**INS PRACTICALS**

**Practical 1A:**

public class CaesarCipher {

public static String encrypt(String plaintext, int shift) {

StringBuilder encryptedText = new StringBuilder();

shift = shift % 26; // Ensure shift is within the range of the alphabet

for (char i : plaintext.toCharArray()) {

if (Character.isLetter(i)) {

char base = Character.isUpperCase(i) ? 'A' : 'a';

char encryptedChar = (char) ((i - base + shift) % 26 + base);

encryptedText.append(encryptedChar);

} else {

encryptedText.append(i); // Non-letter characters remain unchanged

}

}

return encryptedText.toString();

}

public static String decrypt(String ciphertext, int shift) {

return encrypt(ciphertext, -shift);

}

public static void main(String[] args) {

String plaintext = "Hello, World!";

int shift = 3;

String encrypted = encrypt(plaintext, shift);

String decrypted = decrypt(encrypted, shift);

System.out.println("Original: " + plaintext);

System.out.println("Encrypted: " + encrypted);

System.out.println("Decrypted: " + decrypted);

}

}

**Practical1B:**

import java.util.HashMap;

import java.util.Map;

public class MonoalphabeticCipher {

public static String encrypt(String plaintext, String substitutionAlphabet) {

String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

substitutionAlphabet = substitutionAlphabet.toUpperCase();

Map<Character, Character> translationMap = new HashMap<>();

for (int i = 0; i < alphabet.length(); i++) {

translationMap.put(alphabet.charAt(i), substitutionAlphabet.charAt(i));

}

StringBuilder encryptedText = new StringBuilder();

for (char i : plaintext.toCharArray()) {

if (Character.isLetter(i)) {

char base = Character.isUpperCase(i) ? 'A' : 'a';

char encryptedChar = translationMap.get(Character.toUpperCase(i));

if (Character.isLowerCase(i)) {

encryptedChar = Character.toLowerCase(encryptedChar);

}

encryptedText.append(encryptedChar);

} else {

encryptedText.append(i); // Non-letter characters remain unchanged

}

}

return encryptedText.toString();

}

public static String decrypt(String ciphertext, String substitutionAlphabet) {

String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

substitutionAlphabet = substitutionAlphabet.toUpperCase();

Map<Character, Character> reverseTranslationMap = new HashMap<>();

for (int i = 0; i < alphabet.length(); i++) {

reverseTranslationMap.put(substitutionAlphabet.charAt(i), alphabet.charAt(i));

}

StringBuilder decryptedText = new StringBuilder();

for (char i : ciphertext.toCharArray()) {

if (Character.isLetter(i)) {

char base = Character.isUpperCase(i) ? 'A' : 'a';

char decryptedChar = reverseTranslationMap.get(Character.toUpperCase(i));

if (Character.isLowerCase(i)) {

decryptedChar = Character.toLowerCase(decryptedChar);

}

decryptedText.append(decryptedChar);

} else {

decryptedText.append(i); // Non-letter characters remain unchanged

}

}

return decryptedText.toString();

}

public static void main(String[] args) {

String plaintext = "Hello, World!";

String substitutionAlphabet = "QWERTYUIOPASDFGHJKLZXCVBNM";

String encrypted = encrypt(plaintext, substitutionAlphabet);

String decrypted = decrypt(encrypted, substitutionAlphabet);

System.out.println("Original: " + plaintext);

System.out.println("Encrypted: " + encrypted);

System.out.println("Decrypted: " + decrypted);

}

}

**Practical 2A:**

import java.nio.charset.StandardCharsets;

import java.util.Random;

public class VernamCipher {

// Generate a random key of the same length as the plaintext

public static String generateKey(int length) {

Random random = new Random();

StringBuilder key = new StringBuilder();

for (int i = 0; i < length; i++) {

key.append((char) (random.nextInt(256))); // Generate random byte

}

return key.toString();

}

// Encrypt the plaintext using the key

public static String encrypt(String plaintext, String key) {

return xor(plaintext, key);

}

// Decrypt the ciphertext using the key (same operation as encryption)

public static String decrypt(String ciphertext, String key) {

return xor(ciphertext, key);

}

// XOR operation

private static String xor(String input, String key) {

byte[] inputBytes = input.getBytes(StandardCharsets.ISO\_8859\_1);

byte[] keyBytes = key.getBytes(StandardCharsets.ISO\_8859\_1);

byte[] resultBytes = new byte[inputBytes.length];

for (int i = 0; i < inputBytes.length; i++) {

resultBytes[i] = (byte) (inputBytes[i] ^ keyBytes[i]);

}

return new String(resultBytes, StandardCharsets.ISO\_8859\_1);

