

## 1. Program to illustrate Caesar Cipher

In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which 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 method is named after Julius Caesar, who used it in his private correspondence.

The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. For instance, here is a Caesar cipher using a left rotation of three places, equivalent to a right shift of 23 (the shift parameter is used as the [key](#)):

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

When encrypting, a person looks up each letter of the message in the "plain" line and writes down the corresponding letter in the "cipher" line.

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG  
 Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

Deciphering is done in reverse, with a right shift of 3.

The encryption can also be represented using [modular arithmetic](#) by first transforming the letters into numbers, according to the scheme,  $A \rightarrow 0, B \rightarrow 1, \dots, Z \rightarrow 25$ .<sup>[2]</sup> Encryption of a letter  $x$  by a shift  $n$  can be described mathematically.

$$\text{En}(x) = (x+n) \bmod 26$$

Decryption is performed similarly,

$$\text{Dn}(x) = (x-n) \bmod 26$$

(There are different definitions for the [modulo operation](#). In the above, the result is in the range 0 to 25; i.e., if  $x + n$  or  $x - n$  are not in the range 0 to 25, we have to subtract or add 26.)

The replacement remains the same throughout the message, so the cipher is classed as a type of [monoalphabetic substitution](#), as opposed to [polyalphabetic substitution](#).

**Program**

```
#include<stdio.h>

int main()
{
char message[100], ch;
int i, key;
printf("Enter a message to encrypt: ");
gets(message);
printf("Enter key: ");
scanf("%d", &key);
for(i = 0; message[i] != '\0'; ++i){
ch = message[i];
if(ch >= 'a' && ch <= 'z'){
ch = ch + key;
if(ch > 'z'){
ch = ch - 'z' + 'a' - 1;
}
message[i] = ch;
}
else if(ch >= 'A' && ch <= 'Z'){
ch = ch + key;
if(ch > 'Z'){
ch = ch - 'Z' + 'A' - 1;
}
message[i] = ch;
}
}
printf("Encrypted message: %s", message);
return 0;
}
```

**Output:**

```
Enter a message to encrypt: SSIT
Enter key: 2
Encrypted message: UUKV
```

**2. Program to illustrate Playfair Cipher****The Playfair Cipher Encryption Algorithm:**

The Algorithm consists of 2 steps:

**1. Generate the key Square(5×5):**

- The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J,

then it is replaced by I.

- The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2. **Algorithm to encrypt the plain text:** The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

**For example:**

**PlainText:** "instruments"

**After Split:** 'in' 'st' 'ru' 'me' 'nt' 'sz'

1. Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter.

**Plain Text:** "hello"

**After Split:** 'he' 'lx' 'lo'

Here 'x' is the bogus letter.

2. If the letter is standing alone in the process of pairing, then add an extra bogus letter with the alone letter

**Plain Text:** "helloe"

**AfterSplit:** 'he' 'lx' 'lo' 'ez'

Here 'z' is the bogus letter.

**Rules for Encryption:**

- **If both the letters are in the same column:** Take the letter below each one (going back to the top if at the bottom).

**For example:**

**Diagraph:** "me"

**Encrypted Text:** cl

**Encryption:**

m -> c

e -> l

- **If both the letters are in the same row:** Take the letter to the right of each one (going back to the leftmost if at the rightmost position).

**For example:**

**Diagraph:** "st"

**Encrypted Text:** tl

**Encryption:**

s -> t

t -> l

- **If neither of the above rules is true:** Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

For example:

Diagraph: "nt"

Encrypted Text: rq

Encryption:

n -> r

t -> q

For example:

Plain Text: "instrumentsz"

Encrypted Text: gatlmzclrqtx

Encryption:

i -> g

n -> a

s -> t

t -> l

r -> m

u -> z

m -> c

e -> l

n -> r

t -> q

s -> t

in:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

st:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

ru:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

me:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

nt:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

sz:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

z -> x

### Code:

```
// C program to implement Playfair Cipher
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define SIZE 30
```

```
// Function to convert the string to lowercase
```

```
void toLowerCase(char plain[], int ps)
```

```
{
    int i;
```

```

    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

// Function to remove all spaces in a string
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

// Function to generate the 5x5 key square
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // a 26 character hashmap
    // to store count of the alphabet
    dicty = (int*)calloc(26, sizeof(int));
    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }

    dicty['j' - 97] = 1;

    i = 0;
    j = 0;

    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }

    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {

```

```

        i++;
        j = 0;
    }
}

// Function to search for the characters of a digraph
// in the key square and return their position
void search(char keyT[5][5], char a, char b, int arr[])
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}

// Function to find the modulus with 5
int mod5(int a)
{
    return (a % 5);
}

// Function to make the plain text length to be even
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

// Function for performing the encryption
void encrypt(char str[], char keyT[5][5], int ps)

```

```

{
    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] + 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] + 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

// Function to encrypt using Playfair Cipher
void encryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];

    // Key
    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    // Plaintext
    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);
}

// Driver code
int main()
{
    char str[SIZE], key[SIZE];

    // Key to be encrypted
    strcpy(key, "Monarchy");
    printf("Key text: %s\n", key);
}

```

```

// Plaintext to be encrypted
strcpy(str, "instruments");
printf("Plain text: %s\n", str);

// encrypt using Playfair Cipher
encryptByPlayfairCipher(str, key);

printf("Cipher text: %s\n", str);

return 0;
}

```

### 3. Program to illustrate Hill Cipher

```

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int **matrixmultiply (int **a,int r1,int c1,int **b,int r2,int
c2)
{
    int **resultmatrix;
    int i,j,k,r,c;
    r=r1;c=c2;
    resultmatrix=(int**)malloc(sizeof(int*)*r);
    for(i=0;i<r;i++)
    resultmatrix[i]=(int*)malloc(sizeof(int)*c);
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        {
            resultmatrix[i][j]=0;
            for(k=0;k<c1;k++)
            resultmatrix[i][j]+=a[i][k]*b[k][j];
        }
    }
    return resultmatrix;
}
void printmatrix(int** matrix,int r,int c)
{
    int i,j;
    for(i=0;i<r;i++)
    {
        for(j=0;j<c;j++)
        printf("%d",matrix[i][j]);
        printf("\n");
    }
}
int plaintexttociphertext (char plaintext[],int** matrix)
{
    int len,**plaintextmatrix,**resultmatrix,i,j;

