Practical No 1

Aim: Write programs to implement the following Substitution cipher technique.

1. Caesar Cipher

```
public class CaesarCipher {
    public static String encrypt(String text, int shift) {
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text.length(); i++) {
            char ch = text.charAt(i);

            // Check if the character is a letter
            if (Character.isLetter(ch)) {
                char base = Character.isLowerCase(ch) ? 'a' : 'A';
                // Shift the character and wrap around the alphabet
                ch = (char) ((ch - base + shift) % 26 + base);
            }
            result.append(ch); // Append the processed character
        }
        return result.toString(); // Return the encrypted string
    }

    public static String decrypt(String text, int shift) {
        return encrypt(text, 26 - shift); // Decrypt by shifting in the opposite direction
    }

    public static void main(String[] args) {
        String text = "Hello World!";
        int shift = 3;

        String encrypted = encrypt(text, shift);
        String decrypted = decrypt(encrypted, shift);

        System.out.println("Original: " + text);
        System.out.println("Encrypted: " + encrypted);
        System.out.println("Decrypted: " + decrypted);
    }
}
```

2. Monoalphabetic Cipher

```
import java.util.HashMap;
import java.util.Map;

public class MonoalphabeticCipher {
    private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    // Simple key for demonstration. For a more secure key, you should randomize the characters.
    public static String generateKey() {
        return "QWERTYUIOPASDFGHJKLZXCVBNM";
    }

    public static String encrypt(String text, String key) {
```

```
        Map<Character, Character> charMap = createCharMap(ALPHABET, key);
        return transformText(text, charMap);
    }

    public static String decrypt(String text, String key) {
        Map<Character, Character> charMap = createCharMap(key, ALPHABET);
        return transformText(text, charMap);
    }

    private static Map<Character, Character> createCharMap(String from, String to) {
        Map<Character, Character> charMap = new HashMap<>();
        for (int i = 0; i < from.length(); i++) {
            charMap.put(from.charAt(i), to.charAt(i));
        }
        return charMap;
    }

    private static String transformText(String text, Map<Character, Character> charMap) {
        StringBuilder result = new StringBuilder();
        for (char ch : text.toUpperCase().toCharArray()) {
            if (charMap.containsKey(ch)) {
                result.append(charMap.get(ch));
            } else {
                result.append(ch); // Non-alphabetic characters remain unchanged
            }
        }
        return result.toString();
    }

    public static void main(String[] args) {
        String text = "HELLO WORLD";
        String key = generateKey();

        String encrypted = encrypt(text, key);
        String decrypted = decrypt(encrypted, key);

        System.out.println("Original: " + text);
        System.out.println("Encrypted: " + encrypted);
        System.out.println("Decrypted: " + decrypted);
    }
}
```

Practical No 2

Aim: Write programs to implement the following Substitution Cipher Techniques:

1. Vernam Cipher

```
import java.util.Random;

public class VernamCipher {
    // Generate a random key of specified length
    public static String generateKey(int length) {
        Random random = new Random();
```

```
        StringBuilder key = new StringBuilder();
        for (int i = 0; i < length; i++) {
            char ch = (char) (random.nextInt(26) + 'A'); // Generate a random uppercase letter
            key.append(ch);
        }
        return key.toString();
    }

    // Encrypt the text using the key
    public static String encrypt(String text, String key) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.length(); i++) {
            char ch = (char) (text.charAt(i) ^ key.charAt(i)); // XOR operation
            result.append(ch);
        }
        return result.toString();
    }

    // Decrypt the text using the key (same as encryption)
    public static String decrypt(String text, String key) {
        return encrypt(text, key); // Encryption and decryption are the same for Vernam
Cipher
    }

    public static void main(String[] args) {
        String text = "HELLOWORLD";
        String key = generateKey(text.length());

        String encrypted = encrypt(text, key);
        String decrypted = decrypt(encrypted, key);

        System.out.println("Original: " + text);
        System.out.println("Key: " + key);
        System.out.println("Encrypted: " + encrypted);
        System.out.println("Decrypted: " + decrypted);
    }
}
```

