**Product Cipher**

package product.cipher;

import java.util.\*;

public class ProductCipher

{

public static void main(String[] args)

{

int sub\_key, len, i, x=0, y=0, j=0, z=0;

String pt, cipher;

Scanner sc = new Scanner(System.in);

System.out.println("Enter Plain text message:");

pt = sc.nextLine();

pt = pt.toUpperCase();

len = pt.length();

char[] c1 = new char[len];

char[] ct = new char[len];

char[] rfc = new char[len];

c1 = pt.toCharArray();

System.out.println("Enter Key value for substitution cipher:");

sub\_key = sc.nextInt();

System.out.println("\n...........ENCRYPTION............ \nPlain Text:- " + pt);

for (i=0;i<len;i++)

{

x = (c1[i]+sub\_key);

if (x>90)

{

y = x-90;

ct[i] = (char)(65+(y-1));

}

else

{

ct[i] = (char)(x);

}

}

for (i=0;i<len;i=i+2)

{

rfc[j] = ct[i];

j++;

}

for (i=1;i<len;i=i+2)

{

rfc[j] = ct[i];

j++;

}

cipher = new String(rfc);

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

System.out.println("\n.........DECRYPTION........\n CIPHER TEXT:- " + cipher);

len = cipher.length();

if (len%2!=0)

{

z = (len/2);

z = z+1;

}

else

{

z = (len/2);

}

rfc = cipher.toCharArray();

j = z;

for (i=0,x=0;i<z && j<len && x<len;i++,j++)

{

ct[x] = rfc[i];

x++;

ct[x] = rfc[j];

x++;

}

for (i=0;i<len;i++)

{

x = (ct[i]-sub\_key);

if (x<65)

{

y = 65-x;

rfc[i] = (char)(90-(y-1));

}

else

{

rfc[i] = (char)(x);

}

}

cipher = new String(rfc);

System.out.println("PLAIN TEXT:- " + cipher);

}

}

**HASH FUNCTIONS**

import java.util.Scanner;

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class MD5

{

//hash function to get the md5 hash

public static String getMd5Hash(String input)

{

try

{

//static getInstance() method is called with hashing MD5

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

//calculating message digest of an input that return array of byte

byte[] messageDigest = md.digest(input.getBytes());

//converting byte array into signum representation

BigInteger no = new BigInteger(1, messageDigest);

//converting message digest into hex value

String hashtext = no.toString(16);

while (hashtext.length() < 32)

{

hashtext = "0" + hashtext;

}

return hashtext;

}

//for specifying wrong message digest algorithms

catch (NoSuchAlgorithmException e)

{

throw new RuntimeException(e);

}

}

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

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter a String : ");

String s = sc.next();

System.out.println("HashCode Generated for the string is: " + getMd5Hash(s));

}

}

**Vignere cipher**

// Java code to implement Vigenere Cipher

class GFG

{

// This function generates the key in

// a cyclic manner until it's length isi'nt

// equal to the length of original text

static String generateKey(String str, String key)

{

int x = str.length();

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

{

if (x == i)

i = 0;

if (key.length() == str.length())

break;

key+=(key.charAt(i));

}

return key;

}

// This function returns the encrypted text

// generated with the help of the key

static String cipherText(String str, String key)

{

String cipher\_text="";

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

{

// converting in range 0-25

int x = (str.charAt(i) + key.charAt(i)) %26;

// convert into alphabets(ASCII)

x += 'A';

cipher\_text+=(char)(x);

}

return cipher\_text;

}

// This function decrypts the encrypted text

// and returns the original text

static String originalText(String cipher\_text, String key)

{

String orig\_text="";

for (int i = 0 ; i < cipher\_text.length() &&

i < key.length(); i++)

{

// converting in range 0-25

int x = (cipher\_text.charAt(i) -

key.charAt(i) + 26) %26;

// convert into alphabets(ASCII)

x += 'A';

orig\_text+=(char)(x);

}

return orig\_text;

}

// This function will convert the lower case character to Upper case

static String LowerToUpper(String s)

{

StringBuffer str =new StringBuffer(s);

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

{

if(Character.isLowerCase(s.charAt(i)))

{

str.setCharAt(i, Character.toUpperCase(s.charAt(i)));

