mat):
        if char in row:
            return i, row.index(char)
    return None

def displayMat(mat):
    print("\nMatrix: \n")
    for row in mat:
        print("   ".join(row))
    print()

def playFairEncrypt(input_message, key):
    mat = getMat(key)
    displayMat(mat)
    modified_input = modifyInput(input_message)

    encrypted = ""
    i = 0

    while i < len(modified_input):
        a = modified_input[i]
        b = modified_input[i + 1]

        row1, col1 = findPosition(a, mat)
        row2, col2 = findPosition(b, mat)

        if row1 == row2:
            encrypted += mat[row1][(col1 + 1) % 5]
            encrypted += mat[row2][(col2 + 1) % 5]
        elif col1 == col2:
            encrypted += mat[(row1 + 1) % 5][col1]
            encrypted += mat[(row2 + 1) % 5][col2]
        else:
            encrypted += mat[row1][col2]
            encrypted += mat[row2][col1]
```

```
            i += 2

        return encrypted

    def playFairDecrypt(cypher, key):
        mat = getMat(key)
        displayMat(mat)

        plain = ""
        i = 0

        while i < len(cypher):
            a = cypher[i]
            b = cypher[i + 1]

            row1, col1 = findPosition(a, mat)
            row2, col2 = findPosition(b, mat)

            if row1 == row2:
                plain += mat[row1][(col1 - 1) % 5]
                plain += mat[row2][(col2 - 1) % 5]
            elif col1 == col2:
                plain += mat[(row1 - 1) % 5][col1]
                plain += mat[(row2 - 1) % 5][col2]
            else:
                plain += mat[row1][col2]
                plain += mat[row2][col1]

            i += 2

        return plain

    print("\PlayFair Cypher Encryption/Decryption\n")

    input_message = input("\nEnter the message you want to encrypt: ")
    key = input("\nEnter the encryption key you want to use: ")
    encrypted_message = playFairEncrypt(input_message, key)
    print("\nEncrypted Message: ", encrypted_message)
    decrypted_message = playFairDecrypt(encrypted_message, key).replace("X", "")
    print("\nDecrypted Message: ", decrypted_message, "\n")
```

Output:

```
\PlayFair Cypher Encryption/Decryption


Enter the message you want to encrypt: hello

Enter the encryption key you want to use: occur

Matrix:

O   C   U   R   A
B   D   E   F   G
H   I   K   L   M
N   P   Q   S   T
V   W   X   Y   Z


Encrypted Message:  KBKYHR

Matrix:

O   C   U   R   A
B   D   E   F   G
H   I   K   L   M
N   P   Q   S   T
V   W   X   Y   Z


Decrypted Message:  HELLO
```

Improvised Playfair Cipher Code:

```
import math

def autoKeyGeneration(key, input_message): # Generates key of the same length as
the input message
    key = list(key) # Convert key to list
    if len(input_message) == len(key): # If the key is the same length as the input
message, return the key as is
        return "".join(key)
    elif len(input_message) < len(key): # If the key is longer than the input
message
        return "".join(key[:len(input_message)]) # Return the key truncated to the
length of the input message
    else: # If the key is shorter than the input message
        for i in range(len(input_message) - len(key)): # Append the key to itself
until it is the same length as the input message
            key.append(key[i % len(key)])
    return "".join(key)

def getMat(key): # Generate the Playfair matrix
    usedAlphas = set() # Set to keep track of used alphabets
    matList = [] # List to store the matrix
    skipped = False # Flag to check if a character has been skipped
    skippedChar = 'X' # Skipped Character
    replaced = False # Flag to check if a character has been replaced
    replacedChar = 'X' # Replaced Character
```

