**SRI SIVASUBRAMANIYA NADAR**

**COLLEGE OF ENGINEERING**

(AFFILIATED TO ANNA UNIVERSITY, CHENNAI)

RAJIV GANDHI SALAI (OMR), KALAVAKKAM – 603110

**LABORATORY RECORD**

**IT8761 – SECURITY LABORATORY**

Name : ..........P.G.ABINAYA...............................................

Reg. No. : ............312217205003…............................................

Dept. : .........IT........... Sem. : ......VII......... Sec. : ..A.......

****

**ANNA UNIVERSITY**

**BONAFIDE CERTIFICATE**

Certified that this is the bonafide record of

the practical work done for the

# ................................. IT8761 Security Laboratory………….............. lab

by

Name ......................P.G.ABINAYA.........................

Register Number .....................312217205003...........................

Of Department of ..................INFORMATION TECHNOLOGY..........................

Sri Sivasubramaniya Nadar College of Engineering,

Kalavakkam, Chennai

during the academic year ...........2020-2021................

Staff In-Charge Head

Department of ...............IT..............

*Submitted for the University Examination held at*

*Sri Sivasubramaniya Nadar College of Engineering on ....................................*

Internal Examiner External Examiner

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **INDEX**  Name: \_\_\_\_\_\_P.G.ABINAYA\_\_\_\_\_\_\_ Reg. No. : \_\_\_\_\_312217205003\_\_\_\_\_\_\_\_  Sem. : \_\_\_\_\_\_\_\_\_\_VII\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Sec. : \_\_\_\_\_\_\_\_\_\_\_A\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |
| Ex.No. | Date of Expt. | Title of the Experiment | Page No. | Signature of the Faculty | Remarks |
| 1 | 11-08-2020 | **SubstitutionTechniques**  - Ceaser Cipher  - Playfair Cipher  - Vigenere Cipher  - Hill Cipher | 4 |  |  |
| 2 | 25-08-2020 | **Transposition Techniques**  -Rail fence  -Row and Column  -One Time Pad | 36 |  |  |
| 3 | 08-09-2020 | DES Algorithm | 48 |  |  |
| 4 | 15-09-2020 | AES Algorithm | 62 |  |  |
| 5 | 22-09-2020 | RSA Algorithm | 99 |  |  |
| 6 | 29-09-2020 | **Key Exchange Algorithms**  -Diffie Hellman  - ElGamal Key Exchange | 106 |  |  |
| 7 | 06-10-2020 | SHA-1 Algorithm | 113 |  |  |
| 8 | 13-10-2020 | **Digital Signature Algorithms**  -ElGamal Digital Signature  -Digital Signature Algorithm | 120 |  |  |
| 9 | 20-10-2020 | Intrusion Detection System using Snort | 126 |  |  |
| 10 | 27-10-2020 | Web application security using N-Stalker | 133 |  |  |
| 11 | 03-11-2020 | Defeating malware  Building Rootkit hunter | 139 |  |  |

Ex: 1 [Classical Encryption Substitution Techniques](https://lms.ssn.edu.in/mod/assign/view.php?id=83301)

Date : 11.08.2020

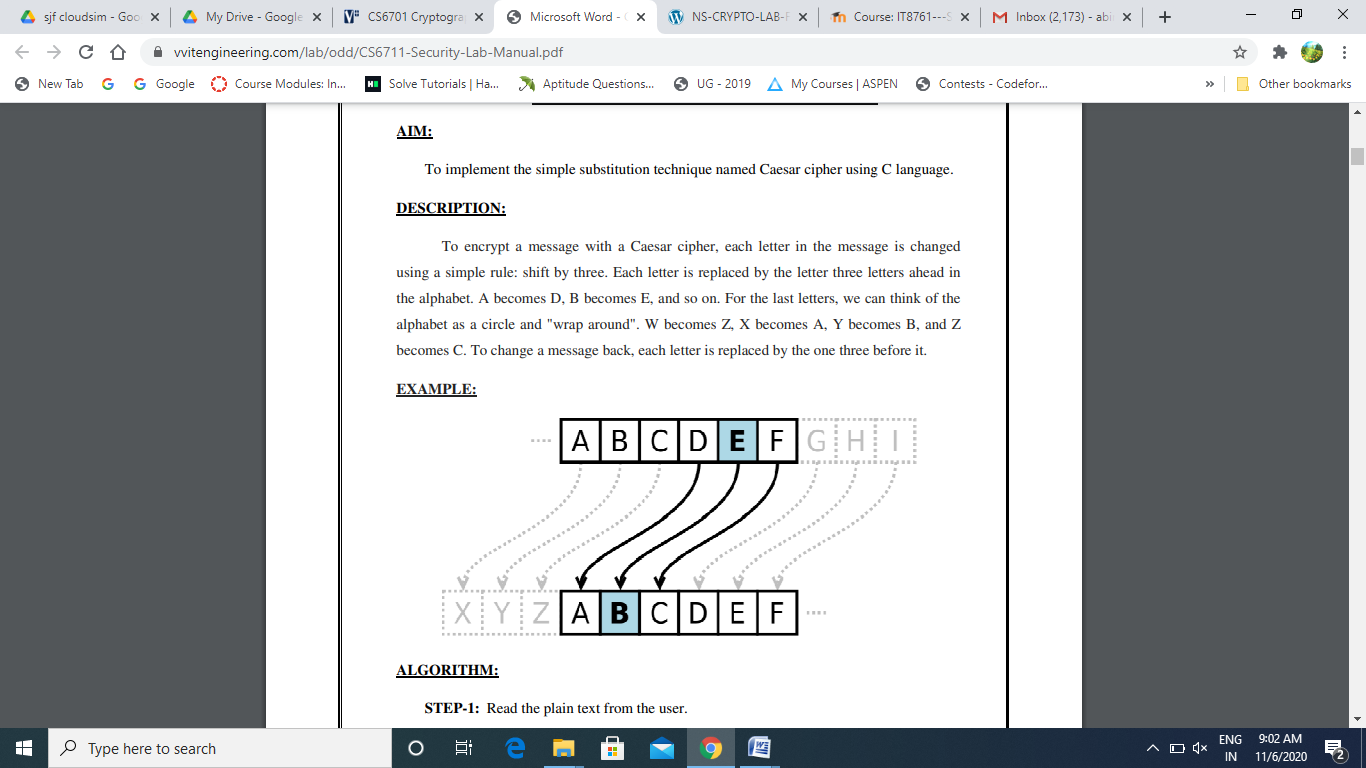
**Aim:**

To implement Classical Encryption Substitution techniques (Caesar Cipher, Vigenere Cipher, PlayFair Cipher and Hill Cipher) using java.

1. **Caesar Cipher:**

**Description:**

To encrypt a message with a Caesar cipher, each letter in the message is changed using a simple rule: shift by three. Each letter is replaced by the letter three letters ahead in the alphabet. A becomes D, B becomes E, and so on. For the last letters, we can think of alphabet as a circle and "wrap around". W becomes Z, X becomes A, Y becomes B, and Z becomes C. To change a message back, each letter is replaced by the one three before it.



**Algorithm:**

* In Ceaser Cipher each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.
* For example, with a **left shift of 3**, **D** would be replaced by **A**, **E** would become **B**, and so on.
* The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, **A = 0, B = 1, Z = 25.**
* Encryption of a letter x by a shift n can be described mathematically as,

**En(x) = (x + n) mod (total no. of letters)**

* Decryption is performed similarly,

**Dn (x)=(x – n+ total no. of letters) mod26(total no. of letters)**

**Program:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

public class CeaserCipher {

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

{

int choice;

do

{

System.out.println("Enter the choice(1)Encryption by Shifting\n(2)Decryption by Shifting\n(3)Encryption by Modulos\n(4)Decryption by Modulos\n(5)Exit ");

Scanner sc=new Scanner(System.in);

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

choice=sc.nextInt();

switch(choice)

{

case 1:

{

System.out.print("Enter Plain Text: ");

String str = br.readLine();

System.out.print("\nEnter the Key: ");

int key = sc.nextInt();

String encrypted = encrypt(str, key);

System.out.println("\nEncrypted String by Shifting is: " +encrypted);

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

break;

}

case 2:

{

System.out.print("Enter Cipher Text: ");

String str = br.readLine();

System.out.print("\nEnter the Key: ");

int key = sc.nextInt();

String decrypted = decrypt(str, key);

System.out.println("\nDecrypted String by Shifting is: "+decrypted);

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

break;

}

case 3:

{

System.out.print("Enter Plain Text: ");

String str = br.readLine();

System.out.print("\nEnter the Key: ");

int key = sc.nextInt();

String encrye=encry(str,key);

System.out.println("\nEncrypted String is: "+encrye);

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

break;

}

case 4:

{

System.out.print("Enter Cipher Text: ");

String str = br.readLine();

System.out.print("\nEnter the Key: ");

int key = sc.nextInt();

String decrye=decry(str,key);

System.out.println("\nDecrypted String is: "+decrye);

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

break;

}

}

}while(choice!=5);

}

public static String decry(String str, int key)

{

String decrypted = "";

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

{

int c = str.charAt(i);

if (Character.isUpperCase(c))

{

c = (((c-65) - key) % 26)+65;

if (c < 'A')

c = c + 26;

}

else if (Character.isLowerCase(c))

{

c = (((c-97) - key) % 26)+97;

if (c < 'a')

c = c + 26;

}

decrypted += (char) c;

}

return decrypted;

}

public static String encry(String str,int key)

{

String encrypted = "";

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

{

int c = str.charAt(i);

//System.out.println((((c-97)+key)%26)+97);

if (Character.isUpperCase(c))

{

c = (((c-65) + key) % 26)+65;

if (c > 'Z')

c = c - 26;

}

else if (Character.isLowerCase(c))

{

c = (((c-97) + key) % 26)+97;

if (c > 'z')

c = c - 26;

}

encrypted += (char) c;

}

return encrypted;

}

public static String encrypt(String str, int key)

{

String encrypted = "";

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

{

int c = str.charAt(i);

if (Character.isUpperCase(c))

{

c = c + (key % 26);

if (c > 'Z')

c = c - 26;

}

else if (Character.isLowerCase(c))

{

c = c + (key % 26);

if (c > 'z')

c = c - 26;

}

encrypted += (char) c;

}

return encrypted;

}

public static String decrypt(String str, int key)

{

String decrypted = "";

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

{

int c = str.charAt(i);

if (Character.isUpperCase(c))

{

c = c - (key % 26);

if (c < 'A')

c = c + 26;}

else if (Character.isLowerCase(c)) {

c = c - (key % 26);

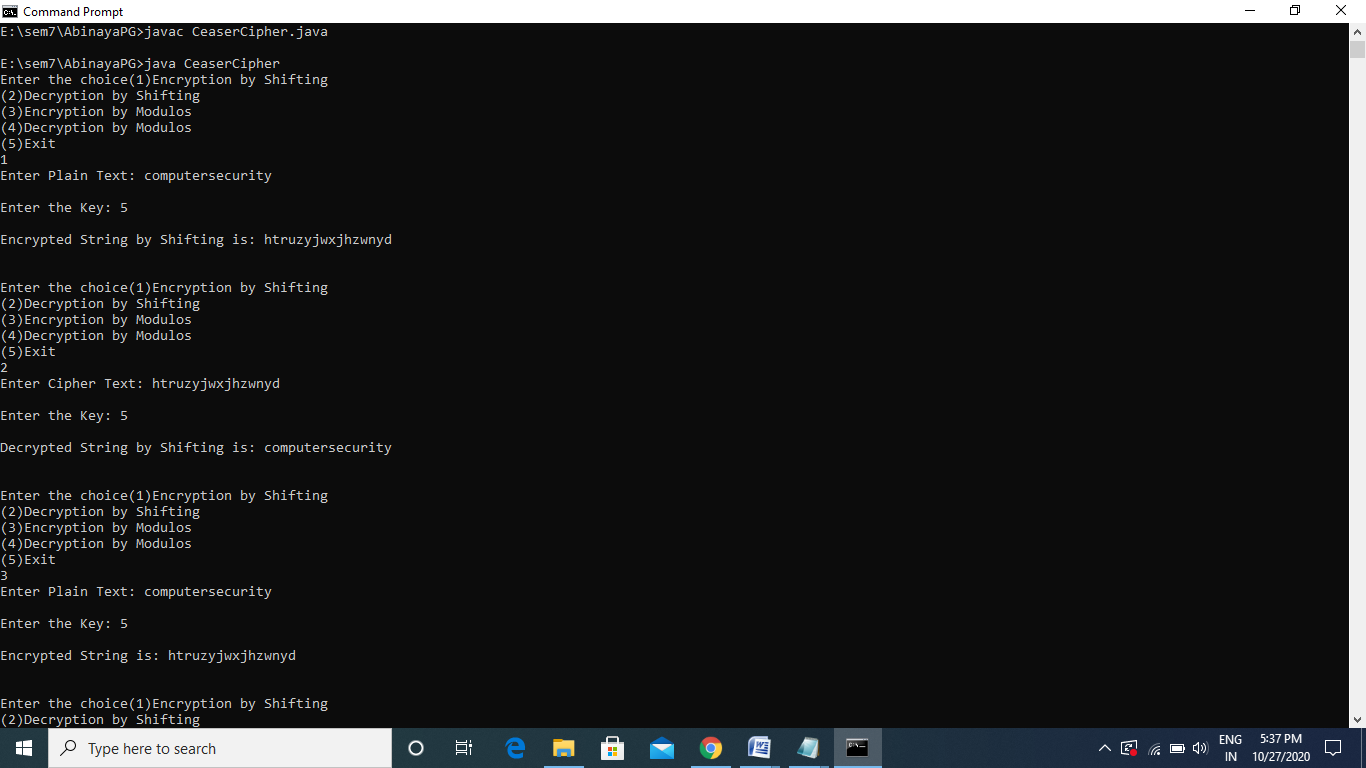
if (c < 'a')

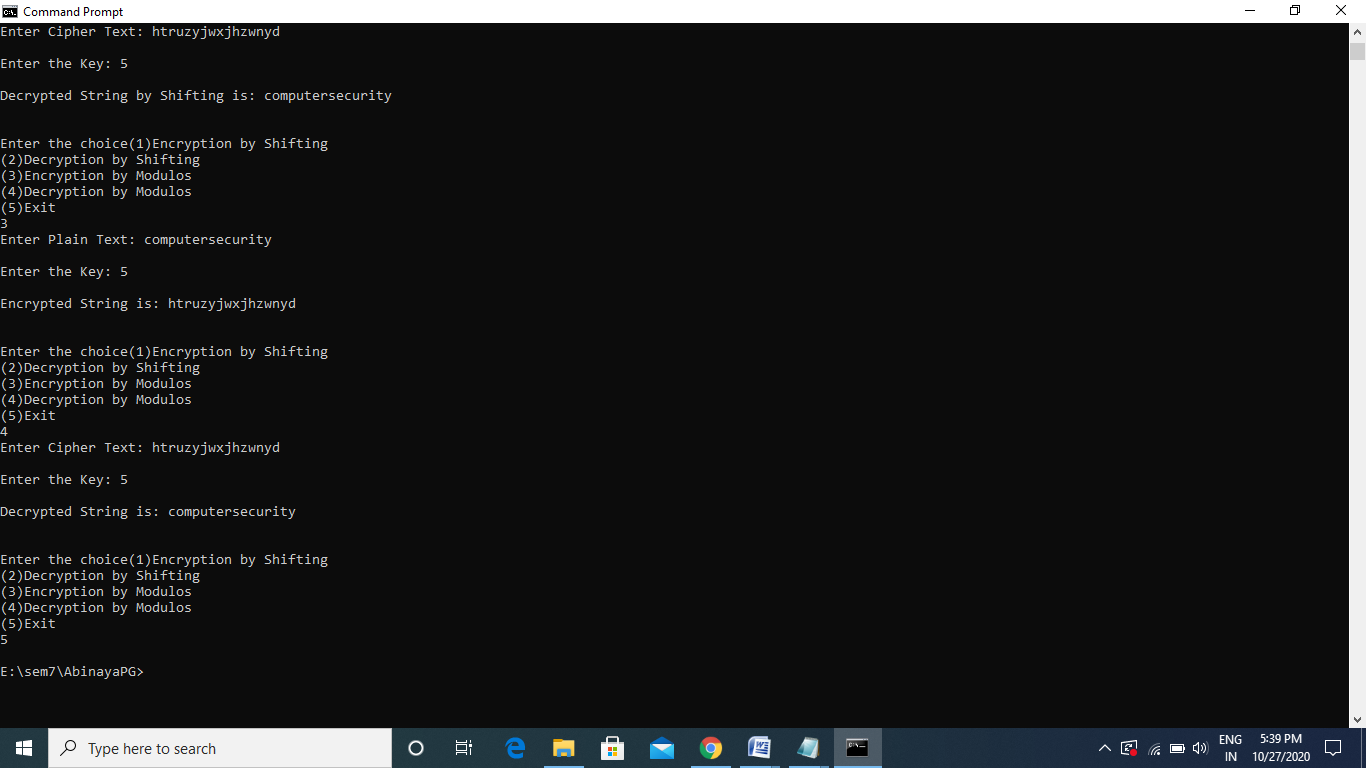
c = c + 26;}

decrypted += (char) c;}

return decrypted;}}

**Output:**





1. **Vigenere Cipher:**

**Description:**

To encrypt, a table of alphabets can be used, termed a Vigenère table. It consists of the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the prev 26 possible Caesar ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows. The alphabet used at each point depends on a repeating keyword. Each row starts with a key letter. The remainder of the row holds the letters A to Z. Although there are 26 key rows shown, you will only use as many keys as there are unique letters in the key string, here just 5 keys, {L, E, M, O, N}. For successive letters of the message, we are going to take successive letters of the key string, and encipher each message letter using its corresponding key row. Choose the next letter of the key, go along that row to find the column heading that matches the message character; the letter at the [key-row, msg-col] is the enciphered letter.



**Algorithm:**

* The Vigenere cipher is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword.
* It is a simple form of *polyalphabetic* substitution.
* Let P=p1p1.. be the plain text ,C=c1c2.. be the cipher textand K=(k1k2…kn,k1k2…kn,..) be the key stream and the key stream is the repetition of initial key stream(as many times as needed).
* The encryption can be performed as,

**Ci=(Pi+Ki) mod 26**

* Do the above steps for all the characters in the plain text.
* The decryption can be performed as,

**Pi=(Ci-Ki+26) mod 26**

**Program:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

public class VigenereCipher {

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

{

Scanner sc=new Scanner(System.in);

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int choice;

do{

System.out.println("Enter the choice(1)Encrypt\n(2)Decrypt\n3)Exit");

choice=sc.nextInt();

switch(choice)

{

case 1:

{

encrypt();

break;

}

case 2:

{

decrypt();

break;

}

}

}while(choice!=3);

}

public static void encrypt()throws IOException

{

String abc="";

int[][] vig=new int[26][26];

for(int i=0;i<26;i++)

{

for(int j=0;j<26;j++)

{

vig[i][j]=(i+j)%26;

}

}

Scanner sc=new Scanner(System.in);

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter any Plain Text: ");

String str = br.readLine();

System.out.print("\nEnter the Key: ");

String key = br.readLine();

System.out.println("Resultant Cipher Text:");

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

{

if(j>=key.length())

{

j=0;

}

int c=str.charAt(i);

int a=key.charAt(j);

if(c>=97 && a>=97)

abc+=(char)(vig[c-97][a-97]+97);

else if(c>=97)

abc+=(char)(vig[c-97][a-65]+97);

else if(a>=97){

abc+=(char)(vig[c-65][a-97]+65);

}

else

{

abc+=(char)(vig[c-65][a-65]+65);

}

//System.out.println(c+" "+a);

}

System.out.println(abc);

}

public static void decrypt() throws IOException

{

String abc="";

int[][] vig=new int[26][26];

for(int i=0;i<26;i++)

{

for(int j=0;j<26;j++)

{

vig[i][j]=(i+j)%26;

}

}

Scanner sc=new Scanner(System.in);

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter any Cipher Text: ");

String str = br.readLine();

System.out.print("\nEnter the Key: ");

String key = br.readLine();

System.out.println("Resultant Plain Text:");

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

{

if(j>=key.length()){

j=0;}

int c=str.charAt(i);

int a=key.charAt(j);

int k=0;

while(vig[k][a-97]!=c-97){

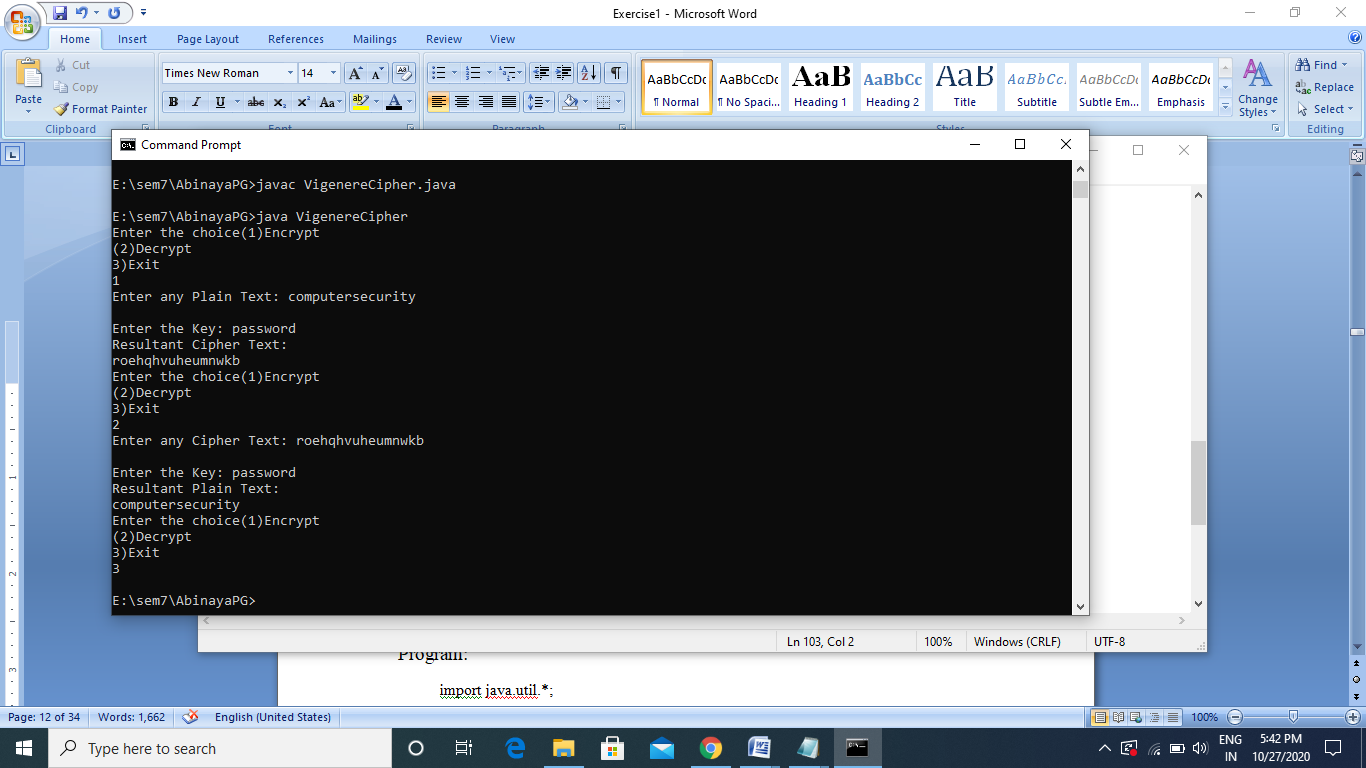
k++;}

abc+=(char)(k+97);}

System.out.println(abc);