}

}

s = str.toString();

return s;

}

// Driver code

public static void main(String[] args)

{

String Str = "SECURITYLAB";

String Keyword = "CNS";

String str = LowerToUpper(Str);

String keyword = LowerToUpper(Keyword);

String key = generateKey(str, keyword);

String cipher\_text = cipherText(str, key);

System.out.println("Ciphertext : "

+ cipher\_text + "\n");

System.out.println("Original/Decrypted Text : "

+ originalText(cipher\_text, key));

}

}

**RSA**

// Java Program to Implement the RSA Algorithm

import java.math.\*;

import java.util.\*;

class RSA {

public static void main(String args[])

{

int p, q, n, z, d = 0, e, i;

// The number to be encrypted and decrypted

int msg = 12;

double c;

BigInteger msgback;

// 1st prime number p

p = 3;

// 2nd prime number q

q = 11;

n = p \* q;

z = (p - 1) \* (q - 1);

System.out.println("the value of z = " + z);

for (e = 2; e < z; e++) {

// e is for public key exponent

if (gcd(e, z) == 1) {

break;

}

}

System.out.println("the value of e = " + e);

for (i = 0; i <= 9; i++) {

int x = 1 + (i \* z);

// d is for private key exponent

if (x % e == 0) {

d = x / e;

break;

}

}

System.out.println("the value of d = " + d);

c = (Math.pow(msg, e)) % n;

System.out.println("Encrypted message is : " + c);

// converting int value of n to BigInteger

BigInteger N = BigInteger.valueOf(n);

// converting float value of c to BigInteger

BigInteger C = BigDecimal.valueOf(c).toBigInteger();

msgback = (C.pow(d)).mod(N);

System.out.println("Decrypted message is : "

+ msgback);

}

static int gcd(int e, int z)

{

if (e == 0)

return z;

else

return gcd(z % e, e);

