Pabna University of Science and Technology

Faculty of Engineering and Technology

Department of Information and Communication Engineering

Practical Lab Report

Course Code: ICE-4108

Course title: Cryptography and Computer Security Sessional.

Submitted By:	Submitted To:
Naimur Rahman Roll: 200626 Session:2019-2020 4th Year 1st semester Department of Information and Communication Engineering, PUST	**Md. Anwar Hossain** Professor Department of Information and Communication Engineering, PUST

Date of Submission: 12-11-2024 Signature

Experiment No:1

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

Objective:

1. To develop a foundational cryptographic structure based upon the Caesar Cipher technique.

2. To execute the processes of encrypting and decrypting the phrase "Naimur Rahman" utilizing a range of shift values.

3. To analyze the attributes of character alterations during the encryption process.
4. To evaluate the accuracy of the decryption technique.

Theory:

The Caesar Cipher, categorized as a traditional substitution cipher, substitutes each symbol in the plaintext with a corresponding symbol from the alphabet by advancing a predetermined number of positions. This pre-established shift value serves as a critical key for both the processes of encryption and decryption. While the cipher's straightforward architecture facilitates comprehension, it simultaneously renders itself susceptible to analysis through contemporary cryptanalytic methodologies.

The operations of encryption and decryption are conducted through the following mathematical expressions:

Encryption Formula: En(x) = (x + k) mod 26

In this expression, x denotes the position of a character within the alphabet, whereas k represents the number of positions to be advanced.

Decryption Formula: Dn(x) = (x - k) mod 26

In this context, x indicates the position of the character in the ciphertext, and k signifies the shift value. At present, this cipher is primarily regarded as inadequate for secure communication due to its susceptibility to basic attack strategies.

Procedure:

Character Mapping:

The cipher operates utilizing the conventional English alphabet (A-Z), wherein each letter is assigned its corresponding position within the alphabet. Encryption Process: The function processes the message by shifting each character by a specified number of positions.

Decryption Process:

The decryption function retrieves the original message by reversing the shift implemented during encryption. Verification: Validate the integrity of the encryption by employing the identical shift for both the encryption and decryption processes to ensure that the system functions as designed.

Python Code for Caesar Cipher

```
def caesar_cipher(text, shift, decrypt=False):
    shift = -shift if decrypt else shift
    result = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result

# Main code
text = "NAIMUR RAMHAN"
shift = 3
encrypted_text = caesar_cipher(text, shift)
decrypted_text = caesar_cipher(encrypted_text, shift, decrypt=True)

print("Original Text:", text)
print("Encrypted Text:", encrypted_text)
print("Decrypted Text:", decrypted_text)
```

Output: Original Text: NAIMUR RAMHAN

Encrypted Text: QDLPEX UDPKDQ

Decrypted Text: NAIMUR RAMHAN

Algorithm:

Input: Text " NAIMUR RAMHAN "

Output: Encrypted and Decrypted text

Step 1: Initiate

Step 2: Assign INPUT_T = " NAIMUR RAMHAN "

Step 3: Assign SHIFT = 3

Step 4: Invoke CaesarEncryption(INPUT_T, SHIFT)

Preserve resultant value in ENCRYPTED_T

Step 5: Invoke CaesarDecryption(ENCRYPTED_T, SHIFT)

Preserve resultant value in DECRYPTED_T

Step 6: Present INPUT_T

Step 7: Present ENCRYPTED_T

Step 8: Present DECRYPTED_T

Step 9: CONCLUDE

Conclusion: This laboratory exercise elucidates the functional principles of the Caesar Cipher, facilitating the encryption and decryption of messages via systematic alphabetic shifts; however, despite its simplicity, this cipher is considered insufficient for modern cryptographic applications, serving instead as a foundational introduction to basic encryption concepts and a gateway to more advanced cryptographic methodologies.

Experiment No:2

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

Objective: The Mono-Alphabetic Cipher constitutes a conventional form of substitution cipher wherein each symbol of the plaintext is replaced by a symbol from a fixed cipher alphabet. This encryption methodology is predicated upon a one-to-one correlation between the original alphabet and the cipher alphabet.

Theory: Within the framework of this cipher, the key consists of a randomized permutation of the original alphabet. The key retains its constancy throughout both the encryption and decryption phases, thereby designating it as a symmetric cipher. The primary merit of the Mono-Alphabetic Cipher is its simplicity; nonetheless, it is vulnerable to frequency analysis, which renders it inadequate when evaluated against contemporary cryptographic criteria.

For instance:

Plaintext: m y n a m e i s n a i m u r r a h m a n

Ciphertext: G Y F W C H G G D T Y S F H V V A

The characters present in the plaintext exhibit a correlation with the ciphertext based on a predetermined cryptographic key. This key may encompass a random permutation of the alphabet; however, the same key is utilized for both the encryption and decryption operations.

Procedure:

Establish the Cipher Key: Initially, it is imperative to create a systematic correspondence between each letter of the English alphabet (A-Z) and a letter derived from a randomized sequence of the alphabet. This systematic correspondence will function as the key for both the encryption and decryption operations.

Encryption Methodology: In the encryption phase, the plaintext is subjected to a letter-by-letter transformation process. Each letter is replaced by its corresponding letter derived from the cipher key, while non-alphabetic symbols, including spaces, remain unchanged.