```



```

        // the matrix will be of dimensions strlen (plain text)
        by strlen(plain text)
        char * ciphertext;
        len=strlen(plaintext);
        ciphertext=(char*) malloc(sizeof(char)*1000);
        //plaintextmatrix should be of dimension strlen(plain
text) by 1
        // allocating memory to plaintextmatrix
        plaintextmatrix=(int**)malloc(sizeof(int*)*len);
        for(i=0;i<len;i++)
        plaintextmatrix[i]=(int*)malloc(sizeof(int)*1);
        //populating the plaintextmatrix
        for(i=0;i<len;i++)
        for(j=0;j<1;j++)
            plaintextmatrix[i][j]=plaintext[i]-'a';
        resultmatrix=matrixmultiply(matrix,len,len,plaintextmatr
ix,len,1);
        //taking mod26 of each element of result matrix
        for(i=0;i<len;i++)
        for(j=0;j<1;j++)
            resultmatrix[i][j]%=26;
        //printing the cipher text
        printf(" the cipher text is as follows:");
        for(i=0;i<len;i++)
        for(j=0;j<1;j++)
        printf("%c",resultmatrix[i][j]+'a');
        printf("\n");
    }
    int main()
    {
        int len,i,j,**matrix;
        char plaintext[1000];
        printf(" enter the word to be encrypted:");
        scanf("%s",plaintext);
        len=strlen(plaintext);
        //allocating memory to matrix
        matrix=(int**)malloc(sizeof(int*)*len);
        for(i=0;i<len;i++)
        matrix[i]=(int*)malloc(sizeof(int)*len);
        printf(" enter matrix of %d by %d to be used in
encryption process:\n",len,len);
        for(i=0;i<len;i++)
        for(j=0;j<len;j++)
        scanf("%d",&matrix[i][j]);
        plaintexttociphertext(plaintext,matrix);
        return 0;
    }

```

```

Activities  Terminal  Wed 15:24
ise11@ise11-V5305-07ICR: ~/18is060
File Edit View Search Terminal Help
ise11@ise11-V5305-07ICR:~/18is060$ gedit hillcipher.c
ise11@ise11-V5305-07ICR:~/18is060$ cc hillcipher.c
ise11@ise11-V5305-07ICR:~/18is060$ ./a.out
enter the word to be encrypted: pay
enter matrix of 3 by 3 to be used in encryption process:
17 21 2
17 18 2
5 21 19
the cipher text is as follows:rrl
ise11@ise11-V5305-07ICR:~/18is060$ ./a.out
enter the word to be encrypted: mor
enter matrix of 3 by 3 to be used in encryption process:
17 21 2
17 18 2
5 21 19
the cipher text is as follows:mwb
ise11@ise11-V5305-07ICR:~/18is060$ ./a.out
enter the word to be encrypted: emo
enter matrix of 3 by 3 to be used in encryption process:
17 21 2
17 18 2
5 21 19
the cipher text is as follows:kas
ise11@ise11-V5305-07ICR:~/18is060$ ./a.out
enter the word to be encrypted: ney
enter matrix of 3 by 3 to be used in encryption process:
17 21 2
17 18 2
5 21 19
the cipher text is as follows:pdh
ise11@ise11-V5305-07ICR:~/18is060$ 

```

#### 4. Program to illustrate Vigenere Cipher

##### Vigenere Cipher:

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of [polyalphabetic substitution](#). A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the [Vigenère square or Vigenère table](#).

- The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 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.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

**Table to encrypt Geeks**

```
#include<stdio.h>
#include<string.h>

int main(){
    char msg[] = "THECRAZYPROGRAMMER";
    char key[] = "HELLO";
    int msgLen = strlen(msg), keyLen = strlen(key), i, j;

    char newKey[msgLen], encryptedMsg[msgLen],
    decryptedMsg[msgLen];

    //generating new key
    for(i = 0, j = 0; i < msgLen; ++i, ++j){
        if(j == keyLen)
            j = 0;
    }
```

```

        newKey[i] = key[j];
    }

    newKey[i] = '\0';

    //encryption
    for(i = 0; i < msgLen; ++i)
        encryptedMsg[i] = ((msg[i] + newKey[i]) % 26) +
'A';

    encryptedMsg[i] = '\0';

    //decryption
    for(i = 0; i < msgLen; ++i)
        decryptedMsg[i] = (((encryptedMsg[i] - newKey[i])
+ 26) % 26) + 'A';

    decryptedMsg[i] = '\0';

    printf("Original Message: %s", msg);
    printf("\nKey: %s", key);
    printf("\nNew Generated Key: %s", newKey);
    printf("\nEncrypted Message: %s", encryptedMsg);
    printf("\nDecrypted Message: %s", decryptedMsg);

    return 0;
}

```

Output: **Message Text:** THECRAZYPROGRAMMER

**Key:** HELLO

**New Generated Key:** HELLOHELLOHELLOHEL

## 5. Implement the following TRANSPOSITION TECHNIQUES

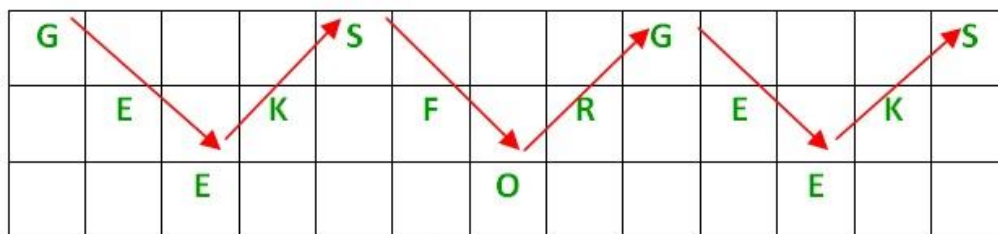
**concepts: Rail fence - row & Column Transformation**

### Encryption

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

- 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.

