

List of Experiments and Continuous Evaluation

A: Attendance **OE:** Observation & Experiment Conduction

R: Record **T:** Total marks

| S.no. | Name of the experiment | Page no. | Marks | | | | | Remarks | Date of exp. | Sign. with date |
|-------|---|----------|----------|------------|------------|----------|-----------|---------|--------------|-----------------|
| | | | A (2) | OBS (2) | EXP (3) | R (3) | T (10) | | | |
| 1 | Write a C/JAVA program to perform encryption and decryption using Ceaser cipher algorithm. | | | | | | | | | |
| 2 | Write a C/JAVA program to perform encryption and decryption using Substitution cipher algorithm. | | | | | | | | | |
| 3 | Write a C/JAVA program to perform encryption and decryption using hill cipher algorithm. | | | | | | | | | |
| 4 | Write C /JAVA program to implement the DES algorithm logic. | | | | | | | | | |
| 5 | Write C/JAVA program that contains functions, which accept a key and input text to be encrypted/decrypted. This program should use the key to encrypt/decrypt the input by using the triple DES algorithm. Make use of Java Cryptography package. | | | | | | | | | |
| 6 | Write C/JAVA program to implement the Blowfish algorithm logic | | | | | | | | | |
| 7 | Write C/JAVA program to implement RSA algorithm. | | | | | | | | | |

| | | | | | | | | | | |
|----|--|--|--|--|--|--|--|--|--|--|
| 8 | Write C/JAVA to Calculate the message digest of a text using the SHA-1 algorithm . | | | | | | | | | |
| 9 | write a JAVA program to Calculate the message digest of a text using the MD5 algorithm. | | | | | | | | | |
| 10 | Write a JAVA program to implement digital certificates | | | | | | | | | |
| 11 | Write a java program Create a digital certificate of your own by using the Java key tool | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

Name and Signature of the Faculty member

PAC Member

HOD

AIM: Write a C/JAVA program to perform encryption and decryption using Ceaser cipher algorithm.

DESCRIPTION

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet.

$$E_n(x)=(x+n)\%26$$

$$D_n(x)=(x-n)\%26$$

ALGORITHM

1. Input a String of lower case letters, called Text.
2. Input an Integer between 0-25 denoting the required shift.
3. Traverse the given text one character at a time.
4. For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
5. Return the new string generated.

PROGRAM

```
#include <stdio.h>

#include <string.h>

int main(){

    char text[50];

    char enctext[50];

    char dectext[50];

    int key,temp,i,l;

    printf("\nEnter the text:");

    scanf("%s",text);

    printf("\nEnter the key:");

    scanf("%d",&key);

    l=strlen(text);

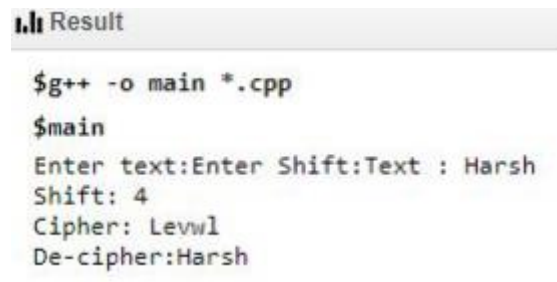
    for(i=0;i<l;i++){
```

```

        enctext[i]=((text[i]-97+key)%26)+97;
    }
    enctext[l]='\0';
    printf("\nencrypted text is: %s",enctext);
    for (i=0;i<l;i++){
        temp=(enctext[i]-97-key)%26;
        if(temp>=0)
            dectext[i]=temp+97;
        else
            dectext[i]=123+temp;
    }
    dectext[l]='\0';
    printf("\nthey decrypted text is:%s",dectext);
}

```

OUTPUT



```

Result
$g++ -o main *.cpp
$main
Enter text:Enter Shift:Text : Harsh
Shift: 4
Cipher: Levwl
De-cipher:Harsh

```

CONCLUSION

Ceasar Cipher has been successfully implemented

AIM: Write a C/JAVA program to perform encryption and decryption using Substitution cipher algorithm.

DESCRIPTION

The simple substitution cipher is a cipher that has been in use for many hundreds of years. It basically consists of substituting every plaintext character for a different ciphertext character. It differs from the Caesar cipher in that the cipher alphabet is not simply the alphabet shifted, it is completely jumbled.