}}

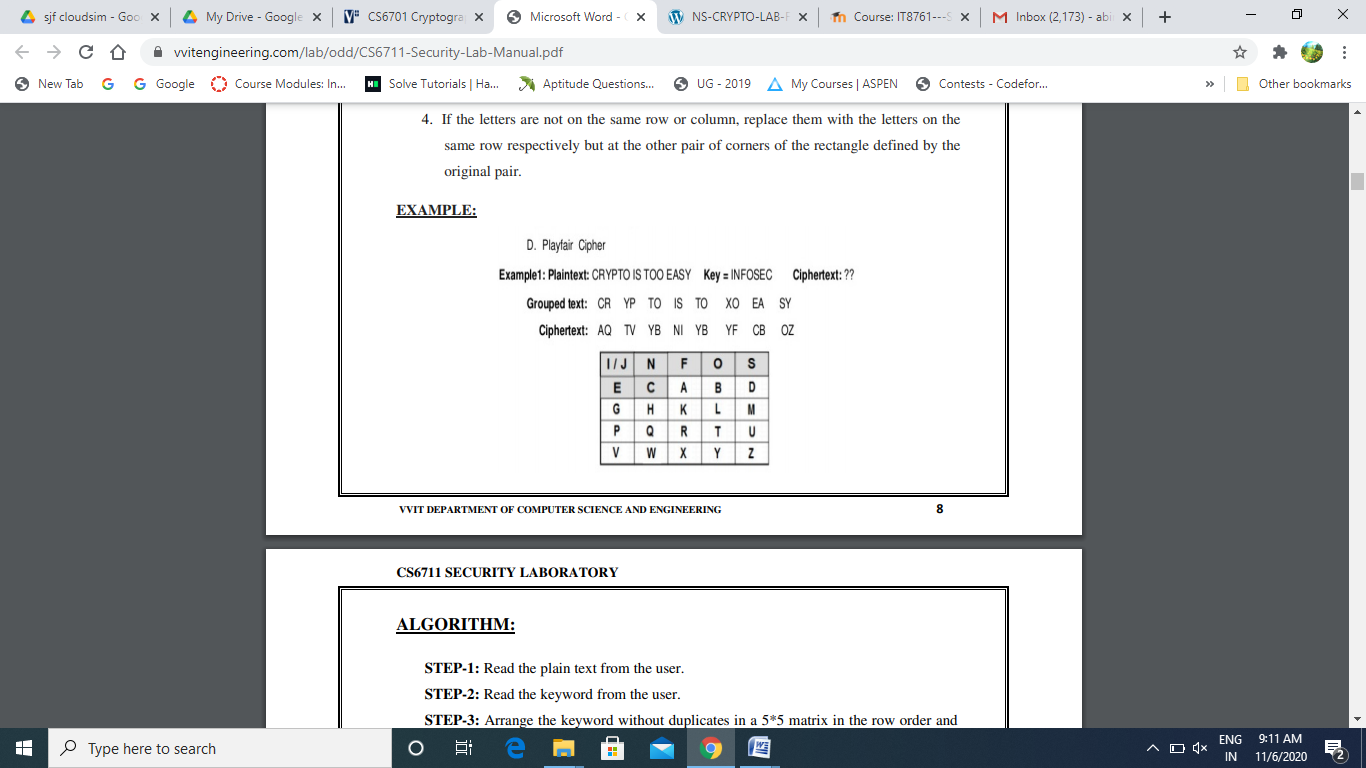
**Output:**



1. **Play Fair Cipher:**

**Description:**

The Playfair cipher starts with creating a key table. The key table is a 5×5 grid of letters that will act as the key for encrypting your plaintext. Each of the 25 letters must be unique and one letter of the alphabet is omitted from the table (as there are 25 spots and 26 letters in the alphabet). To encrypt a message, one would break the message into digrams (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD", and map them out on the key table. The two letters of the diagram are considered as the opposite corners of a rectangle in the key table. Note the relative position of the corners of this rectangle. Then apply the following 4 rules, in order, to each pair of letters in the plaintext: 1. If both letters are the same (or only one letter is left), add an "X" after the first letter 2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively 3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively 4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair.



**Algorithm:**

* To encrypt a message, one would break the message into digrams (groups of 2 letters)

For example, "HelloWorld" becomes "HE LL OW OR LD".

* These digrams will be substituted using the key table.
* Since encryption requires pairs of letters, messages with an odd number of characters usually append an uncommon letter, such as "X", to complete the final digram.
* The two letters of the digram are considered opposite corners of a rectangle in the key table. To perform the substitution, apply the following 4 rules, in order, to each pair of letters in the plaintext:
* If a pair is a repeated letter, insert filler like 'X’.
* If both letters fall in the same row, replace each with the letter to its right (circularly).
* If both letters fall in the same column, replace each with the the letter below it (circularly).
* Otherwise, each letter is replaced by the letter in the same row but in the column of the other letter of the pair.

**Program:**

import java.util.\*;

class PlayFairCipher{

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

{

String key,cipherText,plainText;

int choice;

do{

Scanner scn=new Scanner(System.in);

System.out.println("Enter choice 1)encrypt 2)Decrypt 3)exit");

choice=scn.nextInt();

switch(choice)

{

case 1:

{

Scanner sc=new Scanner(System.in);

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

plainText=sc.nextLine();

System.out.println("Enter Key:");

key=sc.nextLine();

cipherText=Encrypt(plainText,key);

System.out.println("Encrypted text:");

System.out.println("\n"+cipherText+"\n");

break;

}

case 2:

{

Scanner scan=new Scanner(System.in);

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

plainText=scan.nextLine();

System.out.println("Enter Key:");

key=scan.nextLine();

String encryptedText=Decrypt(plainText, key);

System.out.println("Decrypted text:" );

System.out.println("\n"+encryptedText+"\n");

break;

}

}

}while(choice!=3);

}

static char[][] insertKey(String key)

{

char keyMatrix[][]=new char[5][5];

key=key.toUpperCase();

key=key.replaceAll("J", "I");

key=key.replaceAll(" ", "");

int a=0,b=0;

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

{

if(!repeat(key.charAt(k),keyMatrix))

{

keyMatrix[a][b++]=key.charAt(k);

if(b>4)

{

b=0;

a++;

}

}

}

char p='A';

while(a < 5)

{

while(b < 5)

{

if(!repeat(p,keyMatrix))

{

keyMatrix[a][b++]=p;

}

p++;

}

b=0;

a++;

}

System.out.print("-------------------------Key Matrix-------------------");

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

{

System.out.println();

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

{

System.out.print("\t"+keyMatrix[i][j]);

}

}

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

return keyMatrix;

}

static boolean repeat(char c,char keyMatrix[][])

{

if(!indexOfChar(c))

{

return true;

}

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

{

for(int j=0;j < keyMatrix[i].length;j++)

{

if(keyMatrix[i][j]==c || c=='J')

return true;

}

}

return false;

}

static int rowPos(char c,char keyMatrix[][])

{

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

{

for(int j=0;j < keyMatrix[i].length;j++)

{

if(keyMatrix[i][j]==c)

return i;

}

}

return -1;

}

static int columnPos(char c,char keyMatrix[][])

{

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

{

for(int j=0;j < keyMatrix[i].length;j++)

{

if(keyMatrix[i][j]==c)

return j;

}

}

return -1;

}

static boolean indexOfChar(char c)

{

String allChar="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

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

{

if(allChar.charAt(i)==c)

return true;

}

return false;

}

static String encryptChar(String plain,char keyMatrix[][])

{

plain=plain.toUpperCase();

char a=plain.charAt(0),b=plain.charAt(1);

String cipherChar="";

int r1,c1,r2,c2;

r1=rowPos(a,keyMatrix);

c1=columnPos(a,keyMatrix);

r2=rowPos(b,keyMatrix);

c2=columnPos(b,keyMatrix);

if(c1==c2)

{

++r1;

++r2;

if(r1>4)

r1=0;

if(r2>4)

r2=0;

cipherChar+=keyMatrix[r1][c2];

cipherChar+=keyMatrix[r2][c1];

}

else if(r1==r2)

{

++c1;

++c2;

if(c1>4)

c1=0;

if(c2>4)

c2=0;

cipherChar+=keyMatrix[r1][c1];

cipherChar+=keyMatrix[r2][c2];

}

else{

cipherChar+=keyMatrix[r1][c2];

cipherChar+=keyMatrix[r2][c1];

}

return cipherChar;

}

static String Encrypt(String plainText,String key)

{

char keyMatrix[][]=new char[5][5];

keyMatrix= insertKey(key);

String cipherText="";

plainText=plainText.replaceAll("j", "i");

plainText=plainText.replaceAll(" ", "");

plainText=plainText.toUpperCase();

int len=plainText.length();

// System.out.println(plainText.substring(1,2+1));

if(len/2!=0)

{

plainText+="X";

++len;

}

for(int i=0;i < len-1;i=i+2)

{

cipherText+=encryptChar(plainText.substring(i,i+2),keyMatrix);

cipherText+=" ";

}

return cipherText;

}

static String decryptChar(String cipher,char keyMatrix[][])

{

cipher=cipher.toUpperCase();

char a=cipher.charAt(0),b=cipher.charAt(1);

String plainChar="";

int r1,c1,r2,c2;

r1=rowPos(a,keyMatrix);

c1=columnPos(a,keyMatrix);

r2=rowPos(b,keyMatrix);

c2=columnPos(b,keyMatrix);

if(c1==c2)

{

--r1;

--r2;

if(r1 < 0)

r1=4;

if(r2 < 0)

r2=4;

plainChar+=keyMatrix[r1][c2];

plainChar+=keyMatrix[r2][c1];

}

else if(r1==r2)

{

--c1;

--c2;

if(c1 < 0)

c1=4;

if(c2 < 0)

c2=4;

plainChar+=keyMatrix[r1][c1];

plainChar+=keyMatrix[r2][c2];

}

else{

plainChar+=keyMatrix[r1][c2];

plainChar+=keyMatrix[r2][c1];

}

return plainChar;

}

static String Decrypt(String cipherText,String key)

{

char keyMatrix[][]=new char[5][5];

keyMatrix= insertKey(key);

String plainText="";

cipherText=cipherText.replaceAll("j", "i");

cipherText=cipherText.replaceAll(" ", "");

cipherText=cipherText.toUpperCase();

int len=cipherText.length();

for(int i=0;i < len-1;i=i+2)

{

plainText+=decryptChar(cipherText.substring(i,i+2),keyMatrix);

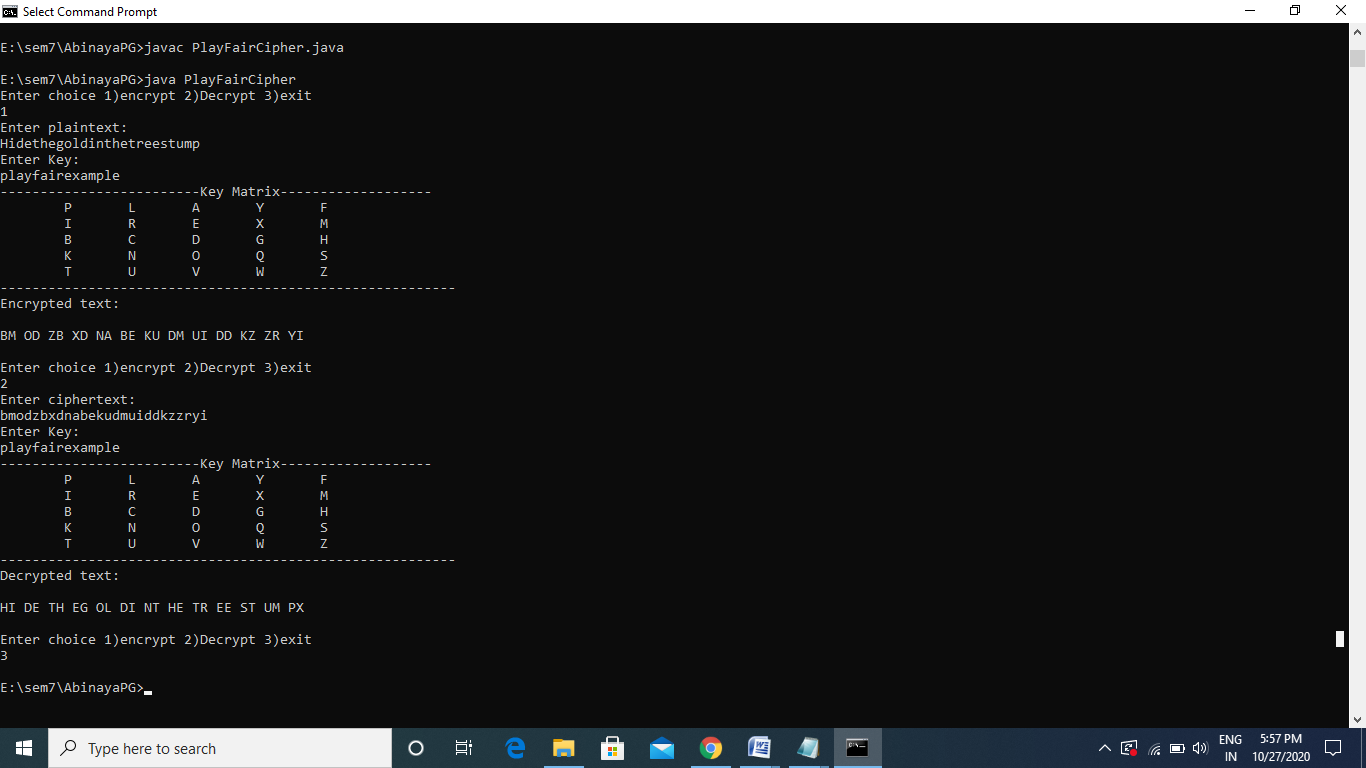
plainText+=" ";

}

return plainText;

}}

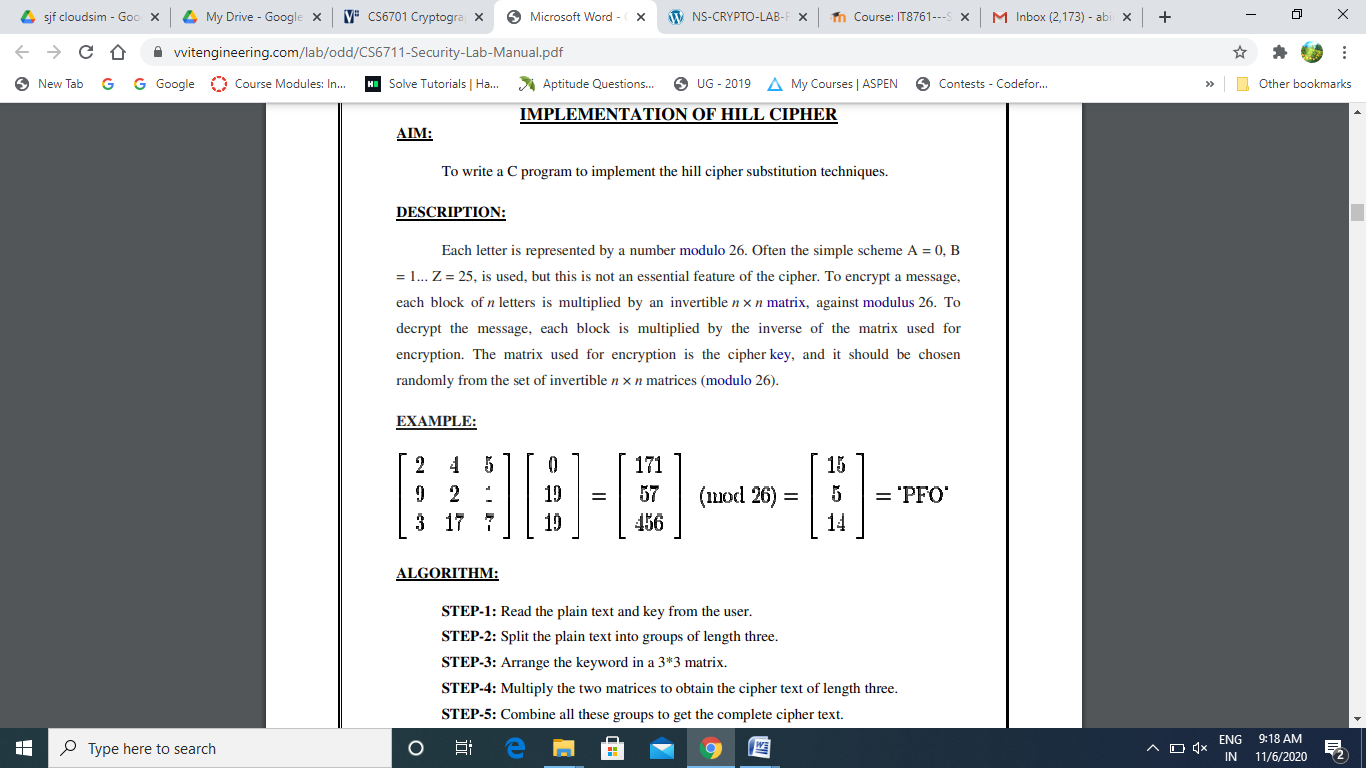
**Output:**



1. **Hill Cipher:**

**Description:**

Each letter is represented by a number modulo 26.Often the simple scheme A=0, B= 1... Z = 25, is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters is multiplied by an invertible n\*n matrix against 26.To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n\*n matrix (modulo 26).



**Algorithm:**

* In the Hill cipher, each letter is represented by a number modulo 26.
* Divide the input string into blocks of size n and Identify A=0, B=1, C=2, …, Z=25.
* To encrypt a message, each block of n letters is multiplied by an invertible ***n x n*** matrix, again ***modulus 26***. i.e.
* **C=[P][K] mod 26**
* Where C is the cipher text, P is the plain text and K is the key.
* Perform the above step for each block of the plain text.
* To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption i.e. **P=[C][K inverse] mod 26.**

**Program:**

import java.util.\*;

class HillCipher{

public static void main(String args[])

{

String plainText,cipherText;

int block;

int choice;

Scanner scn=new Scanner(System.in);

do{

System.out.println("Enter choice 1)encrypt 2)Decrypt: 3)exit");

choice=scn.nextInt();

switch(choice)

{

case 1:

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter plain-text:");

plainText=sc.nextLine();

System.out.println("Enter block size of matrix:");

block=sc.nextInt();

System.out.println(encrypt(plainText,block));

break;

}

case 2:

{

System.out.println("Enter cipher text:");

Scanner scan=new Scanner(System.in);

cipherText=scan.nextLine();

System.out.println("Enter block size of matrix:");

block=scan.nextInt();

System.out.println(Decrypt(cipherText,block));

break;

}

}

}while(choice!=3);

}

static int[][] keyInsert(int block)

{

int key[][]=new int[block][block];

Scanner scn=new Scanner(System.in);

System.out.println("Enter key Matrix");

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

{

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

{

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

}

}

return key;

}

static int[][] KeyInverseInsert(int block)

{

Scanner scn=new Scanner(System.in);

int key[][]=new int[block][block];

System.out.println("Enter key Inverse Matrix:");

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

{

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

{

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

}

}

return key;

}

static int indexOfChar(char c)

{

String allChar="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

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

{

if(allChar.charAt(i)==c)

return i;

}

return -1;

}

static char charAtIndex(int pos)

{

String allChar="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

return allChar.charAt(pos);

}

static String encryptBlock(String plain,int block,int key[][])

{

plain=plain.toUpperCase();

String b1="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int a[][]=new int[block][1],sum=0;

int cipherMatrix[][]=new int[block][1];

String cipher="";

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

{

a[i][0]=indexOfChar(plain.charAt(i));

}

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

{

for(int j=0;j < 1;j++)

{

for(int k=0;k < block;k++)

{

sum=sum+key[i][k]\*a[k][j];

}

cipherMatrix[i][j] = sum%26;

sum = 0;

}

}

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

{

cipher+=charAtIndex(cipherMatrix[i][0]);

}

return cipher;

}

static String encrypt(String plainText,int block)

{

String cipherText="";

int key[][]=new int[block][block];

key=keyInsert(block);

String b1="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

plainText=plainText.toUpperCase();

int len=plainText.length();

// System.out.println(plainText.substring(1,2+1));

while(len%block!=0)

{

plainText+="X";

System.out.println(len);

len=plainText.length();

}

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

{

cipherText+=encryptBlock(plainText.substring(i,i+block),block,key);

cipherText+=" ";

}

return cipherText;

}

static String decryptBlock(String cipher,int block,int key[][])

{

cipher=cipher.toUpperCase();

int a[][]=new int[block][1],sum=0;

int plainMatrix[][]=new int[block][1];

String plain="";

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

{

a[i][0]=indexOfChar(cipher.charAt(i));

}

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

{

for(int j=0;j < 1;j++)

{

for(int k=0;k < block;k++)

{

sum=sum+key[i][k]\*a[k][j];

}

while(sum < 0)

{

sum+=26;

}

plainMatrix[i][j] = sum%26;

sum = 0;} }

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

plain+=charAtIndex(plainMatrix[i][0]);}

return plain;}

static String Decrypt(String cipherText,int block){

String plainText="";

int key[][]=new int[block][block];

key=KeyInverseInsert(block);

cipherText=cipherText.replaceAll(" ", "");

cipherText=cipherText.toUpperCase();

int len=cipherText.length();

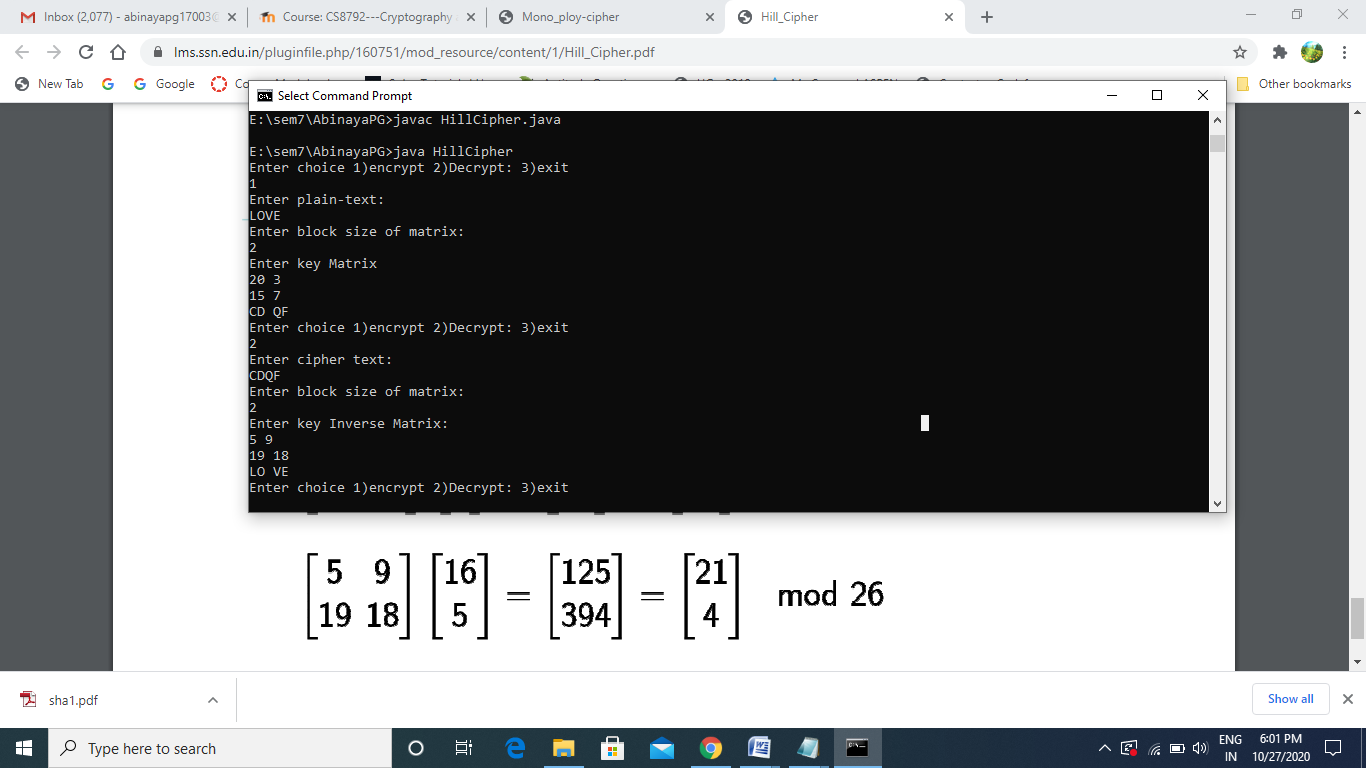
for(int i=0;i < len-1;i=i+block){

plainText+=decryptBlock(cipherText.substring(i,i+block),block,key);

plainText+=" "; }

return plainText; }}

**Output:**



**Result:**

The above classical encryption substitution techniques have been successfully executed.

