1) a) PROGRAM:

```
import java.util.*;
import java.io.*;
public class Caesercipher {
public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";
public static String encrypt(String ptext, int cserkey) {
String ctext = "";
for (int i = 0; i < ptext.length(); i++) {
int plainnumeric = ALPHABET.indexOf(ptext.charAt(i));
int ciphernumeric = (plainnumeric + cserkey) % 26;
char cipherchar = ALPHABET.charAt(ciphernumeric);
ctext += cipherchar;
} return ctext;
}
public static String decrypt(String ctext, int cserkey) {
String ptext = "";
for (int i = 0; i < ctext.length(); i++) {
int ciphernumeric = ALPHABET.indexOf(ctext.charAt(i));
int plainnumeric = (ciphernumeric - cserkey) % 26;
if (plainnumeric < 0) {
plainnumeric = ALPHABET.length() + plainnumeric;
}
char plainchar = ALPHABET.charAt(plainnumeric);
ptext += plainchar;
} return ptext;
}
public static void main(String[] args) throws IOException {
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Enter the PLAIN TEXT for Encryption: ");
String plaintext = br.readLine();
System.out.print("Enter the CAESERKEY between 0 and 25:");
int cserkey = Integer.parseInt(br.readLine());
System.out.println("ENCRYPTION");
String ciphertext = encrypt(plaintext, cserkey);
System.out.println("CIPHER TEXT : " + ciphertext);
System.out.println("DECRYPTION");
plaintext = decrypt(ciphertext, cserkey);
System.out.println("PLAIN TEXT : " + plaintext);
}
}
```

```
Enter the PLAIN TEXT for
Encryption:  hello
Enter the CAESERKEY
between 0 and 25: 3
ENCRYPTION
CIPHER TEXT : khoor
DECRYPTION
PLAIN TEXT : hello
```

1) b) PROGRAM:

```
import java.util.*;
import java.io.*;
public class Playfairs {
private char pfmatrix[][] = new char[5][5];
public String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
String plain, cipher;
int jflg = 0, xpad = 0;
int row, col;
public void matrixgen(String key) {
key = key.toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I");
boolean[] used = new boolean[26];
int i = 0, j = 0;
for (int k = 0; k < key.length(); k++) {
char ch = key.charAt(k);
if (!used[ch - 'A']) {
pfmatrix[i][j] = ch;
used[ch - 'A'] = true;
j++;
if (j == 5) {
j = 0;
i++;
}
}
}
for (char ch = 'A'; ch <= 'Z'; ch++) {
if (ch == 'J') continue;
if (!used[ch - 'A']) {
pfmatrix[i][j] = ch;
used[ch - 'A'] = true;
j++;
if (j == 5) {
j = 0;
i++;
}
}
}
}
public void matrixdisplay() {
for (int i = 0; i < 5; i++) {
for (int j = 0; j < 5; j++)
```

```
System.out.print(pfmatrix[i][j] + " ");
System.out.println();
}
}
public String pfencryption(String txt) {
int ch1row, ch2row, ch1col, ch2col;
char ch1, ch2, tmp1, tmp2;
String nutext = "", text = "";
int i = 0;
txt = txt.toUpperCase().replace("J", "I").replaceAll("[^A-Z]", "");
while (i < txt.length()) {
text += txt.charAt(i);
if (i + 1 < txt.length()) {
if (txt.charAt(i) == txt.charAt(i + 1)) {
text += 'X';
xpad++;
} else {
text += txt.charAt(i + 1);
i++;
}
} else {
text += 'X';
xpad++;
}
i++;
}
System.out.println("TEXT : " + text);
for (int k = 0; k < text.length(); k += 2) {
ch1 = text.charAt(k);
ch2 = text.charAt(k + 1);
System.out.println("CHARACTER PAIR : " + ch1 + " " + ch2);
matsearch(ch1);
ch1row = row;
ch1col = col;
matsearch(ch2);
ch2row = row;
ch2col = col;
//System.out.println("ch1row: " + ch1row + " ch1col: " + ch1col);
//System.out.println("ch2row: " + ch2row + " ch2col: " + ch2col);
if (ch1row == ch2row) {
tmp1 = pfmatrix[ch1row][(ch1col + 1) % 5];
```

```
tmp2 = pfmatrix[ch2row][(ch2col + 1) % 5];
} else if (ch1col == ch2col) {
tmp1 = pfmatrix[(ch1row + 1) % 5][ch1col];
tmp2 = pfmatrix[(ch2row + 1) % 5][ch2col];
} else {
tmp1 = pfmatrix[ch1row][ch2col];
tmp2 = pfmatrix[ch2row][ch1col];
}
nutext += tmp1;
nutext += tmp2;
System.out.println("TRANSLATED TEXT : " + tmp1 + " " + tmp2);
}
return nutext;
}
public String pfdecryption(String text) {
int ch1row, ch2row, ch1col, ch2col;
char ch1, ch2, tmp1, tmp2;
String nutext = "", txt = "";
for (int k = 0; k < text.length(); k += 2) {
ch1 = text.charAt(k);
ch2 = text.charAt(k + 1);
System.out.println("CHARACTER PAIR : " + ch1 + " " + ch2);
matsearch(ch1);
ch1row = row;
ch1col = col;
matsearch(ch2);
ch2row = row;
ch2col = col;
//System.out.println("ch1row: " + ch1row + " ch1col: " + ch1col);
//System.out.println("ch2row: " + ch2row + " ch2col: " + ch2col);
if (ch1row == ch2row) {
int c1 = ch1col - 1;
if (c1 < 0) c1 += 5;
int c2 = ch2col - 1;
if (c2 < 0) c2 += 5;
tmp1 = pfmatrix[ch1row][c1];
tmp2 = pfmatrix[ch2row][c2];
} else if (ch1col == ch2col) {
int r1 = ch1row - 1;
int r2 = ch2row - 1;
if (r1 < 0) r1 += 5;
```

```
 if (r2 < 0) r2 += 5;
 tmp1 = pfmatrix[r1][ch1col];
 tmp2 = pfmatrix[r2][ch2col];
 } else {
 tmp1 = pfmatrix[ch1row][ch2col];
 tmp2 = pfmatrix[ch2row][ch1col];
 }
 nutext += tmp1;
 nutext += tmp2;
 System.out.println("TRANSLATED TEXT : " + tmp1 + " " + tmp2);
 }
 if (xpad != 0) {
 int i = 0;
 while (i < nutext.length()) {
 if (nutext.charAt(i) == 'X') {
 i++;
 continue;
 }
 txt += nutext.charAt(i);
i++;
}
System.out.println("TEXT : " + txt);
return txt;
} else {
System.out.println("TEXT : " + nutext);
return nutext;
}
}
public void matsearch(char ch) {
if (ch == 'J') ch = 'I';
for (int i = 0; i < 5; i++) {
for (int j = 0; j < 5; j++) {
if (pfmatrix[i][j] == ch) {
row = i;
col = j;
return;
}
}
}
}
public static void main(String[] args) {
```

```
Playfairs pf = new Playfairs();
Scanner sc = new Scanner(System.in);
System.out.println("Enter the PLAYFAIR KEY: ");
String pfkey = sc.nextLine();
System.out.println("PLAYFAIR MATRIX");
pf.matrixgen(pfkey);
pf.matrixdisplay();
System.out.println("Enter PLAIN TEXT");
String ptext = sc.nextLine();
String ctext = pf.pfencryption(ptext);
System.out.println("\nCIPHER TEXT :" + ctext);
String plaintext = pf.pfdecryption(ctext);
System.out.println("\nPLAIN TEXT :" + plaintext);
sc.close();
}
}
```

