Experiment No:1

Experminet Name: Write a program to implement encryption and decryption using Caesar cipher.

Objective:

1.To implement a basic cryptographic system using the Caesar Cipher algorithm

2.To encrypt and decrypt the text "ASHIKUR RAHMAN" using different shift values

3.To analyze the pattern of character transformations in the encryption process

4.To verify the accuracy of the decryption mechanism

Theory:

The Caesar Cipher, a classical substitution cipher, replaces each letter in the plaintext with another letter from the alphabet by shifting a set number of positions. The shift value is a key to both the encryption and decryption processes. The cipher's simplicity makes it easy to understand, but also relatively easy to break using modern cryptanalysis techniques.

The encryption and decryption are performed using these formulas:

- **Encryption Formula:** $En(x) = (x + k) \bmod 26$

  Where x represents the position of a character in the alphabet, and k is the number of positions to shift.

- **Decryption Formula:** $Dn(x) = (x - k) \bmod 26$

  Where x represents the position of the letter in the ciphertext, and k is the shift value.

This cipher is typically not used for secure communication today due to its vulnerability to simple attacks.

Procedure:

1. **Character Mapping:** The cipher works with the standard English alphabet (A-Z), where each letter is mapped to its respective position in the alphabet.
2. **Encryption Process:** The function takes the message and shifts each letter by the specified number of positions.
3. **Decryption Process:** The decryption function reverses the encryption shift, returning the original message.
4. **Verification:** Test the encryption by applying the same shift for both encryption and decryption to confirm the system works as expected.

Python Code for Caesar Cipher

```
def caesar_encrypt(text, shift):
    encrypted_text = ""
    for char in text:
        if char.isalpha():
            shift_base = ord('A') if char.isupper() else ord('a')
            encrypted_char = chr((ord(char) - shift_base + shift) % 26 + shift_base)
            encrypted_text += encrypted_char
        else:
            encrypted_text += char
    return encrypted_text
def caesar_decrypt(text, shift):
    decrypted_text = " "
    for char in text:
        if char.isalpha():
            shift_base = ord('A') if char.isupper() else ord('a')
            decrypted_char = chr((ord(char) - shift_base - shift) % 26 + shift_base)
            decrypted_text += decrypted_char
        else:
            decrypted_text += char
    return decrypted_text
text = "ASHIKUR RAHMAN"
shift = 3
encrypted_text = caesar_encrypt(text, shift)
decrypted_text = caesar_decrypt(encrypted_text, shift)
print("Original Text:", text)
print("Encrypted Text:", encrypted_text)
print("Decrypted Text:", decrypted_text)
```

Algorithm:

Algorithm Name: CaesarCipherMain

Input: Text "ASHIKUR RAHMAN", Shift value

Output: Encrypted and Decrypted text

Step 1: START

Step 2: Set INPUT_TEXT = "ASHIKUR RAHMAN"

Step 3: Set SHIFT = 3

Step 4: CALL CaesarEncryption(INPUT_TEXT, SHIFT)

Store result in ENCRYPTED_TEXT

Step 5: CALL CaesarDecryption(ENCRYPTED_TEXT, SHIFT)

Store result in DECRYPTED_TEXT

Step 6: Display INPUT_TEXT

Step 7: Display ENCRYPTED_TEXT

Step 8: Display DECRYPTED_TEXT

Step 9: END

Expected Output:

Enter the message: ASHIKUR RAHMAN
Encrypted Message: DVKLNXU UDKPDQ
Decrypted Message: ASHIKUR RAHMAN

Conclusion:

This lab shows how the Caesar Cipher works, allowing us to encrypt and decrypt messages by shifting the alphabet's letters. Despite its simplicity, the cipher is not considered secure for modern cryptographic applications. The process provides insight into basic encryption principles and serves as an introduction to more complex cryptographic methods.

Experiment No:2

Experminet Name: Write a program to implement encryption and decryption using Mono-Alphabetic cipher.

Objective:

This experiment aims to demonstrate the process of encrypting and decrypting messages using the Mono-Alphabetic Cipher. The objective is to substitute each letter of the plaintext with a specific letter from a predefined cipher key. This lab will help in understanding substitution ciphers and how they can be used to encrypt and decrypt messages.

Theory:

The **Mono-Alphabetic Cipher** is a classical substitution cipher in which each letter of the plaintext is replaced by a letter from a fixed cipher alphabet. This method of encryption is based on a one-to-one mapping between the original alphabet and the cipher alphabet.

In this cipher, the key is a shuffled version of the original alphabet. The key remains constant for both encryption and decryption, making it a symmetric cipher. The main advantage of the Mono-Alphabetic Cipher is its simplicity, but it is vulnerable to frequency analysis, meaning it is not secure enough for modern cryptography.

For example:

- Plaintext: **m y n a m e i s a s h i k u r r a h m a n**
- Ciphertext: **D N F Q D T O L Q L I O A X K K Q I D Q F**

The letters of the plaintext are mapped to the ciphertext according to a predefined key. The key can be a random shuffle of the alphabet, but the same key is used for both encryption and decryption.

Procedure:

1. **Define the Cipher Key:** First, create a mapping of each letter of the English alphabet (A-Z) to another letter in a shuffled alphabet. This mapping will be used as the key for encryption and decryption.
2. **Encryption Process:** For encryption, the plaintext is processed letter by letter. Each letter is replaced by the corresponding letter from the cipher key. Non-alphabetic characters, such as spaces, remain unchanged.
3. **Decryption Process:** To decrypt the ciphertext, use the reverse of the cipher key. Each letter in the ciphertext is replaced with the corresponding letter from the original alphabet, restoring the original plaintext.
4. **Testing:** The system is tested with a sample message. The message is encrypted using the cipher key and then decrypted back to verify the correctness of the process.