2. Playfair Cipher

```
import java.util.HashSet;
import java.util.Set;

public class PlayfairCipher {
    private static char[][] matrix = new char[5][5];
    private static final String ALPHABET = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

    // Generate the key matrix based on the provided key
    public static void generateKeyMatrix(String key) {
        Set<Character> usedChars = new HashSet<>();
```

```java
        key = key.toUpperCase().replaceAll("J", "I");
        StringBuilder keyBuilder = new StringBuilder(key);

        // Add characters from the key to the keyBuilder
        for (char ch : ALPHABET.toCharArray()) {
            if (!usedChars.contains(ch) && keyBuilder.indexOf(String.valueOf(ch)) == -1) {
                keyBuilder.append(ch);
            }
        }

        // Fill the matrix with characters from the keyBuilder
        int k = 0;
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                char ch = keyBuilder.charAt(k++);
                matrix[i][j] = ch;
                usedChars.add(ch);
            }
        }
    }

    // Preprocess the text for encryption/decryption
    public static String preprocessText(String text) {
        text = text.toUpperCase().replaceAll("[^A-Z]", "").replaceAll("J", "I");
        StringBuilder processed = new StringBuilder();

        for (int i = 0; i < text.length(); i++) {
            char ch = text.charAt(i);
            processed.append(ch);
            // Insert 'X' between duplicate letters
            if (i < text.length() - 1 && text.charAt(i) == text.charAt(i + 1)) {
                processed.append('X');
            }
        }

        // Append 'X' if the length is odd
        if (processed.length() % 2 != 0) {
            processed.append('X');
        }

        return processed.toString();
    }

    // Encrypt the text using the Playfair cipher
    public static String encrypt(String text, String key) {
        generateKeyMatrix(key);
        text = preprocessText(text);
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text.length(); i += 2) {
            char a = text.charAt(i);
            char b = text.charAt(i + 1);
```

```
            int[] posA = findPosition(a);
            int[] posB = findPosition(b);

            if (posA[0] == posB[0]) {
                result.append(matrix[posA[0]][(posA[1] + 1) % 5]);
                result.append(matrix[posB[0]][(posB[1] + 1) % 5]);
            } else if (posA[1] == posB[1]) {
                result.append(matrix[(posA[0] + 1) % 5][posA[1]]);
                result.append(matrix[(posB[0] + 1) % 5][posB[1]]);
            } else {
                result.append(matrix[posA[0]][posB[1]]);
                result.append(matrix[posB[0]][posA[1]]);
            }
        }

        return result.toString();
    }

    // Decrypt the text using the Playfair cipher
    public static String decrypt(String text, String key) {
        generateKeyMatrix(key);
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text.length(); i += 2) {
            char a = text.charAt(i);
            char b = text.charAt(i + 1);
            int[] posA = findPosition(a);
            int[] posB = findPosition(b);

            if (posA[0] == posB[0]) {
                result.append(matrix[posA[0]][(posA[1] + 4) % 5]);
                result.append(matrix[posB[0]][(posB[1] + 4) % 5]);
            } else if (posA[1] == posB[1]) {
                result.append(matrix[(posA[0] + 4) % 5][posA[1]]);
                result.append(matrix[(posB[0] + 4) % 5][posB[1]]);
            } else {
                result.append(matrix[posA[0]][posB[1]]);
                result.append(matrix[posB[0]][posA[1]]);
  }
        }

        return result.toString();
    }

    // Find the position of a character in the matrix
    private static int[] findPosition(char ch) {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (matrix[i][j] == ch) {
                    return new int[] { i, j };
                }
            }
```

```
        }

        return null;
    }

    public static void main(String[] args) {
        String text = "HELLO WORLD";
        String key = "KEYWORD";
        String encrypted = encrypt(text, key);
        String decrypted = decrypt(encrypted, key);

        System.out.println("Original: " + text);
        System.out.println("Key: " + key);
        System.out.println("Encrypted: " + encrypted);
        System.out.println("Decrypted: " + decrypted);
    }
}
```

Practical No. 3

Aim: Write programs to implement the following Transposition Cipher Techniques.