Decryption Methodology: For the objective of decrypting the ciphertext, the inverse of the cipher key is utilized. Each letter in the ciphertext is replaced with the corresponding letter from the original alphabet, thus facilitating the restoration of the plaintext.

Verification: The system undergoes testing with a representative message. The message is encrypted utilizing the cipher key and subsequently decrypted to confirm the accuracy of the methodology.

Code:

```
normal_chars = ['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_chars = ['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 encrypt_message(message):
    encrypted_message = ""
    for char in message:
        if char in normal_chars:
            index = normal_chars.index(char)
            encrypted_message += coded_chars[index]
        else:
            encrypted_message += char
```

```
    return encrypted_message


def decrypt_message(message):
    decrypted_message = ""
    for char in message:
        if char in coded_chars:
            index = coded_chars.index(char)
            decrypted_message += normal_chars[index]
        else:
            decrypted_message += char
    return decrypted_message


# Main code
plain_text = "my name is Naimur"
print("Plain text:", plain_text)
encrypted_text = encrypt_message(plain_text.lower())
print("Encrypted message:", encrypted_text)
decrypted_text = decrypt_message(encrypted_text)
print("Decrypted message:", decrypted_text)
```

Output:

Plain text: my name is Naimur

Encrypted message: dk ftlt eo Fqokyi

Decrypted message: my name is naimur

Algorithm:

Input: Unadulterated text string

Output: Encoded communication and Decoded representations

Step 1: Initialize

Step 2: Assign PLAIN_T= "my name is naimur"

Step 3: Exhibit PLAIN_T

Step 4: Transform PLAIN_T to lowercase

Step 5: Invoke String_Encryption(PLAIN_T)

Preserve outcome in ENCRYPTED_T

Step 6: Exhibit ENCRYPTED_T

Step 7: Invoke String_Decryption(ENCRYPTED_T)

Preserve outcome in DECRYPTED_T

Step 8: Exhibit DECRYPTED_T

Step 9: Conclude

Conclusion: This experimental procedure clarifies the technique of utilizing the Mono-Alphabetic Cipher for the purposes of both encryption and decryption. By creating a mapping between each character of the conventional alphabet and an associated symbol from a rearranged alphabet, the initial message is transformed into ciphertext. The same key is employed in both the encryption and decryption stages, thus classifying it as a symmetric cipher. Although the implementation of the Mono-Alphabetic Cipher is straightforward, its vulnerability to frequency analysis renders it unsuitable for secure communication in modern scenarios.

Experiment No:3

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

Objective: The purpose of this laboratory exercise is to demonstrate the effectiveness of a Brute Force Attack in the decryption of a message encoded via a Caesar Cipher. The brute force method aims to examine every potential key within the encryption scheme until the genuine plaintext is accurately uncovered. This study seeks to clarify the vulnerabilities of basic ciphers, such as the Caesar Cipher, to brute force attack strategies and to carry out this specific assault technique.

Theory: A Caesar Cipher represents a type of substitution cipher in which each character in the plaintext is shifted by a specified number of positions within the alphabet. For example, with a shift of 3, 'A' becomes 'D', 'B' turns into 'E', and so forth.

While the execution of the Caesar Cipher is ostensibly straightforward, it harbors a considerable weakness: if an opponent discerns that the cipher utilized is a Caesar Cipher, they may engage in a brute force attack to decipher the information. In a brute force assault, each conceivable key (shift value) is methodically assessed, and the resulting decrypted output is examined to identify the original plaintext.

For example, if the ciphertext is "nz obnf jt btijl" and the shift key remains unknown, a brute force attack will systematically investigate all viable shift values (ranging from 1 to 25) until the correct plaintext "My name is Ashik" is revealed. In the brute force methodology, considering the restricted key space (only 25 possible shifts), it is computationally efficient to analyze all potential shifts.

Procedure:

1. Initially, employ a Caesar Cipher to encode a message. Choose a predetermined shift value for the encryption of a plaintext message, which will serve as the ciphertext for subsequent brute force evaluation.

2. Subsequently, construct a brute force algorithm that systematically examines all feasible shift values of the Caesar Cipher, ranging from 1 to 25. For each shift value, execute decryption on the ciphertext and present the resultant output.

3. During each decryption attempt, evaluate whether the resultant output constitutes coherent and meaningful text. The appropriate shift will facilitate the accurate decryption of the message.

4. Ultimately, execute the brute force attack on various ciphertexts, verifying that the algorithm adeptly identifies the correct plaintext by comprehensively investigating all potential keys..