Python Code for Mono-Alphabetic Cipher

```
normal_char = ['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']
coded_char = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O',
              'P', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K',
              'L', 'Z', 'X', 'C', 'V', 'B', 'N', 'M']
def string_encryption(s):
    encrypted_string = ""
    for char in s:
        if char in normal_char:
            index = normal_char.index(char)
            encrypted_string += coded_char[index]
        else:
            encrypted_string += char
    return encrypted_string
def string_decryption(s):
    decrypted_string = ""
    for char in s:
        if char in coded_char:
            index = coded_char.index(char)
            decrypted_string += normal_char[index]
        else:
            decrypted_string += char
    return decrypted_string
```

```
if __name__ == "__main__":
    plain_text = "my name is Ashikur Rahman"
    print("Plain text:", plain_text)
    encrypted_text = string_encryption(plain_text.lower())
    print("Encrypted message:", encrypted_text)
    decrypted_text = string_decryption(encrypted_text)
    print("Decrypted message:", decrypted_text)
```

Algorithm:

Input: Plain text string

Output: Encrypted and Decrypted messages

Step 1: START

Step 2: Set PLAIN_TEXT = "my name is Ashikur Rahman"

Step 3: Display PLAIN_TEXT

Step 4: Convert PLAIN_TEXT to lowercase

Step 5: CALL String_Encryption(PLAIN_TEXT)

Store result in ENCRYPTED_TEXT

Step 6: Display ENCRYPTED_TEXT

Step 7: CALL String_Decryption(ENCRYPTED_TEXT)

Store result in DECRYPTED_TEXT

Step 8: Display DECRYPTED_TEXT

Step 9: END

Expected Output:

Plain Text: my name is Ashikur Rahman
Encrypted Message: DN FQDT OL QLIOAXK KQIDQF
Decrypted Message: my name is ashikur rahman

Conclusion:

This lab demonstrates the process of using the Mono-Alphabetic Cipher for encryption and decryption. By creating a mapping between each letter of the alphabet and a corresponding letter in the shuffled alphabet, the message is transformed into a ciphertext. The same key is used for both encryption and decryption, making it a symmetric cipher. Although the Mono-Alphabetic Cipher is easy to implement, its vulnerability to frequency analysis makes it unsuitable for secure communications in modern applications.

Experiment No:3

Experminet Name: Write a program to implement encryption and decryption using Brute force attack cipher.

Objective:

The objective of this lab is to demonstrate how a Brute Force Attack can be used to decrypt a message that has been encrypted using a Caesar Cipher. The brute force method attempts every possible key in the encryption process until the original plaintext is found. This experiment aims to understand how simple ciphers like the Caesar Cipher are vulnerable to brute force attacks and to implement this attack method.

Theory:

A Caesar Cipher is a type of substitution cipher where each letter in the plaintext is shifted by a fixed number of positions in the alphabet. For example, with a shift of 3, 'A' would become 'D', 'B' would become 'E', and so on.

While the Caesar Cipher is straightforward to implement, it has a major flaw: if an attacker knows that the cipher used is a Caesar Cipher, they can use a brute force attack to decrypt the message. In a brute force attack, every possible key (shift value) is tested, and the resulting decrypted text is examined to determine the original plaintext.

For example, if the ciphertext is " nz obnf jt btijl" and the shift key is unknown, a brute force attack will try all possible shift values (from 1 to 25) until the correct plaintext "My name is Ashik" is found. In the brute force method, since the key space is small (only 25 possible shifts), it’s computationally feasible to test all possible shifts.

Procedure:

1.First, encrypt a message using a Caesar Cipher. Choose a known shift to encrypt a plaintext message. This will serve as the ciphertext to be attacked using brute force.

2.Then, implement a brute force algorithm that tries all possible Caesar Cipher shift values (from 1 to 25). For each shift, decrypt the ciphertext and display the result.

3.After each decryption attempt, check if the output makes sense as meaningful text. Once the correct shift is found, the message will be correctly decrypted.

4.Finally, test the brute force attack on various ciphertexts, ensuring that the algorithm successfully identifies the correct plaintext by trying all possible keys.

Python Code for Brute Force Attack on Caesar Cipher:

```
def brute_force_decrypt(ciphertext):
    print("\nAttempting all possible decryption shifts:")
    print("-" * 50)
    for shift in range(26):
        decrypted_text = caesar_decrypt(ciphertext, shift)
        print(f"Shift {shift:2d}: {decrypted_text}")
def brute_force_encrypt(plainText):
    print("\nGenerating all possible encryption shifts:")
    print("-" * 50)
    for shift in range(26):
        encrypted_text = caesar_encrypt(plainText, shift)
        print(f"Shift {shift:2d}: {encrypted_text}")
def caesar_encrypt(plainText, shift):
    encrypted_text = ""
    for char in plainText:
        if char.isalpha():
            ascii_base = ord('A') if char.isupper() else ord('a')
            shifted = (ord(char) - ascii_base + shift) % 26
            encrypted_text += chr(ascii_base + shifted)
        else:
            encrypted_text += char
    return encrypted_text
def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""
```

```
    for char in ciphertext:
        if char.isalpha()
            ascii_base = ord('A') if char.isupper() else ord('a')
            shifted = (ord(char) - ascii_base - shift) % 26
            decrypted_text += chr(ascii_base + shifted)
        else:
                decrypted_text += char
    return decrypted_text
def main():
    plaintext = input("Enter the text to encrypt: ")
    print("\n=== ENCRYPTION RESULTS ===")
    print(f"Original text: {plaintext}")
    brute_force_encrypt(plaintext)
    ciphertext = caesar_encrypt(plaintext, 1)
    print("\n=== DECRYPTION RESULTS ===")
    print(f"Encrypted text (shift 1): {ciphertext}")
    brute_force_decrypt(ciphertext)
if __name__ == "__main__":
    main()
```