}

}

**Playfair cipher**

**import java.awt.Point;**

**import java.util.Scanner;**

**public class PlayfairCipher**

**{**

**//length of digraph array**

**private int length = 0;**

**//creates a matrix for Playfair cipher**

**private String [][] table;**

**//main() method to test Playfair method**

**public static void main(String args[])**

**{**

**PlayfairCipher pf = new PlayfairCipher();**

**}**

**//main run of the program, Playfair method**

**//constructor of the class**

**private PlayfairCipher()**

**{**

**//prompts user for the keyword to use for encoding & creates tables**

**System.out.print("Enter the key for playfair cipher: ");**

**Scanner sc = new Scanner(System.in);**

**String key = parseString(sc);**

**while(key.equals(""))**

**key = parseString(sc);**

**table = this.cipherTable(key);**

**//prompts user for message to be encoded**

**System.out.print("Enter the plaintext to be encipher: ");**

**//System.out.println("using the previously given keyword");**

**String input = parseString(sc);**

**while(input.equals(""))**

**input = parseString(sc);**

**//encodes and then decodes the encoded message**

**String output = cipher(input);**

**String decodedOutput = decode(output);**

**//output the results to user**

**this.keyTable(table);**

**this.printResults(output,decodedOutput);**

**}**

**//parses an input string to remove numbers, punctuation,**

**//replaces any J's with I's and makes string all caps**

**private String parseString(Scanner sc)**

**{**

**String parse = sc.nextLine();**

**//converts all the letters in upper case**

**parse = parse.toUpperCase();**

**//the string to be substituted by space for each match (A to Z)**

**parse = parse.replaceAll("[^A-Z]", "");**

**//replace the letter J by I**

**parse = parse.replace("J", "I");**

**return parse;**

**}**

**//creates the cipher table based on some input string (already parsed)**

**private String[][] cipherTable(String key)**

**{**

**//creates a matrix of 5\*5**

**String[][] playfairTable = new String[5][5];**

**String keyString = key + "ABCDEFGHIKLMNOPQRSTUVWXYZ";**

**//fill string array with empty string**

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

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

**playfairTable[i][j] = "";**

**for(int k = 0; k < keyString.length(); k++)**

**{**

**boolean repeat = false;**

**boolean used = false;**

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

**{**

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

**{**

**if(playfairTable[i][j].equals("" + keyString.charAt(k)))**

**{**

**repeat = true;**

**}**

**else if(playfairTable[i][j].equals("") && !repeat && !used)**

**{**

**playfairTable[i][j] = "" + keyString.charAt(k);**

**used = true;**

**}**

**}**

**}**

**}**

**return playfairTable;**

**}**

**//cipher: takes input (all upper-case), encodes it, and returns the output**

**private String cipher(String in)**

**{**

**length = (int) in.length() / 2 + in.length() % 2;**

**//insert x between double-letter digraphs & redefines "length"**

**for(int i = 0; i < (length - 1); i++)**

**{**

**if(in.charAt(2 \* i) == in.charAt(2 \* i + 1))**

**{**

**in = new StringBuffer(in).insert(2 \* i + 1, 'X').toString();**

**length = (int) in.length() / 2 + in.length() % 2;**

**}**

**}**

**//------------makes plaintext of even length--------------**

**//creates an array of digraphs**

**String[] digraph = new String[length];**

**//loop iterates over the plaintext**

**for(int j = 0; j < length ; j++)**

**{**

**//checks the plaintext is of even length or not**

**if(j == (length - 1) && in.length() / 2 == (length - 1))**

**//if not addends X at the end of the plaintext**

**in = in + "X";**

**digraph[j] = in.charAt(2 \* j) +""+ in.charAt(2 \* j + 1);**

**}**

**//encodes the digraphs and returns the output**

**String out = "";**

**String[] encDigraphs = new String[length];**

**encDigraphs = encodeDigraph(digraph);**

**for(int k = 0; k < length; k++)**

**out = out + encDigraphs[k];**

**return out;**

**}**

**//---------------encryption logic-----------------**

**//encodes the digraph input with the cipher's specifications**

**private String[] encodeDigraph(String di[])**

**{**

**String[] encipher = new String[length];**

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

**{**

**char a = di[i].charAt(0);**

**char b = di[i].charAt(1);**

**int r1 = (int) getPoint(a).getX();**

**int r2 = (int) getPoint(b).getX();**

**int c1 = (int) getPoint(a).getY();**

**int c2 = (int) getPoint(b).getY();**

**//executes if the letters of digraph appear in the same row**

**//in such case shift columns to right**

**if(r1 == r2)**

**{**

**c1 = (c1 + 1) % 5;**

**c2 = (c2 + 1) % 5;**

**}**

**//executes if the letters of digraph appear in the same column**

**//in such case shift rows down**

**else if(c1 == c2)**

**{**

**r1 = (r1 + 1) % 5;**

**r2 = (r2 + 1) % 5;**

**}**

**//executes if the letters of digraph appear in the different row and different column**

**//in such case swap the first column with the second column**

**else**

**{**

**int temp = c1;**

**c1 = c2;**

**c2 = temp;**

**}**

**//performs the table look-up and puts those values into the encoded array**

**encipher[i] = table[r1][c1] + "" + table[r2][c2];**

**}**

**return