OUTPUT:

```
srmvec@cse-2:~/Downloads/Code$ java Playfairs.java
Enter the PLAYFAIR KEY:
MONARCHY
PLAYFAIR MATRIX
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z
Enter PLAIN TEXT
INSTRUMENTS
TEXT : INSTRUMENTSX
CHARACTER PAIR : I N
TRANSLATED TEXT : G A
CHARACTER PAIR : S T
TRANSLATED TEXT : T L
CHARACTER PAIR : R U
TRANSLATED TEXT : M Z
CHARACTER PAIR : M E
TRANSLATED TEXT : C L
CHARACTER PAIR : N T
TRANSLATED TEXT : R Q
CHARACTER PAIR : S X
TRANSLATED TEXT : X A
```

```
CIPHER TEXT :GATLMZCLRQXA
CHARACTER PAIR : G A
TRANSLATED TEXT : I N
CHARACTER PAIR : T L
TRANSLATED TEXT : S T
CHARACTER PAIR : M Z
TRANSLATED TEXT : R U
CHARACTER PAIR : C L
TRANSLATED TEXT : M E
CHARACTER PAIR : R Q
TRANSLATED TEXT : N T
CHARACTER PAIR : X A
TRANSLATED TEXT : S X
TEXT : INSTRUMENTS
PLAIN TEXT :INSTRUMENTS
```

