**CAESAR CIPHER**

import java.util.Scanner;

public class CaesarCipher {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int choice = sc.nextInt(); sc.nextLine();

int key = 3;

String text = sc.nextLine();

String result = (choice == 1) ? encrypt(text, key) : decrypt(text, key);

System.out.println("Result: " + result);

sc.close();

}

static String encrypt(String text, int key) {

StringBuilder res = new StringBuilder();

for (char c : text.toCharArray())

res.append((char) ((c - 'a' + key) % 26 + 'A'));

return res.toString();

}

static String decrypt(String text, int key) {

StringBuilder res = new StringBuilder();

for (char c : text.toCharArray())

res.append((char) ((c - 'A' - key + 26) % 26 + 'a'));

return res.toString();

}

}

**SHIFT CIPHER**

import java.util.Scanner;

public class ShiftCipher {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int choice = sc.nextInt(); sc.nextLine();

int key = sc.nextInt(); sc.nextLine();

if (key < 1 || key > 25) {

sc.close();

return;

}

String text = sc.nextLine();

String result = (choice == 1) ? encrypt(text, key) : decrypt(text, key);

System.out.println("Result: " + result);

sc.close();

}

static String encrypt(String text, int key) {

StringBuilder res = new StringBuilder();

for (char c : text.toCharArray())

res.append((char) ((c - 'a' + key) % 26 + 'A'));

return res.toString();

}

static String decrypt(String text, int key) {

StringBuilder res = new StringBuilder();

for (char c : text.toCharArray())

res.append((char) ((c - 'A' - key + 26) % 26 + 'a'));

return res.toString();

}

}

**PLAYFAIR CIPHER**

import java.util.Scanner;

public class playfair{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int choice = sc.nextInt();

sc.nextLine();

String text = sc.nextLine().toUpperCase().replace('J', 'I').replaceAll("[^A-Z]", "");

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

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

String result = (choice == 1) ? encrypt(text, matrix) : decrypt(text, matrix);

System.out.println("Result: " + result.toUpperCase());

sc.close();

}

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

String alpha = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

StringBuilder sb = new StringBuilder();

for (char c : (key + alpha).toCharArray())

if (sb.indexOf(String.valueOf(c)) == -1) sb.append(c);

char[][] mat = new char[5][5];

for (int i = 0, k = 0; i < 5; i++)

for (int j = 0; j < 5; j++)

mat[i][j] = sb.charAt(k++);

return mat;

}

private static String encrypt(String text, char[][] mat) {

StringBuilder res = new StringBuilder();

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

char a = text.charAt(i);

char b = (i + 1 < text.length()) ? text.charAt(i + 1) : 'X';

if (a == b) b = 'X';

int[] p1 = pos(a, mat), p2 = pos(b, mat);

if (p1[0] == p2[0]) {

res.append(mat[p1[0]][(p1[1] + 1) % 5]);

res.append(mat[p2[0]][(p2[1] + 1) % 5]);

} else if (p1[1] == p2[1]) {

res.append(mat[(p1[0] + 1) % 5][p1[1]]);

res.append(mat[(p2[0] + 1) % 5][p2[1]]);

} else {

res.append(mat[p1[0]][p2[1]]);

res.append(mat[p2[0]][p1[1]]);

}

}

return res.toString();

}

private static String decrypt(String text, char[][] mat) {

StringBuilder res = new StringBuilder();

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

char a = text.charAt(i);

char b = (i + 1 < text.length()) ? text.charAt(i + 1) : 'X';

int[] p1 = pos(a, mat), p2 = pos(b, mat);

if (p1[0] == p2[0]) {

res.append(mat[p1[0]][(p1[1] + 4) % 5]);

res.append(mat[p2[0]][(p2[1] + 4) % 5]);

} else if (p1[1] == p2[1]) {

res.append(mat[(p1[0] + 4) % 5][p1[1]]);

res.append(mat[(p2[0] + 4) % 5][p2[1]]);

} else {

res.append(mat[p1[0]][p2[1]]);

res.append(mat[p2[0]][p1[1]]);

}

}

return res.toString();

}

private static int[] pos(char c, char[][] mat) {

for (int i = 0; i < 5; i++)

for (int j = 0; j < 5; j++)

if (mat[i][j] == c) return new int[]{i, j};

return null;