For example, if the message is "GeeksforGeeks" and the number of rails = 3 then cipher is prepared as:



© copyright geeksforgeeks.org

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
main()
{
    int i,j,len,rails,count,code[100][1000];
    char str[1000];
    printf("Enter a Secret Message\n");
    gets(str);
    len=strlen(str);
    printf("Enter number of rails\n");
    scanf("%d",&rails);
    for(i=0;i<rails;i++)
    {
        for(j=0;j<len;j++)
        {
            code[i][j]=0;
        }
    }
}
```

```

count=0;
j=0;
while (j<len)
{
    if (count%2==0)
    {
        for (i=0;i<rails;i++)
        {
            //strcpy(code[i][j],str[j]);
            code[i][j]=(int)str[j];
            j++;
        }
    }
    else
    {
        for (i=rails-2;i>0;i--)
        {
            code[i][j]=(int)str[j];
            j++;
        }
    }

    count++;
}

for (i=0;i<rails;i++)
{
    for (j=0;j<len;j++)
    {
        if (code[i][j]!=0)
            printf("%c",code[i][j]);
    }
}
printf("\n");
}

```

Original Message: Hello World  
 Encrypted Message: Horel ollWd  
 Decrypted Message: Hello World

## 6. Implement the DES algorithm

STEP-1: Read the 64-bit plain text.  
 STEP-2: Split it into two 32-bit blocks and store it in two different arrays.  
 STEP-3: Perform XOR operation between these two arrays.  
 STEP-4: The output obtained is stored as the second 32-bit sequence and the original

second 32-bit sequence forms the first part.  
 STEP-5: Thus the encrypted 64-bit cipher text is obtained in this way. Repeat the same process for the remaining plain text characters.

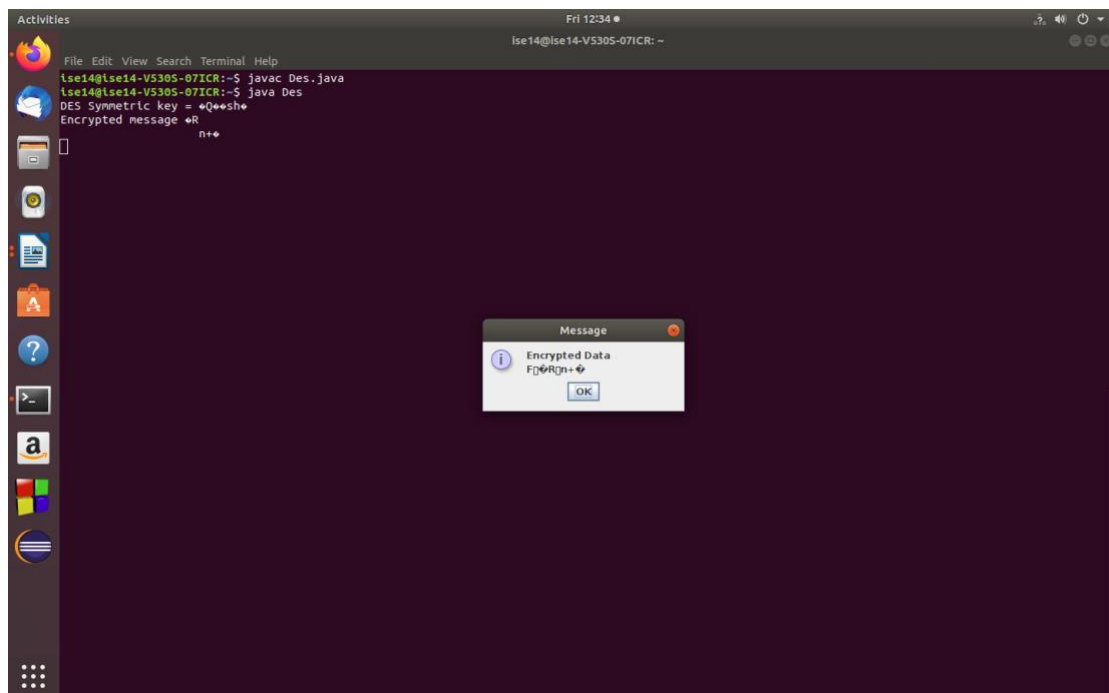
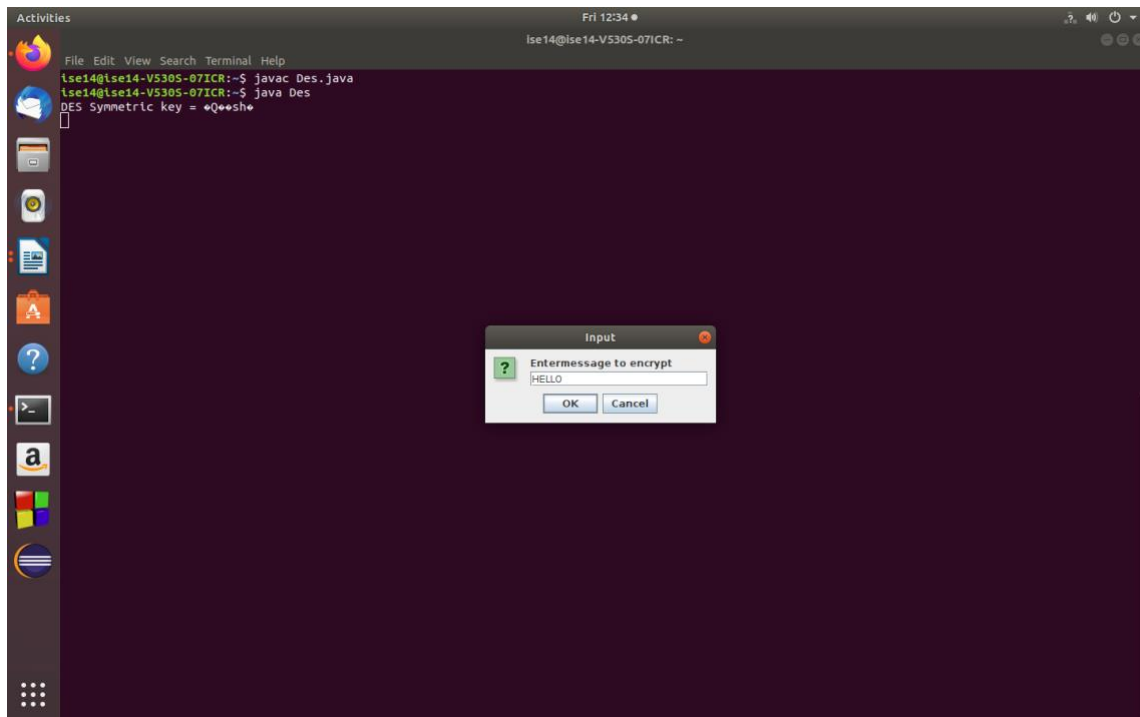
```
import javax.swing.*;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Random ;
class Des {
byte[] skey = new byte[1000];
String skeyString;
static byte[] raw;
String inputMessage, encryptedData, decryptedMessage;
public Des ()