1) c) PROGRAM:

```
import java.util.*;
import java.io.*
public class Hillcipher {
public int keyinverse[][] = new int[3][3];
public int key[][] = { {17, 17, 5}, {21, 18, 21}, {2, 2, 19} };
public int plainmat[][] = new int[8][3];
public int ciphermat[][] = new int[8][3];
public String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
String plain, cipher;
int row, flag = 0, decrypt = 0;
public void matdisplay(int mat[][]) {
for (int i = 0; i < row; i++) {
for (int j = 0; j < 3; j++)
System.out.print(mat[i][j] + " ");
System.out.println();
}
}
public void keydisplay(int mat[][]) {
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++)
System.out.print(mat[i][j] + " ");
System.out.println();
}
}
public void inverse() {
int dtrmnt = 0, mulinvdtrmnt = 0, x, y, z, i, j, a, tmp, p, q;
int transkey[][] = new int[3][3];
int minormat[][] = new int[3][3];
int temp[][] = new int[2][2];
System.out.println("HILL CIPHER KEY");
keydisplay(key);
x = key[0][0] * ((key[1][1] * key[2][2]) - (key[1][2] * key[2][1]));
y = key[0][1] * ((key[1][0] * key[2][2]) - (key[1][2] * key[2][0]));
z = key[0][2] * ((key[1][0] * key[2][1]) - (key[1][1] * key[2][0]));
dtrmnt = (x - y + z) % 26;
if (dtrmnt < 0) dtrmnt += 26;
System.out.println("DETERMINANT :" + dtrmnt);
a = dtrmnt;
for (i = 0; i < 26; i++) {
tmp = (a * i) % 26;
if (tmp == 1) {
mulinvdtrmnt = i;
break;
}
 }
 System.out.println("MULTIPLICATIVE INVERSE OF DETERMINANT:" +
mulinvdtrmnt);
 for (i = 0; i < 3; i++) {
 for (j = 0; j < 3; j++)
 transkey[i][j] = key[j][i];
 }
 System.out.println("TRANSPOSED KEY");
 keydisplay(transkey);
 for (i = 0; i < 3; i++) {
 for (j = 0; j < 3; j++) {
 p = 0; q = 0;
 for (x = 0; x < 3; x++) {
 for (y = 0; y < 3; y++) {
 if (x != i && y != j) {
 temp[p][q] = transkey[x][y];
 q++;
 if (q == 2) { q = 0; p++; }
 }
 }
 }
 minormat[i][j] = (temp[0][0] * temp[1][1]) - (temp[0][1] * temp[1][0]);
 minormat[i][j] = minormat[i][j] * (int) Math.pow(-1, (i + j));
 }
 }
 for (i = 0; i < 3; i++) {
 for (j = 0; j < 3; j++) {
 keyinverse[i][j] = (mulinvdtrmnt * minormat[i][j]) % 26;
 if (keyinverse[i][j] < 0) keyinverse[i][j] += 26;
 }
 }
 System.out.println("KEY INVERSE");
 keydisplay(keyinverse);
 }
 public void str2matrix(String text) {
 int k, p, n;
 flag = 0;
```

```
if ((text.length() % 3) == 1) {
n = text.length(); text += "XX"; flag = 2;
} else if ((text.length() % 3) == 2) {
text += "X"; flag = 1;
}
row = text.length() / 3; k = 0;
for (int i = 0; i < row; i++) {
for (int j = 0; j < 3; j++) {
for (p = 0; p < 26; p++) {
if (text.charAt(k) == ALPHABET.charAt(p)) {
if (decrypt == 1) ciphermat[i][j] = p;
else plainmat[i][j] = p;
k++; break;
}
}
}
}
System.out.println((decrypt == 1 ? "CIPHER" : "PLAIN") + " TEXT MATRIX");
if (decrypt == 1) matdisplay(ciphermat);
else matdisplay(plainmat);
System.out.println();
}
public String matrix2str(int mat[][]) {
String txt = "", tmp = "";
for (int i = 0; i < row; i++) {
for (int j = 0; j < 3; j++) {
int k = mat[i][j];
txt += ALPHABET.charAt(k);
}
}
if (decrypt == 1) {
if (flag == 1) { tmp = txt.substring(0, txt.length() - 1); return tmp; }
if (flag == 2) { tmp = txt.substring(0, txt.length() - 2); return tmp; }
}
return txt;
}
public String hcencryption(String ptxt) {
int sum; String ctxt = ""; decrypt = 0;
System.out.println("HILL CIPHER ENCRYPTION");
str2matrix(ptxt);
for (int i = 0; i < row; i++) {
```

```
for (int j = 0; j < 3; j++) {
sum = 0;
for (int k = 0; k < 3; k++) sum += plainmat[i][k] * key[k][j];
ciphermat[i][j] = sum % 26;
}
}
System.out.println("CIPHER TEXT MATRIX");
matdisplay(ciphermat);
ctxt = matrix2str(ciphermat);
return ctxt;
}
public String hcdecryption(String ctxt) {
int sum; String ptxt = ""; decrypt = 1;
System.out.println("HILL CIPHER DECRYPTION");
str2matrix(ctxt);
for (int i = 0; i < row; i++) {
for (int j = 0; j < 3; j++) {
sum = 0;
for (int k = 0; k < 3; k++) sum += ciphermat[i][k] * keyinverse[k][j];
plainmat[i][j] = sum % 26;
if (plainmat[i][j] < 0) plainmat[i][j] += 26;
}
}
System.out.println("PLAIN TEXT MATRIX");
matdisplay(plainmat);
ptxt = matrix2str(plainmat);
return ptxt;
}
public static void main(String[] args) {
Hillcipher hc = new Hillcipher();
Scanner sc = new Scanner(System.in);
hc.inverse();
System.out.println("Enter PLAIN TEXT");
String ptext = sc.next().toUpperCase();
String ctext = hc.hcencryption(ptext);
System.out.println("\nCIPHER TEXT :" + ctext);
String plaintext = hc.hcdecryption(ctext);
System.out.println("\nPLAIN TEXT :" + plaintext);
sc.close();
}
}
```