ALGORITHM

1. String of lower or upper case letters as input
2. A string containing the substituted characters is taken
3. Traverse the given initial plain text one character at a time
4. For each character, transform or substitute given character
5. Return the new string generated

PROGRAM

ENCRYPTION:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
    int n=13,i,k;
    char c[100];
    printf("Enter a plain text ");
    for(i=0;i<100;i++)
    {
        scanf("%c",&c[i]);
        if(c[i]=='\n') break;
    }
    for(i=0;i<100;i++)
    {
        if(c[i]=='\0')
            break;
        else if(c[i]>=97 && c[i]<=122)
        {
            k=c[i];
            k=k+n;
            if(k>122) k=k-26;
            c[i]=(char)k;
        }
        else if(c[i]>=64 && c[i]<=90)
        {
            k=c[i];
            k=k+n;
```

```

        if(k>90) k=k-26;
        c[i]=(char)k;
    }
}
printf("Cipher Text is %s",c);
}

```

DECRYPTION:

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
    int n=13,i,k;
    char c[100];
    printf("Enter a cipher text ");
    for(i=0;i<100;i++)
    {
        scanf("%c",&c[i]);
        if(c[i]=='\n') break;
    }
    for(i=0;i<100;i++)
    {
        if(c[i]=='\0')
            break;
        else if(c[i]>=97 && c[i]<=122)
        {
            k=c[i];
            k=k-n;
            if(k<97) k=k+26;
            c[i]=(char)k;
        }
        else if(c[i]>=64 && c[i]<=90)
        {
            k=c[i];
            k=k-n;
            if(k<64) k=k+26;
            c[i]=(char)k;
        }
    }
    printf("Plain Text is %s",c);
}

```

OUTPUT:**Result**

```
$g++ -o main *.cpp
$main
Enter text:
Text : Harsh
Cipher: Unefu
De-cipher:Harsh
```

Result

```
$g++ -o main *.cpp
$main
Enter text:Text : Harsh
Cipher: Kexbu
De-cipher:Harsh
```

CONCLUSION:

Substitution cipher has been implemented successfully showing encryption and decryption.

AIM: Write a C/JAVA program to perform encryption and decryption using hill cipher algorithm.

DESCRIPTION

Hill cipher is a polygraphic substitution cipher based on linear algebra. 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 (considered as an n-component vector) is multiplied by an invertible $n \times n$ matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

ALGORITHM

1. Input plain text and key
2. Verify if key is non-singular
3. Transform plain to cipher text with the key
4. Find det inverse to decrypt
5. Multiply det inverse to the adjoint of key matrix
6. Transform cipher to plain text with the decryption key

PROGRAM

```
import sys

import numpy

import math

chartext=list(input('Entr the text to be encrypted:')) #only capital letters. No spaces

print('the text entered is:',chartext)

asciitext=[]

for d in range(len(chartext)):

    asciitext.append(ord(chartext[d]))

asciitext=numpy.asarray(asciitext)

asciitext=asciitext-65

print('the numeric form of text is:',asciitext)

length=len(chartext)

print('length of the text is:',length)
```



```

key=list(input('Enter key')) #only capital letters

for d in range(len(key)):

    key[d]=ord(key[d])

key=numpy.asarray(key)

key=key-65

print('the numeric of key is:',key)

keylen=int(input('enter matrix dimension'))

keymatrix=numpy.asarray(key[0:keylen*keylen])

keymatrix=numpy.reshape(keymatrix,(keylen,keylen))

print('keymatrix:')

print(keymatrix)

detkeymatrix=numpy.linalg.det(keymatrix)

detkeymatrix=math.ceil(detkeymatrix%26)

print('det of keymatrix is:',detkeymatrix)

cokeymatrix=numpy.linalg.inv(keymatrix).T*numpy.linalg.det(keymatrix)

adjkeymatrix=numpy.matrix.transpose(cokeymatrix)

adjkeymatrix=adjkeymatrix%26

print('adjoint of keymatrix is:')

print(adjkeymatrix)

detinverse=-1

for z in range(1,25):

    if (detkeymatrix*z)%26==1:

        detinverse=z

        break

if detinverse==-1:

```

```

    print('detinversen not initialized')

    exit()

print('det inverse is:',detinverse)

keyinverse=detinverse*adjkeymatrix

keyinverse=keyinverse%26

print('keyinverse is:')

print(keyinverse)

for x in range(0,length-keylen+1,keylen):

    tempmatrix=numpy.asarray(asciitext[x:x+keylen])

    tempmatrix=numpy.reshape(tempmatrix,(keylen,1))

    alpha=numpy.array(tempmatrix).tolist()

    print('temp matrix is:')

    print(tempmatrix)

    enc=numpy.matmul(keymatrix,tempmatrix)

    enc=numpy.mod(enc,26)

    print('encrypted text is:')

    print(enc)

    dec=numpy.matmul(keyinverse,enc)

    dec=numpy.mod(dec,26)

    print('decrypted text is:')

    print(dec)

```

RESULTS

Result

```
$g++ -o main *.cpp
$main
Name:-Harsh
Roll No:-160115733030
Enter 3x3 matrix for key (It should be inversible):

Enter a 3 letter string:
Encrypted string is: poh
Decrypted string is: act
```

CONCLUSION

Hill cipher has been implemented successfully- both encryption and decryption.