1. Rail Fence Cipher

```
public class RailFenceCipher {
    // Encrypt the text using the Rail Fence cipher
    public static String encrypt(String text, int key) {
        if (key == 1) return text; // No encryption needed for key 1

        StringBuilder[] rail = new StringBuilder[key];
        for (int i = 0; i < key; i++) {
            rail[i] = new StringBuilder(); // Initialize each rail
        }

        int row = 0;
        boolean down = true; // Direction flag
        for (char ch : text.toCharArray()) {
            rail[row].append(ch); // Place character in the current rail
            // Change direction at the top or bottom rail
            if (row == 0) {
                down = true;
            } else if (row == key - 1) {
                down = false;
            }
            row += down ? 1 : -1; // Move to the next rail
        }

        // Combine all rails to get the encrypted text
```

```
        StringBuilder result = new StringBuilder();
        for (StringBuilder sb : rail) {
            result.append(sb);
        }
        return result.toString();
    }

    // Decrypt the text using the Rail Fence cipher
    public static String decrypt(String text, int key) {
        if (key == 1) return text; // No decryption needed for key 1

        char[] decrypted = new char[text.length()];
        boolean[] visited = new boolean[text.length()]; // Track visited characters
        int index = 0;

        // Fill the decrypted array with characters in the correct order
        for (int k = 0; k < key; k++) {
            int row = 0;
            boolean down = true; // Direction flag
            for (int i = 0; i < text.length(); i++) {
                // If we are at the current rail and the character hasn't been visited
                if (row == k && !visited[i]) {
                    decrypted[i] = text.charAt(index++);
                    visited[i] = true; // Mark this character as visited
                }
                // Change direction at the top or bottom rail
                if (row == 0) {
                    down = true;
                } else if (row == key - 1) {
                    down = false;
                }
                row += down ? 1 : -1; // Move to the next rail
            }
        }
        return new String(decrypted);
    }

    public static void main(String[] args) {
        String text = "HELLO WORLD";
        int key = 3; // Number of rails
        String encrypted = encrypt(text, key);
        String decrypted = decrypt(encrypted, key);

        System.out.println("Original: " + text);
        System.out.println("Encrypted: " + encrypted);
        System.out.println("Decrypted: " + decrypted);
    }
}
```

2. Simple Columnar Technique

```java
import java.util.Arrays;

public class SimpleColumnarCipher {
    // Encrypt the text using the Simple Columnar Cipher
    public static String encrypt(String text, int key) {
        int length = text.length();
        int numRows = (int) Math.ceil((double) length / key);
        char[][] grid = new char[numRows][key];

        // Fill the grid with spaces initially
        for (char[] row : grid) {
            Arrays.fill(row, ' ');
        }

        // Fill the grid with characters from the text
        int index = 0;
        for (int r = 0; r < numRows; r++) {
            for (int c = 0; c < key; c++) {
                if (index < length) {
                    grid[r][c] = text.charAt(index++);
                }
            }
        }

        // Read the grid column-wise to create the encrypted text
        StringBuilder result = new StringBuilder();
        for (int c = 0; c < key; c++) {
            for (int r = 0; r < numRows; r++) {
                if (grid[r][c] != ' ') {
                    result.append(grid[r][c]);
                }
            }
        }
        return result.toString();
    }

    // Decrypt the text using the Simple Columnar Cipher
    public static String decrypt(String text, int key) {
        int length = text.length();
        int numRows = (int) Math.ceil((double) length / key);
        char[][] grid = new char[numRows][key];

        // Fill the grid with spaces initially
        for (char[] row : grid) {
            Arrays.fill(row, ' ');
        }

        // Fill the grid column-wise with characters from the text
        int index = 0;
        for (int c = 0; c < key; c++) {
            for (int r = 0; r < numRows; r++) {
```

```
                if (index < length) {
                    grid[r][c] = text.charAt(index++);
                }
            }
        }

        // Read the grid row-wise to create the decrypted text
        StringBuilder result = new StringBuilder();
        for (int r = 0; r < numRows; r++) {
            for (int c = 0; c < key; c++) {
                if (grid[r][c] != ' ') {
                    result.append(grid[r][c]);
                }
            }
        }
        return result.toString();
    }

    public static void main(String[] args) {
        String text = "HELLO WORLD";
        int key = 5; // Number of columns
        String encrypted = encrypt(text, key);
        String decrypted = decrypt(encrypted, key);

        System.out.println("Original: " + text);
        System.out.println("Encrypted: " + encrypted);
        System.out.println("Decrypted: " + decrypted);
    }
}
```