OUTPUT:

```
srmvec@cse-2:~/Downloads/Code$ java Hillcipher.java
HILL CIPHER KEY
17 17 5
21 18 21
2 2 19
DETERMINANT :23
MULTIPLICATIVE INVERSE OF DETERMINANT:17
TRANSPOSED KEY
17 21 2
17 18 2
5 21 19
KEY INVERSE
4 9 15
15 17 6
24 0 17
Enter PLAIN TEXT
PAY
HILL CIPHER ENCRYPTION
PLAIN TEXT MATRIX
15 0 24
CIPHER TEXT MATRIX
17 17 11
CIPHER TEXT :RRL
HILL CIPHER DECRYPTION
CIPHER TEXT MATRIX
17 17 11
PLAIN TEXT MATRIX
15 0 24
PLAIN TEXT :PAY
```

1) d) PROGRAM:

```
import java.util.*;
import java.io.*;
public class Vigenerecipher {
public static String key = new String();
public String extndkey;
public String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
public String keyextnsn(String ptxt, String keytxt) {
int j = 0;
String nukey = "";
for (int i = 0; i < ptxt.length(); i++) {
nukey += keytxt.charAt(j);
j++;
if (j == keytxt.length()) j = 0;
}
return nukey;
}
public int valueofchar(char x) {
for (int i = 0; i < 26; i++) {
if (x == ALPHABET.charAt(i)) return i;
}
return 0;
}
public char charofvalue(int y) {
return ALPHABET.charAt(y);
}
public String vcencryption(String txt) {
int p, k, tmp1;
char tmp;
String ctxt = "";
extndkey = keyextnsn(txt, key);
System.out.println("VIGENERE ENCRYPTION");
System.out.println("PLAIN TEXT : " + txt);
System.out.println("VIGENERE KEY : " + extndkey);
for (int i = 0; i < txt.length(); i++) {
p = valueofchar(txt.charAt(i));
k = valueofchar(extndkey.charAt(i));
tmp1 = (p + k) % 26;
tmp = charofvalue(tmp1);
ctxt += tmp;
}
return ctxt;
}
public String vcdecryption(String txt) {
int c, k, tmp1;
char ch;
String ptxt = "";
System.out.println("VIGENERE DECRYPTION");
System.out.println("CIPHER TEXT : " + txt);
System.out.println("VIGENERE KEY : " + extndkey);
for (int i = 0; i < txt.length(); i++) {
c = valueofchar(txt.charAt(i));
k = valueofchar(extndkey.charAt(i));
tmp1 = (c - k + 26) % 26;
ch = charofvalue(tmp1);
ptxt += ch;
}
return ptxt;
}
public static void main(String[] args) {
Vigenerecipher vc = new Vigenerecipher();
Scanner sc = new Scanner(System.in);
System.out.println("ENTER KEY");
key = sc.next().toUpperCase();
System.out.println("Enter PLAIN TEXT");
String text = sc.next().toUpperCase();
String ciphertext = vc.vcencryption(text);
System.out.println("\nCIPHER TEXT :" + ciphertext);
String plaintext = vc.vcdecryption(ciphertext);
System.out.println("\nPLAIN TEXT :" + plaintext);
sc.close();
}
}
```

```
ENTER KEY
LEMON
Enter PLAIN TEXT
ATTACKATDAWN
VIGENERE ENCRYPTION
PLAIN TEXT : ATTACKATDAWN
VIGENERE KEY : LEMONLEMONLE
      CIPHER TEXT :LXFOPVEFRNHR
```

```
VIGENERE DECRYPTION
CIPHER TEXT : LXFOPVEFRNHR
VIGENERE KEY : LEMONLEMONLE
PLAIN TEXT :ATTACKATDAWN
```