AIM: Write C /JAVA program to implement the DES algorithm logic.

DESCRIPTION

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST). DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only).

ALGORITHM

- 1) Feed 64 bit key to PC1 to obtain 56 bit key
- 2) Modify the two halves of 28 bits each with respective left shifts to obtain 16 keys for 16 rounds
- 3) Combine Right half followed by left half
- 4) Feed 56 bit key to PC2 to obtain 48 bit key
- 5) Feed plain message to IP
- 6) Obtain 16 rounds of data from each of the 32 halves of the message by encrypting with 16 rounds of key

PROGRAM

USING PYDES PACKAGE:

```
from pydes import des
```

```
key = "secret_k"
```

```
text= "anirudhi"
```

```
d = des()
```

```
ciphered = d.encrypt(key,text)
```

```
plain = d.decrypt(key,ciphered)
```

```
print "Ciphered: %r" % ciphered
```

```
print "Deciphered: ", plain
```

OUTPUT:

```
C:\Users\admin\Desktop>python des.py  
Encrypted: b'o\xce5ef\xe6\xa6\x8f\x82\x98\xc7V\xd0I\xdc\x03\x1e\x07\xe4\x998\x07\x9cI'  
Decrypted: b'Please encrypt my data'
```

CONCLUSION:

DES has been successfully implemented with the help of Pydes package.

AIM: Write C/JAVA program that contains functions, which accept a key and input text to be encrypted/decrypted. This program should use the key to encrypt/decrypt the input by using the triple DES algorithm. Make use of Java Cryptography package.

DESCRIPTION:

Triple Data Encryption Standard (DES) is a type of computerized cryptography where block cipher algorithms are applied three times to each data block. The key size is increased in Triple DES to ensure additional security through encryption capabilities. Each block contains 64 bits of data. Three keys are referred to as bundle keys with 56 bits per key.

ALGORITHM:

1. Triple DES encrypts input data three times.
2. The three keys are referred to as k1, k2 and k3 or two keys referred to as k1 and k2.
3. Encrypt three times
4. Decrypt the same way

PROGRAM

USING PYDES PACKAGE:

```
from pydes import des

key1 = "secret_k"

key2 = "secret_l"

text= "anirudhi"

d = des()

c1 = d.encrypt(key1,text)

c2 = d.decrypt (key2,c1)

ciphered=d.encrypt(key1,c2)

p1 = d.decrypt(key1,ciphered)

p2=d.encrypt(key2,p1)

plain=d.decrypt(key1,p2)

print "Ciphered: %r" % ciphered

print "Deciphered: ", plain
```

OUTPUT:

```
C:\Users\abhis\Desktop\New folder\BE 44 2ND SEM\infosec\des>C:\Python27\python.exe ddes.py  
CIPHERED: '\xd2Y30\xae\x9esk'  
Deciphered: anirudhi
```

```
C:\Users\abhis\Desktop\New folder\BE 44 2ND SEM\infosec\des>^S
```

CONCLUSION:

Triple DES has been successfully implemented.

AIM: Write C/JAVA program to implement RSA algorithm.

DESCRIPTION:

The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

ALGORITHM:

1. Enter 2 prime numbers p and q
2. Find product of 2 prime numbers to be N
3. Find the relative prime of (p-1)*(q-1) to be encryption key
4. Find decryption key by solving the equation
$$(K_e * K_d) \% ((p-1) * (q-1)) == 1$$
5. Encrypt with
$$C = M^{K_e} \% N$$
6. Decrypt with
$$M = C^{K_d} \% N$$

PROGRAM

```
#include<iostream>

#include<strings.h>

#include<math.h>

#include<stdlib.h>

using namespace std;

int gcd(int a, int b)

{

if (a == b)

return a;
```



```

if (a > b)
return gcd(a-b,b);
return gcd(a, b-a);
}

int main()
{
int p,i,m,q,j=0,x,z,n,e,d=0,public_key[2],private_key[2];

double c,f;

char ca[20],s[20];

cout<<"Enter 2 primes ";

cin>>p>>q;

n=p*q;

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

for(i=2;i<z;i++)
{
if(gcd(i,z)==1)
{
e=i;
break;
}
}

for(i=1;i<z;i++)
{
if(((i*e)%z)==1 &&e!=i)
{

