Perform encryption, decryption using the following substitution techniques

1 a) Caesar Cipher

Aim:

To implement and understand the Caesar Cipher encryption technique in Java, which is a simple substitution cipher that shifts characters in the alphabet by a fixed number of positions. The experiment will demonstrate encryption and decryption processes using a user-defined shift value.

Software Required:

- Java Development Kit (JDK)
- Java Integrated Development Environment (IDE) such as Eclipse, IntelliJ, or VS Code

Algorithm:

1. **Input**:
   - Take the plaintext message from the user.
   - Take the shift key (number of positions to shift) from the user.
2. **Encrypt the Message**:
   - Iterate through each character in the plaintext.
   - If the character is uppercase:
     - Shift it forward by the key value within the range of 'A' to 'Z'.
   - If the character is lowercase:
     - Shift it forward within 'a' to 'z'.
   - Retain special characters and spaces as they are.
3. **Output**: Display the encrypted text.
4. **Decrypt the Message**:
   - Apply the reverse shift (subtract instead of add).
   - Restore the original plaintext.
5. **Output**: Display the decrypted text.

Perform encryption, decryption using the following substitution techniques

1 b) Playfair Cipher

Aim:

To implement the Playfair Cipher encryption method in Java, which is a digraph substitution cipher that encrypts letters in pairs. The experiment aims to construct the Playfair 5x5 matrix using a keyword and encrypt a given message.

Software Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ, or VS Code)

Algorithm:

1. **Input**:
    - Take a keyword and a plaintext message from the user.
2. **Generate the Key Matrix**:
    - Remove duplicate letters from the keyword.
    - Construct a 5x5 matrix with the keyword, followed by remaining letters of the alphabet (excluding 'J', which is replaced with 'I').
3. **Prepare the Text**:
    - Split plaintext into letter pairs.
    - Insert 'X' between duplicate letters and pad the last letter if the text has an odd length.
4. **Encrypt the Text**:
    - Locate letter pairs in the matrix.
    - Apply Playfair encryption rules:
        - Same row: Shift right.
        - Same column: Shift down.
        - Rectangle: Swap opposite corners.
5. **Output**: Display the encrypted text.

Perform encryption and decryption using following transposition techniques

2 a) Rail Fence Cipher

Aim:

To implement and understand the Rail Fence Cipher, a transposition cipher that arranges text in a zigzag pattern across multiple rows before reading it row by row. The experiment will cover encryption and decryption processes.

Software Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ, or VS Code)

Algorithm:

1. **Input**:
    - Take plaintext and the number of rails (depth) from the user.
2. **Encrypt the Message**:
    - Arrange characters in a zigzag rail fence pattern.
    - Read row-wise to generate ciphertext.
3. **Output**: Display the encrypted text.
4. **Decrypt the Message**:
    - Reconstruct the zigzag pattern based on rails.
    - Read in the original sequence to retrieve plaintext.
5. **Output**: Display the decrypted message.

2 b) Row Column Transformation

Aim:

To implement Row-Column Transposition Cipher, which rearranges text in a grid format based on user-defined rows and columns. This experiment will showcase how text is encrypted by permuting columns and decrypted by restoring the original order.

Software Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ, or VS Code)

Algorithm:

1. **Input**:
    - Accept plaintext, number of rows, and columns from the user.
2. **Write into a Grid**:
    - Fill characters row-wise into a grid matrix, padding with 'X' if needed.
3. **Encrypt the Message**:
    - Read column-wise in reverse order to form the ciphertext.
4. **Output**: Display the encrypted text.
5. **Decrypt the Message**:
    - Rearrange characters into the original order.
6. **Output**: Display the decrypted text.

3 Apply DES algorithm for practical applications.

Aim:

To implement the Data Encryption Standard (DES), a symmetric key encryption technique that encrypts and decrypts messages using a 56-bit key. The experiment demonstrates the working of block cipher encryption using Java's built-in Cipher class.

Software Required:

- Java Development Kit (JDK)
- Java Cryptography API

Algorithm:

1. **Input**: Take a plaintext message from the user.
2. **Generate Key**:
   - Use KeyGenerator to create a DES encryption key.
3. **Encrypt the Message**:
   - Convert plaintext to bytes.
   - Apply DES encryption using the generated key.
   - Encode the output in Base64.
4. **Output**: Display the encrypted text.
5. **Decrypt the Message**:
   - Decode Base64 text.
   - Use the same key to decrypt the message.
6. **Output**: Display the decrypted text.

4 Implement the Diffie-Hellman Key Exchange algorithm for a given problem.

Aim:

To implement the Diffie-Hellman Key Exchange algorithm, a cryptographic method for securely exchanging cryptographic keys over a public channel. This experiment showcases how two parties establish a shared secret key without direct transmission.

Software Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ, or VS Code)

Algorithm:

1. **Input**:
   - Take prime p, primitive root g, and private keys for Alice and Bob.
2. **Compute Public Keys**:
   - Alice: A=gamod pA = g^a \mod p
   - Bob: B=gbmod pB = g^b \mod p
3. **Compute Shared Secret**:
   - Alice: sharedKey=Bamod psharedKey = B^a \mod p
   - Bob: sharedKey=Abmod psharedKey = A^b \mod p
4. **Output**: Display public keys and shared secret key.

5) Calculate the message digest of a text using the SHA-1 algorithm.

Aim:

To implement the SHA-1 hashing algorithm, which is a cryptographic function that converts a message into a fixed-size 160-bit hash. This experiment demonstrates how messages are hashed securely.

Software Required:

- Java Development Kit (JDK)

Algorithm:

1. **Input**: Accept a message from the user.
2. **Apply SHA-1**: Convert the message to bytes and compute its hash.
3. **Convert to Hexadecimal**: Format the hash output in hexadecimal.
4. **Output**: Display the SHA-1 hash.

6) Calculate the message digest of a text using the MD5 algorithm in JAVA.

Aim:

To generate an MD5 hash of a given message, demonstrating how message integrity can be verified using hashing techniques.

Software Required:

- Java Development Kit (JDK)

Algorithm:

1. **Input**: Accept a message from the user.
2. **Apply MD5 Hashing**: Convert the message to bytes and generate the hash.
3. **Format Output**: Convert hash bytes to a hexadecimal string.
4. **Output**: Display the MD5 hash.

7) Implement the Signature Scheme - Digital Signature Standard

Aim:

To implement Digital Signature generation and verification using DSA. The experiment demonstrates how a sender signs a message and how the receiver verifies its authenticity.

Software Required:

- Java Development Kit (JDK)
- Java Cryptography API

Algorithm:

1. **Input**: Accept a message.
2. **Generate Key Pair**: Create public and private keys using KeyPairGenerator.
3. **Sign the Message**:
   - Use the private key to generate a digital signature.
4. **Verify the Signature**:
   - Use the public key to verify the signature.
5. **Output**: Display the digital signature and verification result.