2) PROGRAM:

```
import java.util.*;
public class Railfence {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
int n = 7;
int[] key = new int[n];
System.out.println("Enter 7-digit key for column permutation (values 1 to 7 in
any order):");
for (int i = 0; i < n; i++) key[i] = sc.nextInt();
System.out.print("\nEnter PLAIN TEXT: ");
String plain = sc.next().toUpperCase();
while (plain.length() % n != 0) plain += "X";
int rows = plain.length() / n;
char[][] mat = new char[rows][n];
int k = 0;
for (int i = 0; i < rows; i++) {
for (int j = 0; j < n; j++) {
mat[i][j] = plain.charAt(k++);
}
}
System.out.println("\nColumn-wise filled Matrix:");
for (char[] r : mat) {
for (char c : r) System.out.print(c + " ");
System.out.println();
}
StringBuilder cipher = new StringBuilder();
for (int col : key) {
for (int i = 0; i < rows; i++) cipher.append(mat[i][col - 1]);
}
System.out.println("\nCIPHER TEXT: " + cipher);
char[][] decMat = new char[rows][n];
k = 0;
for (int col : key) {
for (int i = 0; i < rows; i++) {
decMat[i][col - 1] = cipher.charAt(k++);
}
}
System.out.println("\nRow-wise filled Decryption Matrix:");
for (char[] r : decMat) {
for (char c : r) System.out.print(c + " ");
System.out.println();
}
StringBuilder dec = new StringBuilder();
for (int i = 0; i < rows; i++) {
for (int j = 0; j < n; j++) dec.append(decMat[i][j]);
}
System.out.println("\nDECRYPTED TEXT: " + dec);
}
}
```

OUTPUT:

```
Enter 7-digit key for column permutation (values 1 to 7 in any order):
 3
 1
 4
 7
 2
 6
 5
 Enter PLAIN TEXT: HELLOWORLD
 Column-wise filled Matrix:
 H E L L O W O
 R L D X X X X
 CIPHER TEXT: LDHRLXOXELWXOX
 Row-wise filled Decryption Matrix:
 H E L L O W O
 R L D X X X X
 DECRYPTED TEXT: HELLOWORLDXXXX
```

3) PROGRAM:

```
import javax.swing.*;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.Random;
public class DES {
byte[] skey = new byte[1000];
String skeyString;
static byte[] raw;
String inputMessage, encryptedData, decryptedMessage;
public DES() {
try {
generateSymmetricKey();
inputMessage = JOptionPane.showInputDialog(null, "Enter message to
encrypt");
if (inputMessage == null | inputMessage.isEmpty()) {
JOptionPane.showMessageDialog(null, "No input provided.");
return;
}
byte[] inputBytes = inputMessage.getBytes();
byte[] encryptedBytes = encrypt(raw, inputBytes);
encryptedData = Base64.getEncoder().encodeToString(encryptedBytes);
System.out.println("Encrypted message: " + encryptedData);
JOptionPane.showMessageDialog(null, "Encrypted Data:\n" + encryptedData);
byte[] decryptedBytes = decrypt(raw,
Base64.getDecoder().decode(encryptedData));
decryptedMessage = new String(decryptedBytes);
System.out.println("Decrypted message: " + decryptedMessage);
JOptionPane.showMessageDialog(null, "Decrypted Data:\n" +
decryptedMessage);
} catch (Exception e) {
e.printStackTrace();
}
}
void generateSymmetricKey() {
try {
Random r = new Random();
int num = r.nextInt(10000);
String knum = String.valueOf(num);
byte[] knumb = knum.getBytes();
skey = getRawKey(knumb);
skeyString = Base64.getEncoder().encodeToString(skey);
System.out.println("DES Symmetric key (Base64) = " + skeyString);
} catch (Exception e) {
e.printStackTrace();
}
}
private static byte[] getRawKey(byte[] seed) throws Exception {
KeyGenerator kgen = KeyGenerator.getInstance("DES");
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
sr.setSeed(seed);
kgen.init(56, sr);
SecretKey skey = kgen.generateKey();
raw = skey.getEncoded();
return raw;
}
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
return cipher.doFinal(clear);
}
private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {
SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.DECRYPT_MODE, skeySpec);
return cipher.doFinal(encrypted);
}
public static void main(String[] args) {
new DES();
}
}
```

OUTPUT:

```
srmvec@cse-2:~/Downloads/Code$ java DES.java
DES Symmetric key (Base64) = JQGA7wvHXdM=
Encrypted message: Uqchn8XuG+4=
Decrypted message: Hello
```