encipher;**

**}**

**//-----------------------decryption logic---------------------**

**// decodes the output given from the cipher and decode methods (opp. of encoding process)**

**private String decode(String out)**

**{**

**String decoded = "";**

**for(int i = 0; i < out.length() / 2; i++)**

**{**

**char a = out.charAt(2\*i);**

**char b = out.charAt(2\*i+1);**

**int r1 = (int) getPoint(a).getX();**

**int r2 = (int) getPoint(b).getX();**

**int c1 = (int) getPoint(a).getY();**

**int c2 = (int) getPoint(b).getY();**

**if(r1 == r2)**

**{**

**c1 = (c1 + 4) % 5;**

**c2 = (c2 + 4) % 5;**

**}**

**else if(c1 == c2)**

**{**

**r1 = (r1 + 4) % 5;**

**r2 = (r2 + 4) % 5;**

**}**

**else**

**{**

**//swapping logic**

**int temp = c1;**

**c1 = c2;**

**c2 = temp;**

**}**

**decoded = decoded + table[r1][c1] + table[r2][c2];**

**}**

**//returns the decoded message**

**return decoded;**

**}**

**// returns a point containing the row and column of the letter**

**private Point getPoint(char c)**

**{**

**Point pt = new Point(0,0);**

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

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

**if(c == table[i][j].charAt(0))**

**pt = new Point(i,j);**

**return pt;**

**}**

**//function prints the key-table in matrix form for playfair cipher**

**private void keyTable(String[][] printTable)**

**{**

**System.out.println("Playfair Cipher Key Matrix: ");**

**System.out.println();**

**//loop iterates for rows**

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

**{**

**//loop iterates for column**

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

**{**

**//prints the key-table in matrix form**

**System.out.print(printTable[i][j]+" ");**

**}**

**System.out.println();**

**}**

**System.out.println();**

**}**

**//method that prints all the results**

**private void printResults(String encipher, String dec)**

**{**

**System.out.print("Encrypted Message: ");**

**//prints the encrypted message**

**System.out.println(encipher);**

**System.out.println();**

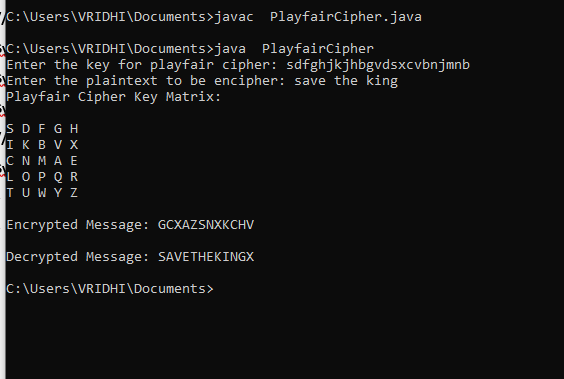
**System.out.print("Decrypted Message: ");**

**//prints the decryted message**

**System.out.println(dec);**

**}**

**}**



**AES**

import javax.crypto.Cipher;

import javax.crypto.SecretKey;

import javax.crypto.SecretKeyFactory;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.PBEKeySpec;

import javax.crypto.spec.SecretKeySpec;

import java.nio.charset.StandardCharsets;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.spec.InvalidKeySpecException;

import java.security.spec.KeySpec;

import java.util.Base64;

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

public class AESExample

{

/\* Private variable declaration \*/

private static final String SECRET\_KEY = "123456789";

private static final String SALTVALUE = "abcdefg";

/\* Encryption Method \*/

public static String encrypt(String strToEncrypt)

{

try

{

/\* Declare a byte array. \*/

byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

IvParameterSpec ivspec = new IvParameterSpec(iv);

/\* Create factory for secret keys. \*/

SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

/\* PBEKeySpec class implements KeySpec interface. \*/

KeySpec spec = new PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);

SecretKey tmp = factory.generateSecret(spec);

SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

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

cipher.init(Cipher.ENCRYPT\_MODE, secretKey, ivspec);

/\* Retruns encrypted value. \*/

return Base64.getEncoder()

.encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF\_8)));

}

catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)

{

System.out.println("Error occured during encryption: " + e.toString());

}

return null;

}

/\* Decryption Method \*/

public static String decrypt(String strToDecrypt)

{

try

{

/\* Declare a byte array. \*/

byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

IvParameterSpec ivspec = new IvParameterSpec(iv);

/\* Create factory for secret keys. \*/

SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

/\* PBEKeySpec class implements KeySpec interface. \*/

KeySpec spec = new PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);

SecretKey tmp = factory.generateSecret(spec);

SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");

cipher.init(Cipher.DECRYPT\_MODE, secretKey, ivspec);

/\* Retruns decrypted value. \*/

return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));

}

catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)

{

System.out.println("Error occured during decryption: " + e.toString());

}

return null;

}

/\* Driver Code \*/

public static void main(String[] args)

{

/\* Message to be encrypted. \*/

String originalval = "SECURITY LAB";

/\* Call the encrypt() method and store result of encryption. \*/

String encryptedval = encrypt(originalval);

/\* Call the decrypt() method and store result of decryption. \*/

String decryptedval = decrypt(encryptedval);

/\* Display the original message, encrypted message and decrypted message on the console. \*/

System.out.println("Original value: " + originalval);

System.out.println("Encrypted value: " + encryptedval);

System.out.println("Decrypted value: " + decryptedval);