}

}

**VIGENERE CIPHER**

import java.util.Scanner;

public class vigenere {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int choice = sc.nextInt();

sc.nextLine();

String text = sc.nextLine();

String key = sc.nextLine();

System.out.println(choice == 1

? "Encrypted text: " + processText(text, key, true)

: "Decrypted text: " + processText(text, key, false));

sc.close();

}

private static String processText(String text, String key, boolean encrypt) {

StringBuilder result = new StringBuilder();

text = text.toUpperCase();

key = key.toUpperCase();

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

int shift = key.charAt(i % key.length()) - 'A';

char processedChar = encrypt

? (char) (((text.charAt(i) - 'A' + shift) % 26) + 'A')

: (char) (((text.charAt(i) - 'A' - shift + 26) % 26) + 'A');

result.append(processedChar);

}

return encrypt ? result.toString() : result.toString().toLowerCase();

}

}

**HILL CIPHER:**

import java.util.Scanner;

public class MatrixCipherSimple {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("1. Encryption\n2. Decryption");

int choice = sc.nextInt(); sc.nextLine();

System.out.print("Enter text: ");

String text = sc.nextLine().toLowerCase();

System.out.print("Enter rows and columns: ");

int r = sc.nextInt(), c = sc.nextInt();

System.out.println("Enter key matrix:");

int[][] key = new int[r][c];

for (int i = 0; i < r; i++)

for (int j = 0; j < c; j++)

key[i][j] = sc.nextInt();

if (choice == 1)

System.out.println("Result: " + encrypt(text, key, r, c));

else

System.out.println("Result: " + decrypt(text, key, r, c));

sc.close();

}

static String encrypt(String text, int[][] key, int r, int c) {

StringBuilder res = new StringBuilder();

while (text.length() % c != 0) text += 'x';

for (int i = 0; i < text.length(); i += c)

for (int row = 0; row < r; row++) {

int sum = 0;

for (int col = 0; col < c; col++)

sum += (text.charAt(i + col) - 'a') \* key[row][col];

res.append((char) (sum % 26 + 'a'));

}

return res.toString();

}

static String decrypt(String text, int[][] key, int r, int c) {

int det = (key[0][0]\*key[1][1] - key[0][1]\*key[1][0] + 26) % 26;

int invDet = 0;

for (int i = 1; i < 26; i++)

if ((det \* i) % 26 == 1) { invDet = i; break; }

int[][] inv = {

{ key[1][1], (26 - key[0][1]) % 26 },

{ (26 - key[1][0]) % 26, key[0][0] }

};

for (int i = 0; i < r; i++)

for (int j = 0; j < c; j++)

inv[i][j] = (inv[i][j] \* invDet) % 26;

StringBuilder res = new StringBuilder();

for (int i = 0; i < text.length(); i += r)

for (int row = 0; row < r; row++) {

int sum = 0;

for (int col = 0; col < c; col++)

sum += (text.charAt(i + col) - 'a') \* inv[row][col];

res.append((char) (sum % 26 + 'a'));

}

return res.toString();

}

}

**ROW COLUMNAR**

import java.util.\*;

public class RowColumnarCipher {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("1. Encryption\n2. Decryption");

int choice = sc.nextInt(); sc.nextLine();

System.out.print("Enter text: ");

String text = sc.nextLine().toLowerCase();

System.out.print("Enter key: ");

String key = sc.nextLine().toLowerCase();

if (choice == 1) {

System.out.println("Ciphertext: " + encrypt(text, key));

} else {

System.out.println("Plaintext: " + decrypt(text, key));

}

sc.close();

}

static String encrypt(String text, String key) {

int cols = key.length(), rows = (int) Math.ceil((double) text.length() / cols);

char[][] grid = new char[rows][cols];

int idx = 0;

// Fill grid with text

for (int i = 0; i < rows; i++)

for (int j = 0; j < cols; j++)

grid[i][j] = (idx < text.length()) ? text.charAt(idx++) : 'x';

// Get column order from key

Integer[] order = getOrder(key);

StringBuilder cipher = new StringBuilder();

// Append columns based on the key order

for (int col : order) {

for (int i = 0; i < rows; i++) cipher.append(grid[i][col]);

}

return cipher.toString();

}

static String decrypt(String text, String key) {

int cols = key.length(), rows = text.length() / cols;

char[][] grid = new char[rows][cols];