Code:

```
def brute_force_decrypt(ct):
    print("\nAttempting all possible decryption shifts:")
    print("-" * 50)
    for s in range(26):
        decrypted_text = caesar_decrypt(ct, s)
        print(f"Shift {s:2d}: {decrypted_text}")


def brute_force_encrypt(pt):
    print("\nGenerating all possible encryption shifts:")
    print("-" * 50)
```

```
    for s in range(26):
        encrypted_text = caesar_encrypt(pt, s)
        print(f"Shift {s:2d}: {encrypted_text}")


def caesar_encrypt(pt, s):
    enc_text = ""
    for ch in pt:
        if ch.isalpha():
            base = ord('A') if ch.isupper() else ord('a')
            enc_text += chr((ord(ch) - base + s) % 26 + base)
        else:
            enc_text += ch
    return enc_text


def caesar_decrypt(ct, s):
    dec_text = ""
    for ch in ct:
        if ch.isalpha():
            base = ord('A') if ch.isupper() else ord('a')
            dec_text += chr((ord(ch) - base - s) % 26 + base)
        else:
            dec_text += ch
    return dec_text


def main():
```

```
    pt = input("Enter the text to encrypt: ")
    print("\n=== ENCRYPTION RESULTS ===")
    print(f"Original text: {pt}")
    brute_force_encrypt(pt)

    ct = caesar_encrypt(pt, 1)
    print("\n=== DECRYPTION RESULTS ===")
    print(f"Encrypted text (shift 1): {ct}")
    brute_force_decrypt(ct)

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

Algorithm:

Step 1: Implement the SRT methodology.

Step 2: Acquire plaintext from the user's input.

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

Step 4: PRESENT the original plaintext.

Step 5: INVOKE the function Brute_Encrypt(plain_text).

Step 6: PERFORM encryption on the plaintext utilizing a shift value of 1.

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

Step 8: PRESENT the encrypted text with a shift of 1.

Step 9: INVOKE the function Brute_Decrypt(cipher_text).

Step 10: TERMINATE the process.

Input :

Enter the text to encrypt: **my name is NAYAN**

Output Is:

Encryption Results:

Original text Is: my name is NAYAN

Generating all possible encryption shifts:

Shift 0: my name is NAYAN
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 NAYAN
Shift 2: lx mzld hr Zrghj
...
Shift 25: oa pcog ku Cujkm

Conclusion: In the context of this laboratory investigation, we demonstrated the effectiveness of a brute force attack on the Caesar Cipher. The brute force methodology successfully decrypted the ciphertext by systematically assessing all conceivable shift values. This highlights a critical vulnerability intrinsic to elementary ciphers such as the Caesar Cipher: their keys can be easily exposed by exhaustively evaluating each potential shift. While this strategy is computationally uncomplicated and efficient for fundamental ciphers, it becomes impractical for more advanced encryption methods utilized in modern cryptography.

Experiment No:4

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

Objective: The fundamental objective of this experimental study is to implement the procedures of encryption and decryption through the application of the Hill cipher methodology. The Hill cipher is characterized as a polygraphic substitution cipher that utilizes the principles of linear algebra to facilitate the secure transmutation of messages. By acquiring a comprehensive understanding and proficient application of matrix operations, we aspire to proficiently transform plaintext into ciphertext and vice versa.

Theory : The Hill cipher was originally devised by Lester S. Hill in the year 1929 and is fundamentally anchored in the principles of linear algebra. Its functionality is predicated upon the encryption of text blocks via matrix multiplication. The encryption key is represented as a square matrix (for example, 2x2 or 3x3), while the message is divided into blocks of uniform size that correspond to the dimensions of the matrix. Each block is subjected to multiplication by the matrix, culminating in the production of ciphertext.

In the process of encryption, the original plaintext message undergoes an initial transformation into numerical equivalents (with A assigned the value of 0, B as 1, and so forth until Z is represented by 25). Following this transformation, the message is systematically arranged into vectors that align in dimensionality with the key matrix. These vectors are subsequently subjected to multiplication by the key matrix, and the resulting product is then converted back into alphabetical symbols, thus generating the ciphertext.

For the purpose of decryption, the inverse of the key matrix is utilized. The ciphertext undergoes multiplication with the inverse key matrix to retrieve the original plaintext message.

Key Components Key Matrix: A square matrix employed in the encryption mechanism. Inverse Matrix: A matrix that enables the decryption process. Modular Arithmetic: All operations are performed modulo 26, corresponding to the total number of letters in the English alphabet.

Given Data

- **Plaintext**: "AHH"
- **Key**: "NAIMUR RAHMAN" (we'll use the first 9 characters to form a 3x3 matrix)

Steps

1. **Convert Plaintext and Key Letters to Numbers**:
   - Plaintext "AHH": `A = 0, H = 7, H = 7`
   - Key (first 9 characters of "NAIMUR RAHMAN"): `N = 13, A = 0, I = 8, M = 12, U = 20, R = 17, R = 17, A = 0, H = 7`
2. **Construct the Key Matrix**: Using the first 9 characters from the key, we get the following 3x3 matrix

$$\text{Key Matrix} = \begin{bmatrix} 13 & 0 & 8 \\ 12 & 20 & 17 \\ 17 & 0 & 7 \end{bmatrix}$$

:

3. **Construct the Plaintext Vector**: Since the plaintext is "AHH", the vector representation is:
4. **Matrix Multiplication for Encryption**: We perform matrix multiplication between the key matrix and the plaintext vector, then apply modulus 26 to each result to map it to letters in the alphabet.

$$\begin{bmatrix} 13 & 0 & 8 \\ 12 & 20 & 17 \\ 17 & 0 & 7 \end{bmatrix} \times \begin{bmatrix} 0 \\ 7 \\ 7 \end{bmatrix} = \begin{bmatrix} (13 \cdot 0 + 0 \cdot 7 + 8 \cdot 7) \\ (12 \cdot 0 + 20 \cdot 7 + 17 \cdot 7) \\ (17 \cdot 0 + 0 \cdot 7 + 7 \cdot 7) \end{bmatrix} = \begin{bmatrix} 56 \\ 259 \\ 49 \end{bmatrix}$$

   Applying modulo 26 to each element in the resulting matrix:

$$\begin{bmatrix} 56 \\ 259 \\ 49 \end{bmatrix} \mod 26 = \begin{bmatrix} 4 \\ 25 \\ 23 \end{bmatrix}$$

5. **Convert Numbers Back to Letters**: Using the alphabet mapping:
   - `4 → E`
   - `25 → Z`
   - `23 → X`

   So, the **ciphertext** is **"EZX"**.

6. Decryption: In order to execute decryption, one must ascertain the inverse of the key matrix under modulo 26. The determination of the inverse matrix can be achieved through various methodologies, including the utilization of the adjoint matrix or the extended Euclidean algorithm. Subsequently, the inverse matrix should be multiplied by the ciphertext vector to retrieve the original message.

Code :