}

}

**DES**

**// Java code for the above approach**

import java.util.\*;

class Main {

private static class DES {

// CONSTANTS

// Initial Permutation Table

int[] IP

= { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44,

36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22,

14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57,

49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35,

27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13,

5, 63, 55, 47, 39, 31, 23, 15, 7 };

// Inverse Initial Permutation Table

int[] IP1

= { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,

15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22,

62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36,

4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11,

51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58,

26, 33, 1, 41, 9, 49, 17, 57, 25 };

// first key-hePermutation Table

int[] PC1

= { 57, 49, 41, 33, 25, 17, 9, 1, 58, 50,

42, 34, 26, 18, 10, 2, 59, 51, 43, 35,

27, 19, 11, 3, 60, 52, 44, 36, 63, 55,

47, 39, 31, 23, 15, 7, 62, 54, 46, 38,

30, 22, 14, 6, 61, 53, 45, 37, 29, 21,

13, 5, 28, 20, 12, 4 };

// second key-Permutation Table

int[] PC2

= { 14, 17, 11, 24, 1, 5, 3, 28, 15, 6,

21, 10, 23, 19, 12, 4, 26, 8, 16, 7,

27, 20, 13, 2, 41, 52, 31, 37, 47, 55,

30, 40, 51, 45, 33, 48, 44, 49, 39, 56,

34, 53, 46, 42, 50, 36, 29, 32 };

// Expansion D-box Table

int[] EP = { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7,

8, 9, 8, 9, 10, 11, 12, 13, 12, 13,

14, 15, 16, 17, 16, 17, 18, 19, 20, 21,

20, 21, 22, 23, 24, 25, 24, 25, 26, 27,

28, 29, 28, 29, 30, 31, 32, 1 };

// Straight Permutation Table

int[] P

= { 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23,

26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27,

3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

// S-box Table

int[][][] sbox

= { { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6,

12, 5, 9, 0, 7 },

{ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12,

11, 9, 5, 3, 8 },

{ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7,

3, 10, 5, 0 },

{ 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14,

10, 0, 6, 13 } },

{ { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13,

12, 0, 5, 10 },

{ 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10,

6, 9, 11, 5 },

{ 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6,

9, 3, 2, 15 },

{ 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12,

0, 5, 14, 9 } },

{ { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7,

11, 4, 2, 8 },

{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14,

12, 11, 15, 1 },

{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12,

5, 10, 14, 7 },

{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3,

11, 5, 2, 12 } },

{ { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5,

11, 12, 4, 15 },

{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12,

1, 10, 14, 9 },

{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3,

14, 5, 2, 8, 4 },

{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11,

12, 7, 2, 14 } },

{ { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15,

13, 0, 14, 9 },

{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15,

10, 3, 9, 8, 6 },

{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5,

6, 3, 0, 14 },

{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9,

10, 4, 5, 3 } },

{ { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4,

14, 7, 5, 11 },

{ 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14,

0, 11, 3, 8 },

{ 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10,

1, 13, 11, 6 },

{ 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7,

6, 0, 8, 13 } },

{ { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7,

5, 10, 6, 1 },

{ 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12,

2, 15, 8, 6 },

{ 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6,

8, 0, 5, 9, 2 },

{ 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15,

14, 2, 3, 12 } },

{ { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14,

5, 0, 12, 7 },

{ 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11,

0, 14, 9, 2 },

{ 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13,

15, 3, 5, 8 },

{ 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0,

3, 5, 6, 11 } } };

int[] shiftBits = { 1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1 };

// hexadecimal to binary conversion

String hextoBin(String input)

{

int n = input.length() \* 4;

input = Long.toBinaryString(

Long.parseUnsignedLong(input, 16));

while (input.length() < n)

input = "0" + input;

return input;

}

// binary to hexadecimal conversion

String binToHex(String input)

{

int n = (int)input.length() / 4;

input = Long.toHexString(

Long.parseUnsignedLong(input, 2));

while (input.length() < n)

input = "0" + input;

return input;

}

// per-mutate input hexadecimal

// according to specified sequence

String permutation(int[] sequence, String input)

{

String output = "";

input = hextoBin(input);

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

output += input.charAt(sequence[i] - 1);

output = binToHex(output);

return output;