Ex: 2 [Classical Encryption Transposition Techniques](https://lms.ssn.edu.in/mod/assign/view.php?id=83301)

Date: 25.08.2020

**Aim:**

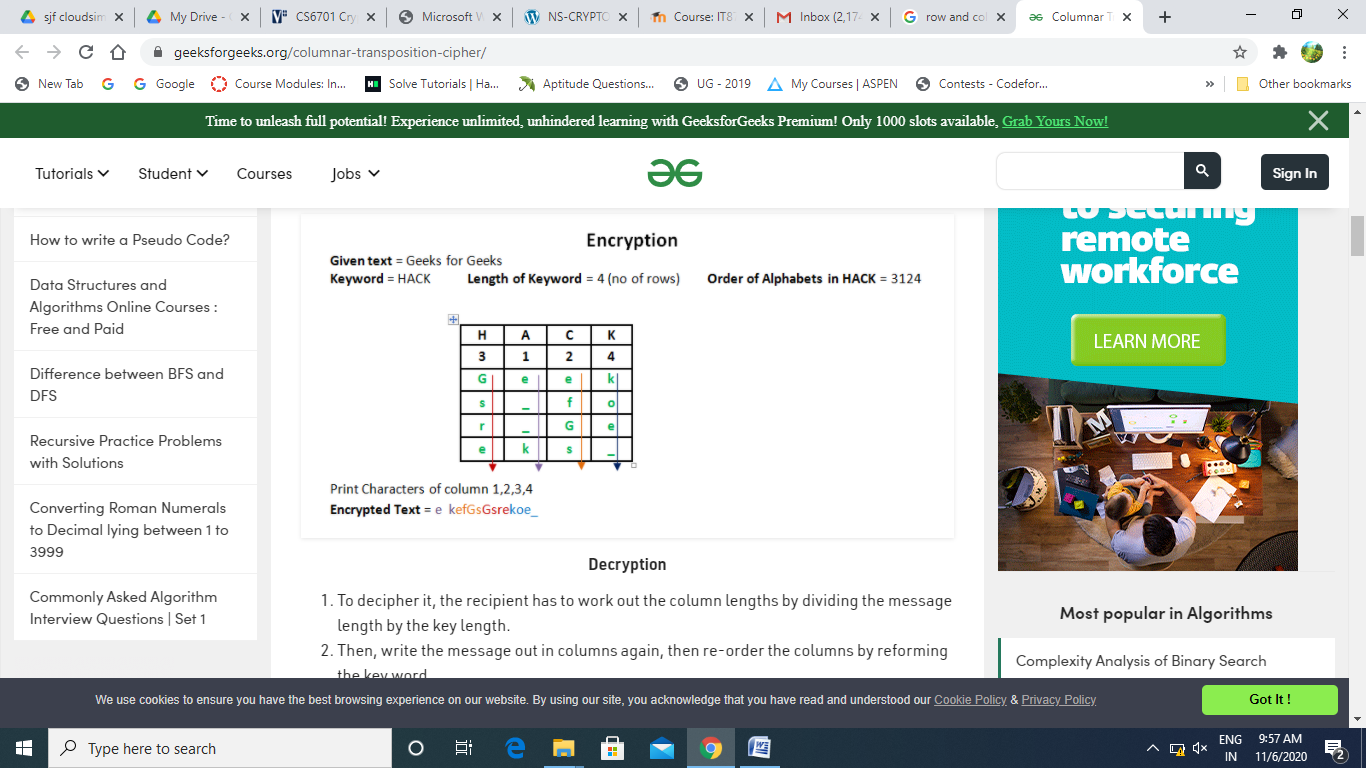
To implement Classical Encryption Transposition Techniques (Row column Transposition, Rail Fence, One Time Pad) using java.

1. **Row Column Transposition**

**Description:**

Given a plain-text message and a numeric key, cipher/de-cipher the given text using Row and Column Transposition Cipher

The Row /Column Transposition Cipher is a form of transposition cipher just like [Rail Fence Cipher](https://www.geeksforgeeks.org/rail-fence-cipher-encryption-decryption/). Row/Column Transposition involves writing the plaintext out in columns/rows, and then reading the cipher text off in rows/columns one by one.



**Algorithm:**

* The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
* Width of the rows and the permutation of the columns are usually defined by a keyword.
* For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.
* Any spare spaces are filled with nulls or left blank or placed by a character (Example: \_).
* Finally, the message is read off in columns, in the order specified by the keyword.
* To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
* Then, write the message out in columns again, then re-order the columns by reforming the key word.

**Program:**

import java.util.\*;

class transposition{

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

{

int choice;

String plainText,cipherText,decryptedText;

do{

Scanner scn=new Scanner(System.in);

System.out.println("Enter choice 1)encrypt 2)Decrypt 3)exit");

choice=scn.nextInt();

switch(choice)

{

case 1:

{

encryption();

break;

}

case 2:

{

decryption();

break;

}

}

}while(choice!=3);

}

static void encryption()

{

String plainText;

int row,col,i,j;

Scanner scan=new Scanner(System.in);

System.out.println("Enter plain text:");

plainText=scan.nextLine();

char[] stringToCharArray = plainText. toCharArray();

System.out.println("Enter row and col");

row=scan.nextInt();

col=scan.nextInt();

int arr[]=new int[col];

System.out.println("Enter keys");

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

{

arr[i]=scan.nextInt();

}

char mat[][]=new char[row][col];

String encryp="";

int k=0;

int val=82;

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

{

for(j=0;j<col;j++)

{

if(k>=plainText.length())

{

mat[i][j]=(char)val;

val=val+1;

}

else

{

mat[i][j]=stringToCharArray[k];

k=k+1;

}

}

}

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

{

for(j=0;j<row;j++)

{

if(mat[j][arr[i]]!='0')

encryp+=mat[j][arr[i]];

}

}

System.out.println(encryp);

}

static void decryption()

{

String cipherText;

Scanner scan=new Scanner(System.in);

System.out.println("Enter cipherText");

cipherText=scan.nextLine();

int row,col;

char[] stringToCharArray = cipherText. toCharArray();

System.out.println("Enter row and col");

row=scan.nextInt();

col=scan.nextInt();

int arr[]=new int[col];

int i;

System.out.println("Enter keys");

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

{

arr[i]=scan.nextInt();

}

int k=0,j;

char mat[][]=new char[row][col];

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

{

for(j=0;j<row;j++)

{

if(k<cipherText.length())

{

mat[j][arr[i]]=stringToCharArray[k];

k=k+1;

}

}

}

String decrypt="";

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

{

for(j=0;j<col;j++)

{

if(Character.isLowerCase(mat[i][j]))

decrypt+=mat[i][j];

}

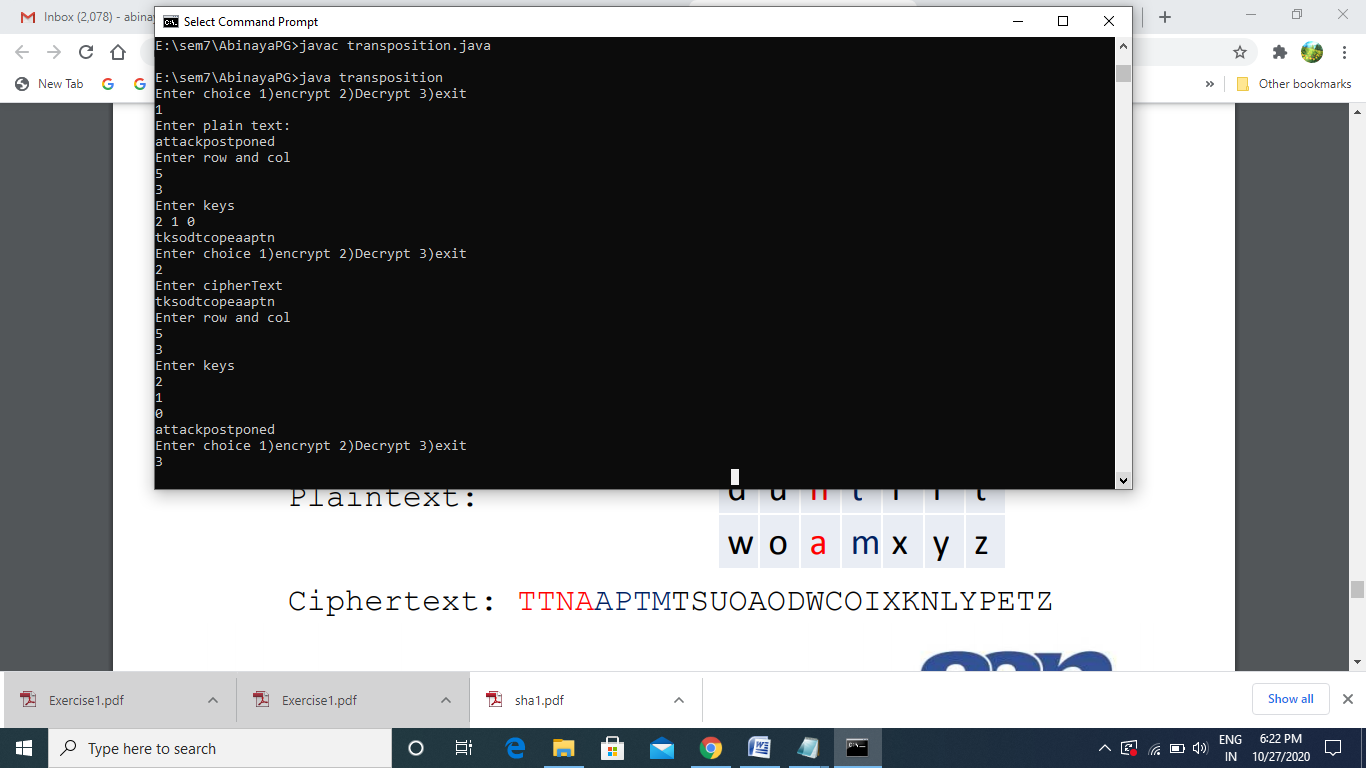
}

System.out.println(decrypt);

}

}

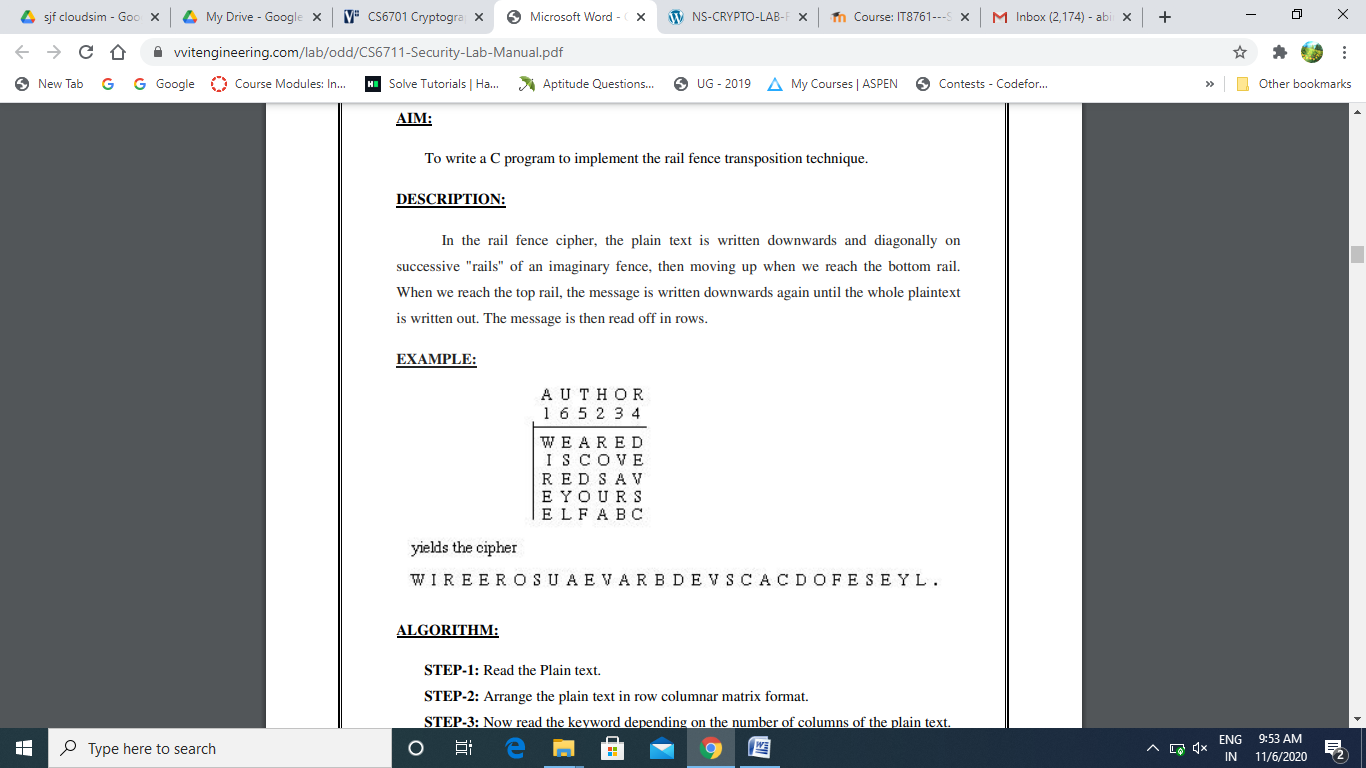
**Output:**



1. **Rail Fence**

**Description:**

In the rail fence cipher, the plain text is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.



**Algorithm:**

* In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.
* When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus, the alphabets of the message are written in a zig-zag manner.
* After each alphabet has been written, the individual rows are combined to obtain the cipher-text.
* The number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.
* Hence, rail matrix can be constructed accordingly. Once we’ve got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively).
* Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

**Program:**

import java.util.\*;

class RailFence{

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

{

int depth=2,choice;

String plainText,cipherText,decryptedText;

do{

Scanner scn=new Scanner(System.in);

System.out.println("Enter choice 1)encrypt 2)Decrypt 3)exit");

choice=scn.nextInt();

switch(choice)

{

case 1:

{

Scanner scan=new Scanner(System.in);

System.out.println("Enter plain text:");

plainText=scan.nextLine();

cipherText=Encryption(plainText,depth);

System.out.println("Encrypted text is:\n"+cipherText);

break;

}

case 2:

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter cipher text:");

cipherText=sc.nextLine();

decryptedText=Decryption(cipherText, depth);

System.out.println("Decrypted text is:\n"+decryptedText);

break;

}

}

}while(choice!=3);

}

static String Encryption(String plainText,int depth)

{

int r=depth,len=plainText.length(),c;

if(len%2!=0)

c=len/depth+1;

else

c=len/depth;

char mat[][]=new char[r][c];

int k=0;

String cipherText="";

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

{

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

{

if(k!=len)

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

else

mat[j][i]='X';

}

}

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

{

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

{

cipherText+=mat[i][j];

}

}

return cipherText;

}

static String Decryption(String cipherText,int depth)

{

int r=depth,len=cipherText.length();

int c=len/depth;

char mat[][]=new char[r][c];

int k=0;

String plainText="";

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

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

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

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

{

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

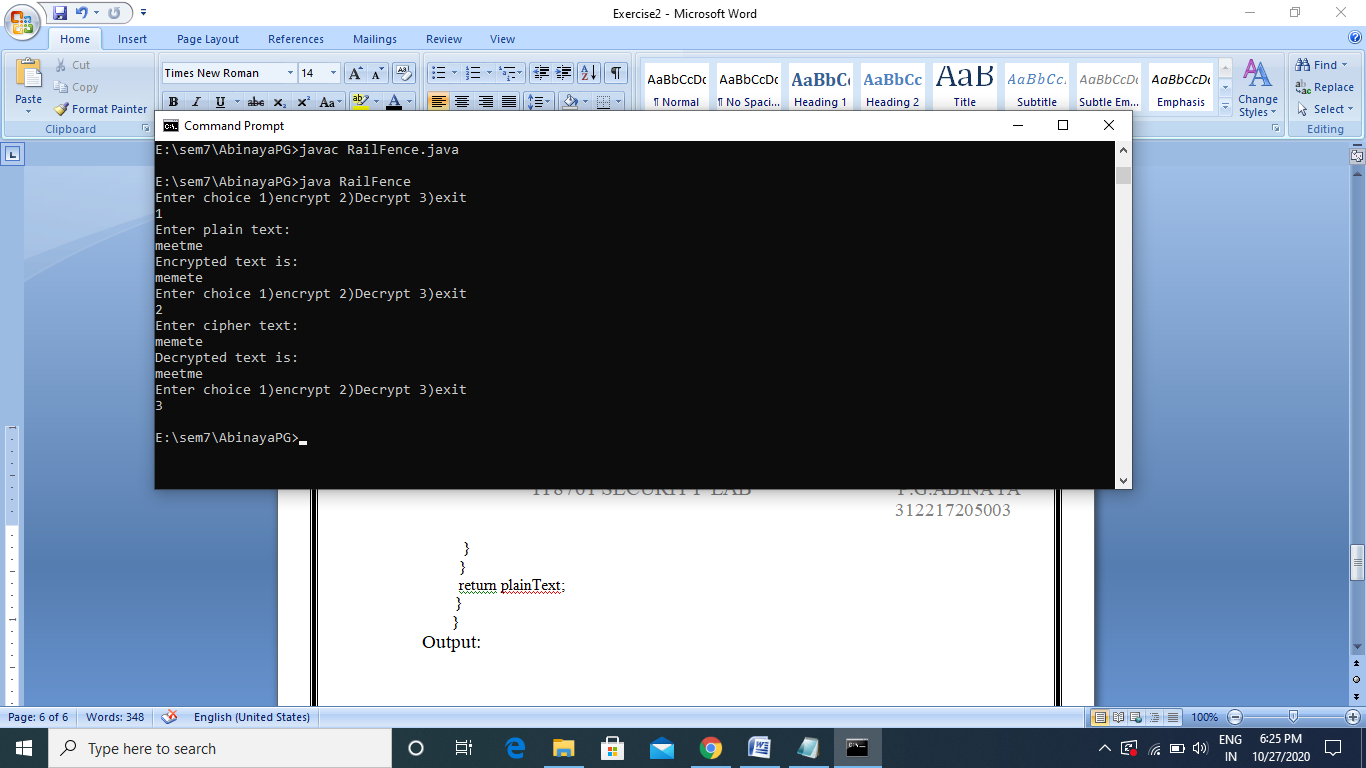
{

if(mat[j][i]!='X')

plainText+=mat[j][i];} }

return plainText; }}

**Output:**



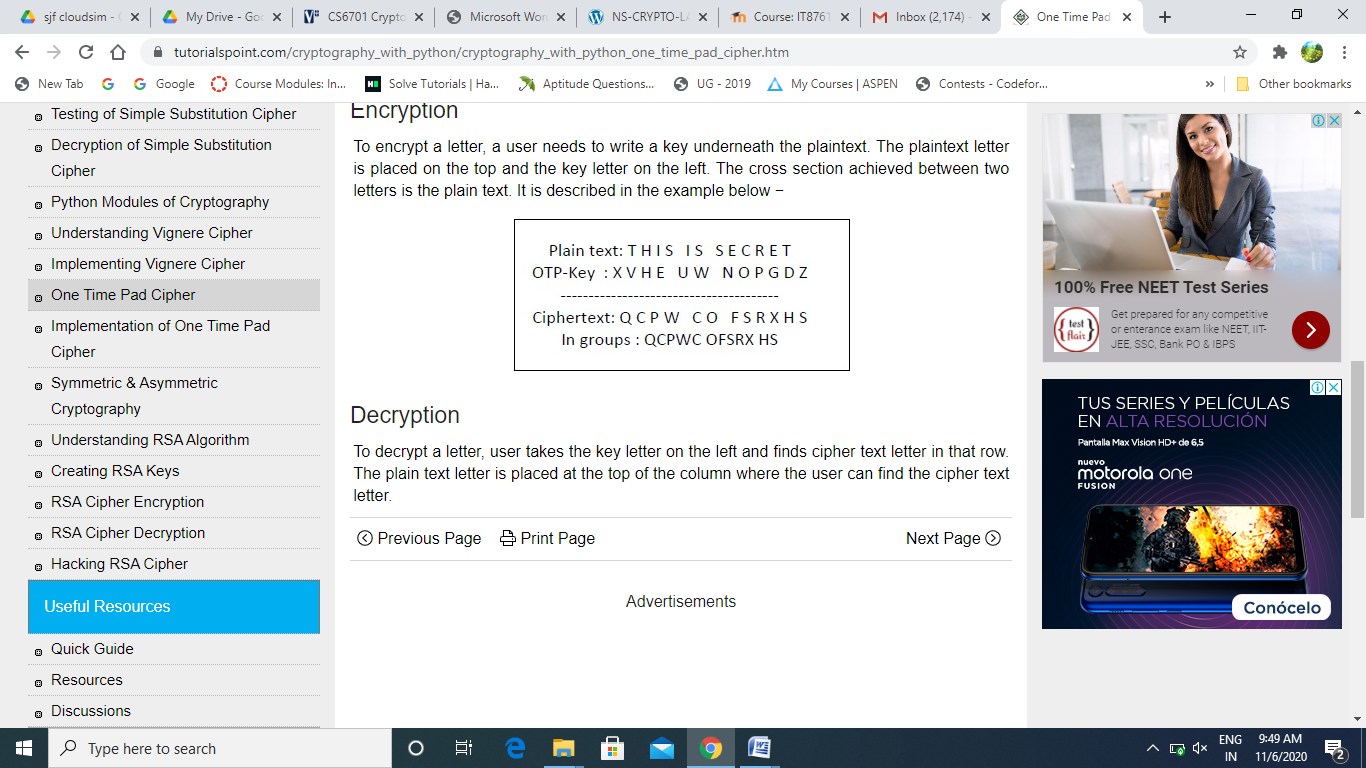
1. **One Time Pad**

**Description:**

One-time pad cipher is a type of Vignere cipher which includes the following features −

* It is an unbreakable cipher.
* The key is exactly same as the length of message which is encrypted.
* The key is made up of random symbols.
* As the name suggests, key is used one time only and never used again for any other message to be encrypted.