```
keyMat = [[0] * 3 for i in range(3)]

messageVec = [[0] for i in range(3)]

cipherMat= [[0] for i in range(3)]

def getKeyMat(key):

    k = 0

    for i in range(3):

        for j in range(3):

            keyMat[i][j] = ord(key[k]) % 65

            k += 1

def encrypt(messageVec):

    for i in range(3):

        for j in range(1):

            cipherMat[i][j] = 0

            for x in range(3):

                cipherMat[i][j] += (keyMat[i][x] * messageVec[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.")
```

```
    getKeyMat(key)
    for i in range(3):
        messageVec[i][0] = ord(message[i]) % 65
    encrypt(messageVec)
    CipherT = []
    for i in range(3):
        CipherT.append(chr(cipherMat[i][0] + 65))
    print("Ciphertext: ", "".join(CipherT))
def main():
    message = "ASH"
    print("Input message:", message)
    key = "NAIMURRAHMAN"
print("Key:", key
    HillCipher(message, key)
if __name__ == "__main__":
    main()
```

7.Output:

Ciphertext: " **EZX** "

Decrypted message: "AHH"

Algorithm

1.STRT OBTAIN the plaintext message alongside the cryptographic key.

2.FORMULATE a 3x3 key matrix derived from the provided string.

3.TRANSLATE the plaintext into a vector of numerical representations (A=0, B=1, ..., Z=25).

4.SECURE the plaintext through the multiplication of the key matrix with the message vector and subsequently apply modulo 26.

5.TRANSLATE the resultant cipher matrix back into its corresponding characters.

6.PRESENT the ciphertext.

7.END

Conclusion: Throughout the duration of this investigation, the Hill cipher was utilized to facilitate the processes of encryption and decryption of a designated message. The methodology incorporated matrix multiplication, modular arithmetic, and the utilization of key matrices during both the encryption and decryption stages. This cipher serves as a prime illustration of the application of linear algebra in the endeavor for secure communication. By employing the inverse of the key matrix, we successfully reverted the ciphertext back to its original plaintext form.

Experiment No:5

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

Objective: The primary objective of this laboratory exercise is to implement encryption through the utilization of the Playfair Cipher. This specific cipher encrypts letter pairs derived from a plaintext message by employing a designated keyword, thereby providing an augmented level of security in comparison to conventional ciphers such as the Caesar Cipher.

Theory: The Playfair Cipher is classified as a digraph substitution cipher, originally developed by Charles Wheatstone and later promoted by Lord Playfair. This cipher operates by encrypting pairs of letters (digraphs) instead of singular letters. The encryption process is based on the following procedural steps:

1.Key Matrix: The cipher utilizes a 5x5 matrix that is formulated from a selected keyword. This matrix includes all letters of the English alphabet (with the exception of 'J', which is typically merged with 'I' to accommodate a total of 25 letters)

2. Encryption Process: The original text is divided into pairs of characters, referred to as digraphs. In cases where a digraph consists of two identical characters, an 'X' is interjected between them. In instances where the original text comprises an odd number of characters, an 'X' is appended at the end. Each character pair is encoded by determining their respective positions within the key matrix and applying a specific set of rules dictated by their locations within the matrix.

3. Encryption Rules: In the event that both characters in a digraph are located within the same row, they are substituted with the characters that are directly adjacent to their right (with an adjustment to wrap around to the beginning of the row as necessary). Conversely, if both characters in a digraph are positioned within the same column, they are replaced with the characters that are directly below them (with a wrap-around to the top if required). When the characters form a rectangle, they are interchanged with the characters located at the opposite corners of the rectangle.

Procedure:

1. Preparation of the Key Matrix: Select a specific keyword and remove any repeating letters. Construct a 5x5 matrix employing the identified keyword, and subsequently fill it with the remaining letters of the alphabet, explicitly excluding the letter 'D'.

2. Preparation of the Plaintext: Divide the plaintext into digraphs (pairs of letters). In the event that a pair consists of identical letters, replace the second letter with 'X'. If the plaintext contains an odd number of letters, affix 'X' to the terminal position.

3. Encryption Process: For each digraph, identify the letters within the key matrix. Apply the established encryption protocols to transform the digraph into its corresponding ciphertext.

4. Presentation of the Ciphertext: Upon the conclusion of the encryption of all digraphs, display the resultant ciphertext.