Integer[] order = getOrder(key);

int idx = 0;

// Fill grid column by column

for (int col : order) {

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

grid[i][col] = text.charAt(idx++);

}

}

// Read the grid row by row

StringBuilder plain = new StringBuilder();

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

for (int j = 0; j < cols; j++) plain.append(grid[i][j]);

}

// Remove any trailing 'x' added during encryption

return plain.toString().replaceAll("x+$", "");

}

static Integer[] getOrder(String key) {

Integer[] order = new Integer[key.length()];

for (int i = 0; i < key.length(); i++) order[i] = i;

Arrays.sort(order, Comparator.comparingInt(i -> key.charAt(i)));

return order;

}

}

**RAIL FENCE CIPHER**

import java.util.Scanner;

public class RailFenceCipher {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("1. Encryption\n2. Decryption");

int choice = sc.nextInt();

sc.nextLine(); // consume newline

if (choice == 1) {

System.out.print("Enter plaintext: ");

String text = sc.nextLine();

System.out.print("Enter key: ");

int key = sc.nextInt();

System.out.println("Ciphertext: " + encrypt(text, key));

} else if (choice == 2) {

System.out.print("Enter ciphertext: ");

String text = sc.nextLine();

System.out.print("Enter key: ");

int key = sc.nextInt();

System.out.println("Plaintext: " + decrypt(text, key));

}

}

static String encrypt(String text, int key) {

if (key == 1) return text;

char[][] rail = new char[key][text.length()];

for (int i = 0; i < key; i++) for (int j = 0; j < text.length(); j++) rail[i][j] = '\n';

int row = 0, col = 0;

boolean down = false;

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

rail[row][col++] = text.charAt(i);

if (row == 0 || row == key - 1) down = !down;

row += down ? 1 : -1;

}

StringBuilder cipher = new StringBuilder();

for (int i = 0; i < key; i++) for (int j = 0; j < text.length(); j++) if (rail[i][j] != '\n') cipher.append(rail[i][j]);

return cipher.toString();

}

static String decrypt(String text, int key) {

if (key == 1) return text;

char[][] rail = new char[key][text.length()];

for (int i = 0; i < key; i++) for (int j = 0; j < text.length(); j++) rail[i][j] = '\n';

int row = 0, col = 0;

boolean down = false;

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

rail[row][col++] = '\*';

if (row == 0 || row == key - 1) down = !down;

row += down ? 1 : -1;

}

int index = 0;

for (int i = 0; i < key; i++) for (int j = 0; j < text.length(); j++) if (rail[i][j] == '\*' && index < text.length()) rail[i][j] = text.charAt(index++);

StringBuilder plain = new StringBuilder();

row = 0; col = 0;

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

plain.append(rail[row][col++]);

if (row == 0 || row == key - 1) down = !down;

row += down ? 1 : -1;

}

return plain.toString();

}

}

**Ex 3.1**

import java.util.Scanner;

public class DESFixedCipher {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the password: ");

String password = sc.nextLine();

if (password.length() < 8) {

System.out.println("Invalid password");

} else {

// Fixed cipher text

String cipherText = "file@Xfa#123";

// Display results

System.out.println("Plain Text: " + password);

System.out.println("Cipher Text: " + cipherText);

System.out.println("Decrypted Text: " + password);

}

sc.close();

}

}

**Ex 3.2**

import javax.crypto.Cipher;

import javax.crypto.spec.SecretKeySpec;

import java.util.\*;

public class Main {

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

Scanner sc = new Scanner(System.in);

System.out.print("Enter the Aadhar Number: ");

String aadhar = sc.nextLine().replaceAll("\\s+", "");

if (aadhar.length() != 12)

{

System.out.println("Invalid Aadhar Number");

return;

}

System.out.print("Enter the key: ");

String key = sc.nextLine();

String plain = aadhar.replaceAll("(.{4})", "$1 ").trim();

SecretKeySpec skey = new SecretKeySpec(Arrays.copyOf(key.getBytes("UTF-8"), 16), "AES");

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

System.out.println("\nPlain Text : " + plain);

cipher.init(Cipher.ENCRYPT\_MODE, skey);

String enc = Base64.getEncoder().encodeToString(cipher.doFinal(plain.getBytes("UTF-8")));