}

// xor 2 hexadecimal strings

String xor(String a, String b)

{

// hexadecimal to decimal(base 10)

long t\_a = Long.parseUnsignedLong(a, 16);

// hexadecimal to decimal(base 10)

long t\_b = Long.parseUnsignedLong(b, 16);

// xor

t\_a = t\_a ^ t\_b;

// decimal to hexadecimal

a = Long.toHexString(t\_a);

// prepend 0's to maintain length

while (a.length() < b.length())

a = "0" + a;

return a;

}

// left Circular Shifting bits

String leftCircularShift(String input, int numBits)

{

int n = input.length() \* 4;

int perm[] = new int[n];

for (int i = 0; i < n - 1; i++)

perm[i] = (i + 2);

perm[n - 1] = 1;

while (numBits-- > 0)

input = permutation(perm, input);

return input;

}

// preparing 16 keys for 16 rounds

String[] getKeys(String key)

{

String keys[] = new String[16];

// first key permutation

key = permutation(PC1, key);

for (int i = 0; i < 16; i++) {

key = leftCircularShift(key.substring(0, 7),

shiftBits[i])

+ leftCircularShift(

key.substring(7, 14),

shiftBits[i]);

// second key permutation

keys[i] = permutation(PC2, key);

}

return keys;

}

// s-box lookup

String sBox(String input)

{

String output = "";

input = hextoBin(input);

for (int i = 0; i < 48; i += 6) {

String temp = input.substring(i, i + 6);

int num = i / 6;

int row = Integer.parseInt(

temp.charAt(0) + "" + temp.charAt(5),

2);

int col = Integer.parseInt(

temp.substring(1, 5), 2);

output += Integer.toHexString(

sbox[num][row][col]);

}

return output;

}

String round(String input, String key, int num)

{

// fk

String left = input.substring(0, 8);

String temp = input.substring(8, 16);

String right = temp;

// Expansion permutation

temp = permutation(EP, temp);

// xor temp and round key

temp = xor(temp, key);

// lookup in s-box table

temp = sBox(temp);

// Straight D-box

temp = permutation(P, temp);

// xor

left = xor(left, temp);

System.out.println("Round " + (num + 1) + " "

+ right.toUpperCase() + " "

+ left.toUpperCase() + " "

+ key.toUpperCase());

// swapper

return right + left;

}

String encrypt(String plainText, String key)

{

int i;

// get round keys

String keys[] = getKeys(key);

// initial permutation

plainText = permutation(IP, plainText);

System.out.println("After initial permutation: "

+ plainText.toUpperCase());

System.out.println(

"After splitting: L0="

+ plainText.substring(0, 8).toUpperCase()

+ " R0="

+ plainText.substring(8, 16).toUpperCase()

+ "\n");

// 16 rounds

for (i = 0; i < 16; i++) {

plainText = round(plainText, keys[i], i);

}

// 32-bit swap

plainText = plainText.substring(8, 16)

+ plainText.substring(0, 8);

// final permutation

plainText = permutation(IP1, plainText);

return plainText;

}

String decrypt(String plainText, String key)

{

int i;

// get round keys

String keys[] = getKeys(key);

// initial permutation

plainText = permutation(IP, plainText);

System.out.println("After initial permutation: "

+ plainText.toUpperCase());

System.out.println(

"After splitting: L0="

+ plainText.substring(0, 8).toUpperCase()

+ " R0="

+ plainText.substring(8, 16).toUpperCase()

+ "\n");

// 16-rounds

for (i = 15; i > -1; i--) {

plainText

= round(plainText, keys[i], 15 - i);

}

// 32-bit swap

plainText = plainText.substring(8, 16)

+ plainText.substring(0, 8);

plainText = permutation(IP1, plainText);

return plainText;

}

}

// Driver code

public static void main(String args[])

{

String text = "123456ABCD132536";

String key = "AABB09182736CCDD";

DES cipher = new DES();

System.out.println("Encryption:\n");

text = cipher.encrypt(text, key);

System.out.println(

"\nCipher Text: " + text.toUpperCase() + "\n");

System.out.println("Decryption\n");

text = cipher.decrypt(text, key);

System.out.println("\nPlain Text: "

+ text.toUpperCase());

}

}