Due to this, encrypted message will be vulnerable to attack for a cryptanalyst. The key used for a one-time pad cipher is called **pad**, as it is printed on pads of paper.



**Algorithm:**

* The plain text and the random keys are read as input
* The plain text and key must be same in size
* The cipher text is generated by doing bit by bit XOR of the plain text with the key.
* For Decryption, the plain text is generated in the same way by doing bit by bit XOR of the cipher text with the shared one time key.

**Program:**

import java.util.\*;

class otp

{

public static void main(String[] args)

{

int choice;

do

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the choice 1)Encrypt 2)Decrypt 3)Exit:");

choice=sc.nextInt();

switch(choice)

{

case 1:

{

encryption();

break;

}

case 2:

{

decryption();

break;

}

}

}while(choice!=3);

}

static void encryption()

{

Scanner scan=new Scanner(System.in);

System.out.println("Enter the plain Text");

String text=scan.nextLine();

char[] stringToCharArray = text.toCharArray();

int i;

String output="";

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

{

int value=(int)stringToCharArray[i];

String binary=Integer.toBinaryString(value);

Scanner sc=new Scanner(System.in);

System.out.println("Enter the key for letter "+ (i+1));

//int bin1=sc.nextInt();

String binarykey=sc.nextLine();

StringBuffer sb=new StringBuffer();

for (int j = 0; j < binary.length(); j++) {

sb.append(binarykey.charAt(j)^binary.charAt(j));

}

output+=sb;

output+=" ";

}

System.out.println("Cipher Text is "+output);

}

static void decryption()

{

Scanner scan=new Scanner(System.in);

System.out.println("Enter the cipher Text length");

int textlen=scan.nextInt();

//char[] stringToCharArray = text.toCharArray();

int i;

String output="";

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

{

System.out.println("Enter the cipher Text for letter "+ (i+1));

//int value=(int)stringToCharArray[i];

//String binary=Integer.toBinaryString(value);

Scanner sc=new Scanner(System.in);

String binary=sc.nextLine();

System.out.println("Enter the key for letter "+ (i+1));

//int bin1=sc.nextInt();

String binarykey=sc.nextLine();

StringBuffer sb=new StringBuffer();

for (int j = 0; j < binary.length(); j++) {

sb.append(binarykey.charAt(j)^binary.charAt(j));

}

String check="";

check+=sb;

int value=Integer.parseInt(check,2);

output+=(char)value;

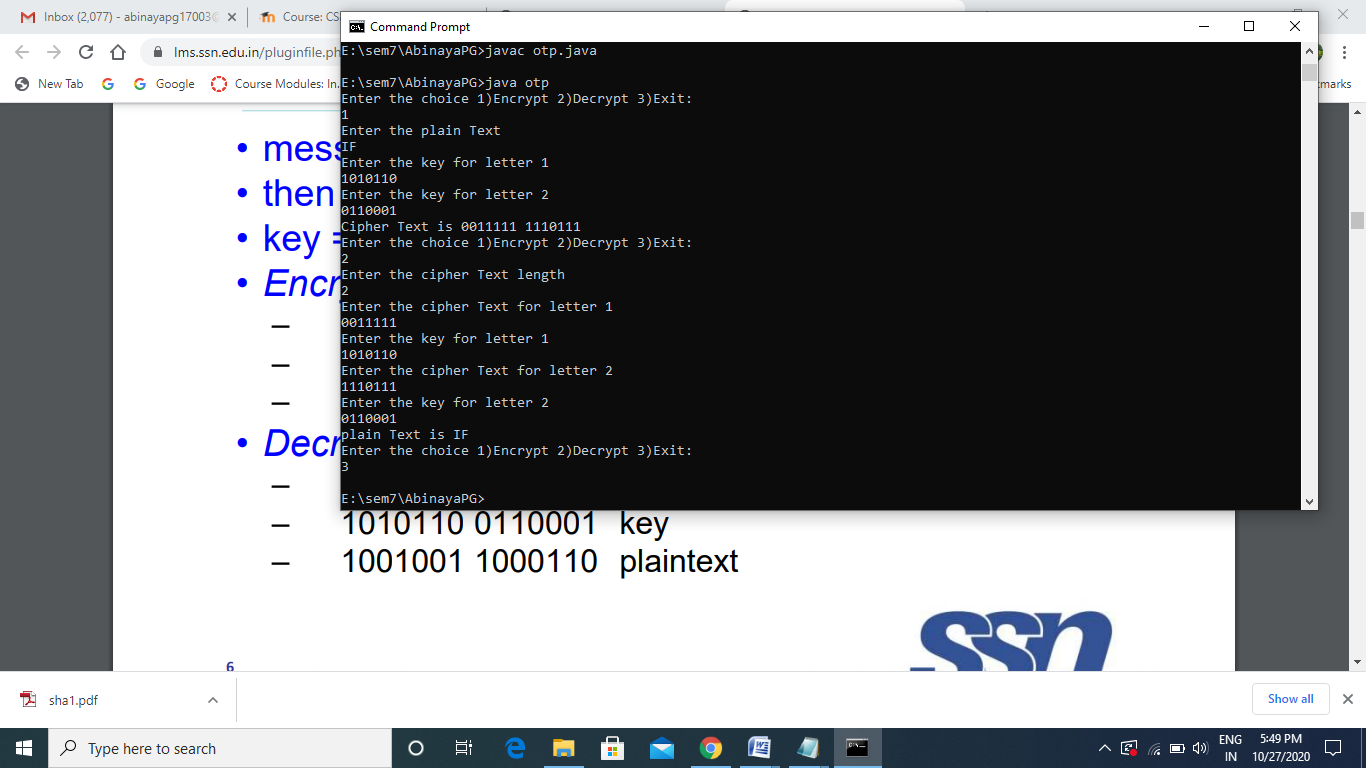
}

System.out.println("plain Text is "+output);

}

}

**Output:**



**Result:**

The above programs to implement classical encryption transposition techniques have been successfully executed.

Ex: 3 [Data](https://lms.ssn.edu.in/mod/assign/view.php?id=83301) Encryption Standard (DES)

Date: 08.09.2020

**Aim:**

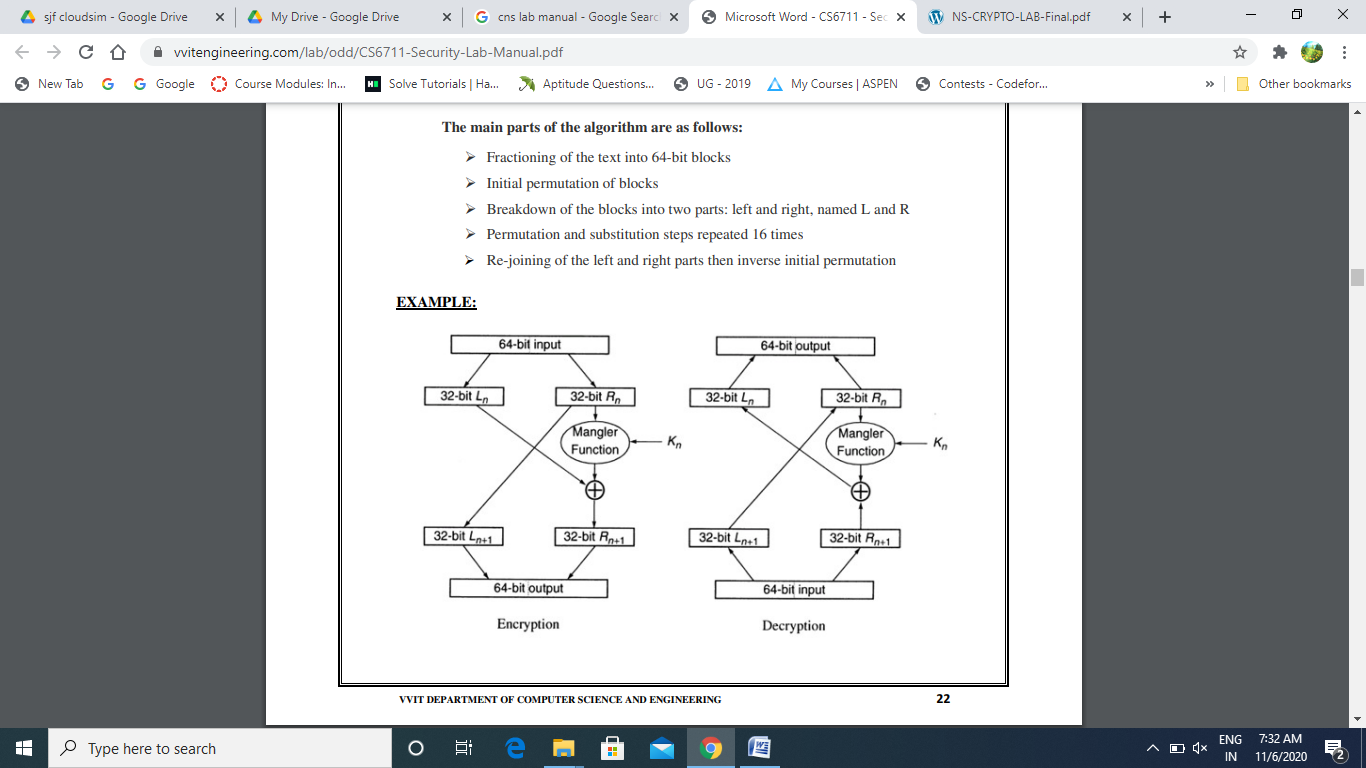
To write a java program to implement Data Encryption Standard (DES).

**Description:**

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted k1 to k16. Given that "only" 56 bits are actually used for encrypting, there can be 256 different keys.

The main parts of the algorithm are as follows:

* Fractioning of the text into 64-bit blocks
* Initial permutation of blocks
* Breakdown of the blocks into two parts: left and right, named L and R
* Permutation and substitution steps repeated 16 times
* Re-joining of the left and right parts then inverse initial permutation



**Algorithm:**

* Read the 64-bit plain text.
* Split it into two 32-bit blocks and store it in two different arrays.
* Perform XOR operation between these two arrays.
* The output obtained is stored as the second 32-bit sequence and the original second 32-bit sequence forms the first part.
* Thus the encrypted 64-bit cipher text is obtained in this way. Repeat the same process for the remaining plain text characters.

**Program:**

import java.util.\*;

class des {

private static class DES {

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-Permutation 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;

}

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)

{

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)

{

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

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

String right = temp;

temp = permutation(EP, temp);

temp = xor(temp, key);

temp = sBox(temp);

temp = permutation(P, temp);

left = xor(left, temp);

System.out.println("Round "+ (num + 1) + " Text: " + hextoBin(left)+" "

+ hextoBin(right) + "(" + left.toUpperCase() +" " + right.toUpperCase()

+")\nKey: "+hextoBin(key)+"("+ key.toUpperCase()+")\n");

System.out.println("------------------------------------------------------------------------");

// 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: "

+ hextoBin(plainText).toUpperCase()+"("+ plainText.toUpperCase()+")");

System.out.println(

"After splitting: L0="

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

+ " R0="

+ hextoBin(plainText.substring(8, 16)).toUpperCase() + "\n");

System.out.println("------------------------------------------------------------------------");

// 16 rounds

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

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

}

//System.out.println("------------------------------------------------------------------------");

// 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;

String keys[] = getKeys(key);

plainText = permutation(IP, plainText);

System.out.println("After initial permutation: "+hextoBin(plainText).toUpperCase() +"("+plainText.toUpperCase()+")");

System.out.println(

"After splitting: L0="

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

+ " R0=" + plainText.substring(8, 16).toUpperCase()

+ "\n");

System.out.println("------------------------------------------------------------------------");

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

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

}

plainText = plainText.substring(8, 16)

+ plainText.substring(0, 8);

plainText = permutation(IP1, plainText);

return plainText; } }

public static void main(String args[]) {

String key,cipherText,plainText;

int choice;

do{

Scanner scn=new Scanner(System.in);

System.out.println("Enter choice 1)encrypt 2)Decrypt 3)exit");

choice=scn.nextInt();

switch(choice){

case 1:{

Scanner sc=new Scanner(System.in);

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

plainText=sc.nextLine();

System.out.println("Enter Key:");

key=sc.nextLine();

DES cipher = new DES();

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

plainText = cipher.encrypt(plainText, key);

System.out.println( "\nCipher Text(hexadecimal): " + plainText.toUpperCase() +"\nCipher Text(Binary): "+cipher.hextoBin(plainText).toUpperCase() +"\n");

break;}

case 2:

{

Scanner scan=new Scanner(System.in);

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

plainText=scan.nextLine();

System.out.println("Enter Key:");

key=scan.nextLine();

DES cipher = new DES();

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

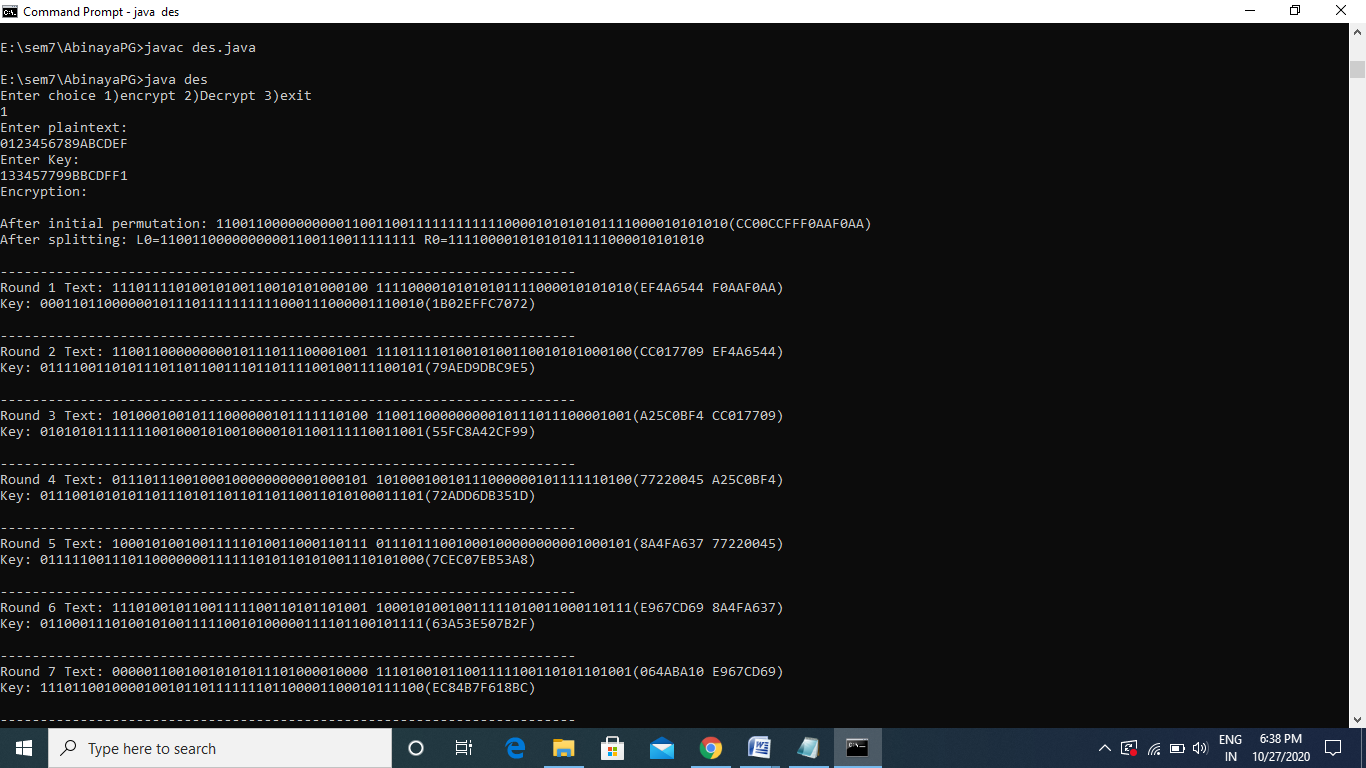
plainText = cipher.decrypt(plainText, key);

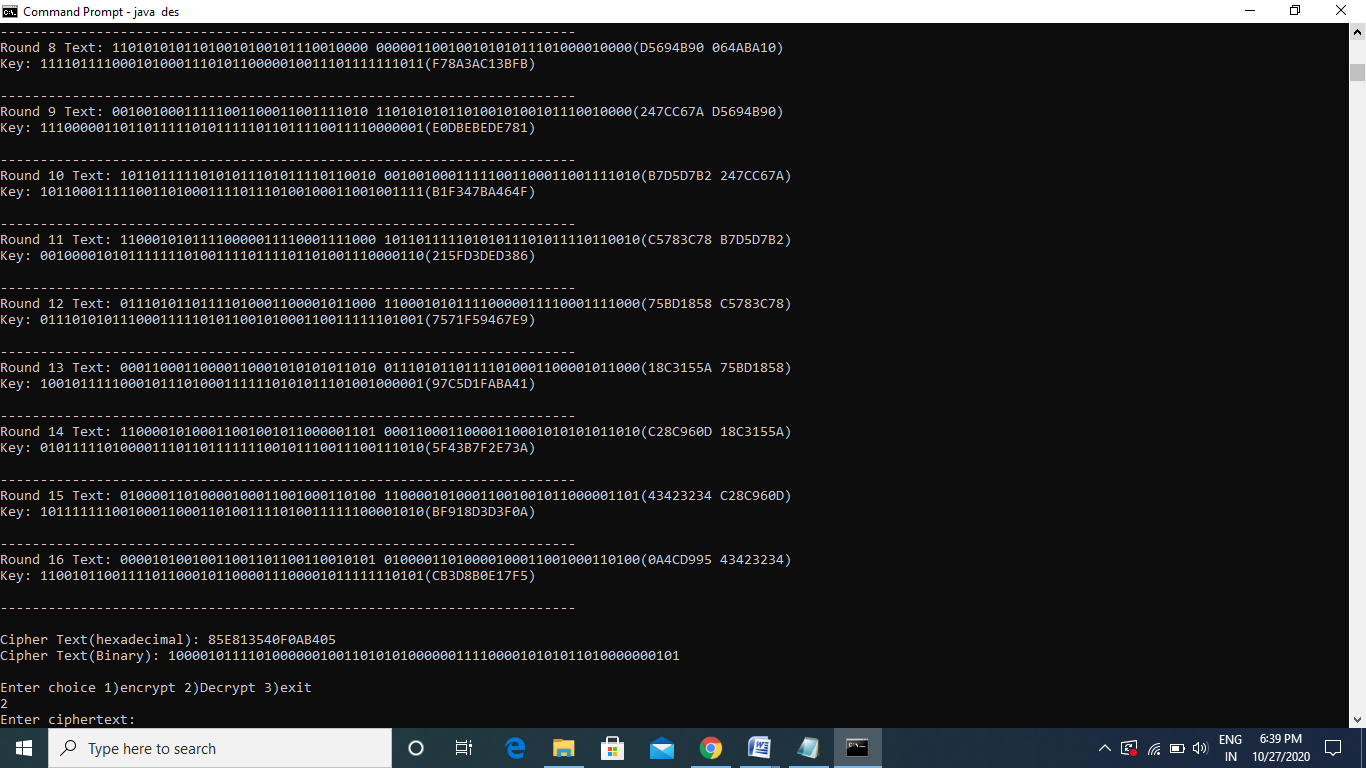
System.out.println( "\nPlain Text(hexadecimal): "+ plainText.toUpperCase() +"\nPlain Text(Binary): "+cipher.hextoBin(plainText).toUpperCase() +"\n" );