Algorithm:

Input: User input text

Output: All possible encryption and decryption combinations

Step 1: START

Step 2: GET plaintext from user input

Step 3: DISPLAY "=== ENCRYPTION RESULTS ==="

Step 4: DISPLAY original plaintext

Step 5: CALL Brute_Force_Encrypt(plaintext)

Step 6: ENCRYPT plaintext with shift value 1

Step 7: DISPLAY "=== DECRYPTION RESULTS ==="

Step 8: DISPLAY encrypted text with shift 1

Step 9: CALL Brute_Force_Decrypt(ciphertext)

Step 10: END

Input:

Enter the text to encrypt:
my name is Ashik

Output:

Encryption Results:

Original text: my name is Ashik
Generating all possible encryption shifts:

Shift 0: my name is Ashik
Shift 1: nz obnf jt Btijl
Shift 2: oa pcog ku Cujkm
Shift 3: pb qdph lv Dvkln
...
Shift 25: oa pcog ku Cujkm

Decryption Results:

Encrypted text (shift 1): nz obnf jt Btijl
Attempting all possible decryption shifts:

Shift 0: nz obnf jt Btijl
Shift 1: my name is Ashik
Shift 2: lx mzld hr Zrghj
...
Shift 25: oa pcog ku Cujkm

Conclusion:

In this lab, we demonstrated the use of a brute force attack on the Caesar Cipher. The brute force method successfully decrypted the ciphertext by testing all possible shift values. This highlights a major vulnerability in simple ciphers like the Caesar Cipher: their keys can be easily discovered by trying all possible shifts. While this method is computationally simple and effective for small ciphers, it becomes impractical for more complex encryption methods used in modern cryptography.

Experiment No:4

Experminet Name: Write a program to implement encryption and decryption using Hill cipher.

Objective

The objective of this experiment is to implement encryption and decryption using the Hill cipher technique. The Hill cipher is a polygraphic substitution cipher that uses linear algebra to encrypt and decrypt messages. By understanding and applying matrix operations, we aim to convert plaintext into ciphertext and vice versa securely.

Theory

The Hill cipher was invented by Lester S. Hill in 1929 and is based on linear algebra concepts. It encrypts blocks of text using matrix multiplication. The key is a square matrix (for example, 2x2 or 3x3), and the message is divided into blocks of equal size, matching the matrix size. Each block is multiplied by the matrix to produce the ciphertext.

For encryption, the plaintext message is first converted into numerical values (using A = 0, B = 1, ..., Z = 25). The message is then grouped into vectors of the same size as the key matrix. These vectors are multiplied by the key matrix, and the resulting product is converted back into letters, forming the ciphertext.

For decryption, the inverse of the key matrix is used. The ciphertext is multiplied by the inverse key matrix to retrieve the original plaintext message.

Key Components

1. **Key Matrix:** A square matrix used for encryption.
2. **Inverse Matrix:** A matrix that helps in the decryption process.
3. **Modular Arithmetic:** All operations are done modulo 26 (the number of letters in the alphabet).

1. Input

- Plaintext message: "ASH"
- Key: "ASHIKRAHM"

2. Convert Letters to Numbers:

- Map each letter of the message and the key to its corresponding number:
- Plaintext "ASH": A = 0, S = 18, H = 7
- Key "ASHIKRAHM": A = 0, S = 18, H = 7, I = 8, K = 10, R = 17, A = 0, H = 7, M = 12

3. Construct the Key Matrix

Use the first 9 characters of the key "ASHIKRAHM" to form a 3x3 matrix:

$$\begin{bmatrix} 0 & 18 & 7 \\ 8 & 10 & 17 \\ 0 & 10 & 12 \end{bmatrix}$$

4.Matrix Multiplication for Encryption:

- Divide the plaintext into blocks of 3 (since the key matrix is 3x3):
- Plaintext "ASH" becomes a vector

$$\begin{bmatrix} 0 \\ 18 \\ 7 \end{bmatrix}$$

- Multiply the key matrix by the plaintext vector:

$$\begin{bmatrix} 0 & 18 & 7 \\ 8 & 10 & 17 \\ 0 & 10 & 12 \end{bmatrix} \times \begin{bmatrix} 0 \\ 18 \\ 7 \end{bmatrix} = \begin{bmatrix} 7 \\ 10 \\ 2 \end{bmatrix}$$

5. **Convert Numbers Back to Letters:**

- Convert the numbers back into letters using the reverse mapping:
- $7 \rightarrow$ H, $10 \rightarrow$ K, $2 \rightarrow$ C
- Thus, the ciphertext is: "JNC".

6.**Decryption:**

- To decrypt, calculate the inverse of the key matrix modulo 26.
- Find the inverse matrix using methods such as the adjoint matrix or extended Euclidean algorithm
- Multiply the inverse matrix by the ciphertext vector to obtain the original message.