```
    mat = [] # Matrix

    key = key.upper() # Convert key to uppercase

    for k in key: # Iterate over the key
        if k not in usedAlphas: # If the alphabet has not been used
            usedAlphas.add(k) # Add the alphabet to the used alphabets set
            matList.append(k) # Add the alphabet to the matrix list

    alphabets = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']

    for a in alphabets: # Iterate over the alphabets
        if a not in usedAlphas: # If the alphabet has not been used
            if len(matList) >= 12 and not skipped: # If the matrix list has 12 or
more elements and a character has not been skipped
                skipped = True
                skippedChar = a
                continue
            if not replaced and len(matList) >= 12: # If the matrix list has 12 or
more elements and a character has not been replaced
                replaced = True
                replacedChar = a
            usedAlphas.add(a)
            matList.append(a)

    for i in range(5): # Generate the matrix
        l = [] # List to store the row
        for j in range(5):
            index = 5 * i + j
            l.append(matList[index]) # Append the element to the row
        mat.append(l) # Append the row to the matrix

    return mat, skippedChar, replacedChar

def modifyInput(input_message): # Modify the input message to fit the Playfair
cipher
    input_message = input_message.upper().replace(" ", "") # Convert the input
message to uppercase and remove spaces
    formatted_message = "" # Formatted message

    i = 0 # Counter
    while i < len(input_message):
        formatted_message += input_message[i] # Append the character to the
formatted message
        if i + 1 < len(input_message): # If there is another character in the input
message
            if input_message[i] == input_message[i + 1]: # If the current character
is the same as the next character
                formatted_message += 'X' # Append 'X' to the formatted message
                i += 1 # Increment the counter
```

```
            else:
                formatted_message += input_message[i + 1] # Append the next
character to the formatted message
                i += 2 # Increment the counter by 2
        else:
            formatted_message += 'X' # Append 'X' to the formatted message
            i += 1 # Increment the counter

    # Example: "HELLOO" -> "HELXLOOX"
    return formatted_message

def findPosition(a, mat): # Find the position of a character in the Playfair matrix
    for i, row in enumerate(mat):
        if a in row:
            return i, row.index(a)
    return None

def displayMat(mat): # Display the Playfair matrix
    print("\nMatrix: \n")
    for row in mat:
        print("   ".join(row))
    print()

def playFairEncrypt(input_message, key):
    mat, skippedChar, replacedChar = getMat(key) # Generate the Playfair matrix
    displayMat(mat) # Display the Playfair matrix
    modified_input = modifyInput(input_message.replace(skippedChar, replacedChar))
# Modify the input message

    encrypted = ""
    i = 0

    while i < len(modified_input):
        a = modified_input[i] # Get the first character
        b = modified_input[i + 1] # Get the second character

        row1, col1 = findPosition(a, mat) # Find the position of the first
character in the matrix
        row2, col2 = findPosition(b, mat) # Find the position of the second
character in the matrix

        if row1 == row2: # If the characters are in the same row
            encrypted += mat[row1][(col1 + 1) % 5] # Append the right character to
the encrypted message
            encrypted += mat[row2][(col2 + 1) % 5] # Append the right character to
the encrypted message
        elif col1 == col2: # If the characters are in the same column
            encrypted += mat[(row1 + 1) % 5][col1] # Append the character below to
the encrypted message
            encrypted += mat[(row2 + 1) % 5][col2] # Append the character below to
the encrypted message
```

```
        else: # If the characters are in different rows and columns
            encrypted += mat[row1][col2] # Append the character at the intersection
to the encrypted message
            encrypted += mat[row2][col1] # Append the character at the intersection
to the encrypted message

        i += 2

    return encrypted

def playFairDecrypt(cypher, key):
    mat, skippedChar, replacedChar = getMat(key)
    displayMat(mat)

    plain = ""
    i = 0

    while i < len(cypher):
        a = cypher[i]
        b = cypher[i + 1]

        row1, col1 = findPosition(a, mat)
        row2, col2 = findPosition(b, mat)

        if row1 == row2:
            plain += mat[row1][(col1 - 1) % 5] # Append the left character to the
decrypted message
            plain += mat[row2][(col2 - 1) % 5] # Append the left character to the
decrypted message
        elif col1 == col2:
            plain += mat[(row1 - 1) % 5][col1] # Append the character above to the
decrypted message
            plain += mat[(row2 - 1) % 5][col2] # Append the character above to the
decrypted message
        else:
            plain += mat[row1][col2]
            plain += mat[row2][col1]

        i += 2

    return plain.replace(replacedChar, skippedChar) # Replace the skipped character
with the original character

def vignereEncrypt(input_message, key):
    encrypted = ""
    i = 0

    input_message = input_message.replace(" ", "")
    key = key.replace(" ", "")
    input_message = input_message.upper()
```