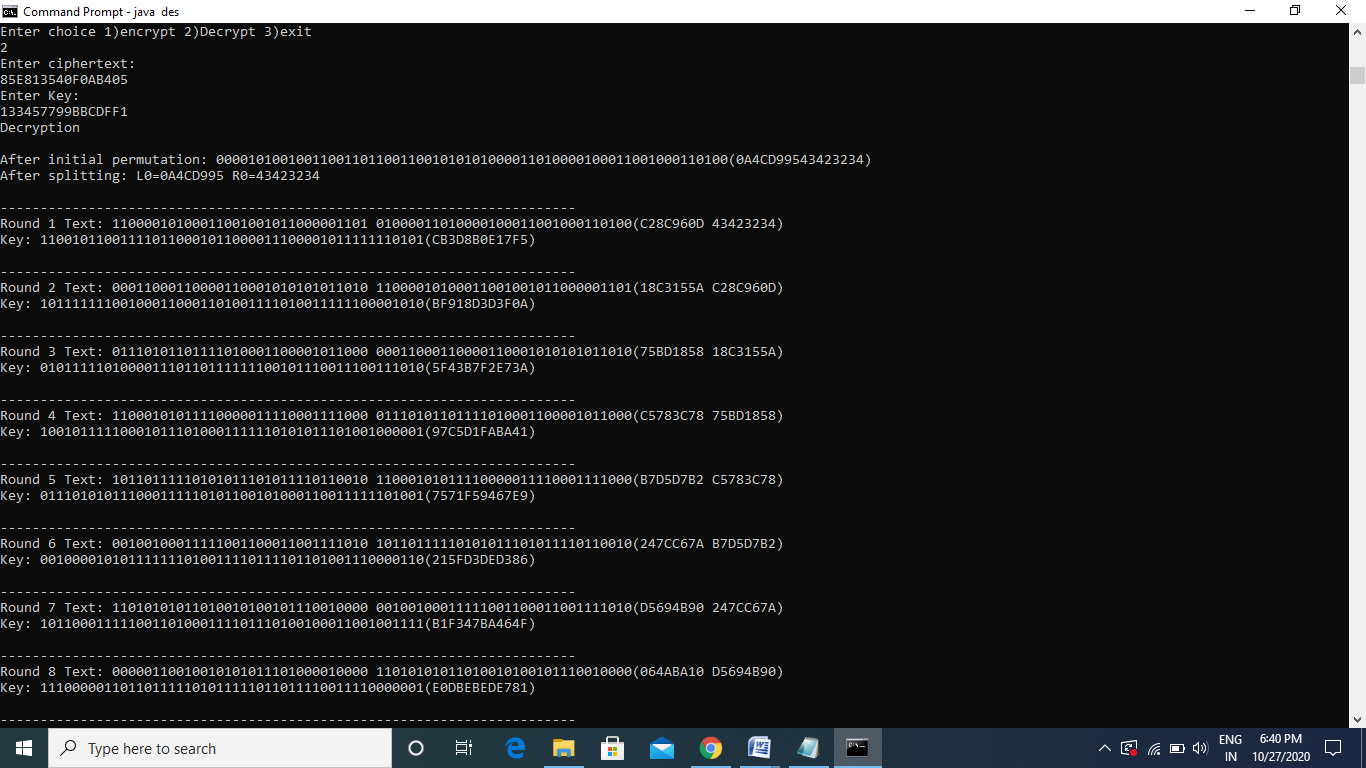
break;}}

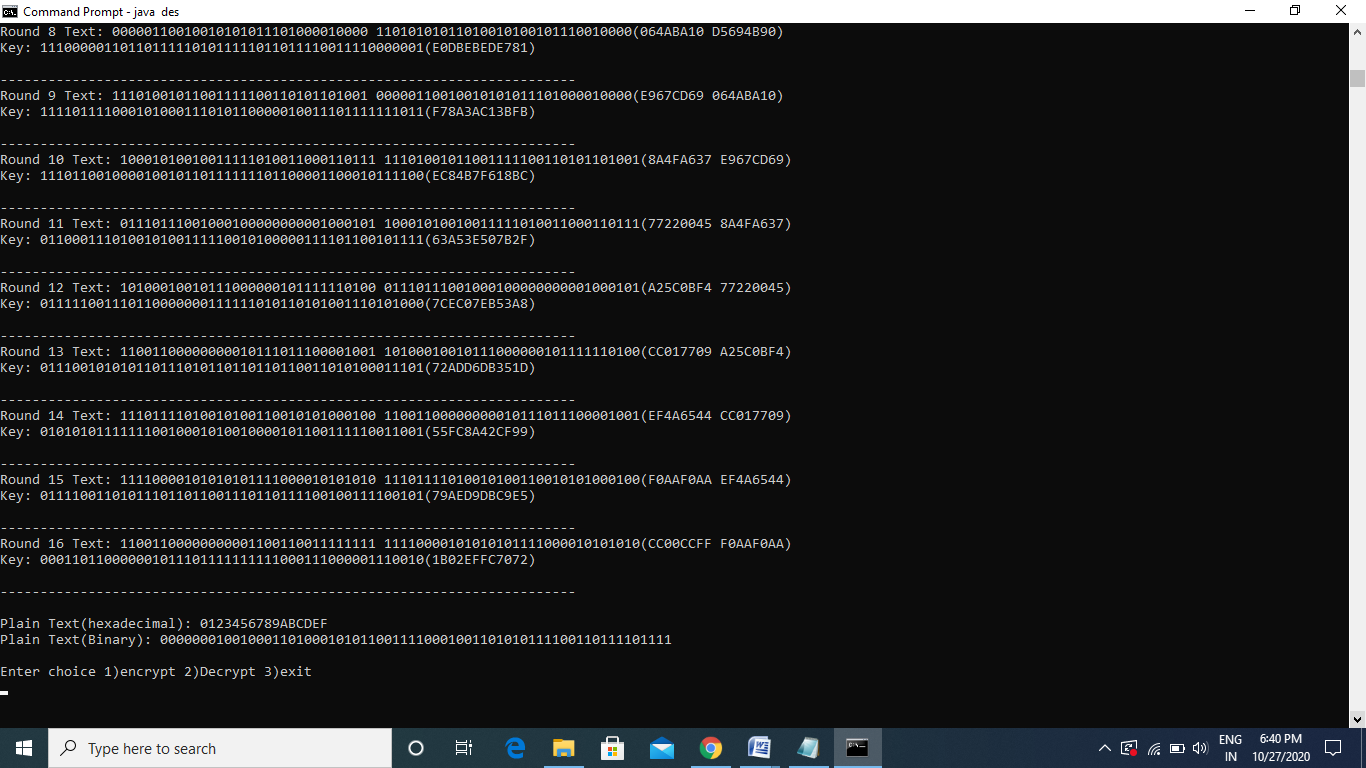
}while(choice!=3); } }

**Output:**









**Result:**

The above java program to implement the DES algorithm has been successfully executed.

Ex: 4 Advanced Encryption Standards (AES)

Date:15.09.2020

**Aim:**

To implement Advanced Encryption Standards (AES) using java.

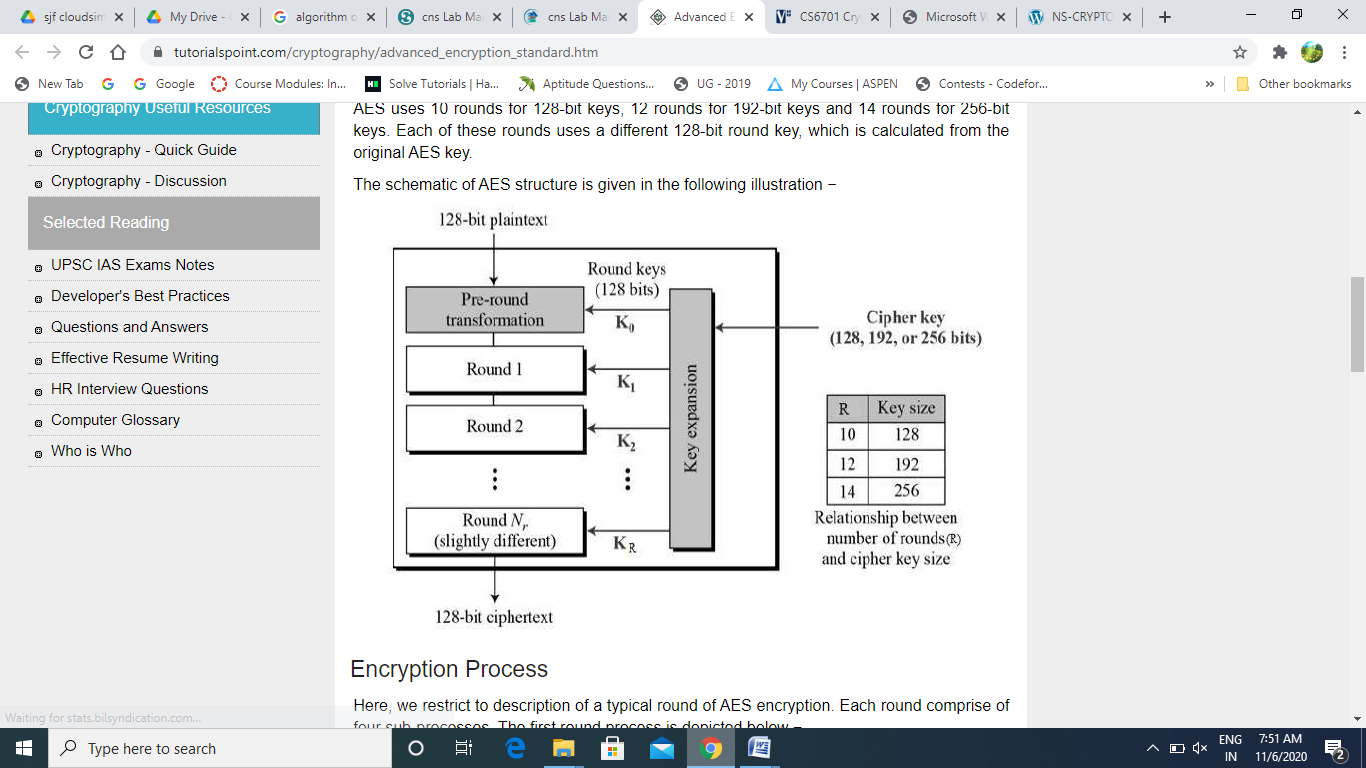
**Description:**

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six time faster than triple DES.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

The features of AES are as follows −

* Symmetric key symmetric block cipher
* 128-bit data, 128/192/256-bit keys
* Stronger and faster than Triple-DES
* Provide full specification and design details
* Software implementable in C and Java



**Algorithm:**

* Read the 128-bit plain text.
* Add the round Key.
* The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.
* Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row.
* The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.
* Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.
* Repeat the same process for all the rounds.
* The final round should be done in similar way without mix columns.
* Decryption is done in reverse way to encryption.

**Program:**

import java.util.\*;

class aes {

private static class AES

{

String[][] sbox={

{"63","7C","77","7B","F2","6B","6F","C5","30","01","67","2B","FE","D7","AB","76"}, {"CA","82","C9","7D","FA","59","47","F0","AD","D4","A2","AF","9C","A4","72","C0"},

{"B7","FD","93","26","36","3F","F7","CC","34","A5","E5","F1","71","D8","31","15"},

{"04","C7","23","C3","18","96","05","9A","07","12","80","E2","EB","27","B2","75"},

{"09","83","2C","1A","1B","6E","5A","A0","52","3B","D6","B3","29","E3","2F","84"},

{"53","D1","00","ED","20","FC","B1","5B","6A","CB","BE","39","4A","4C","58","CF"},

{"D0","EF","AA","FB","43","4D","33","85","45","F9","02","7F","50","3C","9F","A8"}, {"51","A3","40","8F","92","9D","38","F5","BC","B6","DA","21","10","FF","F3","D2"},

{"CD","0C","13","EC","5F","97","44","17","C4","A7","7E","3D","64","5D","19","73"},

{"60","81","4F","DC","22","2A","90","88","46","EE","B8","14","DE","5E","0B","DB"},

{"E0","32","3A","0A","49","06","24","5C","C2","D3","AC","62","91","95","E4","79"},

{"E7","C8","37","6D","8D","D5","4E","A9","6C","56","F4","EA","65","7A","AE","08"},

{"BA","78","25","2E","1C","A6","B4","C6","E8","DD","74","1F","4B","BD","8B","8A"},

{"70","3E","B5","66","48","03","F6","0E","61","35","57","B9","86","C1","1D","9E"},

{"E1","F8","98","11","69","D9","8E","94","9B","1E","87","E9","CE","55","28","DF"},

{"8C","A1","89","0D","BF","E6","42","68","41","99","2D","0F","B0","54","BB","16"}

};

String[][] lbox={

{" ","00","19","01","32","02","1A","C6","4B","C7","1B","68","33","EE","DF","03"},

{"64","04","E0","0E","34","8D","81","EF","4C","71","08","C8","F8","69","1C","C1"},

{"7D","C2","1D","B5","F9","B9","27","6A","4D","E4","A6","72","9A","C9","09","78"},

{"65","2F","8A","05","21","0F","E1","24","12","F0","82","45","35","93","DA","8E"},

{"96","8F","DB","BD","36","D0","CE","94","13","5C","D2","F1","40","46","83","38"},

{"66","DD","FD","30","BF","06","8B","62","B3","25","E2","98","22","88","91","10"},

{"7E","6E","48","C3","A3","B6","1E","42","3A","6B","28","54","FA","85","3D","BA"},

{"2B","79","0A","15","9B","9F","5E","CA","4E","D4","AC","E5","F3","73","A7","57"},

{"AF","58","A8","50","F4","EA","D6","74","4F","AE","E9","D5","E7","E6","AD","E8"},

{"2C","D7","75","7A","EB","16","0B","F5","59","CB","5F","B0","9C","A9","51","A0"},

{"7F","0C","F6","6F","17","C4","49","EC","D8","43","1F","2D","A4","76","7B","B7"},

{"CC","BB","3E","5A","FB","60","B1","86","3B","52","A1","6C","AA","55","29","9D"},

{"97","B2","87","90","61","BE","DC","FC","BC","95","CF","CD","37","3F","5B","D1"},

{"53","39","84","3C","41","A2","6D","47","14","2A","9E","5D","56","F2","D3","AB"},

{"44","11","92","D9","23","20","2E","89","B4","7C","B8","26","77","99","E3","A5"},

{"67","4A","ED","DE","C5","31","FE","18","0D","63","8C","80","C0","F7","70","07"}

};

String[][] ebox={

{"01","03","05","0F","11","33","55","FF","1A","2E","72","96","A1","F8","13","35"},

{"5F","E1","38","48","D8","73","95","A4","F7","02","06","0A","1E","22","66","AA"},

{"E5","34","5C","E4","37","59","EB","26","6A","BE","D9","70","90","AB","E6","31"},

{"53","F5","04","0C","14","3C","44","CC","4F","D1","68","B8","D3","6E","B2","CD"},

{"4C","D4","67","A9","E0","3B","4D","D7","62","A6","F1","08","18","28","78","88"},

{"83","9E","B9","D0","6B","BD","DC","7F","81","98","B3","CE","49","DB","76","9A"},

{"B5","C4","57","F9","10","30","50","F0","0B","1D","27","69","BB","D6","61","A3"},

{"FE","19","2B","7D","87","92","AD","EC","2F","71","93","AE","E9","20","60","A0"},

{"FB","16","3A","4E","D2","6D","B7","C2","5D","E7","32","56","FA","15","3F","41"},

{"C3","5E","E2","3D","47","C9","40","C0","5B","ED","2C","74","9C","BF","DA","75"},

{"9F","BA","D5","64","AC","EF","2A","7E","82","9D","BC","DF","7A","8E","89","80"},

{"9B","B6","C1","58","E8","23","65","AF","EA","25","6F","B1","C8","43","C5","54"},

{"FC","1F","21","63","A5","F4","07","09","1B","2D","77","99","B0","CB","46","CA"},

{"45","CF","4A","DE","79","8B","86","91","A8","E3","3E","42","C6","51","F3","0E"},

{"12","36","5A","EE","29","7B","8D","8C","8F","8A","85","94","A7","F2","0D","17"},

{"39","4B","DD","7C","84","97","A2","FD","1C","24","6C","B4","C7","52","F6","01"}

};

String[] rcon={"01","02","04","08","10","20","40","80","1B","36"};

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;

}

String[] leftCircularShift(String[] input, int numBits)

{

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

String perm[] = new String[5];

if(numBits==1)

{

for (int i = 0; i < 3; i++)

perm[i] = input[i+1];

perm[3] = input[0];

}

else if(numBits==2)

{

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

perm[i] = input[i+2];

perm[3] = input[1];

perm[2] =input[0];

}

else if(numBits==3)

{

perm[0]=input[3];

perm[1]=input[0];

perm[2]=input[1];

perm[3]=input[2];

}

else

return input;

return perm;

}

String[] invleftCircularShift(String[] input, int numBits)

{

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

String perm[] = new String[5];

if(numBits==1)

{

for (int i = 0; i < 3; i++)

perm[i+1] = input[i];

perm[0] = input[3];

}

else if(numBits==2)

{

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

perm[i+2] = input[i];

perm[1] = input[3];

perm[0] =input[2];

}

else if(numBits==3)

{

perm[0]=input[3];

perm[1]=input[0];

perm[2]=input[1];

perm[3]=input[2];

}

else

return input;

return perm;

}

String[][] mix(String[][] pt)

{

int i,j,k;

String[][] mixcol=new String[4][4];

String[][] key={{"02","03","01","01"},

{"01","02","03","01"}, {"01","01","02","03"},

{"03","01","01","02"}};

//String temp="";

int value=0;

for (i=0;i<4;i++)

{

for (j=0;j<4;j++)

{

mixcol[i][j]="00";

for (k=0;k<4;k++)

{

value=Integer.parseInt(pt[k][j],16);

if(key[i][k].equals("03"))

{

if(value>=128)

{

int temp=((2\*value)^Integer.parseInt(pt[k][j],16))^283;

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^temp);

}

else

{

int find=(2\*value)^Integer.parseInt(pt[k][j],16);

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^find);

}

}

else if(key[i][k].equals("02"))

{

if(value>=128)

{

int fin=((2\*value)^283);

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^fin);

}

else

{

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^(2\*value));

}

}

else

{

int tem=Integer.parseInt(key[i][k],16);

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^(tem\*value));

}}}}

return mixcol;}

String[][] invmix(String[][] pt)

{

int i,j,k;

String[][] mixcol=new String[4][4];

String[][] key={{"14","11","13","09"},

{"09","14","11","13"}, {"13","09","14","11"},

{"11","13","09","14"}};

//String temp="";

int value=0;

for (i=0;i<4;i++)

{

for (j=0;j<4;j++)

{

mixcol[i][j]="00";

for (k=0;k<4;k++)

{

//mixcol[i][j]=Integer.toHexString((Integer.parseInt(key[i][k])^(Integer.parseInt(pt[k][j],16)))^(Integer.parseInt(mixcol[i][j],16)));

value=Integer.parseInt(pt[k][j],16);

if(key[i][k].equals("09"))

{

//if(value>=128)

//{

int temp=((((2\*value)\*2)\*2)^value);

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^temp);

//}

//else

//{

// int find=(2\*value)^Integer.parseInt(pt[k][j],16);

// mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^find);

//}

}

else if(key[i][k].equals("11"))

{

int temp=(((((2\*value)\*2)^value)\*2)^value);

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^temp);

}

else if(key[i][k].equals("13"))

{

int temp=(((((2\*value)^value)\*2)\*2)^value);

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^temp);

}

else

{

int temp=(((((2\*value)^value)\*2)^value)\*2);

mixcol[i][j]=Integer.toHexString(Integer.parseInt(mixcol[i][j],16)^temp);

}}}}

return mixcol;

}

String xor(String a, String b)

{

int n=a.length();

int i;

String output="";

for(i=0;i<8;i++)

{

if(a.charAt(i)==b.charAt(i))

output+="0";

else

output+="1";

}

return output;

}

String biadd(String a,String b)

{

int b1=Integer.parseInt(a,2);

int b2=Integer.parseInt(b,2);

int sum=b1+b2;

return Integer.toBinaryString(sum);

}

String binadd(String a,String b)

{

int b1=Integer.parseInt(a,16);

int b2=Integer.parseInt(b,16);

int mul=b1\*b2;

return Integer.toHexString(mul);

}

String permutation(String sequence)

{

String output = "";

//input = hextoBin(input);

int flag=0;

String g="",f="";

char a=sequence.charAt(0);

char b=sequence.charAt(1);

if(Character.compare(a,'A')==0 || Character.compare(a,'a')==0)

g="10";

else if(Character.compare(a,'B')==0 || Character.compare(a,'b')==0)

g="11";

else if(Character.compare(a,'C')==0 || Character.compare(a,'c')==0)

g="12";

else if(Character.compare(a,'D')==0 || Character.compare(a,'d')==0)

g="13";

else if(Character.compare(a,'E')==0 || Character.compare(a,'e')==0)

g="14";

else if(Character.compare(a,'F')==0 || Character.compare(a,'f')==0)

g="15";

else

g+=a;

if(Character.compare(b,'A')==0 || Character.compare(b,'a')==0)

f="10";

else if(Character.compare(b,'B')==0 || Character.compare(b,'b')==0)

f="11";

else if(Character.compare(b,'C')==0 || Character.compare(b,'c')==0)

f="12";

else if(Character.compare(b,'D')==0 || Character.compare(b,'d')==0)

f="13";

else if(Character.compare(b,'E')==0 || Character.compare(b,'e')==0)

f="14";

else if(Character.compare(b,'F')==0 || Character.compare(b,'f')==0)

f="15";

else

//a+=g;

f+=b;

//System.out. println("A "+g+" "+f);

//if(box==1)

output+=sbox[Integer.parseInt(g)][Integer.parseInt(f)];

//else if(box==2)

// output+=lbox[Integer.parseInt(g)][Integer.parseInt(f)];

//else

// output+=ebox[Integer.parseInt(g)][Integer.parseInt(f)];

return output;

}

String permuta(char a,char b,int c)

{

//System.out.print(a+" "+b+" ");

String output="",g="",f="";

if(c!=2)

{

//System.out.print("fgf");

if(c==0)

{

//g="0";

//f="0";

output+=sbox[0][0];

}

else

{

//g="0";

if(Character.compare(b,'A')==0 || Character.compare(b,'a')==0)

f="10";

else if(Character.compare(b,'B')==0 || Character.compare(b,'b')==0)

f="11";

else if(Character.compare(b,'C')==0 || Character.compare(b,'c')==0)

f="12";

else if(Character.compare(b,'D')==0 || Character.compare(b,'d')==0)

f="13";

else if(Character.compare(b,'E')==0 || Character.compare(b,'e')==0)

f="14";

else if(Character.compare(b,'F')==0 || Character.compare(b,'f')==0)

f="15";

else

f+=b;

output+=sbox[0][Integer.parseInt(f)];

}

}

else

{

if(Character.compare(a,'A')==0 || Character.compare(a,'a')==0)

g="10";

else if(Character.compare(a,'B')==0 || Character.compare(a,'b')==0)

g="11";

else if(Character.compare(a,'C')==0 || Character.compare(a,'c')==0)

g="12";

else if(Character.compare(a,'D')==0 || Character.compare(a,'d')==0)

g="13";

else if(Character.compare(a,'E')==0 || Character.compare(a,'e')==0)

g="14";

else if(Character.compare(a,'F')==0 || Character.compare(a,'f')==0)

g="15";

else

g+=a;

if(Character.compare(b,'A')==0 || Character.compare(b,'a')==0)

f="10";

else if(Character.compare(b,'B')==0 || Character.compare(b,'b')==0)

f="11";

else if(Character.compare(b,'C')==0 || Character.compare(b,'c')==0)

f="12";

else if(Character.compare(b,'D')==0 || Character.compare(b,'d')==0)

f="13";

else if(Character.compare(b,'E')==0 || Character.compare(b,'e')==0)

f="14";

else if(Character.compare(b,'F')==0 || Character.compare(b,'f')==0)

f="15";

else

//a+=g;

f+=b;

output+=sbox[Integer.parseInt(g)][Integer.parseInt(f)];