4) PROGRAM:

```
package com.includehelp.stringsample;
import java.util.Base64;
import java.util.Scanner;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class AES {
private static final String encryptionKey = "ABCDEFGHIJKLMNOP";
private static final String characterEncoding = "UTF-8";
private static final String cipherTransformation = "AES/CBC/PKCS5PADDING";
private static final String aesEncryptionAlgorithm = "AES";
public static String encrypt(String plainText) {
try {
Cipher cipher = Cipher.getInstance(cipherTransformation);
byte[] keyBytes = encryptionKey.getBytes(characterEncoding);
SecretKeySpec secretKey = new SecretKeySpec(keyBytes,
aesEncryptionAlgorithm);
IvParameterSpec ivParameterSpec = new IvParameterSpec(keyBytes);
cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivParameterSpec);
byte[] encryptedBytes = cipher.doFinal(plainText.getBytes(characterEncoding));
return Base64.getEncoder().encodeToString(encryptedBytes);
} catch (Exception e) {
System.err.println("Encryption error: " + e.getMessage());
return null;
}
}
public static String decrypt(String encryptedText) {
try {
Cipher cipher = Cipher.getInstance(cipherTransformation);
byte[] keyBytes = encryptionKey.getBytes(characterEncoding);
SecretKeySpec secretKey = new SecretKeySpec(keyBytes,
aesEncryptionAlgorithm);
IvParameterSpec ivParameterSpec = new IvParameterSpec(keyBytes);
cipher.init(Cipher.DECRYPT_MODE, secretKey, ivParameterSpec);
byte[] decodedBytes = Base64.getDecoder().decode(encryptedText);
byte[] decryptedBytes = cipher.doFinal(decodedBytes);
return new String(decryptedBytes, characterEncoding);
} catch (Exception e) {
System.err.println("Decryption error: " + e.getMessage());
return null;
```

```
 }
 }
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 System.out.print("Enter String to Encrypt: ");
 String plainText = scanner.nextLine();
String encrypted = encrypt(plainText);
if (encrypted != null) {
System.out.println("Encrypted String: " + encrypted);
String decrypted = decrypt(encrypted);
System.out.println("Decrypted String: " + decrypted);
} else {
System.out.println("Encryption failed.");
}
scanner.close();
}
}
```

OUTPUT:

```
srmvec@cse-2:~/Downloads/Code$ java AES.java
Enter String to Encrypt: Bolt
Encrypted String: yihmWC6eGdhMqZOkEaOSkg==
Decrypted String: Bolt
```

5) PROGRAM:

```
import java.math.BigInteger;
import java.util.Random;
public class RSA {
private BigInteger p, q, n, phi, e, d;
private int bitLen = 1024;
private Random rand;
private static String bytesToString(byte[] bytes) {
StringBuilder result = new StringBuilder();
for (byte b : bytes) {
result.append(b).append(" ");
}
return result.toString();
}
public RSA() {
rand = new Random();
p = BigInteger.probablePrime(bitLen, rand);
q = BigInteger.probablePrime(bitLen, rand);
n = p.multiply(q);
phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
e = BigInteger.probablePrime(bitLen / 2, rand);
while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0) {
e = e.add(BigInteger.ONE);
}
d = e.modInverse(phi);
}
public byte[] encrypt(byte[] message) {
return new BigInteger(message).modPow(e, n).toByteArray();
}
public byte[] decrypt(byte[] encrypted) {
return new BigInteger(encrypted).modPow(d, n).toByteArray();
}
public static void main(String[] args) {
try {
RSA rsa = new RSA();
String message = "SRM Valliammai";
System.out.println("=== RSA Algorithm Simulation ===");
System.out.println("Original message (string): " + message);
System.out.println("Original message (bytes): " +
bytesToString(message.getBytes()));
byte[] encrypted = rsa.encrypt(message.getBytes());
System.out.println("Encrypted message (bytes): " + bytesToString(encrypted));
byte[] decrypted = rsa.decrypt(encrypted);
System.out.println("Decrypted message (bytes): " + bytesToString(decrypted));
System.out.println("Decrypted message (string): " + new String(decrypted));
} catch (Exception e) {
System.err.println("Exception: " + e.getMessage());
e.printStackTrace();
}
}
}
```

OUTPUT:

```
=== RSA Algorithm Simulation ===
Original message (string): Bolt
Original message (bytes): 66 111 108 116
Encrypted message (bytes): 43 -15 -52 -113 -50 -13 3 -7 -77 -28 -95 105 -
94 11 -67 42 11 105 50 58 -82 -2 45 6 -78 47 64 -12 122 -23 40 -109 114 -
21 96 78 27 0 122 -10 -72 -69 -30 99 -49 81 20 -33 -52 113 -89 -52 -101
50 93 -78 53 -25 67 59 -119 -95 35 -59 44 21 -112 -124 -99 83 118 -58 1 -
120
124 -9 7 92 111 36 126 74 -26 -11 -105 -16 41 24 -86 39 -28 -89 -108 -74
90 79 -60 14 -32 -20 -14 47 -28 -56 -60 86 78 76 -34 49 -11 33 17 -55 5
127 -31 110 113 -124 53 16 40 -37 103 23 103 -104 -11 -117 -91 -102 40
62 119 94 -86 39 -100 -116 -73 -40 110 34 119 63 64 -74 28 -65 -102 -87
112 -71 5
118 115 -79 -20 -104 30 -37 -51 -114 96 55 -23 67 -69 61 -77 27 96 15 -
127 -121 119 -65 -52 -49 38
37 115 58 94 112 107 118 -40 11 94 9 -17 7 -50 -90 -21 -20 70 74 25 23 -
108 111 -39 19 -95 46 98 -10 -97 18 -106 42 70 54 -123 -12 62 109 56 82
119 -71 -93 -122 8 40 8 52 -128 119 -74 98 -105 54 -44 -109 -73 -103 -19 -70
-128 18 82 7 6 88 108 31 80 -38 -8 -21 -56 35
Decrypted message (bytes): 66 111 108 116
Decrypted message (string): Bolt
```