Practical No. 4

Aim: Write program to encrypt and decrypt string using

1. DES Algorithm

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;

public class DESAlgorithm {
    // Encrypt the plaintext using the provided secret key
    public static String encrypt(String plainText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }
```

```
    // Decrypt the encrypted text using the provided secret key
    public static String decrypt(String encryptedText, SecretKey secretKey) throws
Exception {
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decodedBytes = Base64.getDecoder().decode(encryptedText);
        byte[] decryptedBytes = cipher.doFinal(decodedBytes);
        return new String(decryptedBytes);
    }

    public static void main(String[] args) throws Exception {
        String plainText = "HELLO WORLD";

        // Generate a DES key
        KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");
        SecretKey secretKey = keyGenerator.generateKey();

        // Encrypt the plaintext
        String encryptedText = encrypt(plainText, secretKey);
        System.out.println("Encrypted: " + encryptedText);

        // Decrypt the encrypted text
        String decryptedText = decrypt(encryptedText, secretKey);
        System.out.println("Decrypted: " + decryptedText);
    }
}
```

2. AES Algorithm

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;

public class AESAlgorithm {
    // Encrypt the plaintext using the provided secret key
    public static String encrypt(String plainText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    // Decrypt the encrypted text using the provided secret key
    public static String decrypt(String encryptedText, SecretKey secretKey) throws
Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decodedBytes = Base64.getDecoder().decode(encryptedText);
```

```
        byte[] decryptedBytes = cipher.doFinal(decodedBytes);
        return new String(decryptedBytes);
    }

    public static void main(String[] args) throws Exception {
        String plainText = "HELLO WORLD";

        // Generate an AES key
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(128); // Key size can be 128, 192, or 256 bits
        SecretKey secretKey = keyGenerator.generateKey();

        // Encrypt the plaintext
        String encryptedText = encrypt(plainText, secretKey);
        System.out.println("Encrypted: " + encryptedText);

        // Decrypt the encrypted text
        String decryptedText = decrypt(encryptedText, secretKey);
        System.out.println("Decrypted: " + decryptedText);
    }
}
```