```
def create_matrix(key):
    key = key.upper().replace('J', 'I')
    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':
        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 = ''.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(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(key, plaintext):
    matrix = create_matrix(key)
    pairs = prepare(plaintext)
    ciphertext = ''

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

    return ciphertext

if __name__ == "__main__":
    key = "NAIMUR"
    plaintext = "TENTCHJOI"

    print("Key:", key)
    print("Original text:", plaintext)
    encrypted = playfair(key, plaintext)

    print("\nEncrypted text:", encrypted)
```

Output:

Key: NAIMUR
Plaintext is the: TENTCHJOI
Ciphertext is the: ZCZJQHUC

Algorithm for Playfair Cipher Encryption:

1.STRT

2.The procedure for the key entails converting it to uppercase, eliminating duplicate characters, and substituting 'J' with 'I'.

3.Construct the 5x5 key matrix by incorporating characters from the key and subsequently filling in the remaining letters (excluding 'J').

4.Transform the plaintext by converting it to uppercase, substituting 'J' with 'I', partitioning it into digraphs (pairs of characters), inserting 'X' if any digraph contains duplicate letters, and appending 'X' at the conclusion if the total length is odd.

5.Encrypt each digraph by determining whether the letters reside in the same row, column, or form a rectangle.

6.For letters situated in the same row, substitute them with the letters located to their right. For letters aligned in the same column, replace them with the letters situated below. For letters that constitute a rectangle, perform a diagonal interchange. Output the resulting ciphertext.

7.end

Conclusion:

Throughout the duration of this study, the Hill cipher was utilized to enable the processes of encryption and decryption of a designated message. The approach involved matrix multiplication, modular arithmetic, and the implementation of key matrices throughout both the encryption and decryption procedures. This cipher serves as a notable illustration of the application of linear algebra in the quest for secure communication. By employing the inverse of the key matrix, we successfully reverted the ciphertext to its initial form..

Experiment No:6

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

Objective: The principal objective of this laboratory exercise is to implement the decryption process of the Playfair Cipher, which is a digraph substitution cipher utilized for the secure transmission of messages. The Playfair Cipher functions by employing a 5x5 key matrix and adhering to specific regulations to revert the ciphertext to plaintext. Through the execution of this decryption process, our goal is to retrieve the original message by utilizing the key in conjunction with the supplied ciphertext.

Theory: The Playfair Cipher exemplifies a symmetric encryption technique that operates on pairs of letters (digraphs) rather than individual characters. The encryption process commences with the formation of a 5x5 matrix of letters derived from a specified keyword, while the decryption process is contingent upon the application of the inverse of the encryption methods. During the decryption phase:Procedure:

1. Prepare the Key: Transform the key into uppercase characters. Discard any superfluous characters. Replace 'J' with 'I' (given that the Playfair Cipher exclusively accommodates 25 letters).

2. Create the 5x5 Key Matrix: The key serves as the foundation for populating the 5x5 matrix, strategically positioning the characters of the key within the matrix. Following the inclusion of the key characters, the remaining letters of the alphabet (with the exception of 'J') are integrated to finalize the matrix.

3. Preprocess the Ciphertext: Convert the ciphertext into uppercase characters. Substitute 'J' with 'I'. Divide the ciphertext into digraphs (pairs of characters). In instances where the ciphertext exhibits an odd length, append 'X' to the conclusion**.**

4. Decrypt the Digraphs: If the letters are located within the same row of the matrix, replace them with the letters that are immediately to their left. Conversely, if the letters are situated within the same column, substitute them with the letters directly above..

Code :

```
def create_mat(key):
    key = key.upper().replace('J', 'I')
    mat = []
    used_chars = set()
    for char in key:
        if char.isalpha() and char not in used_chars:
            mat.append(char)
            used_chars.add(char)
    for char in 'ABCDEFGHIKLMNOPQRSTUVWXYZ':
        if char not in used_chars:
            mat.append(char)
            used_chars.add(char)
    return [mat[i:i+5] for i in range(0, 25, 5)]
def find_position(mat, char):
    for i in range(5):
        for j in range(5):
            if mat[i][j] == char:
                return i, j
    return None
def decrypt_pair(mat, pair):
    row1, col1 = find_position(mat, pair[0])
    row2, col2 = find_position(mat, pair[1])
    if row1 == row2:
        return (mat[row1][(col1-1)%5], mat[row2][(col2-1)%5])
```