```

```

d=i;

break;

}

}

public_key[0]=e;

public_key[1]=n;

private_key[0]=d;

private_key[1]=n;

i=0;

cout<<"enter the text ";

cin>>s;

while(s[i]!='\0')

{

m=s[i]-97+1;

c=fmod(pow(m,public_key[0]),n);

cout<<"encryption of "<<s[i]<<" is "<<c<<endl;

f=fmod(pow(c,private_key[0]),n);

ca[j++]=f+96;

i++;

}

cout<<"Decrypted Text is "<<ca;

return 0;

}

```

OUTPUT

```
enter the text
hello
encryption of h is 17.000000
encryption of e is 26.000000
encryption of l is 12.000000
encryption of l is 12.000000
encryption of o is 9.000000
hello
```

CONCLUSION:

RSA has been implemented successfully.

AIM: WAP to implement Blowfish Algorithm.

DESCRIPTION:

Blowfish is a symmetric encryption algorithm developed to replace DES. Blowfish is a 16-bit round feistel cipher. It's block size is 64 bit and key sizes range from 32 – 448 bits.

ENCRYPTION:

It has two main stages:

1. Round Function
2. Output Function

Round Function:

It has 4 stages

1. Key whitening of the left side of the input with the r^{th} round key.
2. Application of the s-boxes and combination of their results.
3. XOR the right side of the input with the o/p of the F function(key whitening, S- Boxes and combination of s-box o/p).
4. Swapping the sides of o/p.

Output Function:

The final stage of the blowfish cipher involves two steps. Reversing the final swap and performing o/p whitening. In o/p whitening the right side of the o/p is XORed with the 17th round key and the left side is XORed with the 18th round key.

PROGRAM:

```
from Crypto.Cipher import Blowfish
from Crypto import Random
from struct import pack
from os import urandom

cipher = Blowfish.new("cipherkey")
print ("the key is: cipherkey")
encrypted_data = cipher.encrypt("anirudhk")
print (encrypted_data)
print (cipher.decrypt(encrypted_data))
```

OUTPUT:

```
C:\Users\polab3-56\Desktop>python blowfish.py
the key is: cipherkey
b'\x8d\x0e\xe3\xa2\x8e5\x93\x1b'
b'anirudhk'
C:\Users\polab3-56\Desktop>
```

AIM: To implement SHA-1 hash

DESCRIPTION: In cryptography, SHA-1 is a cryptographic hash function which takes an input and produces a 160-bit hash value known as a message digest – typically rendered as a hexadecimal number, 40 digits long.

ALGORITHM:

- 1) Convert message into binary
- 2) Create 5 random hex strings. $h_0, h_1, h_2, \dots, h_4$ each string with 8 hex digits i.e. 32 bits of each h_0, h_1, \dots, h_4
- 3) Pad the message from step 1 by appending a single one on the right side 1 followed by 0s until the message is 448 bits
- 4) add the length of the string represented in 64 bits to the string obtained in 3.
- 5) Divide the input into 512 bit chunks
- 6) Divide each 512 bit chunks into 16 32-bit words say $w_0, w_1, w_2, \dots, w_{15}$
- 7) For each 512 bit chunk begin 80 iterations
 - 80 round requires 80 ws
 - Rounds 0 to 15 we can use $w_0, w_1, w_2, \dots, w_{15}$
 - Rounds 16 to 79 use the formula to calculate $w(i)$ $16 \leq i \leq 79$
$$w(i) = \text{circular_left_shift}(W(i-3) \text{ XOR } W(i-8) \text{ XOR } W(i-14) \text{ XOR } W(i-16))$$
- 8) Store the values from (2) in A, B, C, D, E respectively
- 9) for ($i=0; i < 79; i++$)
 - TEMP = LeftCircularShift(A, 5) + f(i, B, C, D) + E + W(i) + K(i)
 - E = D
 - D = C
 - C = LeftCircularShift(B, 30)
 - B = A
 - A = TEMP
- 10) $H_0 += A$
 - $H_1 += B$
 - $H_2 += C$
 - $H_3 += D$
 - $H_4 += E$
- 11) HashValue = LeftCircularShift($H_1, 128$) OR LeftCircularShift($H_1, 96$) OR LeftCircularShift($H_2, 64$) OR LeftCircularShift($H_3, 32$) OR H_4
 - for 1 to 19 $f(i, B, C, D) = (B \text{ AND } C) \text{ OR } (\sim B \text{ AND } D)$;
 - for 20 to 39 $f(i, B, C, D) = B \text{ XOR } C \text{ XOR } D$
 - for 40 to 59 $f(i, B, C, D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D)$
 - for 60 to 79 $f(i, B, C, D) = B \text{ XOR } C \text{ XOR } D$