{
try
{
generateSymmetricKey();
inputMessage=JOptionPane.showInputDialog(null, "Enter message to encrypt");
byte[] ibyte = inputMessage.getBytes();
byte[] ebyte=encrypt(raw, ibyte);
String encryptedData = new String(ebyte);
System.out.println("Encrypted message "+encryptedData);
JOptionPane.showMessageDialog(null, "Encrypted Data"+"\\n"+encryptedData);
byte[] dbyte= decrypt(raw, ebyte);
String decryptedMessage = new String(dbyte);
System.out.println("Decrypted message"+decryptedMessage);
JOptionPane.showMessageDialog(null, "Decrypted Data"+"\\n"+decryptedMessage);
}
catch(Exception e)
{
System.out.println(e);
}
}
void generateSymmetricKey() {
try {
Random r = new Random();
int num = r.nextInt(10000);
String knum = String.valueOf(num);
byte[] knumb = knum.getBytes();
skey=getRawKey(knumb);
skeyString = new String(skey);
System.out.println("DES Symmetric key = "+skeyString);
}
}
```

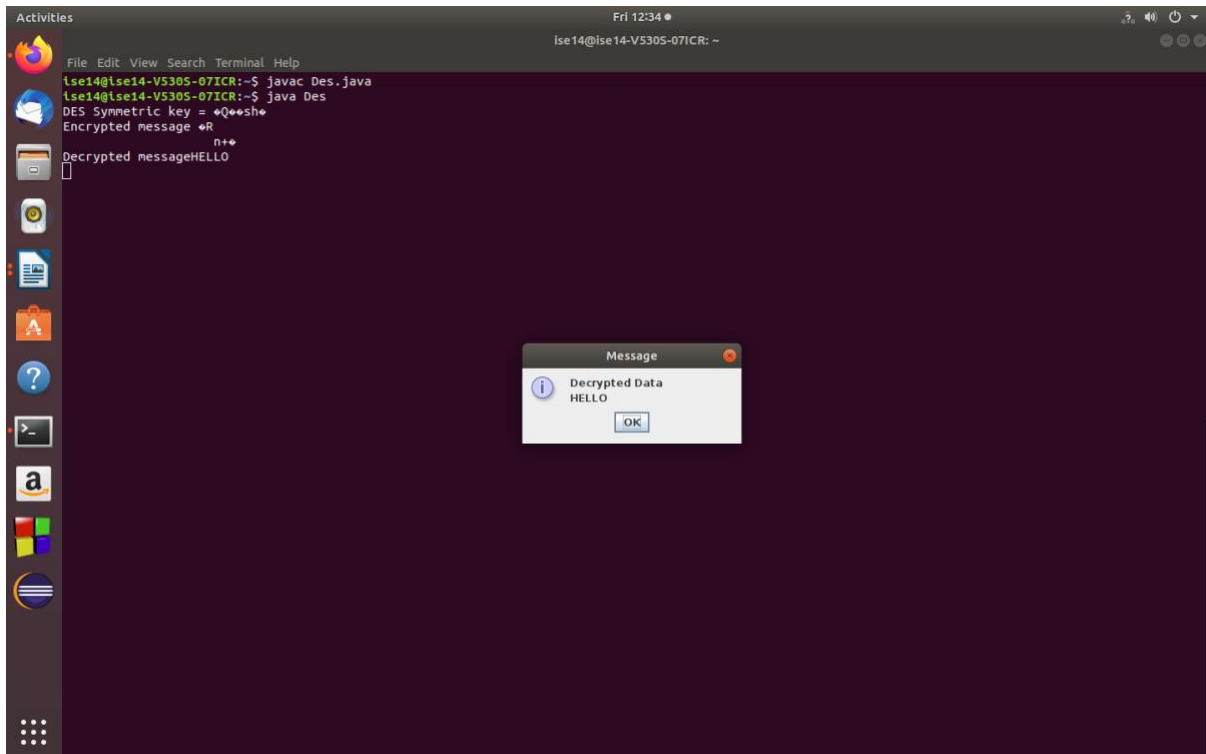
```

catch(Exception e)
{
System.out.println(e);
}
}
private static byte[] getRawKey(byte[] seed) throws Exception
{
KeyGenerator kgen = KeyGenerator.getInstance("DES");
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
sr.setSeed(seed);
kgen.init(56, sr);
SecretKey skey = kgen.generateKey();
raw = skey.getEncoded();
return raw;
}
private static byte[] encrypt(byte[] raw, byte[] clear) throws
Exception {
SecretKeySpec skeySpec = new SecretKeySpec(raw,"DES");
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
byte[] encrypted = cipher.doFinal(clear);
return encrypted;
}
private static byte[] decrypt(byte[] raw, byte[] encrypted)
throws Exception
{
SecretKeySpec skeySpec = new SecretKeySpec(raw,"DES");
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.DECRYPT_MODE, skeySpec);
byte[] decrypted = cipher.doFinal(encrypted);
return decrypted;
}
public static void main(String args[]) {
DES des = new DES();
}
}

```







## 7. Implement the SHA-1 algorithm.