}

return output;

}

String invpermuta(char a,char b,int c)

{

//System.out.print(a+" "+b+" ");

String output="",g="",f="";

if(c!=2)

{

//System.out.print("fgf");

if(c==0)

{

//g="0";

//f="0";

output+=sbox[0][0];

}

else

{

//g="0";

if(Character.compare(b,'A')==0 || Character.compare(b,'a')==0)

f="10";

else if(Character.compare(b,'B')==0 || Character.compare(b,'b')==0)

f="11";

else if(Character.compare(b,'C')==0 || Character.compare(b,'c')==0)

f="12";

else if(Character.compare(b,'D')==0 || Character.compare(b,'d')==0)

f="13";

else if(Character.compare(b,'E')==0 || Character.compare(b,'e')==0)

f="14";

else if(Character.compare(b,'F')==0 || Character.compare(b,'f')==0)

f="15";

else

f+=b;

output+=sbox[0][Integer.parseInt(f)];

}

}

else

{

if(Character.compare(a,'A')==0 || Character.compare(a,'a')==0)

g="10";

else if(Character.compare(a,'B')==0 || Character.compare(a,'b')==0)

g="11";

else if(Character.compare(a,'C')==0 || Character.compare(a,'c')==0)

g="12";

else if(Character.compare(a,'D')==0 || Character.compare(a,'d')==0)

g="13";

else if(Character.compare(a,'E')==0 || Character.compare(a,'e')==0)

g="14";

else if(Character.compare(a,'F')==0 || Character.compare(a,'f')==0)

g="15";

else

g+=a;

if(Character.compare(b,'A')==0 || Character.compare(b,'a')==0)

f="10";

else if(Character.compare(b,'B')==0 || Character.compare(b,'b')==0)

f="11";

else if(Character.compare(b,'C')==0 || Character.compare(b,'c')==0)

f="12";

else if(Character.compare(b,'D')==0 || Character.compare(b,'d')==0)

f="13";

else if(Character.compare(b,'E')==0 || Character.compare(b,'e')==0)

f="14";

else if(Character.compare(b,'F')==0 || Character.compare(b,'f')==0)

f="15";

else

//a+=g;

f+=b;

output+=invsbox[Integer.parseInt(g)][Integer.parseInt(f)];

}

return output;

}

String[][] getKeys(String[] key)

{

String keys[][] = new String[12][16];

int i=0,j,k;

for(j=0;j<16;j++)

{

keys[0][j]=key[j];

}

for (i = 0; i < 10; i++)

{

k=0;

String[] word=new String[5];

for(j=12;j<16;j++)

{

word[k]=keys[i][j];

k++;

}

word=leftCircularShift(word,1);

k=0;

for(j=12;j<16;j++)

{

word[k]=permutation(word[k]);

k++;

}

String bin=hextoBin(word[0]);

String con=hextoBin(rcon[i]);

String res=xor(bin,con);

res=binToHex(res);

word[0]=res;

j=0;

for(k=0;k<4;k++)

{

keys[i+1][j]=binToHex(xor(hextoBin(word[k]),hextoBin(keys[i][j])));

j++;

}

for(k=0;k<4;k++)

{

keys[i+1][j]=binToHex(xor(hextoBin(keys[i+1][k]),hextoBin(keys[i][j])));

j++;

}

for(k=0;k<4;k++)

{

keys[i+1][j]=binToHex(xor(hextoBin(keys[i+1][k+4]),hextoBin(keys[i][j])));

j++;

}

for (k=0;k<4;k++)

{

keys[i+1][j]=binToHex(xor(hextoBin(keys[i+1][k+8]),hextoBin(keys[i][j])));

j++;

}

}

for(i=0;i<11;i++)

{

for (j=0;j<16;j++)

{

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

}

System.out.println(" ");

}

return keys;

}

String[] converttohex(String text)

{

String[] arr=new String[16];

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

{

char a=text.charAt(i);

int val=(int)a;

arr[i]=Integer.toHexString(val);

}

return arr;

}

String convertostr(String[][] pt)

{

int i,j;

String out="";

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

int val=Integer.parseInt(pt[j][i],16);

//System.out.print(val+" ");

char a=(char)val;

out+=a;

}

}

return out;

}

String[][] matxor(String[][] key,String[][] pt)

{

String[][] xk=new String[4][4];

//int k=0;

for(int i=0;i<4;i++)

{

for(int j=0;j<4;j++)

{

xk[i][j]=Integer.toHexString(Integer.parseInt(pt[i][j],16)^Integer.parseInt(key[i][j],16));

//k++;

}

}

return xk;

}

void encrypt(String plainText, String key)

{

int i,j,k=0;

// get round keys

String[] keyhex=converttohex(key);

String[] plainhex=converttohex(plainText);

String[][] keys = getKeys(keyhex);

String[][] pt=new String[4][4];

String[][] keynew=new String[4][4];

String[][] plain=new String[11][16];

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

{

plain[0][i]=plainhex[i];

}

System.out.println("Round state matrix");

//int l=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

pt[j][i]=plain[0][k];

//plain[0][k]=plainhex[k];

k++;

}

}

k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

keynew[j][i]=keys[0][k];

k++;

}

}

pt=matxor(keynew,pt);

for(int l=0;l<9;l++)

{

for(j=0;j<4;j++)

{

for (k=0;k<4;k++)

{

if(pt[j][k].length()<=1)

{

if(pt[j][k].equals("0"))

{

pt[j][k]=permuta('a','a',0);

}

else

{

pt[j][k]=permuta('a',pt[j][k].charAt(0),1);

//System.out.println(pt[j][k]+" ");

}

}

else

pt[j][k]=permuta(pt[j][k].charAt(0),pt[j][k].charAt(1),2);

}

pt[j]=leftCircularShift(pt[j],j);

}

pt=mix(pt);

k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

keynew[j][i]=keys[l+1][k];

k++;

}

}

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

pt[i][j]=Integer.toHexString(Integer.parseInt(pt[i][j],16)^(Integer.parseInt(keynew[i][j],16)));

}

//System.out.println(" ");

}

//k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

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

//plain[l+1][k]=pt[i][j];

//k++;

}

//System.out.println(" ");

}

System.out.println(" ");

}

for(j=0;j<4;j++)

{

for (k=0;k<4;k++)

{

if(pt[j][k].length()<=1)

{

if(pt[j][k].equals("0"))

{

pt[j][k]=permuta('a','a',0);

}

else

{

pt[j][k]=permuta('a',pt[j][k].charAt(0),1);

//System.out.println(pt[j][k]+" ");

}

}

else

pt[j][k]=permuta(pt[j][k].charAt(0),pt[j][k].charAt(1),2);

}

pt[j]=leftCircularShift(pt[j],j);

}

k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

keynew[j][i]=keys[10][k];

k++;

}

}

//}

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

pt[i][j]=Integer.toHexString(Integer.parseInt(pt[i][j],16)^Integer.parseInt(keynew[i][j],16));

}

//System.out.println(" ");

}

String cipher=convertostr(pt);

//k=0;

String output="";

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

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

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

//plain[9][k]=pt[i][j];

//k++;

}

//System.out.println(" ");

}

//System.out.println("\noutput"+cipher);

}

void decrypt(String key)

{

int i,j;

// get round keys

//String keys[] = getKeys(key);

String[] keyhex=converttohex(key);

String[] plainhex=new String[16];

Scanner scan=new Scanner(System.in);

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

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

{

plainhex[i]=scan.nextLine();

}

String[][] keys = getKeys(keyhex);

String[][] plain=new String[10][16];

String[][] pt=new String[4][4];

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

{

plain[0][i]=plainhex[i];

}

System.out.println("Round state matrix");

int k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

pt[j][i]=plain[0][k];

//plain[0][k]=plainhex[k];

k++;

}

}

String[][] keynew=new String[4][4];

k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

keynew[j][i]=keys[9][k];

k++;

}

}

pt=matxor(keynew,pt);

for(int l=0;l<9;l++)

{

for(j=0;j<4;j++)

{

pt[j]=invleftCircularShift(pt[j],j);

for (k=0;k<4;k++)

{

if(pt[j][k].length()<=1)

{

if(pt[j][k].equals("0"))

{

pt[j][k]=invpermuta('a','a',0);

}

else

{

pt[j][k]=invpermuta('a',pt[j][k].charAt(0),1);

//System.out.println(pt[j][k]+" ");

}

}

else

pt[j][k]=invpermuta(pt[j][k].charAt(0),pt[j][k].charAt(1),2);

}

//pt[j]=leftCircularShift(pt[j],j);

}

k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

keynew[j][i]=keys[8-l][k];

k++;

}

}

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

pt[i][j]=Integer.toHexString(Integer.parseInt(pt[i][j],16)^(Integer.parseInt(keynew[i][j],16)));

}

}

pt=invmix(pt);

//k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

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

//plain[l+1][k]=pt[i][j];

//k++;

}

//System.out.println(" ");

}

System.out.println(" ");

}

for(j=0;j<4;j++)

{

pt[j]=invleftCircularShift(pt[j],j);

for (k=0;k<4;k++)

{

if(pt[j][k].length()<=1)

{

if(pt[j][k].equals("0"))

{

pt[j][k]=invpermuta('a','a',0);

}

else

{

pt[j][k]=invpermuta('a',pt[j][k].charAt(0),1);

//System.out.println(pt[j][k]+" ");

}

}

else

pt[j][k]=invpermuta(pt[j][k].charAt(0),pt[j][k].charAt(1),2);

}

}

k=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

keynew[j][i]=keys[0][k];

k++;

}

}

//}

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

pt[i][j]=Integer.toHexString(Integer.parseInt(pt[i][j],16)^Integer.parseInt(keynew[i][j],16));

}

//System.out.println(" ");

}

//k=0;

String output="";

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

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

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

//plain[9][k]=pt[i][j];

//k++;

}

//System.out.println(" ");

}

System.out.println(" ");

}

}

public static void main(String args[])

{

String key,cipherText,plainText;

int choice;

do{

Scanner scn=new Scanner(System.in);

System.out.println("\nEnter choice 1)encrypt 2)Decrypt 3)exit");

choice=scn.nextInt();

switch(choice)

{

case 1:

{

Scanner sc=new Scanner(System.in);

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

plainText=sc.nextLine();

System.out.println("Enter Key:");

key=sc.nextLine();

AES cipher = new AES();

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

cipher.encrypt(plainText, key);

break;

}

case 2:

{

Scanner scan=new Scanner(System.in);

System.out.println("Enter Key:");

key=scan.nextLine();

AES cipher = new AES();

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

cipher.decrypt(key);

break;

}

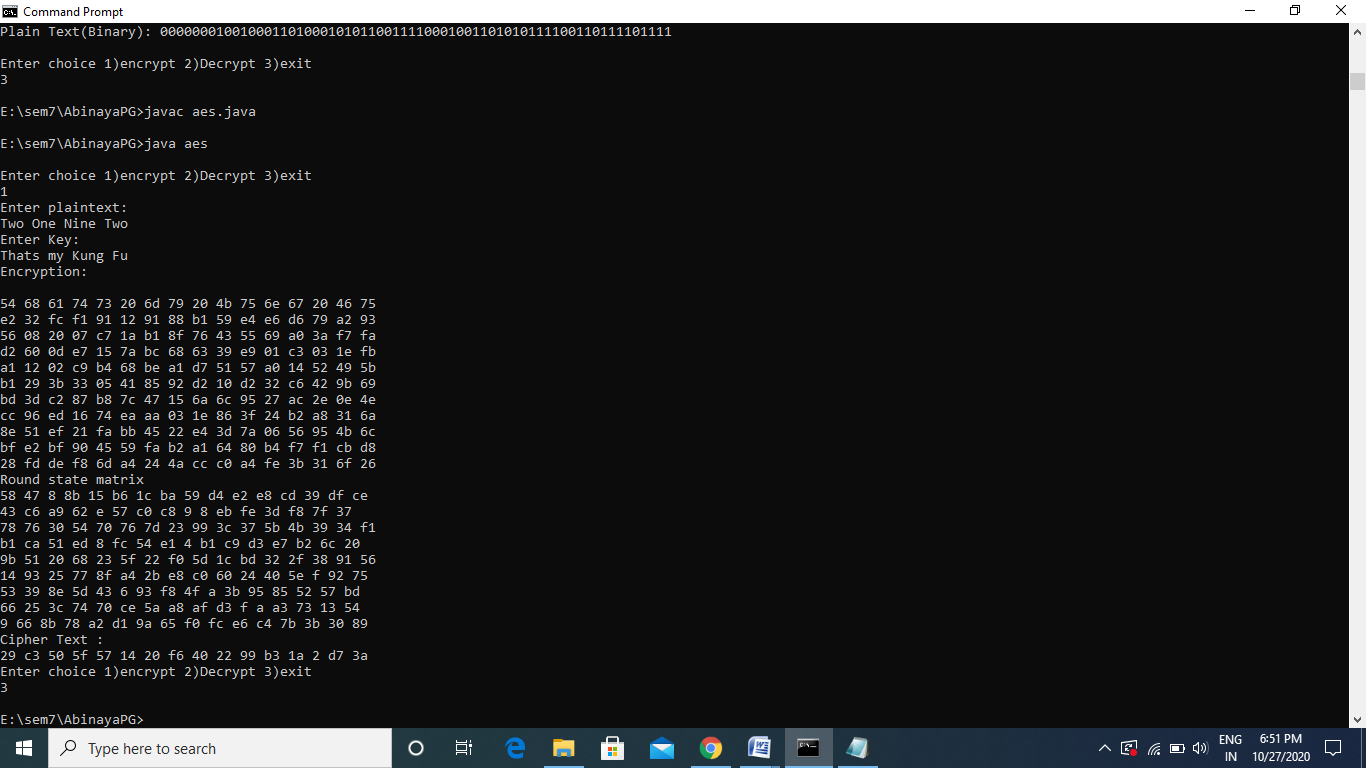
}

}while(choice!=3);

}

}

**Output:**



**Result:**

The above program to implement AES algorithm has been successfully executed.

Ex: 5 Rivest–Shamir–Adleman (RSA) Algorithm

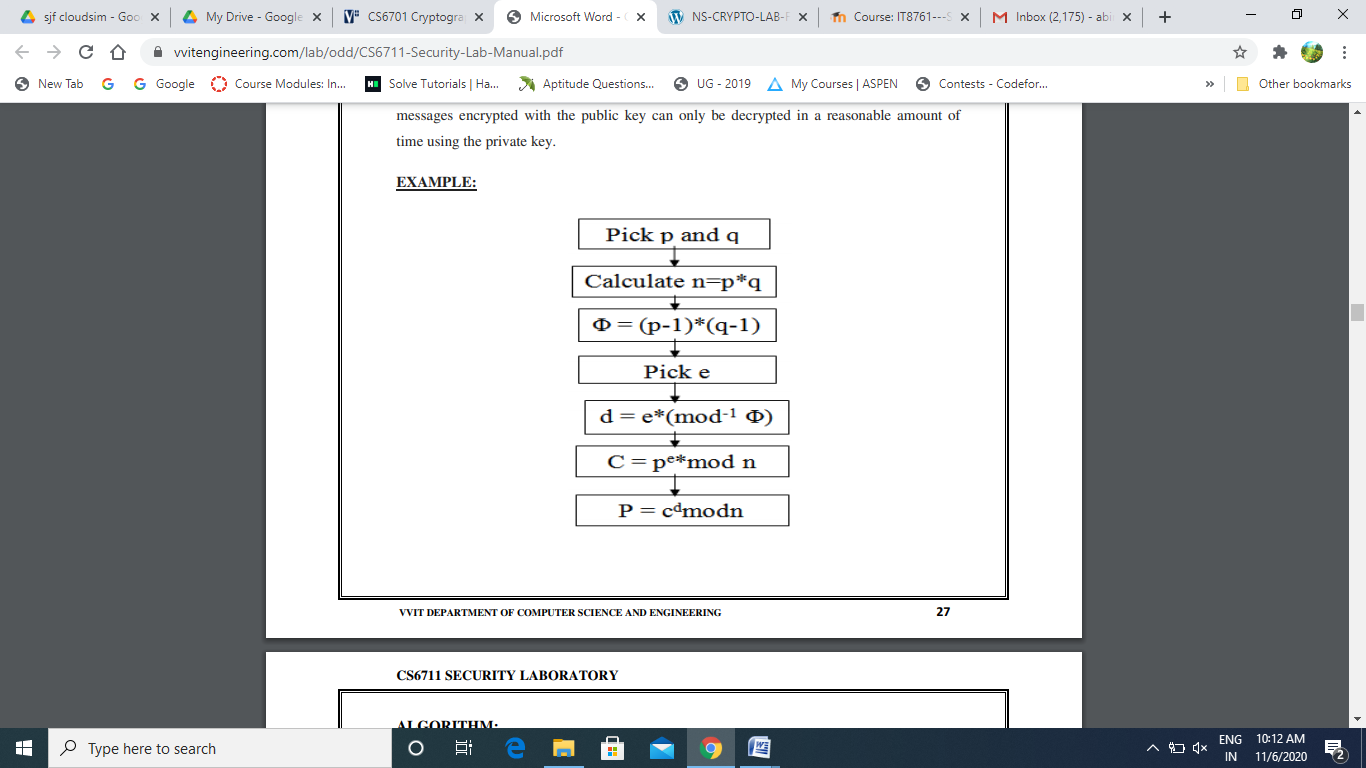
Date:22.09.2020

**Aim:**

To implement Rivest-Shamir-Adleman (RSA) Algorithm using HTML and Javascript.

**Description:**

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integer m: (me ) d = m (mod n) The public key is represented by the integers n and e; and, the private key, by the integer d. m represents the message. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.



**Algorithm:**

* Pick two large  primes numbers p and q.
* Calculate  n as n = pq and Φ(n) as Φ(n)=(p-1)(q-1).
* Choose a relatively small integer d such that GCD(d,Φ(n))=1
* Find e, the multiplicative inverse of d mod Φ(n)
* (d,n) is the public key and (e,n) is the private key.
* To encrypt M, compute  Encrypt(M) = M\*e (mod n) .
* To decrypt C, compute Decrypt(C) = Cd (mod n).

**Program:**

<html>

<head>

<title>Input</title>

<script language="JavaScript">

function gcd (a, b){

var r;

while (b>0){

r=a%b;

a=b;

b=r;}

return a;}

function rel\_prime(phi){

var rel=5;

var i;

for(i=2;i<phi;i++){

if(gcd(i,phi)==1){

return i;} }}

function power(a, b){

var temp=1, i;

for(i=1;i<=b;i++)

temp\*=a;

return temp;}

function encrypt(N, e, M){

var r,i=0,prod=1,rem\_mod=0;

while (e>0){

r=e % 2;

if (i++==0)

rem\_mod=M % N;

else

rem\_mod=power(rem\_mod,2) % N;

if (r==1){

prod\*=rem\_mod;

prod=prod % N;}

e=parseInt(e/2); }

return prod;}

function calculate\_d(phi,e){

var x,y,x1,x2,y1,y2,temp,r,orig\_phi;

orig\_phi=phi;

x2=1;x1=0;y2=0;y1=1;

while (e>0){

temp=parseInt(phi/e);

r=phi-temp\*e;

x=x2-temp\*x1;

y=y2-temp\*y1;

phi=e;e=r;

x2=x1;x1=x;

y2=y1;y1=y;

if (phi==1){

y2+=orig\_phi;

break;}}

return y2;}

function decrypt(c, d, N){

var r,i=0,prod=1,rem\_mod=0;

while (d>0){

r=d % 2;

if (i++==0)

rem\_mod=c % N;

else

rem\_mod=power(rem\_mod,2) % N;

if (r==1){

prod\*=rem\_mod;

prod=prod % N;}

d=parseInt(d/2);}

return prod;}

function openNew(){

//var subWindow=window.open("output.html", "Obj","HEIGHT=400,WIDTH=600,SCROLLBARS=YES");

var p=parseInt(document.Input.p.value);

var q=parseInt(document.Input.q.value);

var M=parseInt(document.Input.M.value);

var N=p \* q;

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

var e=rel\_prime(phi);

var c=encrypt(N,e,M);

var d=calculate\_d(phi,e);

var x=decrypt(c,d,N);

window.alert("Encrypted Text : " +parseInt(c));

}

function openn(){

//var subWindow=window.open("output.html", "Obj","HEIGHT=400,WIDTH=600,SCROLLBARS=YES");

var p=parseInt(document.Input.p.value);

var q=parseInt(document.Input.q.value);

var M=parseInt(document.Input.M.value);

var N=p \* q;

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

var e=rel\_prime(phi);