7.**Output:**

- Ciphertext: "JNC"
- Decrypted message: "ASH"

Code Implementation (Python):

```
keyMatrix = [[0] * 3 for i in range(3)]
messageVector = [[0] for i in range(3)]
cipherMatrix = [[0] for i in range(3)]
def getKeyMatrix(key):
    k = 0
    for i in range(3):
        for j in range(3):
            keyMatrix[i][j] = ord(key[k]) % 65
            k += 1
def encrypt(messageVector):
    for i in range(3):
        for j in range(1):
            cipherMatrix[i][j] = 0
            for x in range(3):
                cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
            cipherMatrix[i][j] = cipherMatrix[i][j] % 26
def HillCipher(message, key):
    if len(message) != 3:
        raise ValueError("Message length must be 3 for this implementation.")
    getKeyMatrix(key)
    for i in range(3):
        messageVector[i][0] = ord(message[i]) % 65
    encrypt(messageVector)
    CipherText = []
```

```
    for i in range(3):
        CipherText.append(chr(cipherMatrix[i][0] + 65))  # Convert number back to character
    print("Ciphertext: ", "".join(CipherText))
def main():
    message = "ASH"  # You can change this message
    print("Input message:", message)
    key = "ASHIKRAHM"
print("Key:", key
    HillCipher(message, key)
if __name__ == "__main__":
    main()
```

Algorithm:

1. **START**
2. **GET** the plaintext message and key.
3. **CREATE** a 3x3 key matrix from the key string.
4. **CONVERT** the plaintext into a vector of numbers (A=0, B=1, ..., Z=25).
5. **ENCRYPT** the plaintext by multiplying the key matrix with the message vector and applying modulo 26.
6. **CONVERT** the resulting cipher matrix back to characters.
7. **DISPLAY** the ciphertext.
8. **END**

Conclusion: In this experiment, the Hill cipher was implemented to encrypt and decrypt a given message. The process involved matrix multiplication, modular arithmetic, and the use of key matrices for both encryption and decryption. This cipher demonstrates how linear algebra can be applied to achieve secure communication. By using the inverse of the key matrix, we successfully decrypted the ciphertext back to the original message.

Experiment No:5

Experminet Name: Write a program to implement encryption using Playfair cipher.

Objective:

The objective of this lab is to implement encryption using the **Playfair Cipher**. This cipher encrypts pairs of letters from a plaintext message using a keyword, providing a more secure encryption compared to classical ciphers like Caesar Cipher.

Theory:

The **Playfair Cipher** is a digraph substitution cipher invented by Charles Wheatstone and later promoted by Lord Playfair. It encrypts pairs of letters (digraphs) instead of single letters. The encryption process is based on the following steps:

1. Key Matrix: The cipher uses a 5x5 matrix generated from a keyword. The matrix contains all the letters of the alphabet (except 'J', which is typically combined with 'I' to fit 25 letters). The keyword is written into the matrix, and the remaining letters of the alphabet are filled in.

2. **Encryption Process:**

- The plaintext is broken into pairs of letters (digraphs).
- If the pair contains identical letters, an 'X' is inserted between them.
- If the plaintext has an odd number of characters, an 'X' is added at the end.
- Each pair of letters is encrypted by locating them in the key matrix and applying a set of rules based on their positions in the matrix.

3. **Encryption Rules:**

- If both letters of a pair are in the same row, replace them with the letters to their immediate right (wrapping around to the beginning of the row if needed).
- If both letters of a pair are in the same column, replace them with the letters immediately below (wrapping to the top if needed).
- If the letters form a rectangle, replace them with the letters on the same row but in the opposite corners of the rectangle.

Procedure:

1.Prepare the Key Matrix:

- Choose a keyword and remove any duplicate letters.
- Create a 5x5 matrix using the keyword, and then fill in the remaining letters of the alphabet (excluding 'J').

2.Prepare the Plaintext:

- Divide the plaintext into digraphs (pairs of letters).
- f a pair has repeated letters, replace the second one with 'X'.
- If the plaintext has an odd number of letters, append 'X' at the end.

3.Encryption:

- For each digraph, find the letters in the key matrix.
- Apply the encryption rules to transform the digraph into ciphertext.

4. Output the Ciphertext:

- After all digraphs have been encrypted, display the resulting ciphertext.

Python Implementation

```
def create_matrix(key):
    # Create a 5x5 matrix using the key
    key = key.upper().replace('J', 'I')  # Replace J with I as per Playfair rules
    matrix = []
    used_chars = set()
    for char in key:
        if char.isalpha() and char not in used_chars:
            matrix.append(char)
            used_chars.add(char)

    for char in 'ABCDEFGHIKLMNOPQRSTUVWXYZ':  # Note: No J
        if char not in used_chars:
            matrix.append(char)
            used_chars.add(char)

    return [matrix[i:i+5] for i in range(0, 25, 5)]
def find_position(matrix, char):
    for i in range(5):
        for j in range(5):
            if matrix[i][j] == char:
                return i, j
    return None

def prepare_text(text):
    text = ''.join(c.upper() for c in text if c.isalpha())
    text = text.replace('J', 'I')
      pairs = []
    i = 0
    while i < len(text):
        if i == len(text) - 1:
            pairs.append(text[i] + 'X')
            break
        elif text[i] == text[i + 1]:
            pairs.append(text[i] + 'X')
            i += 1
        else:
            pairs.append(text[i] + text[i + 1])
            i += 2

    return pairs

def encrypt_pair(matrix, pair):
```

```
    row1, col1 = find_position(matrix, pair[0])
    row2, col2 = find_position(matrix, pair[1])

    if row1 == row2:  # Same row
        return (matrix[row1][(col1 + 1) % 5] +
                matrix[row2][(col2 + 1) % 5])
    elif col1 == col2:  # Same column
        return (matrix[(row1 + 1) % 5][col1] +
                matrix[(row2 + 1) % 5][col2])
    else:  # Rectangle case
        return matrix[row1][col2] + matrix[row2][col1]

def playfair_encrypt(key, plaintext):
    matrix = create_matrix(key)
    pairs = prepare_text(plaintext)
    ciphertext = ''

    for pair in pairs:
        ciphertext += encrypt_pair(matrix, pair)

    return ciphertext

if __name__ == "__main__":
    key = "ASHIKUR"
    plaintext = "TECHJOINT"

    print("Key:", key)
    print("Original text:", plaintext
    encrypted = playfair_encrypt(key, plaintext)
    print("\nEncrypted text:", encrypted)
```