```
    while i < len(input_message):
        a = input_message[i] # Get the character
        b = key[i % len(key)] # Get the key character

        encrypted += chr((ord(a) + ord(b) - 2 * ord('A')) % 26 + ord('A')) #
Encrypt the character
        i += 1

    return encrypted

def vignereDecrypt(cypher, key):
    decrypted = ""
    i = 0

    cypher = cypher.replace(" ", "")
    key = key.replace(" ", "")
    cypher = cypher.upper()

    while i < len(cypher):
        a = cypher[i]
        b = key[i % len(key)]

        decrypted += chr((ord(a) - ord(b) + 26) % 26 + ord('A'))

        i += 1

    return decrypted

def mod_inverse(a, m):
    # Function to find the modular inverse of a under modulo 26
    for x in range(1, 26):
        if (a * x) % 26 == 1:
            return x
    raise ValueError("No modular inverse found for a = {} and 26 = {}".format(a,
26))

def nextCoPrime(a):
    # Function to find the next co-prime of a
    for i in range(a + 1, 26):
        if math.gcd(a, i) == 1:
            return i
    return 1

def affineEncrypt(plaintext, a, b):

    a = nextCoPrime(a)

    ciphertext = ''
    for char in plaintext:
        x = ord(char.upper()) - ord('A') # Convert the character to a number
        encrypted_char = (a * x + b) % 26 # Encrypt the character
```

```
        ciphertext += chr(encrypted_char + ord('A')) # Convert the number to a
character

    return ciphertext

def affineDecrypt(ciphertext, a, b):
    plaintext = ''
    a = nextCoPrime(a)
    a_inv = mod_inverse(a, 26) # Find the modular inverse of a
    for char in ciphertext:
        y = ord(char.upper()) - ord('A') # Convert the character to a number
        decrypted_char = (a_inv * (y - b)) % 26 # Decrypt the character
        plaintext += chr(decrypted_char + ord('A')) # Convert the number to a
character
    return plaintext

def shuffleTwo(cipher):
    cipher = list(cipher) # Convert the cipher to a list
    for i in range(0, len(cipher), 2):
        if i + 1 < len(cipher):
            cipher[i], cipher[i + 1] = cipher[i + 1], cipher[i] # Swap the
characters
    return "".join(cipher)

    # Example: "EHLLO" -> "HELLO"

print("\nImproved PlayFair-Vigenère Cypher Encryption/Decryption\n")


input_message = input("Enter the message you want to encrypt: ").upper()
key = input("Enter the encryption key: ").upper()
keyA = int(input("Enter the key A value: "))
keyB = int(input("Enter the key B value: "))

# Step 1: Playfair
pf_encrypted = playFairEncrypt(input_message, key)

# Step 2: Vigenère
vig_encrypted = vignereEncrypt(pf_encrypted, key)

# Step 3: Affine
affine_encrypted = affineEncrypt(vig_encrypted, keyA, keyB)

# Step 4: Shuffle
shuffled = shuffleTwo(affine_encrypted)

print("\nEncrypted Message: ", shuffled)

# input_message = input("Enter the message you want to decrypt: ").upper()
# key = input("Enter the decryption key: ").upper()
```

```
# Step 4: Shuffle
shuffled = shuffleTwo(shuffled)

# Step 3: Affine
affine_decrypted = affineDecrypt(shuffled, keyA, keyB)

# Step 2: Vigenère
vig_decrypted = vignereDecrypt(affine_decrypted, key)

# Step 1: Playfair
final_decrypted = playFairDecrypt(vig_decrypted, key).replace('X', '')

print("\nDecrypted Message: ", final_decrypted)
```

Output:

```
Improved PlayFair-Vigenère Cypher Encryption/Decryption

Enter the message you want to encrypt: hello
Enter the encryption key: occur
Enter the key A value: 2
Enter the key B value: 4

Matrix:

O   C   U   R   A
B   D   E   F   G
H   I   K   L   M
N   P   Q   S   T
V   W   X   Y   Z


Encrypted Message:  NYGOTY

Matrix:

O   C   U   R   A
B   D   E   F   G
H   I   K   L   M
N   P   Q   S   T
V   W   X   Y   Z


Decrypted Message:  HELLO
```