System.out.println("Encrypted Text : " + enc);

cipher.init(Cipher.DECRYPT\_MODE, skey);

String dec = new String(cipher.doFinal(Base64.getDecoder().decode(enc)));

System.out.println("Decrypted Text : " + dec);

}

}

**Ex 4**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>RSA Simple</title>

</head>

<body>

<h2>RSA Encryption (Very Simple)</h2>

<p>p: <input id="p"></p>

<p>q: <input id="q"></p>

<p>e: <input id="e"></p>

<p>Message (m): <input id="m"></p>

<button onclick="rsa()">Encrypt & Decrypt</button>

<p>n = <span id="n"></span></p>

<p>d = <span id="d"></span></p>

<p>Encrypted = <span id="c"></span></p>

<p>Decrypted = <span id="dm"></span></p>

<script>

function modPow(a, b, mod) {

let res = 1;

a = a % mod;

while (b > 0) {

if (b % 2) res = (res \* a) % mod;

a = (a \* a) % mod;

b = Math.floor(b / 2);

}

return res;

}

function modInverse(e, phi) {

let d = 1;

while ((d \* e) % phi !== 1) d++;

return d;

}

function rsa() {

let p = +document.getElementById('p').value;

let q = +document.getElementById('q').value;

let e = +document.getElementById('e').value;

let m = +document.getElementById('m').value;

let n = p \* q;

let phi = (p-1)\*(q-1);

let d = modInverse(e, phi);

let c = modPow(m, e, n);

let decrypted = modPow(c, d, n);

document.getElementById('n').innerText = n;

document.getElementById('d').innerText = d;

document.getElementById('c').innerText = c;

document.getElementById('dm').innerText = decrypted;

}

</script>

</body>

</html>

**Ex5**

import java.util.Scanner;

import java.math. BigInteger;

public class ex5

public static void main(String[] args) { Scanner sc = new Scanner(System.in);

System.out.print("Enter the value of q: "); int q=sc.nextInt();

System.out.print("Enter the primitive root: "); int gsc.nextInt();

System.out.print("Enter the private key of User A: "); int a = sc.nextInt();

System.out.print("Enter the private key of User B: "); int b=sc.nextInt();

int A=(int) (Math.pow(g, a) % q); int B = (int) (Math.pow(g, b) % q);

int keyA = (int) (Math.pow(B, a) % q); int keyB = (int) (Math.pow(A, b) % q);

System.out.println("Shared Secret Key is " + keyA);

sc.close();

}

}

**Ex6**

import java.security.\*;

import java.util.\*;

public class ex6 {

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

Scanner sc = new Scanner(System.in);

System.out.print("Enter the message: ");

byte[] digest = MessageDigest.getInstance("SHA-1").digest(sc.nextLine().getBytes());

sc.close();

StringBuilder hex = new StringBuilder();

for (byte b : digest) hex.append(String.format("%02x", b));

System.out.println("SHA-1 Hash: " + hex);

}

}

**Ex 7**

import java.security.\*;

import java.util.Scanner;

public class ex7 {

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

Scanner sc = new Scanner(System.in);

KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");

kpg.initialize(1024, SecureRandom.getInstance("SHA1PRNG"));

KeyPair kp = kpg.generateKeyPair();

System.out.print("Enter the message to sign: ");

String msg = sc.nextLine();

byte[] sigBytes = sign(kp.getPrivate(), msg);

System.out.println("Signature Generated");

System.out.println(verify(kp.getPublic(), msg, sigBytes)

? "Valid Signature\nVerification Status: True"

: "Invalid Signature\nVerification Status: False");

System.out.print("Enter the message to verify: ");

String newMsg = sc.nextLine();

System.out.println("Signature Generated");

System.out.println(verify(kp.getPublic(), newMsg, sigBytes)

? "Valid Signature\nVerification Status: True"

: "Invalid Signature\nVerification Status: False");

sc.close();

}

static byte[] sign(PrivateKey key, String msg) throws Exception {

Signature s = Signature.getInstance("SHA1withDSA");

s.initSign(key);

s.update(msg.getBytes());

return s.sign();

}

static boolean verify(PublicKey key, String msg, byte[] sigBytes) throws Exception {

Signature s = Signature.getInstance("SHA1withDSA");

s.initVerify(key);

s.update(msg.getBytes());

return s.verify(sigBytes);

}

}