Expected Output

Key: ASHIKUR
Plaintext: TECHJOINT
Ciphertext: ZCQHUCZJ

Algorithm for Playfair Cipher Encryption:

1. START
2. PREPROCESS the key by converting it to uppercase, removing duplicate letters, and replacing 'J' with 'I'.
3. CREATE the 5x5 key matrix by adding characters from the key and filling in the remaining letters (excluding 'J').
4. PREPROCESS the plaintext by converting it to uppercase, replacing 'J' with 'I', splitting it into digraphs (pairs of letters), inserting 'X' if any pair has duplicate letters, and adding 'X' at the end if the length is odd.
5. ENCRYPT each digraph by checking if the letters are in the same row, column, or form a rectangle. For letters in the same row, replace them with letters to the right. For letters in the same column, replace them with letters below. For letters forming a rectangle, swap them diagonally.
6. OUTPUT the ciphertext.
7. END

Conclusion:

In this lab, we successfully implemented encryption using the **Playfair Cipher**, a digraph substitution cipher that enhances security by encrypting pairs of letters. The key matrix, derived from the keyword, played a crucial role in the encryption process. By applying specific rules for handling letters in the same row, column, or rectangle, we were able to generate the ciphertext from the plaintext message.

The **Playfair Cipher** is more secure than simpler ciphers like the Caesar Cipher, as it encrypts two letters at a time and uses a key matrix to create a more complex encryption. While it is not as secure as modern cryptographic algorithms, it demonstrates important concepts in classical cryptography and provides a foundation for understanding more advanced encryption techniques.

Through this implementation, we gained hands-on experience with key generation, text preparation, and applying encryption rules, which are essential for mastering the principles of cryptography.

Experiment No:6

Experminet Name: Write a program to implement Decryption using Playfair cipher.

Objective:

The objective of this lab is to implement the decryption process of the **Playfair Cipher**, which is a digraph substitution cipher used to securely encrypt messages. The Playfair Cipher works by using a 5x5 key matrix and applying certain rules to decrypt the ciphertext back into plaintext. By performing this decryption, we aim to retrieve the original message using the key and the given ciphertext.

Theory:

The **Playfair Cipher** is a symmetric encryption technique that encrypts pairs of letters (digraphs) rather than individual letters. The encryption is performed by creating a 5x5 matrix of letters from a keyword, and the decryption process is based on applying the reverse of the encryption rules. During decryption:

1. If both letters of the digraph appear in the same row of the matrix, they are replaced by the letters to their immediate left.
2. If both letters appear in the same column, they are replaced by the letters immediately above them.
3. If the letters form a rectangle, the letters are replaced with the ones on the opposite corners.

Procedure:

1.Prepare the Key:

- Convert the key to uppercase.
- Remove any duplicate letters.
- Replace 'J' with 'I' (since the Playfair Cipher uses only 25 letters).

2. Create the 5x5 Key Matrix::

- The key is used to fill the 5x5 matrix, placing the characters of the key in the matrix.
- After the key characters, the remaining alphabet letters (excluding 'J') are added to fill the rest of the matrix.

3.Preprocess the Ciphertext:

- Convert the ciphertext to uppercase.
- Replace 'J' with 'I'.
- Split the ciphertext into digraphs (pairs of letters).

- If the ciphertext has an odd length, append 'X' at the end.

4.Preprocess the Ciphertext:

- If the letters are in the same row of the matrix, replace them with the letters to their immediate left.
- If the letters are in the same column, replace them with the letters immediately above.
- If the letters form a rectangle, replace them with the letters at the opposite corners of the rectangle.

5.Reconstruct the Plaintext:

- Combine the decrypted pairs to reconstruct the original message.
- If 'X' was added during the preprocessing, remove it (if it's not part of the original message).