Practical No 5

Aim: Write a program to implement RSA algorithm to perform encryption/decryption of a given string.

```
import java.security.*;
import javax.crypto.Cipher;
import java.util.Base64;

public class RSAAlgorithm {
    // Generate a key pair for RSA encryption
    public static KeyPair generateKeyPair() throws NoSuchAlgorithmException {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(2048); // You can use 1024, 2048, or 4096 bits for stronger security
        return keyGen.genKeyPair();
    }

    // Encrypt the plaintext using the provided public key
    public static String encrypt(String plainText, PublicKey publicKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    // Decrypt the encrypted text using the provided private key
    public static String decrypt(String encryptedText, PrivateKey privateKey) throws
Exception {
```

```
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
        return new String(decryptedBytes);
    }

    public static void main(String[] args) {
        try {
            // Generate RSA key pair
            KeyPair keyPair = generateKeyPair();
            PublicKey publicKey = keyPair.getPublic();
            PrivateKey privateKey = keyPair.getPrivate();

            // Plain text
            String plainText = "HELLO WORLD";

            // Encrypt the plain text
            String encryptedText = encrypt(plainText, publicKey);
            System.out.println("Encrypted: " + encryptedText);

            // Decrypt the encrypted text
            String decryptedText = decrypt(encryptedText, privateKey);
            System.out.println("Decrypted: " + decryptedText);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Practical No. 6

Aim: Write a program to implement the Diffe-Hellman key agreement algorithm to generate symmetric keys.

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Scanner;

public class DiffieHellmanKeyAgreement {
    public static void main(String[] args) {
        // Alice's and Bob's public values
        BigInteger p = new BigInteger("23"); // prime number
        BigInteger g = new BigInteger("5"); // generator

        // Alice's secret value
        BigInteger a = new BigInteger("6"); // Alice's private key

        // Bob's secret value
        BigInteger b = new BigInteger("15"); // Bob's private key

        // Alice computes her public value
```

```
        BigInteger A = g.modPow(a, p);

        // Bob computes his public value
        BigInteger B = g.modPow(b, p);

        // Alice computes the shared secret key
        BigInteger sharedSecretAlice = B.modPow(a, p);

        // Bob computes the shared secret key
        BigInteger sharedSecretBob = A.modPow(b, p);

        // Print the shared secret keys
        System.out.println("Alice's shared secret key: " + sharedSecretAlice);
        System.out.println("Bob's shared secret key: " + sharedSecretBob);

        // Verify that the shared secret keys are equal
        if (sharedSecretAlice.equals(sharedSecretBob)) {
            System.out.println("Shared secret keys match!");
        } else {
            System.out.println("Shared secret keys do not match!");
        }
    }
}
```

Practical No. 7

Aim: Write a program to implement MD5 Algorithm compute the message digest.

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class MD5Digest {
    public static void main(String[] args) throws NoSuchAlgorithmException {
        String message = "Hello, World!"; // input message
        byte[] messageBytes = message.getBytes();

        // Create an instance of the MD5 message digest algorithm
        MessageDigest md = MessageDigest.getInstance("MD5");

        // Compute the message digest
        byte[] digestBytes = md.digest(messageBytes);

        // Convert the digest bytes to a hexadecimal string
        StringBuilder hexString = new StringBuilder();
        for (byte b : digestBytes) {
            String hex = Integer.toHexString(0xFF & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
```

```
        // Print the MD5 message digest
        System.out.println("MD5 Message Digest: " + hexString.toString());
    }
}
```

Practical No. 8

Aim: Write a program to calculate HMAC-SHA1 signature.

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.nio.charset.StandardCharsets;

public class HmacSha1Signature {
    public static void main(String[] args) throws NoSuchAlgorithmException,
InvalidKeyException {
        String message = "Hello, World!";
        String secretKey = "my_secret_key";

        // Create a SecretKeySpec object
        SecretKeySpec secretKeySpec = new
SecretKeySpec(secretKey.getBytes(StandardCharsets.UTF_8), "HmacSHA1");

        // Create a Mac object
        Mac mac = Mac.getInstance("HmacSHA1");
        mac.init(secretKeySpec);

        // Update the Mac object with the message
        mac.update(message.getBytes(StandardCharsets.UTF_8));

        // Get the HMAC-SHA1 signature
        byte[] signatureBytes = mac.doFinal();

        // Convert the signature to a hexadecimal string
        String signatureHex = bytesToHex(signatureBytes);

        System.out.println("HMAC-SHA1 Signature: " + signatureHex);
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xFF & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
```

```
        }
        return hexString.toString();
    }
}
```