```
    elif col1 == col2:
        return (mat[(row1-1)%5][col1], mat[(row2-1)%5][col2])
    else:
        return (mat[row1][col2], mat[row2][col1])
def playfair_decrypt(ciphertext, key):
    ciphertext = ciphertext.upper().replace('J', 'I')
    ciphertext = ''.join(c for c in ciphertext if c.isalpha())
    mat= create_mat(key)
    decrypted = []
    for i in range(0, len(ciphertext), 2):
        if i+1 < len(ciphertext):
            pair = decrypt_pair(mat, (ciphertext[i], ciphertext[i+1]))
            decrypted.extend(pair)
    return ''.join(decrypted)
def main():
    key = "NAIMUR"
    ciphertext = "NMBISQAQPZ"
    print("Key:", key)
    print("Ciphertext:", ciphertext)
    decrypted = playfair_decrypt(ciphertext, key)
    print("Decrypted message is:", decrypted)
if __name__ == "__main__":
    main()
```

Output:

Key: NAIMUR

Ciphertext Is: ISQNQAMBPZ

Decrypted Message is: TINCHTXEIO

Algorithm for Playfair Cipher Decryption:

1. Construct the key matrix utilizing the distinct characters from the key alongside the residual letters from the English alphabet (A-Z, excluding 'J').

2. Divide the ciphertext into pairs of letters for analysis.

3. For each pair:

If the letters reside within the same row, substitute them with the letters immediately to their left. If the letters are positioned within the same column, replace them with the letters located directly above. If the letters delineate a rectangle, interchange them with their corresponding opposite corners.

4. Integrate the decrypted pairs to formulate the plaintext.

5. Eliminate any filler 'X' that may have been introduced during the encryption process.

Conclusion: The Playfair Cipher represents a traditional encryption methodology that enhances security through the encryption of digraphs, as opposed to individual alphabetic characters. The decryption process functions as the reciprocal of the encryption mechanism, wherein the key matrix is utilized to invert the encryption rules and recover the original message. In the context of this laboratory exercise, we effectively decrypted the ciphertext " ISQNQAMBPZ " into the plaintext " TINCHTXEIO" by employing the Playfair Cipher. This algorithm emphasizes the importance of key management and matrix manipulation within the realms of encryption and decryption processes, alongside the pivotal role of symmetry in the domain of cryptography**.**

Experiment No:7

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

Objective: The foremost objective of this laboratory exercise is to implement encryption through the application of the Poly-Alphabetic Cipher. This cipher exemplifies a variant of substitution cipher in which each character in the plaintext is subjected to a transformation in accordance with a cyclic key sequence. By utilizing multiple alphabets (polyalphabetic), this cipher substantially augments security relative to conventional substitution ciphers, thereby rendering it more resistant to frequency analysis.

Theory: The Poly-Alphabetic Cipher represents an encryption technique that employs a repetitive sequence of keys to variably shift each character of the plaintext. One of the most widely recognized methodologies employed within polyalphabetic ciphers is the Vigenère Cipher, whereby each character in the plaintext is shifted based on the corresponding character in the key. For example, in a scenario where the key is "ICE," each character in the message will be subjected to a shift determined by the letters 'I', 'C', and 'E' in a cyclical manner..

For each character in the original text: Convert the character into its numerical equivalent (A=0, B=1, ..., Z=25). Employ the respective character from the key to execute the shift, cycling through the key as required. Transform the shifted numerical representation back to its alphabetical counterpart.

This approach obscures a direct correspondence between characters, thereby enhancing the complexity of the decryption process for the encoded communication in the absence of the key.

Procedure:

1. Input the Original Text and Key: Obtain the message designated for encryption along with the appropriate key from the user**.**

2. Extend Key to Correspond with Original Text Length: In cases where the key is of lesser length than the original text, duplicate it until it aligns with the length of the original text.

3. Encrypt Each Character: For every character present in the plaintext: Convert the character into its numerical equivalent (A=0 to Z=25). Utilize the relevant character from the repeated key to ascertain the shift magnitude. Execute the shift of the character in accordance with the letter from the key.

4. Generate Ciphertext:

Transform the shifted numerical values back into letters to construct the encrypted message.

Code:

```
def generate_encryption_key(text, secret_key):
    secret_key = list(secret_key)
    if len(text) == len(secret_key):
        return "".join(secret_key)
    else:
        for i in range(len(text) - len(secret_key)):
            secret_key.append(secret_key[i % len(secret_key)])
    return "".join(secret_key)


def encrypt_message():
    plain_text = input("Enter message to encrypt: ")
    secret_key = input("Enter key: ")


    secret_key = generate_encryption_key(plain_text, secret_key)
    cipher_text = []


    for i in range(len(plain_text)):
```

```
        if plain_text[i].isalpha():  # Encrypt only alphabetic characters
            if plain_text[i].isupper():  # For uppercase letters
                shifted_value = (ord(plain_text[i]) + ord(secret_key[i].upper()) - 2 * ord('A')) %
26
                cipher_text.append(chr(shifted_value + ord('A')))
            else:  # For lowercase letters
                shifted_value = (ord(plain_text[i]) + ord(secret_key[i].lower()) - 2 * ord('a')) %
26
                cipher_text.append(chr(shifted_value + ord('a')))
        else:  # Non-alphabetic characters are added without encryption
            cipher_text.append(plain_text[i])


    encrypted_message = "".join(cipher_text)
    print(f"\nEncrypted Message: {encrypted_message}")