// var c=encrypt(N,e,M);

var d=calculate\_d(phi,e);

var x=decrypt(M,d,N);

window.alert("Decrypted Text : " +parseInt(x));}

// end scripting here -->

</script>

</head>

<body>

<p><font size="6">Input Form</font></p>

<hr>

<form name="Input">

<table border="0" width="100%" height="109">

<tr>

<td width="24%" height="23">

<font color="#0000FF">Enter P</font></td>

<td width="76%" height="23">

<input type="text" name="p" size="20"></td>

</tr>

<tr>

<td width="24%" height="23"><font color="#0000FF">

Enter Q</font></td>

<td width="76%" height="23">

<input type="text" name="q" size="20"></td>

</tr>

<tr>

<td width="24%" height="20">

<font color="#0000FF">Enter any Number ( M )</font></td>

<td width="76%" height="20"><input type="text" name="M" size="20">

<font size="1" color="#FF0000">(1-1000)</font></td>

</tr>

<tr>

<td width="24%" height="19"><input type="button"

value="encrypt" name="encrypt" onClick="openNew()"></td>

<td width="76%" height="19"><input type="button"

value="decrypt" name="decrypt" onClick="openn()"></td>

</tr>

</table>

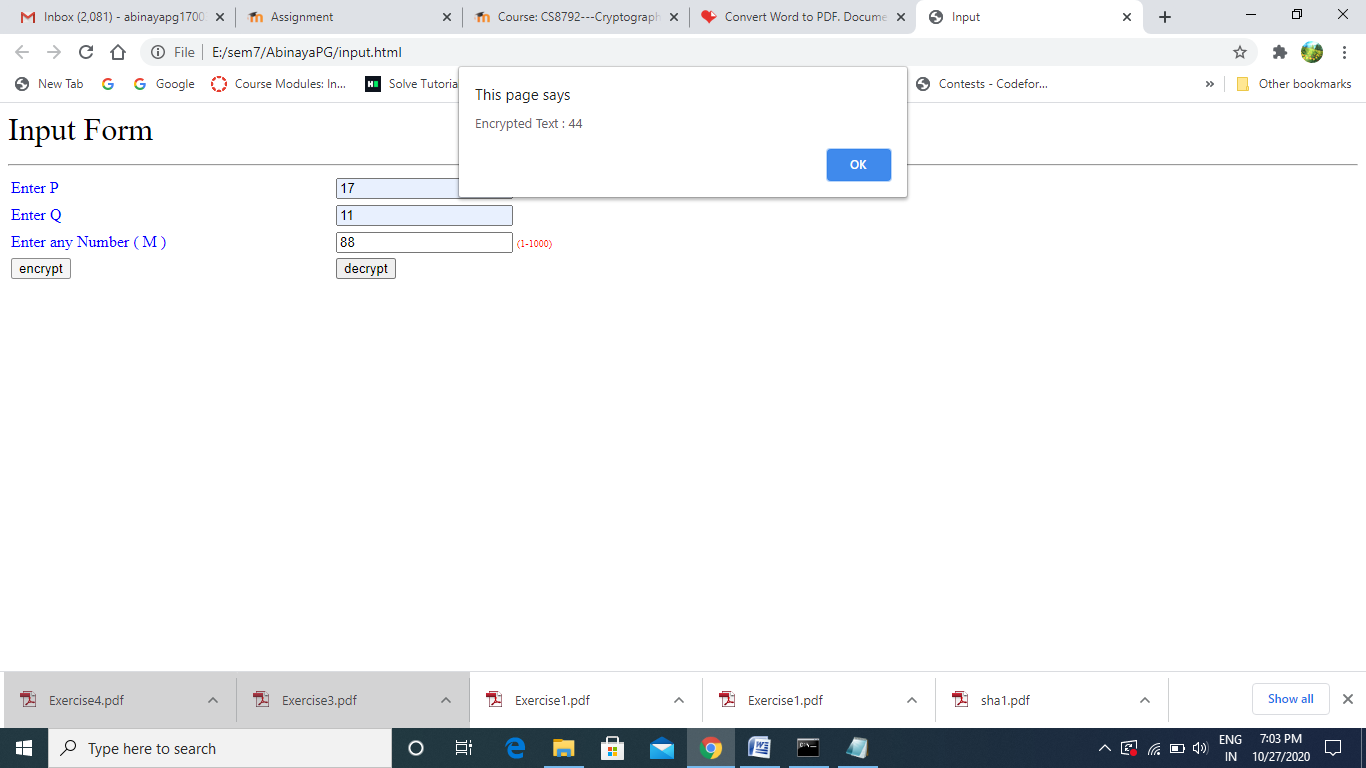
</form>

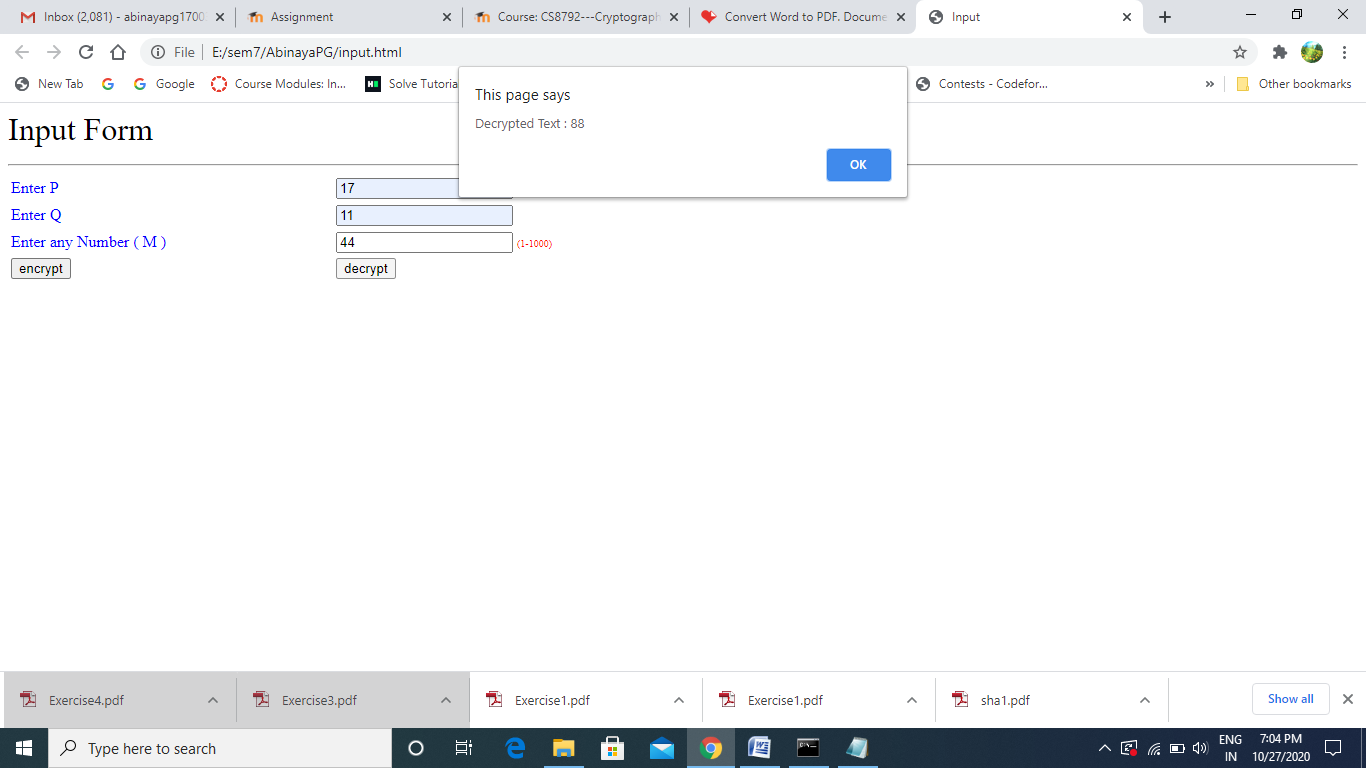
<p>&nbsp;</p>

</body>

</html>

**Output:**





**Result:**

The above program to implement RSA Algorithm has been successfully executed.

Ex: 6 Key Exchange Algorithms

## Date: 29.09.2020

## Aim:

To implement Diffie-Hellman and ElGamal Algorithm in java.

## Diffie-Hellman Algorithm

## Description:

## Diffie–Hellman Key Exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network. It is primarily used as a method of exchanging cryptography keys for use in symmetric encryption algorithms like AES. The algorithm in itself is very simple. The process begins by having the two parties, Alice and Bob. Let's assume that Alice wants to establish a shared secret with Bob.

## 

## Algorithm:

## Both Alice and Bob shares the same public keys g and p.

## Alice selects a random public key a.

## Alice computes his secret key A as ga mod p.

## Then Alice sends A to Bob

## Similarly Bob also selects a public key b and computes his secret key as B and sends the same back to Alice.

## Now both of them compute their common secret key as the other one’s secret key power of a mod p.

**Program:**

import java.util.\*;

import java.lang.\*;

import java.math.\*;

public class diffie

{

public static BigInteger power(BigInteger a,int b,BigInteger p)

{

if(b==1)

return a;

else

return (a.pow(b)).mod(p);

}

public static void keygen()

{

Scanner sc=new Scanner(System.in);

int b,i,a;

BigInteger x,y,ka,kb,g,p;

System.out.println("Enter p");

p=sc.nextBigInteger();

System.out.println("Enter g");

g=sc.nextBigInteger();

System.out.println("Enter a");

a=sc.nextInt();

x=power(g,a,p);

System.out.println("Enter b");

b=sc.nextInt();

y=power(g,b,p);

ka=power(y,a,p);

System.out.println("Alice:");

System.out.println("Private Key : "+a);

System.out.println("Computed public key : "+x);

System.out.println("Shared Secret : "+ka);

kb=power(x,b,p);

System.out.println("Bob:");

System.out.println("Private Key : "+b);

System.out.println("Computed public key : "+y);

System.out.println("Shared Secret : "+kb);

}

public static void main(String[] args)

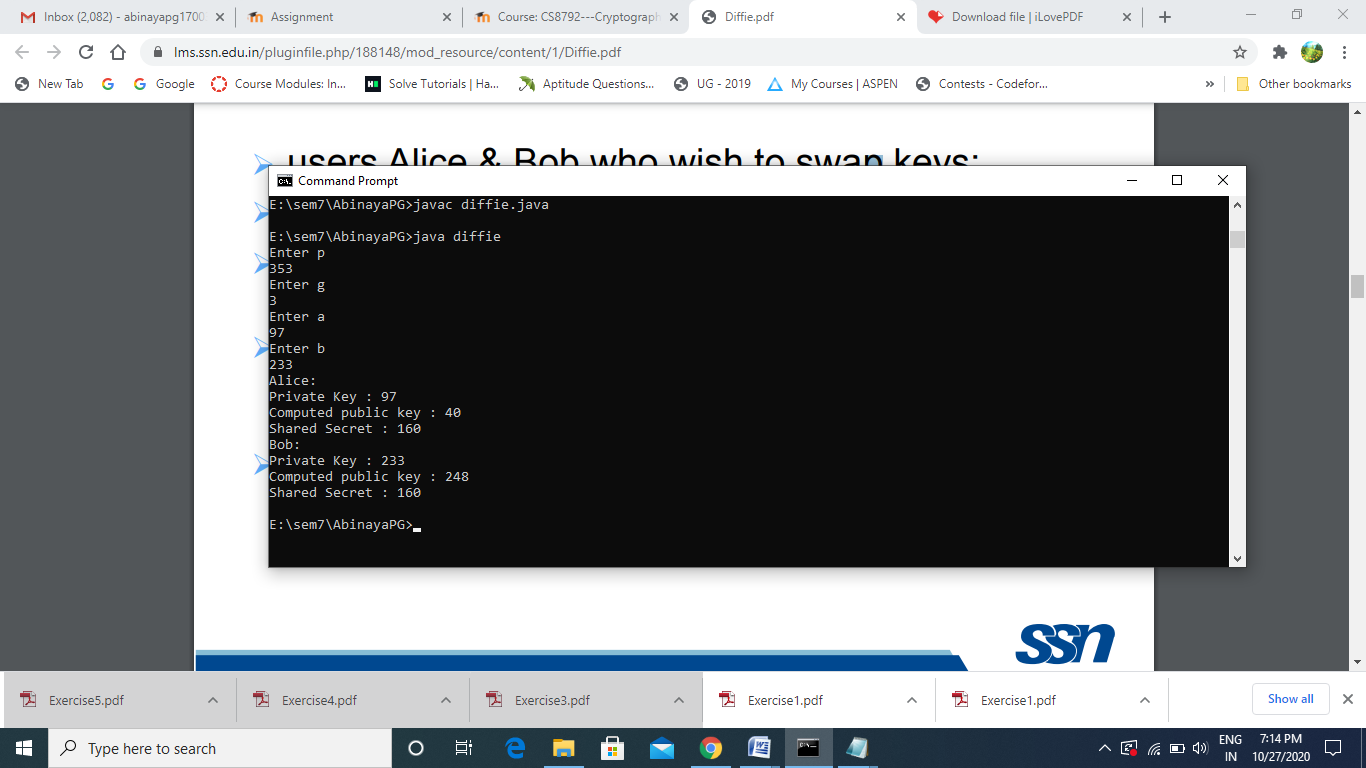
{

keygen();

}

}

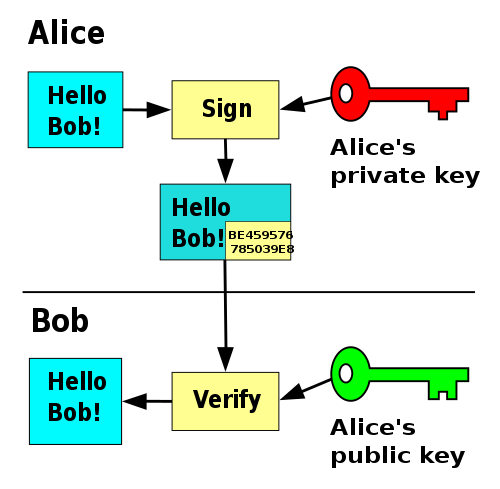
**Output:**



1. **ElGamal Algorithm**

**Description:**

**ElGamal encryption**is an public-key cryptosystem. It uses asymmetric key encryption for communicating between two parties and encrypting the message.  
This cryptosystem is based on the difficulty of finding **discrete logarithm** in a cyclic group that is even if we know ga and gk, it is extremely difficult to compute gak.



**Algorithm:**

1. Bob generates public and private key :
   * Bob chooses a very large number **q** and a cyclic group **F**q.
   * From the cyclic group **F**q, he choose any element **g** and  
     an element **a** such that gcd(a, q) = 1.
   * Then he computes h = ga.
   * Bob publishes **F**, **h = ga**, **q** and **g** as his public key and retains **a** as private key.
2. Alice encrypts data using Bob’s public key :
   * Alice selects an element **k** from cyclic group **F**  
     such that gcd(k, q) = 1.
   * Then she computes p = gk and s = hk = gak.
   * She multiples s with M.
   * Then she sends (p, M\*s) = (gk, M\*s).
3. Bob decrypts the message :
   * Bob calculates s′ = pa = gak.
   * He divides M\*s by s′ to obtain M as s = s′.

**Program:**

import java.util.\*;

import java.lang.\*;

import java.math.\*;

public class elgamal

{

public static BigInteger power(BigInteger a,int b,BigInteger p)

{

if(b==1)

return a;

else

return (a.pow(b)).mod(p);

}

public static void keygen()

{

Scanner sc=new Scanner(System.in);

int b,i,xa;

BigInteger x,y,ka,kb,q,a;

System.out.println("Enter q");

q=sc.nextBigInteger();

System.out.println("Enter a");

a=sc.nextBigInteger();

System.out.println("Enter xa");

xa=sc.nextInt();

x=power(a,xa,q);

System.out.println("Enter k");

b=sc.nextInt();

y=power(x,b,q);

BigInteger c1,c2,m;

c1=power(a,b,q);

System.out.println("Enter M");

m=sc.nextBigInteger();

c2=(m.multiply(y)).mod(q);

System.out.println("Alice:");

System.out.println("K : "+y);

System.out.println("C1 : "+c1);

System.out.println("C2 : "+c2);

ka=power(c1,xa,q);

kb=ka.modInverse(q);

BigInteger m2;

m2=(c2.multiply(kb)).mod(q);

System.out.println("Bob:");

System.out.println("K : "+ka);

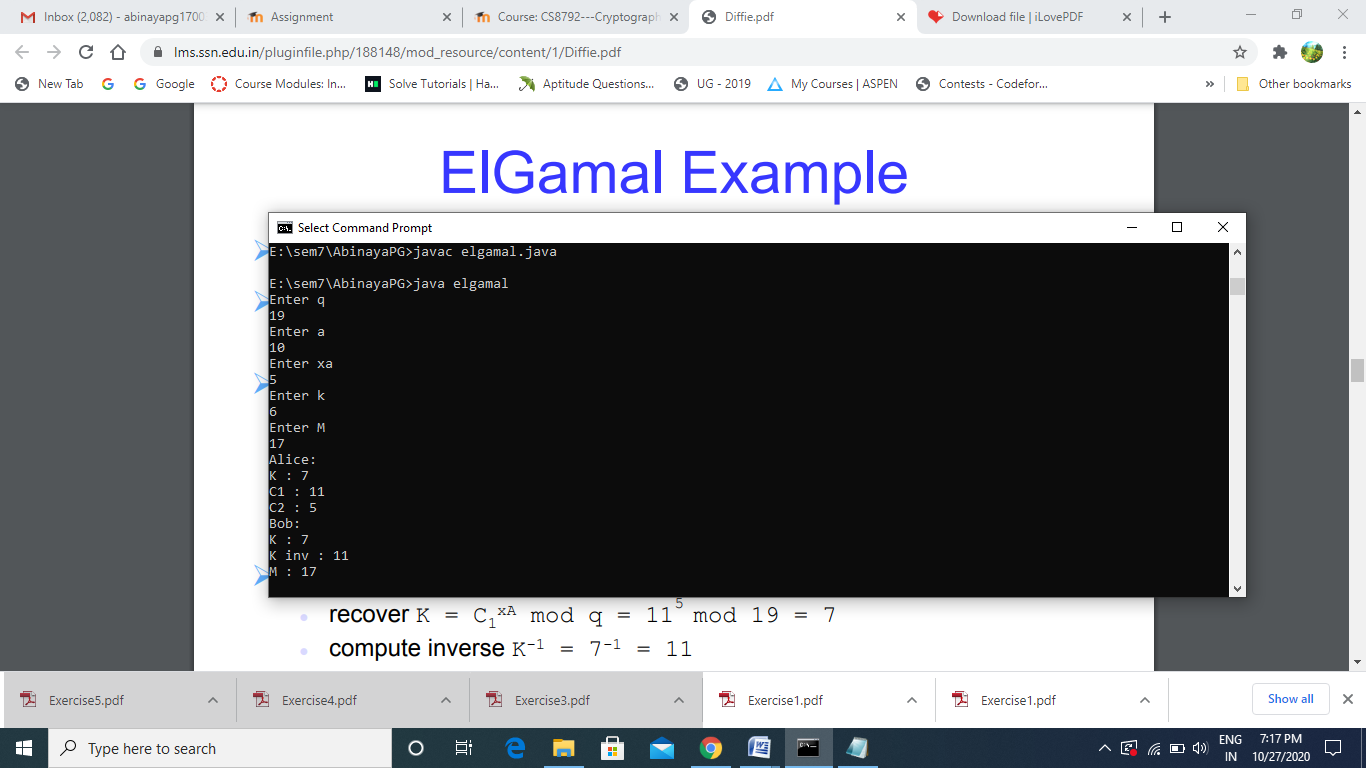
System.out.println("K inv : "+kb);

System.out.println("M : "+m2);}

public static void main(String[] args){

keygen();}}

**Output:**



**Result:**

The above programs to implement Diffie-Hellman Key Exchange Algorithm and ElGamal Algorithm have been successfully executed.

## Ex: 7 SECURE HASH ALGORITHM (SHA1)

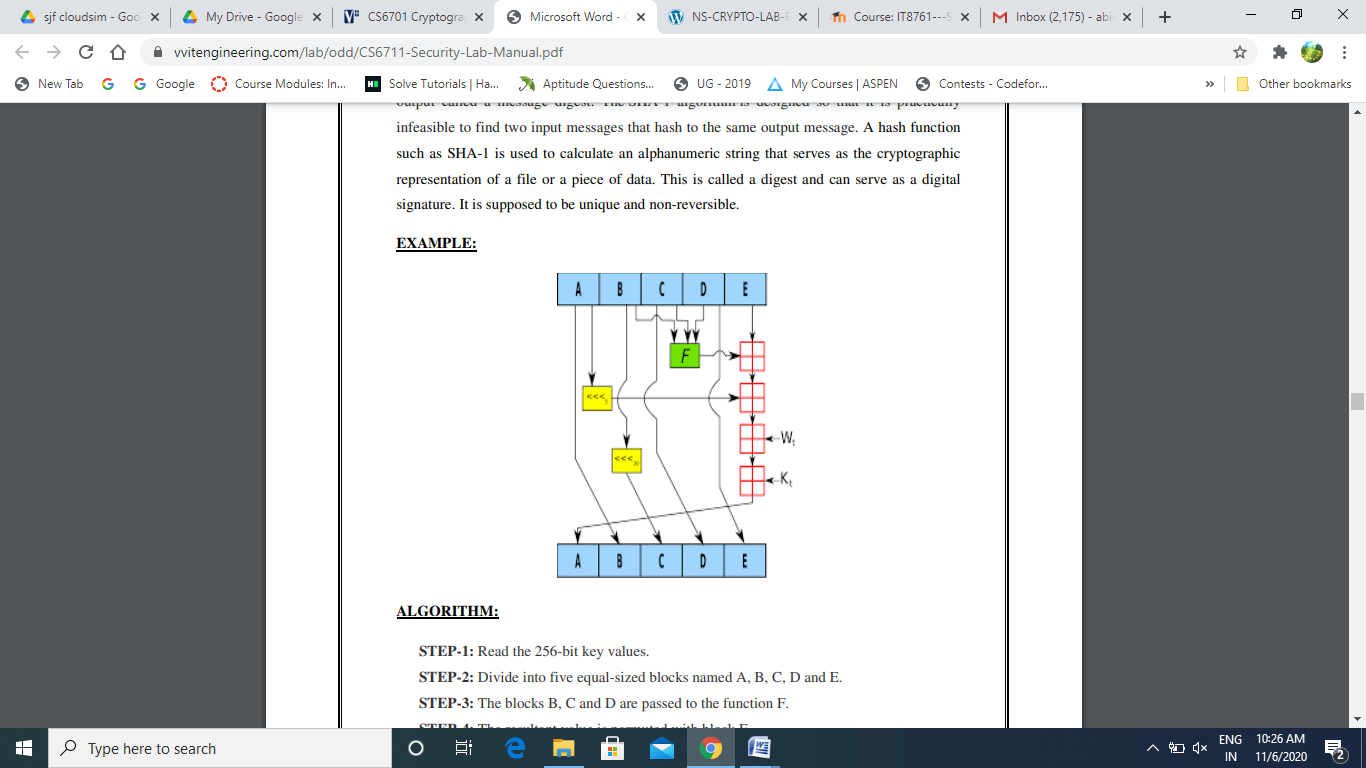
## Date: 06.10.2020

**Aim:**

To implement Secure hash algorithm (SHA1) using java.

**Description:**

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function. SHA-1 produces a 160-bit hash value known as a message digest. The way this algorithm works is that for a message of size < 264 bits it computes a 160-bit condensed output called a message digest. The SHA-1 algorithm is designed so that it is practically infeasible to find two input messages that hash to the same output message. A hash function such as SHA-1 is used to calculate an alphanumeric string that serves as the cryptographic representation of a file or a piece of data. This is called a digest and can serve as a digital signature. It is supposed to be unique and non-reversible.