```

import java.security.*;
class JceSha1Test {
    public static void main(String[] a) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA1");
            System.out.println("Message digest object info: ");
            System.out.println("    Algorithm =
"+md.getAlgorithm());
            System.out.println("    Provider =
"+md.getProvider());
            System.out.println("    toString = "+md.toString());

            String input = "";
            md.update(input.getBytes());
            byte[] output = md.digest();
            System.out.println();
            System.out.println("SHA1(\""+input+"") =");
            System.out.println("    "+bytesToHex(output));

            input = "abc";
            md.update(input.getBytes());
            output = md.digest();
            System.out.println();
  
```

```

System.out.println("SHA1(\""+input+"\"") =");
System.out.println("    "+bytesToHex(output));

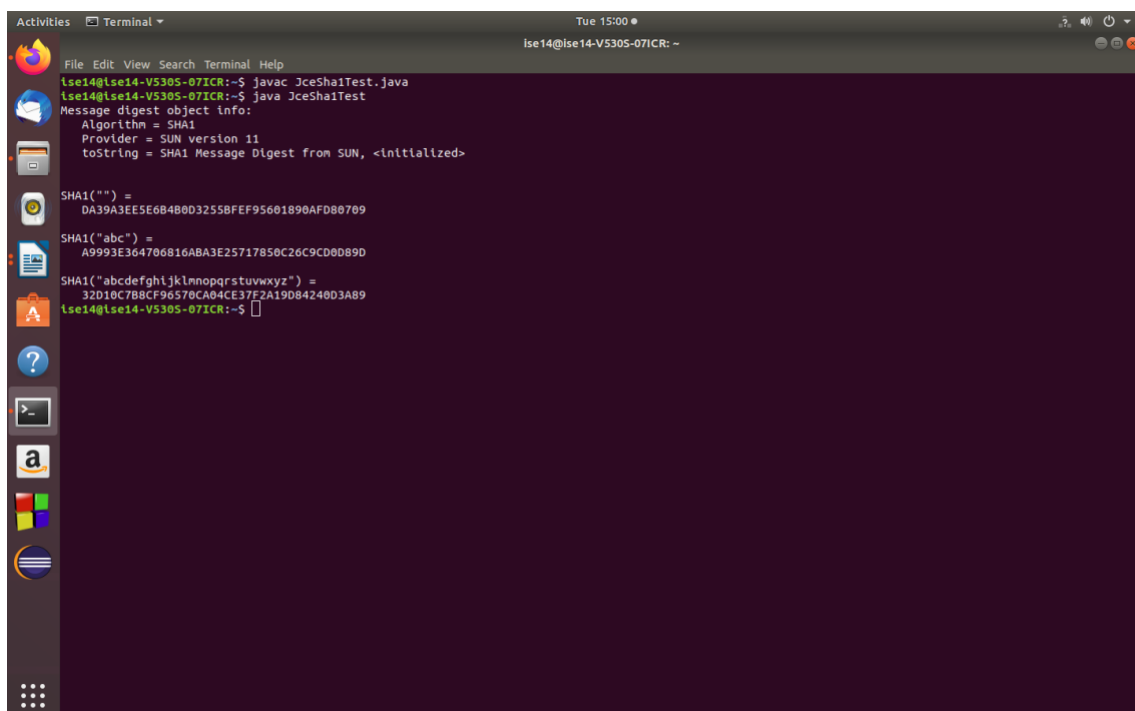
input = "abcdefghijklmnopqrstuvwxyz";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("SHA1(\""+input+"\"") =");
System.out.println("    "+bytesToHex(output));

} catch (Exception e) {
    System.out.println("Exception: "+e);
}
}

public static String bytesToHex(byte[] b) {
    char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6',
'7',
                        '8', '9', 'A', 'B', 'C', 'D', 'E',
'F'};
    StringBuffer buf = new StringBuffer();
    for (int j=0; j<b.length; j++) {
        buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
        buf.append(hexDigit[b[j] & 0x0f]);
    }
    return buf.toString();
}
}

```

output:



```

Tue 15:00
lse14@lse14-V5305-07ICR: ~
File Edit View Search Terminal Help
lse14@lse14-V5305-07ICR:~$ javac JceSha1Test.java
lse14@lse14-V5305-07ICR:~$ java JceSha1Test
Message digest object info:
  Algorithm = SHA1
  Provider = SUN version 11
  toString = SHA1 Message Digest from SUN, <initialized>

SHA1("") =
DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

SHA1("abc") =
A9993E364706816ABA3E25717850C26C9CD0D89D

SHA1("abcdefghijklmnopqrstuvwxyz") =
32D18C7B8CF96570CA04CE37F2A19084240D3A89
lse14@lse14-V5305-07ICR:~$

```

## 8. Implementation of MD5 Algorithm.

The Message-Digest Algorithm (MD5) is a widely used cryptographic hash function, which generates a 16-byte (128-bit) hash value. It is very simple and easy to understand. The main concept is to do the mapping the data sets of variable length to data sets of a constant length.

To achieve the same, the input message is broken into a group of 512-bit blocks. Padding is joined to the end such that its size can be divided by 512. Now, the blocks are processed using the MD5 algorithm that operates in the 128-bit state, and the result is a 128-bit hash value. After applying the MD5 algorithm, the generated hash is usually a 32-digit hexadecimal number. Here, the string that has to be encoded is usually called the "message" and the hash value that is generated after the hashing is known as the "digest" or "message digest".