Code Implementation for Decryption:

```
def create_matrix(key):
    key = key.upper().replace('J', 'I')  # Replace J with I as per Playfair rules
    matrix = []
    used_chars = set()
    for char in key:
        if char.isalpha() and char not in used_chars:
            matrix.append(char)
            used_chars.add(char)
    for char in 'ABCDEFGHIKLMNOPQRSTUVWXYZ':  # Note: No J
        if char not in used_chars:
            matrix.append(char)
            used_chars.add(char)
    return [matrix[i:i+5] for i in range(0, 25, 5)]
def find_position(matrix, char):
```

```
    for i in range(5):
        for j in range(5):
            if matrix[i][j] == char:
                return i, j
    return None
def decrypt_pair(matrix, pair):
    row1, col1 = find_position(matrix, pair[0])
    row2, col2 = find_position(matrix, pair[1])
    if row1 == row2:  # Same row
        return (matrix[row1][(col1-1)%5], matrix[row2][(col2-1)%5])
    elif col1 == col2:  # Same column
        return (matrix[(row1-1)%5][col1], matrix[(row2-1)%5][col2])
    else:  # Rectangle case
        return (matrix[row1][col2], matrix[row2][col1])
def playfair_decrypt(ciphertext, key):
    ciphertext = ciphertext.upper().replace('J', 'I')
    ciphertext = ''.join(c for c in ciphertext if c.isalpha())
    matrix = create_matrix(key)
    decrypted = []
    for i in range(0, len(ciphertext), 2):
        if i+1 < len(ciphertext):
            pair = decrypt_pair(matrix, (ciphertext[i], ciphertext[i+1]))
            decrypted.extend(pair)
    return ''.join(decrypted)
def main():
```

```
    key = "ASHIKUR"
    ciphertext = "NMBISQAQPZ"  # Example ciphertext
    print("Key:", key)
    print("Ciphertext:", ciphertext)
    decrypted = playfair_decrypt(ciphertext, key)
    print("Decrypted:", decrypted)
if __name__ == "__main__":
    main()
```

Expected Output:

Key: ASHIKUR

Ciphertext: NMBISQAQPZ

Decrypted Message: TECHIOINTX

Algorithm for Playfair Cipher Decryption:

1 .PREPARE the key matrix using the unique characters from the key and remaining letters (A-Z, excluding 'J').

2.SPLIT the ciphertext into pairs of letters.

3.For each pair:

- If letters are in the same row, replace them with letters to the left.
- If letters are in the same column, replace them with letters above.
- If letters form a rectangle, swap to opposite corners.

4.COMBINE the decrypted pairs to get the plaintext.
5.REMOVE any filler 'X' if it was added during encryption.

Conclusion:

The Playfair Cipher is a classical encryption algorithm that provides enhanced security by encrypting digraphs instead of individual letters. The decryption process is the reverse of the encryption process, where the key matrix is used to reverse the encryption rules and retrieve the original message. In this lab, we successfully decrypted the ciphertext "NMBISQAQPZ" into the plaintext "TECHIOINTX" using the Playfair Cipher. The algorithm demonstrates the importance of key management and matrix manipulation in encryption and decryption processes, as well as the role of symmetry in cryptography.

Experiment No:7

Experminet Name: Write a program to implement encryption using Poly-Alphabetic cipher.

Objective:

The objective of this lab is to implement encryption using the **Poly-Alphabetic Cipher**. This cipher is a form of substitution cipher in which each letter in the plaintext is shifted according to a repeating key sequence. By using multiple alphabets (polyalphabetic), this cipher provides enhanced security over simple substitution ciphers, making it more resistant to frequency analysis.

Theory:

The **Poly-Alphabetic Cipher** is an encryption technique that uses a repeating sequence of keys to shift each letter of the plaintext by different amounts. One of the most commonly used methods in polyalphabetic ciphers is the **Vigenère Cipher**, where each letter in the plaintext is shifted according to the corresponding letter in the key. For example, if the key is "ICE," each letter in the message will be shifted by the letters 'I', 'C', and 'E' repeatedly, forming a cycle.

For each letter in the plaintext:

1. Convert the letter to its numerical position (A=0, B=1, ..., Z=25).
2. Shift it by the corresponding letter in the key, cycling through the key repeatedly.
3. Convert the shifted number back to a letter.

This technique prevents a direct one-to-one mapping of letters, making the encrypted message harder to decipher without the key.

Procedure:

1. Input the Plaintext and Key:

- Get the message to be encrypted and the key from the user.

2. **Repeat Key to Match Plaintext Length:**

- If the key is shorter than the plaintext, repeat it until it matches the plaintext length.

3.**Encrypt Each Character:**

- For each character in the plaintext:
- Convert the character to a number (A=0 to Z=25).
- Use the corresponding character in the repeated key to determine the shift amount.
- Shift the character according to the key's letter.

4. **Generate Ciphertext:**

- Convert the shifted numbers back to letters to form the encrypted message.

Code Implementation for Poly-Alphabetic Cipher Encryption:

```
def generate_key_encrypt(message, key):
    key = list(key)
    if len(message) == len(key):
        return key
    else:
        for i in range(len(message) - len(key)):
            key.append(key[i % len(key)])
    return "".join(key)
def encryption():
        message = input("Enter message to encrypt: ")
    key = input("Enter key: ")
    key = generate_key_encrypt(message, key)
    cipher_text = []
    for i in range(len(message)):
        if message[i].isalpha():
            # Handle uppercase letters
            if message[i].isupper():
```

```
            value = (ord(message[i]) + ord(key[i].upper()) - 2 * ord('A')) % 26
            cipher_text.append(chr(value + ord('A')))
        else:
            value = (ord(message[i]) + ord(key[i].lower()) - 2 * ord('a')) % 26
            cipher_text.append(chr(value + ord('a')))
    else:
        cipher_text.append(message[i])
encrypted_text = "".join(cipher_text)
print(f"\nEncrypted Message: {encrypted_text}")
if __name__ == "__main__":
    print("Poly-Alphabetic Cipher Encryption")
    print("--------------------------------")
    encryption()
```

Sample Output:

For the input:

- **Plaintext**: Mr Ashikur Rahman
- **Key**: ICE

The output will be:

Plaintext: Mr Ashikur Rahman
Key: ICE
Encrypted Message: UT IULQMYZ VIJQIP

Algorithm for Poly-Alphabetic Cipher Encryption:

1. INPUT plaintext and key
2. CONVERT plaintext and key to uppercase
3. EXTEND key to match the length of plaintext
4. INITIALIZE an empty ciphertext
5. For each character in plaintext:

    - If alphabetic, SHIFT by key character position and ADD to ciphertext
    - If non-alphabetic, ADD directly to ciphertext

6. OUTPUT the ciphertext

Conclusion:

The Poly-Alphabetic Cipher offers a significant improvement in security over monoalphabetic ciphers by using multiple letters in the key to encrypt the message. This method makes it more challenging to decrypt without knowing the key, as each letter in the plaintext can be shifted by a different amount, based on the key's pattern. This lab successfully demonstrates encryption using the Poly-Alphabetic Cipher, providing insight into the advantages of polyalphabetic encryption methods over simpler ciphers in cryptographic security.

Experiment No:8

Experminet Name: Write a program to implement decryption using Poly-Alphabetic cipher.

Objective:

The objective of this lab is to implement decryption using the **Poly-Alphabetic Cipher**. This cipher is a type of substitution cipher that uses a repeating key to determine the decryption shift for each character. The goal is to recover the original plaintext message from the ciphertext by reversing the encryption process.

Theory:

The **Poly-Alphabetic Cipher** is a type of cipher that uses multiple alphabets based on a repeating key, commonly implemented using the **Vigenère Cipher** method. In decryption, each letter in the ciphertext is shifted backward according to the position of the corresponding letter in the key. By shifting each letter according to this dynamic key-based scheme, the original message can be accurately reconstructed.

The decryption process involves:

1. Converting each character in the ciphertext back to a number (A=0, B=1, ..., Z=25).
2. Shifting each character by subtracting the corresponding letter in the key (also converted to a number).
3. Converting the result back to letters to form the decrypted message.

Procedure:

1. Input the Ciphertext and Key:

- Prompt the user for the encrypted message and the key.

2. **Repeat Key to Match Plaintext Length:**

- Repeat the key as necessary to match the length of the ciphertext.

3. **Decrypt Each Character:**

- For each character in the cipertext:
- Convert it to a number, find the corresponding shift from the key, and shift backward.
- Apply modulo 26 to ensure it falls within the alphabet range.

4. **Generate Plaintext:**

- Convert the shifted values back to characters to reconstruct the original message.

Code Implementation for Poly-Alphabetic Cipher Decryption:

```
def generate_key_decrypt(cipher_text, key):
    key = list(key)
    if len(cipher_text) == len(key):
        return key
    else:
        for i in range(len(cipher_text) - len(key)):
            key.append(key[i % len(key)])
    return "".join(key)
def decryption():
    """Decrypt a message using Poly-Alphabetic cipher"""
    cipher_text = input("Enter message to decrypt: ")
    key = input("Enter key: ")
    key = generate_key_decrypt(cipher_text, key)
    plain_text = []
    for i in range(len(cipher_text)):
        if cipher_text[i].isalpha():
            if cipher_text[i].isupper():
                value = (ord(cipher_text[i]) - ord(key[i].upper()) + 26) % 26
                plain_text.append(chr(value + ord('A')))
            else:
                value = (ord(cipher_text[i]) - ord(key[i].lower()) + 26) % 26
```

```
            plain_text.append(chr(value + ord('a')))
        else:
            plain_text.append(cipher_text[i])
    decrypted_text = "".join(plain_text)
    print(f"\nDecrypted Message: {decrypted_text}")
if __name__ == "__main__":
    print("Poly-Alphabetic Cipher Decryption")
    print("--------------------------------")
    decryption()
```

Sample Output:

For the input:

- **Ciphertext**: Ut Iulqmyz Vijqip
- **Key**: ICE

The output will be:

Ciphertext: Ut Iulqmyz Vijqip
Key: ICE
Decrypted Message: Mr Ashikur Rahman

Algorithm for Poly-Alphabetic Cipher Decryption

1. INPUT the ciphertext and key

2.CONVERT both ciphertext and key to uppercase

3.EXTEND the key to match the length of the ciphertext

4.INITIALIZE an empty string for the plaintext

5.For each character in the ciphertext:

- If alphabetic, CALCULATE the original position by subtracting the key character's position and applying modulo 26
- CONVERT the result back to a character and ADD to the plaintext
- If non-alphabetic, ADD directly to the plaintext

6.OUTPUT the final plaintext

Conclusion:

The Poly-Alphabetic Cipher, using a repeating key, provides strong encryption by altering each character with a unique shift determined by the key. Decrypting the message involves reversing these shifts according to the key. This lab demonstrates how to successfully retrieve the original message from the ciphertext using the Poly-Alphabetic Cipher decryption technique, providing valuable insight into the effectiveness of polyalphabetic ciphers in cryptographic security.

Experiment No:9

Experminet Name: Write a program to implement encryption using Vernam cipher.

Objective:

The objective of this lab is to implement encryption using the **Vernam Cipher** (also known as the One-Time Pad cipher). This cipher employs a unique, random key equal in length to the plaintext to create ciphertext, ensuring that each encryption is unique and theoretically unbreakable if the key is used only once.

Theory:

The **Vernam Cipher** is a symmetric encryption technique where each character in the plaintext is combined with a corresponding character in the key using a bitwise XOR operation. This operation produces a unique encrypted character that depends on both the plaintext and the key. The Vernam Cipher is unbreakable when:

1. The key is truly random.
2. The key length matches the plaintext length.
3. The key is used only once.

In this method:

1. Each character in the plaintext is XORed with the corresponding character in the key.
2. This XOR operation yields an encrypted character, producing the ciphertext.