}

public static void main(String[] args) {

String plaintext = "Hello, World!";

String key = generateKey(plaintext.length());

String encrypted = encrypt(plaintext, key);

String decrypted = decrypt(encrypted, key);

System.out.println("Original: " + plaintext);

System.out.println("Key: " + key);

System.out.println("Encrypted: " + encrypted);

System.out.println("Decrypted: " + decrypted);

}

}

**Practical 2B:**

import java.util.HashMap;

import java.util.Map;

public class PlayfairCipher {

private static final int MATRIX\_SIZE = 5;

// Prepare Playfair matrix with the key

private static char[][] prepareMatrix(String key) {

String processedKey = key.toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I");

StringBuilder sb = new StringBuilder();

Map<Character, Boolean> usedChars = new HashMap<>();

for (char c : processedKey.toCharArray()) {

if (!usedChars.containsKey(c)) {

usedChars.put(c, true);

sb.append(c);

}

}

for (char c = 'A'; c <= 'Z'; c++) {

if (c != 'J' && !usedChars.containsKey(c)) {

sb.append(c);

}

}

char[][] matrix = new char[MATRIX\_SIZE][MATRIX\_SIZE];

int index = 0;

for (int i = 0; i < MATRIX\_SIZE; i++) {

for (int j = 0; j < MATRIX\_SIZE; j++) {

matrix[i][j] = sb.charAt(index++);

}

}

return matrix;

}

// Find position of a character in the matrix

private static int[] findPosition(char[][] matrix, char c) {

for (int i = 0; i < MATRIX\_SIZE; i++) {

for (int j = 0; j < MATRIX\_SIZE; j++) {

if (matrix[i][j] == c) {

return new int[]{i, j};

}

}

}

return null;

}

// Encrypt the plaintext using Playfair cipher

public static String encrypt(String plaintext, String key) {

char[][] matrix = prepareMatrix(key);

plaintext = plaintext.toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I");

if (plaintext.length() % 2 != 0) {

plaintext += 'X'; // Padding if the length is odd

}

StringBuilder ciphertext = new StringBuilder();

for (int i = 0; i < plaintext.length(); i += 2) {

char a = plaintext.charAt(i);

char b = plaintext.charAt(i + 1);

int[] posA = findPosition(matrix, a);

int[] posB = findPosition(matrix, b);

if (posA[0] == posB[0]) { // Same row

ciphertext.append(matrix[posA[0]][(posA[1] + 1) % MATRIX\_SIZE]);

ciphertext.append(matrix[posB[0]][(posB[1] + 1) % MATRIX\_SIZE]);

} else if (posA[1] == posB[1]) { // Same column

ciphertext.append(matrix[(posA[0] + 1) % MATRIX\_SIZE][posA[1]]);

ciphertext.append(matrix[(posB[0] + 1) % MATRIX\_SIZE][posB[1]]);

} else { // Rectangle

ciphertext.append(matrix[posA[0]][posB[1]]);

ciphertext.append(matrix[posB[0]][posA[1]]);

}

}

return ciphertext.toString();

}

public static void main(String[] args) {

String plaintext = "HELLO";

String key = "KEYWORD";

String ciphertext = encrypt(plaintext, key);

System.out.println("Ciphertext: " + ciphertext);

}

}

**Practical 6:**

import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.Scanner;

public class DiffieHellmanKeyAgreement{

public static void main(String[] args){

BigInteger p=new BigInteger("23");

BigInteger g=new BigInteger("5");

BigInteger a=new BigInteger("6");

BigInteger b=new BigInteger("15");

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

BigInteger B=g.modPow(b,p);

BigInteger sharedSecretAlice=B.modPow(a,p);

BigInteger sharedSecretBob=A.modPow(a,p);

System.out.println("Alice's shared secret key:"+sharedSecretAlice);

System.out.println("Bob's shared secret key:"+sharedSecretBob);

if(sharedSecretAlice.equals(sharedSecretBob))

{

System.out.println("Shared secret keys match!");

}

else{

System.out.println("Shared secret keys do not match!");

}

}

}

**Practical 7:**

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class MD5Digest{

public static void main(String[] args) throws NoSuchAlgorithmException{

String message="Hello World";

byte[] messageBytes=message.getBytes();

MessageDigest md=MessageDigest.getInstance("MD5");

byte[] digestBytes=md.digest(messageBytes);

StringBuilder hexString=new StringBuilder();

for(byte b:digestBytes){

String hex=Integer.toHexString(0xFF & b);

if(hex.length()==1){

hexString.append('0');

}

hexString.append(hex);

}

System.out.println("MD5 Message Digest:"+hexString.toString());

}

}

**Practical 8:**

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();

}

}