6) PROGRAM:

```
import java.io.*;
import java.math.BigInteger;
class DHKE {
public static void main(String[] args) throws IOException {
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Enter a prime number (p): ");
BigInteger p = new BigInteger(br.readLine());
System.out.print("Enter a primitive root modulo p (g): ");
BigInteger g = new BigInteger(br.readLine());
System.out.print("Enter private key for sender (x), 0 < x < p: ");
BigInteger x = new BigInteger(br.readLine());
BigInteger R1 = g.modPow(x, p);
System.out.println("Sender's public key (R1) = " + R1);
System.out.print("Enter private key for receiver (y), 0 < y < p: ");
BigInteger y = new BigInteger(br.readLine());
BigInteger R2 = g.modPow(y, p);
System.out.println("Receiver's public key (R2) = " + R2);
BigInteger k1 = R2.modPow(x, p);
System.out.println("Shared secret key computed by sender: " + k1);
BigInteger k2 = R1.modPow(y, p);
System.out.println("Shared secret key computed by receiver: " + k2);
if (k1.equals(k2)) {
System.out.println("Success! Both shared keys match.");
} else {
System.out.println("Error! Shared keys do not match.");
}
}
}
```

OUTPUT:

```
Enter a prime number (p): 23
Enter a primitive root modulo p (g): 5
Enter private key for sender (x), 0 < x < p: 6
Sender's public key (R1) = 8
Enter private key for receiver (y), 0 < y < p: 15
Receiver's public key (R2) = 19
Shared secret key computed by sender: 2
Shared secret key computed by receiver: 2
Success! Both shared keys match.
```

7) PROGRAM:

```
import java.security.*;
public class SHA {
public static void main(String[] a) {
try {
MessageDigest md = MessageDigest.getInstance("SHA-1");
String input = "srm";
md.update(input.getBytes());
byte[] output = md.digest();
System.out.println();
System.out.println("SHA1(\"" + input + "\") = " + bytesToHex(output));
input = "vec";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("SHA1(\"" + input + "\") = " + bytesToHex(output));
input = "valliammai";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("SHA1(\"" + input + "\") = " + bytesToHex(output));
System.out.println();
} catch (Exception e) {
System.out.println("Exception: " + e);
}
}
public static String bytesToHex(byte[] b) {
char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
StringBuffer buf = new StringBuffer();
for (int j = 0; j < b.length; j++) {
buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
buf.append(hexDigit[b[j] & 0x0f]);
}
return buf.toString();
}
}
```

```
SHA1("srm") =
FEB97F07D083079080E3AE6A9523F1FC3FFF6833
SHA1("vec") =
B803E7CA5C714DBD85BF00D511FBC99A77690AD2
SHA1("valliammai") =
BD46D71D6A8424C88ABBC25B40050B370A23B0FB
```