Procedure:

1 Input the Plaintext and Key:

- Ensure the key length is equal to the plaintext length.

2. **Convert Plaintext and Key Characters to Numeric Values:**

- Map each character to its corresponding ASCII value or its position in the alphabet.

3. **Perform XOR Operation on Each Character:**

- For each character, XOR the plaintext character with the corresponding key character.
- Convert the result back to a character.

4. **Generate Plaintext:**

- Combine all encrypted characters to form the final ciphertext.

Code Implementation for Vernam Cipher Encryption:

```
def stringEncryption(text, key):
    cipherText = ""
    cipher = []
    for i in range(len(key)):
        cipher.append(ord(text[i]) - ord('A') + ord(key[i]) - ord('A'))
    for i in range(len(key)):
        if cipher[i] > 25:
            cipher[i] = cipher[i] - 26
    for i in range(len(key)):
        x = cipher[i] + ord('A')
        cipherText += chr(x)
    return cipherText
plainText = "howareyou"
key = "ncbtzqarx"
encryptedText = stringEncryption(plainText.upper(), key.upper())
print("Cipher Text - " + encryptedText)
```

Sample Output:

For the input:

- **Plaintext**: howareyou
- **Key**: ncbtzqarx

The output will be:

Plaintext: howareyou
Key: ncbtzqarx
Cipher Text: UQXTQUYFR

Algorithm for Vernam Cipher Encryption

1. INPUT the ciphertext and key

2. CHECK that the length of the key matches the length of the plaintext. If not, RETURN an error.

3. INITIALIZE an empty string for the ciphertext.

4. For each character in the plaintext and corresponding character in the key:

- CONVERT each character to its position in the alphabet (e.g., 'a' = 0, 'b' = 1, ..., 'z' = 25).
- PERFORM the XOR operation on the positions of the plaintext character and key character.
- MAP the resulting position back to a character in the alphabet.
- ADD the resulting character to the ciphertext string.

5. OUTPUT the ciphertext as the encrypted message.

Conclusion:

The Vernam Cipher, through the XOR operation with a unique key, provides an encryption method that is theoretically unbreakable if the key is used only once and remains secret. This lab demonstrates the encryption process of the Vernam Cipher, showcasing how each character in the plaintext can be uniquely encrypted by pairing it with a key character, illustrating the fundamental principles of a one-time pad.

Experiment No:10

Experminet Name: Write a program to implement Deycryption using Vernam cipher.

Objective:

The objective of this lab is to implement decryption using the **Vernam Cipher**. This cipher, also known as the One-Time Pad, is a symmetric encryption technique that uses a unique, random key equal in length to the ciphertext for decryption. When the key is known, this method provides a theoretically unbreakable means of recovering the original message.

Theory:

The **Vernam Cipher** relies on a bitwise XOR operation to encrypt and decrypt messages. The XOR operation is reversible, meaning that if a character in the plaintext is XORed with a character in the key to produce the ciphertext, then XORing the ciphertext character with the same key character will recover the original plaintext character. This decryption process relies on:

1. The ciphertext and key lengths being equal.
2. The key being known and used only once.

The decryption process involves:

1. Each character in the ciphertext is XORed with the corresponding character in the key.
2. This operation yields the original plaintext character.

Procedure:

1 Input the Ciphertext and Key:

- Ensure the key length matches the ciphertext length.

2. **Convert Cipertext and Key Characters to Numeric Values:**

- Map each character to its corresponding position in the alphabet.

3. **Perform XOR Operation on Each Character:**

- XOR each character in the ciphertext with the corresponding character in the key..
- Convert the result back to a character.

4. **Construct the Plaintext:**

- Combine all decrypted characters to form the plaintext.

Code Implementation for Vernam Cipher Decryption:

```
def stringDecryption(s, key):
    plainText = ""
    plain = []
    s = s.upper()
    key = key.upper()
    for i in range(len(key)):
        plain.append(ord(s[i]) - ord('A') - (ord(key[i]) - ord('A')))
    for i in range(len(key)):
        if plain[i] < 0:
            plain[i] = plain[i] + 26
    for i in range(len(key)):
        x = plain[i] + ord('A')
        plainText += chr(x)
    return plainText
cipherText = "uqxtquyfr"  # Encrypted text from the encryption script output
key = "ncbtzqarx"
decryptedText = stringDecryption(cipherText, key)
print("Decrypted Message - " + decryptedText)
```

Sample Output:

For the input:

- **Ciphertext**: uqxtquyfr
- **Key**: ncbtzqarx

The output will be:

Ciphertext: uqxtquyfr
Key: ncbtzqarx
Decrypted Message: HOWAREYOU

Algorithm for Vernam Cipher Decryption

1. INPUT the ciphertext and key

2. **Check** if ciphertext and key lengths match

3. **For each character** in ciphertext and key:

- Convert characters to numeric values.
- Perform XOR operation.
- Convert result back to character.

4. Output **the decrypted message**

Conclusion:

This program demonstrates the decryption process of the Vernam Cipher, where each character in the ciphertext is XORed with the corresponding character in the key to recover the original plaintext. This lab emphasizes the Vernam Cipher's security, which remains unbreakable if the key is used only once, is truly random, and is kept secret. This exercise illustrates the practical decryption mechanism of the Vernam Cipher, validating its use as a one-time pad in cryptographic applications.