```
// import statements
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.MessageDigest;

// A Java program that uses the MD5 to do the hashing
public class MD5
{
    public static String getMd5(String input)
    {
        try
        {
            // invoking the static getInstance() method of the
            // MessageDigest class
            // Notice it has MD5 in its parameter.
            MessageDigest msgDst = MessageDigest.getInstance("MD5");

            // the digest() method is invoked to compute the message
            // digest
            // from an input digest() and it returns an array of byte
            byte[] msgArr = msgDst.digest(input.getBytes());

            // getting signum representation from byte array msgArr
```

```

BigInteger bi = new BigInteger(1, msgArr);

// Converting into hex value
String hshtxt = bi.toString(16);

while (hshtxt.length() < 32)
{
    hshtxt = "0" + hshtxt;
}
return hshtxt;
}
// for handling the exception
catch (NoSuchAlgorithmException abc)
{
    throw new RuntimeException(abc);
}
}
// main method code
public static void main(String args[]) throws
NoSuchAlgorithmException
{
    String str = "ssitise";
    String hash = getMd5(str);
    str = "'isedept'";
    System.out.println("The HashCode Generated for " + str + "
is: " + hash);
}
}

```

Output:

The screenshot shows a terminal window with the following commands and output:

```

ise14@ise14-V5305-07ICR:~$ javac MD5.java
ise14@ise14-V5305-07ICR:~$ java MD5
The HashCode Generated for 'isedept' is: 27bd37830958dfd64c47dfbed57db718
ise14@ise14-V5305-07ICR:~$

```

## 9. Implement the Signature Scheme - Digital Signature Standard.

The **digital signature** is an electronic signature to sign a document, mail, messages, etc. It validates the **authenticity**, and **integrity** of a message or document. It is the same as a handwritten signature, seal, or stamp. It is widely used to verify a digital message, financial documents, identity cards, etc.

In short, we can say that it ensures the following:

- **Integrity:** It ensures the message or a document cannot be altered while transmitting.
- **Authenticity:** The author of the message is really who they claim to be.
- **Non-repudiation:** The author of the message can't later deny that they were the source.

The digital signature is used by the **Directory Server** that preserves the integrity of information. When we apply **encryption** and **message digests** to the information to be transmitted, the receiver of the information can easily determine that the information is not tempered during transmission.

```
import java.util.*;
import java.math.BigInteger;
class dsaAlg {
final static BigInteger one = new BigInteger("1");
final static BigInteger zero = new BigInteger("0");
public static BigInteger getNextPrime(String ans)
{
BigInteger test = new BigInteger(ans);
while (!test.isProbablePrime(99))
e:
{
test = test.add(one);
}
return test;
}
public static BigInteger findQ(BigInteger n)
{
BigInteger start = new BigInteger("2");
while (!n.isProbablePrime(99))
{
while (!((n.mod(start)).equals(zero)))
{
start = start.add(one);
}
n = n.divide(start);
}
```

```

}
return n;
}
public static BigInteger getGen(BigInteger p, BigInteger
q, Random r)
{
BigInteger h = new BigInteger(p.bitLength(), r);
h = h.mod(p);
return h.modPow((p.subtract(one)).divide(q), p);
}
public static void main (String[] args) throws
java.lang.Exception
{
Random randObj = new Random();
BigInteger p = getNextPrime("10600"); /* approximate prime */
BigInteger q = findQ(p.subtract(one));
BigInteger g = getGen(p,q,randObj);
System.out.println(" \n simulation of Digital Signature
Algorithm \n");
System.out.println(" \n global public key components are:\n");
System.out.println("\np is: " + p);
System.out.println("\nq is: " + q);
System.out.println("\ng is: " + g);
BigInteger x = new BigInteger(q.bitLength(), randObj);
x = x.mod(q);
BigInteger y = g.modPow(x,p);
BigInteger k = new BigInteger(q.bitLength(), randObj);
k = k.mod(q);
BigInteger r = (g.modPow(k,p)).mod(q);
BigInteger hashVal = new BigInteger(p.bitLength(),randObj);
BigInteger kInv = k.modInverse(q);
BigInteger s = kInv.multiply(hashVal.add(x.multiply(r)));
s = s.mod(q);
System.out.println("\nsecret information are:\n");
System.out.println("x (private) is:" + x);
System.out.println("k (secret) is: " + k);
System.out.println("y (public) is: " + y);
System.out.println("h (rndhash) is: " + hashVal);
System.out.println("\n generating digital signature:\n");
System.out.println("r is : " + r);
System.out.println("s is : " + s);
BigInteger w = s.modInverse(q);
BigInteger u1 = (hashVal.multiply(w)).mod(q);
BigInteger u2 = (r.multiply(w)).mod(q);
BigInteger v = (g.modPow(u1,p)).multiply(y.modPow(u2,p));
v = (v.mod(p)).mod(q);
System.out.println("\nverifying          digital          signature
(checkpoints)\n:");
System.out.println("w is : " + w);
System.out.println("u1 is : " + u1);
System.out.println("u2 is : " + u2);
System.out.println("v is : " + v);

```

```

if (v.equals(r))
{
System.out.println("\nsuccess:      digital      signature      is
verified!\n " + r);
}
else
{
System.out.println("\n error:  incorrect  digital  signature\n
");
}
}
}
}

```

### Output:

```

Activities  Terminal  Tue 15:27
ise14@ise14-V5305-07ICR: ~
ise14@ise14-V5305-07ICR:~$ javac dsaAlg.java
ise14@ise14-V5305-07ICR:~$ java dsaAlg
simulation of Digital Signature Algorithm

global public key components are:

p is: 10601
q is: 53
g is: 5582

secret information are:

x (private) is:20
k (secret) is: 5
y (public) is: 479
h (rndhash) is: 10032

generating digital signature:

r is : 31
s is : 21

verifying digital signature (checkpoints)
:
w is : 48
u1 is : 31
u2 is : 4
v is : 31

success: digital signature is verified!
31
ise14@ise14-V5305-07ICR:~$ 

```

## 10. a. C Program to Perform Fermat Primality Test

### Fermat's Little Theorem:

If  $n$  is a prime number, then for every  $a$ ,  $1 < a < n-1$ ,

$$a^{n-1} \equiv 1 \pmod{n}$$

OR

$$a^{n-1} \% n = 1$$

Example: Since 5 is prime,  $2^4 \equiv 1 \pmod{5}$  [or  $2^4 \% 5 = 1$ ],  
 $3^4 \equiv 1 \pmod{5}$  and  $4^4 \equiv 1 \pmod{5}$

Since 7 is prime,  $2^6 \equiv 1 \pmod{7}$ ,



$$3^6 \equiv 1 \pmod{7}, 4^6 \equiv 1 \pmod{7}$$

$$5^6 \equiv 1 \pmod{7} \text{ and } 6^6 \equiv 1 \pmod{7}$$

```

#include <stdio.h>;
#include <stdlib.h>;
#define ll long long
/*
* modular exponentiation
*/ll modulo(ll base, ll exponent, ll mod) {
ll x = 1;
ll y = base;
while (exponent > 0) {
if (exponent % 2 == 1)
x = (x * y) % mod;
y = (y * y) % mod;
exponent = exponent / 2;
}
return x % mod;
}
/*
* Fermat's test for checking primality
*/
int Fermat(ll p, int iterations) {
int i;
if (p == 1) {
return 0;
}
for (i = 0; i < iterations; i++) {
ll a = rand() % (p - 1) + 1;
if (modulo(a, p - 1, p) != 1) {
return 0;
}
}
return 1;
}
/*
* Main
*/
int main() {
int iteration = 50;
ll num;
printf("Enter integer to test primality: ");
scanf("%lld", &num);
if (Fermat(num, iteration) == 1)
printf("%lld is prime", num);
else
printf("%lld is not prime", num);
return 0;
}

```

Cryptography and Network Security Lab Manual  
Department of ISE, SSIT, Tumkur

```
}
```

### Output:

```
Enter integer to test primality
```

```
7
```

```
7 is prime
```

### 10. b. // A simple C program to calculate Euler's Totient Function

**Euler's Totient function  $\Phi(n)$  for an input  $n$  is the count of numbers in  $\{1, 2, 3, \dots, n\}$  that are relatively prime to  $n$ , i.e., the numbers whose GCD (Greatest Common Divisor) with  $n$  is 1.**

**Examples :**

```
 $\Phi(1) = 1$   
gcd(1, 1) is 1
```

```
 $\Phi(2) = 1$   
gcd(1, 2) is 1, but gcd(2, 2) is 2.
```

```
 $\Phi(3) = 2$   
gcd(1, 3) is 1 and gcd(2, 3) is 1
```

```
 $\Phi(4) = 2$   
gcd(1, 4) is 1 and gcd(3, 4) is 1
```

```
 $\Phi(5) = 4$   
gcd(1, 5) is 1, gcd(2, 5) is 1,  
gcd(3, 5) is 1 and gcd(4, 5) is 1
```

```
 $\Phi(6) = 2$   
gcd(1, 6) is 1 and gcd(5, 6) is 1,
```

```
#include <stdio.h>
```

```
// Function to return gcd of a and b  
int gcd(int a, int b)  
{  
    if (a == 0)  
        return b;  
    return gcd(b % a, a);  
}
```

```
// A simple method to evaluate Euler Totient Function  
int phi(unsigned int n)
```

```
{
    unsigned int result = 1;
    for (int i = 2; i < n; i++)
        if (gcd(i, n) == 1)
            result++;
    return result;
}

// Driver program to test above function
int main()
{
    int n;
    for (n = 1; n <= 10; n++)
        printf("phi(%d) = %d\n", n, phi(n));
    return 0;
}
```

**Output :**

```
phi(1) = 1
phi(2) = 1
phi(3) = 2
phi(4) = 2
phi(5) = 4
phi(6) = 2
phi(7) = 6
phi(8) = 4
phi(9) = 6
phi(10) = 4
```

**PART B****1. Write program to implement AES algorithm.**

```
// Java program to demonstrate the creation
// of Encryption and Decryption with Java AES
import java.nio.charset.StandardCharsets;
import java.security.spec.KeySpec;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

class AES {
    // Class private variables
    private static final String SECRET_KEY
        = "my_super_secret_key_ho_ho_ho";

    private static final String SALT = "ssshhhhhhhhhhh!!!!";

    // This method use to encrypt to string
    public static String encrypt(String strToEncrypt)
    {
        try {

            // Create default byte array
            byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0 };
            IvParameterSpec ivspec
                = new IvParameterSpec(iv);

            // Create SecretKeyFactory object
            SecretKeyFactory factory
```

```

        = SecretKeyFactory.getInstance(
            "PBKDF2WithHmacSHA256");

// Create KeySpec object and assign with
// constructor
KeySpec spec = new PBEKeySpec(
    SECRET_KEY.toCharArray(), SALT.getBytes(),
    65536, 256);
SecretKey tmp = factory.generateSecret(spec);
SecretKeySpec secretKey = new SecretKeySpec(
    tmp.getEncoded(), "AES");

Cipher cipher = Cipher.getInstance(
    "AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, secretKey,
    ivspec);
// Return encrypted string
return Base64.getEncoder().encodeToString(
    cipher.doFinal(strToEncrypt.getBytes(
        StandardCharsets.UTF_8)));
}
catch (Exception e) {
    System.out.println("Error while encrypting: "
        + e.toString());
}
return null;
}