8) PROGRAM:

```
import java.util.*;
import java.math.BigInteger;
class DSS {
final static BigInteger one = BigInteger.ONE;
final static BigInteger zero = BigInteger.ZERO;
public static BigInteger getNextPrime(String ans) {
BigInteger test = new BigInteger(ans);
while (!test.isProbablePrime(99)) {
test = test.add(one);
}
return test;
}
public static BigInteger findQ(BigInteger n) {
BigInteger start = new BigInteger("2");
while (!n.isProbablePrime(99)) {
while (!(n.mod(start).equals(zero))) {
start = start.add(one);
}
n = n.divide(start);
}
return n;
}
public static BigInteger getGen(BigInteger p, BigInteger q, Random r) {
BigInteger h;
do {
h = new BigInteger(p.bitLength(), r);
h = h.mod(p.subtract(one)).add(one);
} while (h.equals(one));
return h.modPow((p.subtract(one)).divide(q), p);
}
public static void main(String[] args) throws Exception {
Random randObj = new Random();
BigInteger p = getNextPrime("10600");
BigInteger q = findQ(p.subtract(one));
BigInteger g = getGen(p, q, randObj);
System.out.println("Digital Signature Algorithm");
System.out.println("Global public key components are:");
System.out.println("p is: " + p);
System.out.println("q is: " + q);
System.out.println("g is: " + g);
BigInteger x = new BigInteger(q.bitLength(), randObj).mod(q);
BigInteger y = g.modPow(x, p);
BigInteger k;
do {
k = new BigInteger(q.bitLength(), randObj).mod(q);
} while (k.equals(BigInteger.ZERO) | !k.gcd(q).equals(BigInteger.ONE));
BigInteger hashVal = new BigInteger(p.bitLength(), randObj);
BigInteger r = (g.modPow(k, p)).mod(q);
BigInteger kInv = k.modInverse(q);
BigInteger s = kInv.multiply(hashVal.add(x.multiply(r))).mod(q);
System.out.println("Secret information:");
System.out.println("x (private) is: " + x);
System.out.println("k (secret) is: " + k);
System.out.println("y (public) is: " + y);
System.out.println("h (random hash) is: " + hashVal);
System.out.println("Generating digital signature:");
System.out.println("r is: " + r);
System.out.println("s is: " + s);
BigInteger w = s.modInverse(q);
BigInteger u1 = (hashVal.multiply(w)).mod(q);
BigInteger u2 = (r.multiply(w)).mod(q);
BigInteger v = (g.modPow(u1, p).multiply(y.modPow(u2, p))).mod(p).mod(q);
System.out.println("Verifying digital signature (checkpoints):");
System.out.println("w is: " + w);
System.out.println("u1 is: " + u1);
System.out.println("u2 is: " + u2);
System.out.println("v is: " + v);
if (v.equals(r)) {
System.out.println("Success: digital signature is verified! " + r);
} else {
System.out.println("Error: incorrect digital signature");
}
}
}
```

OUTPUT:

```
Digital Signature Algorithm
Global public key components are:
p is: 10601
q is: 53
g is: 2863
Secret information:
```

```
x (private) is: 47
k (secret) is: 9
y (public) is: 5582
h (random hash) is: 13379
Generating digital signature:
r is: 25
s is: 33
Verifying digital signature (checkpoints):
w is: 45
u1 is: 28
u2 is: 12
v is: 25
Success: digital signature is verified! 25
```

9) PROGRAM:

```
import java.util.Arrays;
import java.security.MessageDigest;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.spec.IvParameterSpec;
import java.util.Base64;
import java.nio.charset.StandardCharsets;
public class TDES {
public static void main(String[] args) throws Exception {
String message = "Bolt";
String secretKey = "SecretKey";
TDES tdes = new TDES();
String encrypted = tdes.encrypt(message, secretKey);
String decrypted = tdes.decrypt(encrypted, secretKey);
System.out.println("Encrypted (Base64): " + encrypted);
System.out.println("Decrypted: " + decrypted);
}
public String encrypt(String message, String secretKey) throws Exception {
MessageDigest md = MessageDigest.getInstance("SHA-1");
byte[] digest = md.digest(secretKey.getBytes(StandardCharsets.UTF_8));
byte[] keyBytes = Arrays.copyOf(digest, 24);
for (int i = 16; i < 24; i++) {
keyBytes[i] = keyBytes[i - 16];
}
SecretKey key = new SecretKeySpec(keyBytes, "DESede");
byte[] ivBytes = new byte[8];
SecureRandom random = new SecureRandom();
random.nextBytes(ivBytes);
IvParameterSpec iv = new IvParameterSpec(ivBytes);
Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key, iv);
byte[] plainTextBytes = message.getBytes(StandardCharsets.UTF_8);
byte[] encryptedBytes = cipher.doFinal(plainTextBytes);
byte[] combined = new byte[ivBytes.length + encryptedBytes.length];
System.arraycopy(ivBytes, 0, combined, 0, ivBytes.length);
System.arraycopy(encryptedBytes, 0, combined, ivBytes.length,
encryptedBytes.length);
return Base64.getEncoder().encodeToString(combined);
```

```
Encrypted (Base64):
wnKIAXvesaoV7q6H+DQHHw==
Decrypted: Bolt
```

```
}
public String decrypt(String encryptedText, String secretKey) throws Exception {
byte[] combined = Base64.getDecoder().decode(encryptedText);
byte[] ivBytes = Arrays.copyOfRange(combined, 0, 8);
IvParameterSpec iv = new IvParameterSpec(ivBytes);
byte[] encryptedBytes = Arrays.copyOfRange(combined, 8, combined.length);
MessageDigest md = MessageDigest.getInstance("SHA-1");
byte[] digest = md.digest(secretKey.getBytes(StandardCharsets.UTF_8));
byte[] keyBytes = Arrays.copyOf(digest, 24);
for (int i = 16; i < 24; i++) {
keyBytes[i] = keyBytes[i - 16];
}
SecretKey key = new SecretKeySpec(keyBytes, "DESede");
Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
cipher.init(Cipher.DECRYPT_MODE, key, iv);
byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
return new String(decryptedBytes, StandardCharsets.UTF_8);
}
}
```