if __name__ == "__main__":
    print("Poly-Alphabetic Cipher Encryption")
    print("--------------------------------")
    encrypt_message()
```

Output:

For the input:

Plaintext: NAIMUR RAHMAN **Key**: ICE

The output is:

Plaintext: NAIMUR RAHMAN
Key: ICE

Encrypted Message: UT IULQMYZ VIJQIP

Algorithm :

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.End

Conclusion: The Poly-Alphabetic Cipher signifies a substantial augmentation in security when compared to monoalphabetic ciphers, as it utilizes a multitude of letters within the key for the purpose of message encryption. This methodology introduces a layer of complexity to the decryption process in the absence of the key, since each character in the plaintext may undergo a unique shift, dependent upon the sequence delineated by the key. This laboratory effectively exemplifies the foundational concepts of encryption through the Poly-Alphabetic Cipher, thereby providing an enriched comprehension of the advantages associated with polyalphabetic encryption strategies in contrast to more rudimentary ciphers within the field of cryptographic security.

Experiment No:8

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

Objective: The principal objective of this laboratory exercise is to carry out the decryption procedure employing the Poly-Alphabetic Cipher. This cipher epitomizes a distinct class of substitution cipher, which utilizes a recurring key to determine the decryption shift applicable to each individual character. The ultimate goal is to recover the initial plaintext message from the ciphertext by reversing the encryption process.

Theory: The Poly-Alphabetic Cipher represents a cipher variation that utilizes multiple alphabets based on a recurring key, often implemented through the Vigenère Cipher technique. During the decryption process, each character in the ciphertext is shifted backward in accordance with the position of the corresponding character in the key. By employing this dynamic key-centric shifting approach, the original message can be accurately reconstructed.The decryption process encompasses:

Translating each symbol in the ciphertext into its corresponding numerical value (A=0, B=1, ..., Z=25). Adjusting each symbol by subtracting the pertinent letter in the key (which has likewise been converted into numerical form). Converting the resulting numerical figures back into letters to construct the decrypted message.

Procedure:

1. Input the Ciphertext and Key: Request that the user submits the encoded message along with the key.

2. Extend Key to Align with Plaintext Length: Replicate the key as required to match the length of the ciphertext.

3. Character Decryption Process: For each individual character present in the encrypted text: Translate it into its numerical representation, ascertain the relevant shift based on the encryption key, and execute a reverse shift. Employ modulo 26 to guarantee that the outcome remains within the permissible limits of the alphabetic spectrum.

4. Reconstruction of Plaintext: Convert the adjusted numerical values back into their respective characters to restore the initial message.

Code:

```
def gen_decrypt_key(text, key):
    key = list(key)
    if len(text) == len(key):
        return "".join(key)
    else:
        for i in range(len(text) - len(key)):
            key.append(key[i % len(key)])
    return "".join(key)


def decrypt():
    """Decrypt a message using Poly-Alphabetic cipher"""
    cipher_text = input("Enter message to decrypt: ")
    key = input("Enter key: ")


    key = gen_decrypt_key(cipher_text, key)
    plain_text = []


    for i in range(len(cipher_text)):
        if cipher_text[i].isalpha():
            if cipher_text[i].isupper():
```

```
            shift_val = (ord(cipher_text[i]) - ord(key[i].upper()) + 26) % 26
            plain_text.append(chr(shift_val + ord('A')))
        else:
            shift_val = (ord(cipher_text[i]) - ord(key[i].lower()) + 26) % 26
            plain_text.append(chr(shift_val + ord('a')))
    else:
        plain_text.append(cipher_text[i])  # Non-alphabet characters remain unchanged

decrypted_text = "".join(plain_text)
print(f"\nDecrypted Message: {decrypted_text}")

if __name__ == "__main__":
    print("Poly-Alphabetic Cipher Decryption")
    print("---------------------------------")
    decrypt()
```

Output:

For the input:

Ciphertext: Uk yulqmIz qVipij **Key**: ICE

The output :

Ciphertext: Uk yulqmIz qVipij
Key: ICE
Decrypted Message: Naimur Rahman

Algorithm for the Decryption of Poly-Alphabetic Ciphers

1. INTRODUCE the ciphertext alongside the key.

2. TRANSFORM both the ciphertext and the key into uppercase letters.

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

4. ESTABLISH an empty string designated for the plaintext.

5. For each character within the ciphertext: If it is alphabetic, DETERMIN the original position by deducting the positional value of the key character and applying modulo 26. TRANSFORM the resulting value back into a character and APPEND it to the plaintext. If it is non-alphabetic, INCORPORATE it directly into the plaintext.

6. DISPLAY the final plaintext.

Conclusion: The Poly-Alphabetic Cipher, which utilizes a repetitive key, provides strong encryption by altering each character with a unique shift determined by the key. The decryption process requires the reversal of these shifts in accordance with the key. This laboratory exercise clarifies the technique for efficiently deciphering..

Experiment No:9

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

Objective: The principal objective of this laboratory exercise is to implement encryption through the use of the Vernam Cipher, which is also known as the One-Time Pad cipher. This cipher employs a unique, random key that matches the length of the plaintext, thereby producing ciphertext and ensuring that each encryption is singular and theoretically resistant to decryption, provided that the key is employed only once**.**

Theory: The Vernam Cipher operates as a symmetric encryption technique in which each character of the plaintext is combined with a corresponding character from the key through the execution of a bitwise XOR operation. This computational method results in a distinctive encrypted character that hinges on both the plaintext and the key. The Vernam Cipher is considered unbreakable under specific conditions:

Procedure:

1. Input the Plaintext and Key: Ensure that the key's length is precisely aligned with that of the plaintext.

2. Convert Plaintext and Key Characters to Numeric Values: Map each character to its corresponding ASCII value or its position within the alphabet.

3. Execute XOR Operation on Each Character: For every character, perform an XOR operation between the plaintext character and the associated key character. Convert the resulting value back into a character.

4. Generate Plaintext: Aggregate all encrypted characters to form the final ciphertext**.**

Code:

```
def encrypt_msg(plain_text, key):
    cipher_text = ""
    cipher = []
    for i in range(len(key)):
        cipher.append((ord(plain_text[i]) - ord('A') + ord(key[i]) - ord('A')) % 26)
    for i in range(len(key)):
        cipher_text += chr(cipher[i] + ord('A'))
    return cipher_text


msg = "howareyou"
key = "ncbtzqarx"
encrypted_msg = encrypt_msg(msg.upper(), key.upper())
print("Cipher Text is - " + encrypted_msg)
```

Output:

For the input:

Plaintext : mynameisnoyon **Key**: arzqncbtx

The output:

Plaintext: mynameisnoyon
Key: arzqncbtx
Cipher Text is – CKQXDUZPM

Algorithm for Vernam Cipher Encryption

1. INITIATE the ciphertext along with the corresponding key.

2. CONFIRM that the length of the key is congruent with the length of the plaintext. Should there be any inconsistencies, RETURN an error message.

3. ESTABLISH an unoccupied string designated to contain the ciphertext.

4. For each character within the plaintext and its affiliated character in the key: TRANSFORM each character into its positional representation within the alphabet (for instance, 'a' = 0, 'b' = 1, ..., 'z' = 25). IMPLEMENT the XOR operation on the positional values derived from the plaintext character and the key character. RECONSTITUTE the resultant positional value back into a character within the alphabet. ATTACH the resulting character to the ciphertext string.

5. PRESENT the ciphertext as the resultant encrypted message.

Conclusion: The Vernam Cipher, by virtue of employing the XOR operation in conjunction with a singularly unique key, provides an encryption framework that is theoretically impervious to decryption, so long as the key is utilized solely once and exclusively.

Experiment No:10

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

Objective: The principal objective of this laboratory exercise is to implement encryption through the use of the Vernam Cipher, which is also known as the One-Time Pad cipher. This cipher employs a unique, random key that matches the length of the plaintext, thereby producing ciphertext and ensuring that each encryption is singular and theoretically resistant to decryption, provided that the key is employed only once**.**

Theory:

The Vernam Cipher fundamentally relies on the utilization of a bitwise XOR operation for the encryption and decryption of messages. The inherent reversibility of the XOR operation indicates that when a character from the plaintext is subjected to an XOR operation with a character from the key to produce the ciphertext, subsequently performing the identical XOR operation on the ciphertext character with the same key character will restore the original plaintext character. This decryption process is dependent on: The lengths of both the ciphertext and the key being equal. The key being known and employed solely once. The decryption process entails: Executing the XOR operation on each character of the ciphertext with its corresponding character from the key. This operation ultimately facilitates the recovery of the original plaintext character.

Procedure:

1. Input the Ciphertext and Key: It is essential to verify that the key's length is equivalent to that of the ciphertext.

2. Convert Ciphertext and Key Characters to Numeric Values: Each character must be assigned to its respective position within the alphabet.

3. Execute XOR Operation on Each Character: Each character in the ciphertext is to be XORed with the corresponding character in the key. The resulting value should then be translated back to its respective character.

4. Construct the Plaintext: All decrypted characters should be combined to create the plaintext

Code :

```
def decrypt_msg(cipher_text, key):
    plain_text = ""
    plain = []
    cipher_text = cipher_text.upper()
    key = key.upper()

    for i in range(len(key)):
        plain.append((ord(cipher_text[i]) - ord('A') - (ord(key[i]) - ord('A'))) % 26)

    for i in range(len(key)):
        plain_text += chr(plain[i] + ord('A'))
    return plain_text
cipher_text = "uqxtquyfr"
key = "ncbtzqarx"
decrypted_msg = decrypt_msg(cipher_text, key)
print("Decrypted Message is - " + decrypted_msg)
```

Output:

For the input is:

Ciphertext: **Key**: nqabrxtzc

The output : Ciphertext: uxfqtquyr

Key: nqabrxtzc Decrypted Message: MYNAMEINOYON

Algorithm for Vernam Cipher Decryption

1. INPUT the ciphertext alongside the key

2. Confirm the congruence in lengths of both the ciphertext and the key

3. For each character within the ciphertext and the corresponding key: Characters must be translated into their respective numerical representations. The XOR operation is to be performed. The resulting value must then be converted back into its respective character.

4. OUTPUT the deciphered message

Conclusion: This program clarifies the decryption mechanism intrinsic to the Vernam Cipher, wherein each character of the ciphertext undergoes an XOR operation with its associated key character to recover the original plaintext. This laboratory activity highlights the security features of the Vernam Cipher, which remains resilient against compromise, provided that the key is utilized exclusively once in the decryption process of the Vernam Cipher, thus validating its suitability as a one-time pad within cryptographic frameworks.