// This method use to decrypt to string
public static String decrypt(String strToDecrypt)
{
    try {

        // Default byte array
        byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0 };
        // Create IvParameterSpec object and assign
with
        // constructor
        IvParameterSpec ivspec
            = new IvParameterSpec(iv);

        // Create SecretKeyFactory Object
        SecretKeyFactory factory
            = SecretKeyFactory.getInstance(
                "PBKDF2WithHmacSHA256");

        // Create KeySpec object and assign with
        // constructor
        KeySpec spec = new PBEKeySpec(
            SECRET_KEY.toCharArray(), SALT.getBytes(),
            65536, 256);

```

```

        SecretKey tmp = factory.generateSecret(spec);
        SecretKeySpec secretKey = new SecretKeySpec(
            tmp.getEncoded(), "AES");

        Cipher cipher = Cipher.getInstance(
            "AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, secretKey,
            ivspec);
        // Return decrypted string
        return new String(cipher.doFinal(

Base64.getDecoder().decode(strToDecrypt)));
    }
    catch (Exception e) {
        System.out.println("Error while decrypting: "
            + e.toString());
    }
    return null;
}

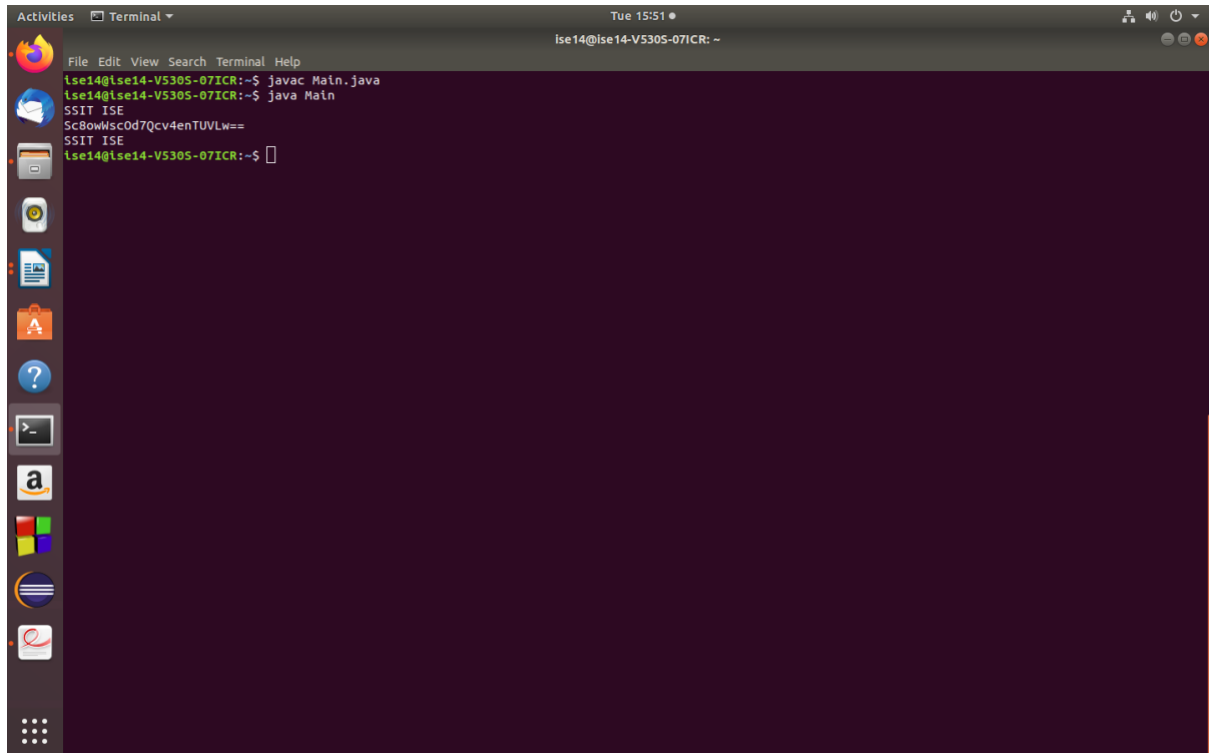
// driver code
public class Main {
    public static void main(String[] args)
    {
        // Create String variables
        String originalString = "SSIT ISE";

        // Call encryption method
        String encryptedString
            = AES.encrypt(originalString);

        // Call decryption method
        String decryptedString
            = AES.decrypt(encryptedString);

        // Print all strings
        System.out.println(originalString);
        System.out.println(encryptedString);
        System.out.println(decryptedString);
    }
}

```



The screenshot shows a terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Tue 15:51, ise14@ise14-V5305-07ICR: ~). The terminal output is as follows:

```
ise14@ise14-V5305-07ICR:~$ javac Main.java
ise14@ise14-V5305-07ICR:~$ java Main
SSIT ISE
Sc8owWscOd7Qcv4enTUVLW==
SSIT ISE
ise14@ise14-V5305-07ICR:~$
```

The left sidebar of the terminal window displays a vertical stack of application icons, including Firefox, Nautilus, and various system utilities.

## **2. Implementation of Elliptic Curve Cryptography.**

## **3. Creation of digital signature secures data storage, secure data transmission using GNUPG.**

AIM:

Demonstrate how to provide secure data storage, secure data transmission and for creating digital signatures (GnuPG).

#### INTRODUCTION:

# Here's the final guide in my PGP basics series, this time focusing on Windows

# The OS in question will be Windows 7, but it should work for Win8 and Win8.1 as well

# Obviously it's not recommended to be using Windows to access the DNM, but I

won't go into the reasons here.

# The tool well be using is GPG4Win

#### INSTALLING THE SOFTWARE:

##### SOFTWARE

1. Visit [www.gpg4win.org](http://www.gpg4win.org)

[www.gpg4win.org](http://www.gpg4win.org). Click on the "Gpg4win 2.3.0" button