**Algorithm:**

* Append Padding Bits
* Append Length - 64 bits are appended to the end
* Prepare Processing Functions
* Prepare Processing Constants
* Initialize Buffers
  1. h0=0x67452301,
* h1=0xEFCDAB89,
* h2=0x98BADCFE,
* h3=0x10325476,
* h4=0xC3D2E1F0
* Processing Message in 512-bit blocks (L blocks in total message)
* Perform message schedule W:
  1. For t=0 to t=15:
     1. W[t]=M[t]
  2. For t=16 to t=79:
     1. W[t]= (W[t-3] xor W[t-8] xor W[t-14] xor W[t-16]) left rotate 1
* Initialize variables a=h0,b=h1,c=h2,d=h3,e=h4.
* Perform the main hash computation
  1. For t=0 to t=79:
     1. T = ( a left rotate 5) + f + e + k + W[t]
     2. e = d
     3. d = c
     4. c = b left rotate 30
     5. b = a
     6. a = T
* Compute the intermediate hash value
  1. h0= a + h0,
* h1= b + h1,
* h2= c + h2,
* h3= d + h3,
* h4= e + h4
* Finally print the message digest

**Program:**

## import java.util.Scanner;

## public class sha {

## public static int messLength = 0;

## public static void main(String[] args) {

## //Getting the word

## System.out.println("Insert a word a phrase to be hashed");

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

## String word = sc.nextLine();

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

## //Converting the word to binary

## String binary = convertToBinary(word);

## messLength = binary.length();

## calculateMod(word, binary);

## }

## public static String convertToBinary(String word) {

## byte[] bytes = word.getBytes();

## StringBuilder binary = new StringBuilder();

## for (byte b : bytes) {

## int val = b;

## for (int i = 0; i < 8; i++) {

## binary.append((val & 128) == 0 ? 0 : 1);

## val <<= 1;

## }

## binary.append(' ');

## }

## return binary.toString();

## }

## public static void calculateMod(String word, String binary) {

## int binaryMessageLength = word.length() \* 8 - 8; //the -8 will be taken into account below.

## String endBitLength = calculateMessageLength(binaryMessageLength+8); //add back 8 for accuracy

## int subMod = endBitLength.length();

## int temp = (binaryMessageLength) % 512;

## if (432 - temp < 0) {

## int x = 512 - temp;

## temp = x + 440 + temp + 64;

## } else {

## temp = 432 - temp;

## }

## int binaryZeros = temp;

## String onePadded = "10000000"; //add back the removed 8

## binary = binary.replaceAll("\\s+", ""); //remove spaces

## createMessageLength(binary, onePadded, binaryZeros, endBitLength); //creates the 512 bit message

## }

## public static String calculateMessageLength(int bitLength) {

## String tempBitsLength = Integer.toBinaryString(bitLength);

## StringBuilder sb = new StringBuilder(tempBitsLength);

## int temp = 64 - tempBitsLength.length();

## while (temp > 0) {

## sb.insert(0, 0);

## temp--;

## }

## return sb.toString();

## }

## //create complete message

## public static String createMessageLength(String message, String paddedOne, int zeros, String endLength) {

## StringBuilder messageBinary = new StringBuilder(message);

## messageBinary.insert(messageBinary.toString().length(), paddedOne);

## while (zeros > 0) {

## messageBinary.insert(messageBinary.toString().length(), 0);

## zeros--;

## }

## messageBinary.insert(messageBinary.toString().length(), endLength);

## String m = printMessage(messageBinary.toString());

## m = m.replaceAll("\\s+", "");

## int[] mArray = new int[m.toString().length()/32];

## for (int i = 0; i < m.toString().length(); i+=32) {

## mArray[i/32] = Integer.valueOf(m.substring(i+1, i+32),2);

## if(m.charAt(i) == '1'){

## mArray[i/32] |= 0X80000000;

## }

## // System.out.printf("Decimal(iterator), String(Binary), Hex values of input: %d %s %x\n", i, m.substring(i, i+32),mArray[i/32]);

## }

## hash(mArray);

## return messageBinary.toString();

## }

## public static String printMessage(String message) {

## StringBuilder sb = new StringBuilder(message);

## int num = message.length();

## while (num > 0) {

## if (num % 32 == 0) {

## sb.insert(num, " ");

## }

## num--;

## }

## return sb.toString();

## }

## private static int leftrotate(int x, int shift) { //leftrotate function

## return ((x << shift) | (x >>> (32 - shift))); //>>> is an UNSIGNED shift compared >> which is not

## }

## //instance variables

## private static int h1 = 0x67452301;

## private static int h2 = 0xEFCDAB89;

## private static int h3 = 0x98BADCFE;

## private static int h4 = 0x10325476;

## private static int h5 = 0xC3D2E1F0;

## private static int k1 = 0x5A827999;

## private static int k2 = 0x6ED9EBA1;

## private static int k3 = 0x8F1BBCDC;

## private static int k4 = 0xCA62C1D6;

## private static String hash(int[] z) {

## //Extend the sixteen 32-bit words into eighty 32-bit words

## int integer\_count = z.length;

## int[] intArray = new int[80];

## int j = 0;

## for(int i = 0; i < integer\_count; i += 16)

## {

## for(j = 0; j <= 15; j++)

## intArray[j] = z[j+i];

## for ( j = 16; j <= 79; j++ ) {

## //w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1

## intArray[j] = leftrotate(intArray[j - 3] ^ intArray[j - 8] ^ intArray[j - 14] ^ intArray[j - 16], 1);

## }

## int A = h1;

## int B = h2;

## int C = h3;

## int D = h4;

## int E = h5;

## int t = 0; //temp

## System.out.println("Round" +" A "+" B "+" C "+" D "+" E ");

## for ( int x = 0; x <= 19; x++ ) {

## //temp = leftrotate(a leftrotate 5) + f(t) + e + w[i] + k

## t = leftrotate(A,5)+((B&C)|((~B)&D))+E+intArray[x]+k1;

## E=D; D=C; C=leftrotate(B,30); B=A; A=t;

## System.out.println(x+" "+Integer.toHexString(A)+" "+Integer.toHexString(B)+" "+Integer.toHexString(C)+" "+Integer.toHexString(D)+" "+Integer.toHexString(E));

## }

## for ( int b = 20; b <= 39; b++ ) {

## t = leftrotate(A,5)+(B^C^D)+E+intArray[b]+k2;

## E=D; D=C; C=leftrotate(B,30); B=A; A=t;

## System.out.println(b+" "+Integer.toHexString(A)+" "+Integer.toHexString(B)+" "+Integer.toHexString(C)+" "+Integer.toHexString(D)+" "+Integer.toHexString(E));

## }

## for (int c = 40; c <= 59; c++ ) {

## t = leftrotate(A,5)+((B&C)|(B&D)|(C&D))+E+intArray[c]+k3;

## E=D; D=C; C=leftrotate(B,30); B=A; A=t;

## System.out.println(c+" "+Integer.toHexString(A)+" "+Integer.toHexString(B)+" "+Integer.toHexString(C)+" "+Integer.toHexString(D)+" "+Integer.toHexString(E));

## }

## for ( int d = 60; d <= 79; d++ ) {

## t = leftrotate(A,5)+(B^C^D)+E+intArray[d]+k4;

## E=D; D=C; C=leftrotate(B,30); B=A; A=t;

## System.out.println(d+" "+Integer.toHexString(A)+" "+Integer.toHexString(B)+" "+Integer.toHexString(C)+" "+Integer.toHexString(D)+" "+Integer.toHexString(E));

## }

## h1+=A; h2+=B; h3+=C; h4+=D; h5+=E;

## }

## String h1Length = Integer.toHexString(h1);

## String h2Length = Integer.toHexString(h2);

## String h3Length = Integer.toHexString(h3);

## String h4Length = Integer.toHexString(h4);

## String h5Length = Integer.toHexString(h5);

## if(h1Length.length() < 8) {

## StringBuilder h1L = new StringBuilder(h1Length);

## h1L.insert(0,0);

## h1Length = h1L.toString();

## } else if(h2Length.length() < 8) {

## StringBuilder h2L = new StringBuilder(h2Length);

## h2L.insert(0,0);

## h2Length = h2L.toString();

## } else if(h3Length.length() < 8) {

## StringBuilder h3L = new StringBuilder(h3Length);

## h3L.insert(0,0);

## h3Length = h3L.toString();

## } else if(h4Length.length() < 8) {

## StringBuilder h4L = new StringBuilder(h4Length);

## h4L.insert(0,0);

## h4Length = h4L.toString();

## } else if(h5Length.length() < 8) {

## StringBuilder h5L = new StringBuilder(h5Length);

## h5L.insert(0,0);

## h5Length = h5L.toString();}

## String hh = h1Length + h2Length + h3Length + h4Length + h5Length;

## System.out.println("Message Digest : " + hh);

## return null; }}

## Output:

## 

## 

## 

**Result:**

The above program to implement secure hash algorithm (SHA1) has been successfully executed.

## Ex: 8 Digital Signature Algorithm

## Date: 13.10.2020

## Aim:

## To implement digital signature algorithms using java.

## 1.Elgamal Digital Signature

## Description:

## The ElGamal digital signature scheme stems from the ElGamal cryptosystem based upon the security of the one-way function of exponentiation in modular rings and the difficulty of solving the discrete logarithm problem.

## 

## Algorithm:

* Choose a large prime p and a primitive root α.
* Input the private key Xa such 1 < Xa < q-1.
* Compute public key Ya as yA = a xA mod q.
* Input the  hash message m= H(M), 0 <= m <= (q-1).
* Choose a random integer K with 1 <= K <= (q-1) and gcd(K,q-1)=1 .
* Compute temporary key: S1 = a k mod q  and compute K-1 the inverse of K mod (q-1).
* Compute the value: S2 = K-1(m-xAS1) mod (q-1) and  signature is:(S1,S2).
* Compute V1 as V1 = a\*m mod q and V2 as V2 = (yA^ S1)\* (S1^S2) mod q .
* Signature is valid if V1 = V2 else invalid.

## Program:

## import java.util.\*;

## import java.lang.\*;

## import java.math.\*;

## public class elgdigsig

## {

## public static BigInteger power(BigInteger a,int b,BigInteger p)

## {

## if(b==1)

## return a;

## else

## return (a.pow(b)).mod(p);

## }

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

## {

## Scanner scan=new Scanner(System.in);

## BigInteger q,a,ya,s1,s2,x,v1,v2;

## int m,k,xa;

## System.out.println("Enter prime number q");

## q=scan.nextBigInteger();

## System.out.println("Enter primitive root");

## a=scan.nextBigInteger();

## System.out.println("Enter Xa");

## xa=scan.nextInt();

## ya=power(a,xa,q);

## System.out.println("Enter K");

## k=scan.nextInt();

## System.out.println("Enter message");

## m=scan.nextInt();

## System.out.println("Public Key (ya) : "+ya);

## s1=power(a,k,q);

## System.out.println("S1 : "+s1);

## int i=1;

## while(true)

## {

## x=BigInteger.valueOf((k\*i)%(q.intValue()-1));

## if(x.intValue()==1)

## break;

## else

## i=i+1;

## }

## BigInteger check;

## Long l=new Long(1L);

## check=BigInteger.valueOf(l);

## s2=(BigInteger.valueOf(i).multiply(BigInteger.valueOf(m).subtract(BigInteger.valueOf(xa).multiply(s1)))).mod(q.subtract(check));

## System.out.println("S2 : "+s2);

## System.out.println("Signature (S1,S2) : ("+s1+","+s2+")" );

## v1=power(a,m,q);

## v2=(ya.pow(s1.intValue()).multiply(s1.pow(s2.intValue()))).mod(q);

## System.out.println("V1 : "+v1+" V2 : "+v2);

## System.out.println("Verification");

## if(v1.compareTo(v2)==0)

## System.out.println("Signature is Valid");

## else

## System.out.println("Signature is invalid");

## }

## }

## Output:

## 

**2.Digital Signature Algorithm**

## Description:

## Like the ElGamal scheme DSA is a digital signature scheme with an appendix meaning that the message cannot be easily recovered from the signature itself.When using DSA it is usually a hash of the message that is signed. A hash is an encoding that reduces the size of a message to a fixed bit length by using a specific encoding. Let the hash of a message m be h(m) = x and let x be a 160-bit message.

## Algorithm:

* Input a  prime number q.
* Input a large prime p with 2^L-1 < p < 2^L,where L= 512 to 1024 bits &  is a multiple of 64.
* Input the primitive root choose g such that g = h^((p-1)/q).
* Input the private key X and compute public key Y as Y = (g\*x) mod p.
* Input a random integer k ,such k<q.
* Input the hash value m of the message (H(M))..
* Compute r as r = (g\*k mod p)mod q  and s as s = [k-1\*(H(M)+ xr)] mod q.
* Compute :
  1. w as w = s-1 mod q ,
  2. u1 as u1= [H(M)w ]mod q
  3. u2 as u2= (r\*w)mod q
  4. v as v = [(g^u1\* y^u2)mod p ]mod q.
* The signature is valid if v==r else invalid.

## Program:

## import java.util.\*;

## import java.lang.\*;

## import java.math.\*;

## public class dsadigsig

## {

## public static BigInteger power(BigInteger a,int b,BigInteger p)

## {

## if(b==1)

## return a;

## else

## return (a.pow(b)).mod(p);

## }

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

## {

## Scanner scan=new Scanner(System.in);

## BigInteger q,p,g,ya;

## System.out.println("Enter prime integer q");

## q=scan.nextBigInteger();

## System.out.println("Enter prime integer p");

## p=scan.nextBigInteger();

## System.out.println("Enter primitive root g");

## g=scan.nextBigInteger();

## int xa;

## System.out.println("Enter xa ");

## xa=scan.nextInt();

## ya=power(g,xa,p);

## int k;

## BigInteger m;

## System.out.println("Enter k");

## k=scan.nextInt();

## System.out.println("Enter hash value ");

## m=scan.nextBigInteger();

## System.out.println("Public Key (ya) : "+ya);

## BigInteger r,x;

## r=power(g,k,p).mod(q);

## System.out.println("r: "+ r);

## int i=1;

## while(true)

## {

## x=(BigInteger.valueOf(k).multiply(BigInteger.valueOf(i))).mod(q);

## if(x.intValue()==1)

## break;

## else

## i=i+1;

## }

## BigInteger s;

## s=(BigInteger.valueOf(i).multiply(m.add(BigInteger.valueOf(xa).multiply(r)))).mod(q);

## System.out.println("s: "+s);

## System.out.println("Signature (r,s) : ("+r+","+s+")");

## i=1;

## while(true)

## {

## x=(s.multiply(BigInteger.valueOf(i))).mod(q);

## if(x.intValue()==1)

## break;

## else

## i=i+1;

## }

## BigInteger w,u1,u2,v;

## w=BigInteger.valueOf(i).mod(q);

## u1=(m.multiply(w)).mod(q);

## u2=(r.multiply(w)).mod(q);

## v=(((g.pow(u1.intValue())).multiply(ya.pow(u2.intValue()))).mod(p)).mod(q);

## System.out.println("W :" +w);

## System.out.println("U1 :"+u1);

## System.out.println("U2 :"+u2);

## System.out.println("V : "+v);

## System.out.println("Verification");

## if(v.compareTo(r)==0)

## System.out.println("Signature is valid");

## else

## System.out.println("Signature is invalid");

## }

## }

## Output:

## 

**Result:**

The above programs to implement Digital Signature Algorithms have been successfully executed.

## Ex: 10 [Intrusion Detection System using SNORT](https://lms.ssn.edu.in/mod/assign/view.php?id=85535)

## Date: 20.10.2020

## Aim:

## To demonstrate intrusion detection system using SNORT.

## Description:

## INTRUSION DETECTION SYSTEM: Intrusion detection is a set of techniques and methods that are used to detect suspicious activity both at the network and host level. Intrusion detection systems fall into two basic categories:

## Signature-based intrusion detection systems

## Anomaly detection systems.

## Intruders have signatures, like computer viruses, that can be detected using software. You try to find data packets that contain any known intrusion-related signatures or anomalies related to Internet protocols. Based upon a set of signatures and rules, the detection system is able to find and log suspicious activity and generate alerts. Anomaly-based intrusion detection usually depends on packet anomalies present in protocol header parts. In some cases these methods produce better results compared to signature-based IDS. Usually an intrusion detection system captures data from the network and applies its rules to that data or detects anomalies in it. Snort is primarily a rule-based IDS, however input plug-ins are present to detect anomalies in protocol headers.

## SNORT TOOL:

## Snort is based on libpcap (for library packet capture), a tool that is widely used in TCP/IPtraffic sniffers and analyzers. Through protocolanalysis and content searching and matching, Snort detects attack methods, including denial of service, buffer overflow, CGI attacks, stealthport scans, and SMB probes. When suspicious behavior is detected, Snort sends a real-time alert to syslog, a separate 'alerts' file, or to apop-up window. Snort is currently the most popular free network intrusion detection software. The advantages of Snort are numerous. According to the snort web site, “It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflow, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more” (Caswell). One of the advantages of Snort is its ease of configuration. Rules are very flexible, easily written, and easily inserted into the rule base. If a new exploit or attack is found a rule for the attack can be added to the rule base in a matter of seconds. Another advantage of snort is that it allows for raw packet data analysis.

## 

## *snort –d c:\log –h ipaddress/24 –c snort.conf*

## This is a configuration file applies rule to each packet to decide it an action based upon the rule type in the file.

## Algorithm:

## Download SNORT from snort.org. Install snort

## Download Rules.

## Install WinPCAP.

## Extract the Rules file. You will need WinRAR for the .gz file.

## Copy all files from the “rules” folder of the extracted folder. Now paste the

## rules into “C:\Snort\rules” folder.

## Copy “snort.conf” file from the “etc” folder of the extracted folder. You must paste it into “C:\Snort\etc” folder. Overwrite any existing file.

## Add the path variable in windows environment variable by selecting new classpath.

## Create a path variable and point it at snort.exe variable name path and variable value c:\snort\bin.

## Click OK button and then close all dialog boxes. Open command prompt and and navigate to folder “C:\Snort\bin” folder.

## Type the following commands:

## *Snort –W*

## This is to find the interface.

## 

## Do the following changes in snort.conf file:

## To specify the network address that you want to protect in snort.conf file,look for the following line.

## var HOME\_NET 192.168.1.0/24 (Add IP)

## Change the RULE\_PATH variable to the path of rules folder.

## var RULE\_PATH c:\snort\rules

## Change the path of all library files with the name and path on your system.

## And you must change the path of snort\_dynamicpreprocessorvariable.

## C:\Snort\lib\snort\_dynamiccpreprocessor

## You need to do this to all library files in the “C:\Snort\lib” folder. The old path might be: “/usr/local/lib/...”. you will need to replace that path with your system path. Using C:\Snort\lib

## Change the path of the “dynamicengine” variable value in the “snort.conf” file.

## dynamicengine C:\Snort\lib\snort\_dynamicengine\sf\_engine.dll

## Add the paths for “include classification.config” and “include reference.config” files.

## include c:\snort\etc\classification.config

## include c:\snort\etc\reference.config

## Remove the comment (#) on the line to allow ICMP rules, if it is commented with a #.

## include $RULE\_PATH/icmp.rules

## You can also remove the comment of ICMP-info rules comment, if it is commented.

## include $RULE\_PATH/icmp-info.rules

## To add log files to store alerts generated by snort, search for the “output log” test in snort.conf and add the following line:

## output alert\_fast: snort-alerts.ids

## Find $WHITE\_LIST\_PATH/white\_list.rules and change it to white.list and similarly black\_list.rules to black.list

## Create white.list and black.list file in C:\Snort\rules.

## Add the ip address that needs to be blacklisted in black.list file.

## 

## Comment out (#) following lines:

## #preprocessor normalize\_ip4

## #preprocessor normalize\_tcp: ips ecn stream

## #preprocessor normalize\_icmp4

## #preprocessor normalize\_ip6

## #preprocessor normalize\_icmp6

## Save the “snort.conf” file.

## To start snort in IDS mode, run the following command:

## *snort -i 3 -c C:\Snort\etc\snort.conf -A console*

## To generate Log files in ASCII mode, you can use following command while running snort in IDS mode:

## *snort -A console -i3 -c c:\Snort\etc\snort.conf -l c:\Snort\log -K ascii*

## Output:

## 

## 

## 

## 

## 

## Result:

## The demonstration of intrusion detection using SNORT tool has been done successfully.

## Ex: 11 Web application security using N-Stalker

## Date: 27.10.2020

## Aim:

## To download the N-Stalker Vulnerability Assessment Tool and exploring the features.

## Description:

## EXPLORING N-STALKER:

## N-Stalker Web Application Security Scanner is a Web security assessment tool. It incorporates with a well-known N-Stealth HTTP Security Scanner and 35,000.Web attack signature database. This tool also comes in both free and paid version. Before scanning the target, go to “License Manager” tab, perform the update. Once update, you will note the status as up to date. You need to download and install N-Stalker from [www.nstalker.com](http://www.nstalker.com).

## Algorithm:

## Install N-Stalker Free edition.

## Go to Global Options->change the proxy settings.

## 

## Click the Test Proxy button for testing the correctness of the proxy.

## 

## Go to Update Manager and then update all.

## 

## Click the start icon and do the following steps.

## 

## 

## Click optimize button.

## 

## 

## Then click start session after that click start scan.

## Output:

## 

## 

## Screenshot (143).png

## Result:

## Thus the N-Stalker Vulnerability Assessment tool has been downloaded, installed and the features has been explored by using a website.

Ex: 12 Defeating malware Building Rootkit hunter

## Date: 03.11.2020

## Aim:

## To install a rootkit hunter and find the malwares in a computer.

## Description:

## ROOTKIT HUNTER:

## rkhunter (Rootkit Hunter) is a Unix-based tool that scans for rootkits,backdoors and possible local exploits.

## It does this by comparing SHA-1 hashes of important files with known good ones in online databases, searching for default directories (of rootkits),wrong permissions, hidden files, suspicious strings in kernel modules, and special tests for Linux and FreeBSD.

## rkhunter is notable due to its inclusion in popular operating systems (Fedora, Debian, etc.)

## The tool has been written in Bourne shell, to allow for portability.

## It can run on almost all UNIX-derived systems.

## GMER ROOTKIT TOOL:

## GMER is a software tool written by a Polish researcher Przemysław Gmerek, for detecting and removing rootkits.

## It runs on Microsoft Windows and has support for Windows NT, 2000, XP,Vista, 7, 8 and 10. With version 2.0.18327 full support for Windows x64 is added.

## Algorithm:

## Install and open GMER .

## 

## Double-click the icon for the program.

## Click the "Scan" button in the lower-right corner of the dialog box. Allow the program to scan your entire hard drive.

## 

## 

## When the program completes its scan, select any program or file listed in red if listed in red.

## Right-click it and select "Delete."

## If the red item is a service, it may be protected. Right-click the service and select

## "Disable." Reboot your computer and run the scan again, this time selecting

## "Delete" when that service is detected.

## When your computer is free of Rootkits, close the program and restart your PC.

## Result:

## In this experiment a rootkit hunter software tool has been installed and the rootkits have been detected.