k(i) values are 4 different values for each of the four stages

PROGRAM:

```
import hashlib  
  
inp=input('Enter text:').encode()  
h = hashlib.sha1(inp)  
print (h.hexdigest())
```

OUTPUT:

```
C:\Users\harshrathi\Desktop\INS>python SHA.py  
Enter text: harsh  
25a9fc4361f0de99d66f569f23dc0154e7064e35
```

AIM: To implement MD-5 hash

DESCRIPTION: The MD5 hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used for authenticating the original message. The MD5 hash function was originally designed for use as a secure cryptographic hash algorithm for authenticating digital signatures. MD5 has been deprecated for uses other than as a non-cryptographic checksum to verify data integrity and detect unintentional data corruption.

ALGORITHM:

- 1) The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words);
- 2) The message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 264.
- 3) The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants.
- 4) The main algorithm then uses each 512-bit message block in turn to modify the state.
- 5) The processing of a message block consists of four similar stages, termed rounds; each round is composed of 16 similar operations based on a non-linear function F, modular addition, and left rotation.

There are four possible functions, a different one is used in each round:

$$F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$$

$$G(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$$

$$H(B, C, D) = B \oplus C \oplus D$$

$$I(B, C, D) = C \oplus (B \vee \neg D)$$

PROGRAM:

```
import hashlib

result = hashlib.md5(input("Enter the data>>\n").encode())

print("Result of md5 encryption in digest format is : ",result.digest())

print("\nResult of md5 encryption in hex format is : ",result.hexdigest())
```


OUTPUT:

```
C:\Users\harshrathi\Desktop\INS>python MD5.py
Enter the data>>
harsh
Result of md5 encryption in digest format is : b'\xd4\xe3s\x0e\x8c\xba!0\x85\xcd\xda\xe5\xf93\x1dt'
Result of md5 encryption in hex format is : d4e3730e8cba214f85cddae5f9331d74
```

AIM: Write a program to implement digital certificate

DESCRIPTION: A digital certificate, also known as a public key certificate, is used to cryptographically link ownership of a public key with the entity that owns it. Digital certificates are for sharing public keys to be used for encryption and authentication. Digital certificates include the public key being certified, identifying information about the entity that owns the public key, metadata relating to the digital certificate and a digital signature of the public key created by the issuer of the certificate.

ALGORITHM:

- 1) Using crypto library create x.509 certificate
- 2) Create the signature with the details like name, locality, country
- 3) Digital certificate will be created

PROGRAM:

```
from OpenSSL import crypto, SSL

from socket import gethostname

from pprint import pprint

from time import gmtime, mktime

from os.path import exists, join

CERT_FILE = "x509.crt"

KEY_FILE = "x509.key"

def create_self_signed_cert(cert_dir):

    """

    If datacard.crt and datacard.key don't exist in cert_dir, create a new

    self-signed cert and keypair and write them into that directory.

    """

    if not exists(join(cert_dir, CERT_FILE)) \
```

```

    or not exists(join(cert_dir, KEY_FILE)):

# create a key pair

k = crypto.PKey()

k.generate_key(crypto.TYPE_RSA, 1024)    #You can use any cipher you want


# create a self-signed cert

cert = crypto.X509()

cert.get_subject().C = "IN"                #Country
cert.get_subject().ST = "TS"              #State
cert.get_subject().L = "HYDERABAD"        #Locality
cert.get_subject().O = "CBIT"             #Organisation
cert.get_subject().OU = "CSE"             #Organisational Unit
print(dir(cert.get_subject()))

cert.get_subject().CN = "ANIRUDH"         #Certificate Name

cert.set_serial_number(1000)              #Serial Number

cert.gmtime_adj_notBefore(0)              #Greenwich Meridian time not valid before

cert.gmtime_adj_notAfter(10*365*24*60*60) #Greenwich Meridian time not valid after

cert.set_issuer(cert.get_subject())

cert.set_pubkey(k)

cert.sign(k, 'sha256')                   #Signing with strong hash


#Write the certificate contents with the file headers as required

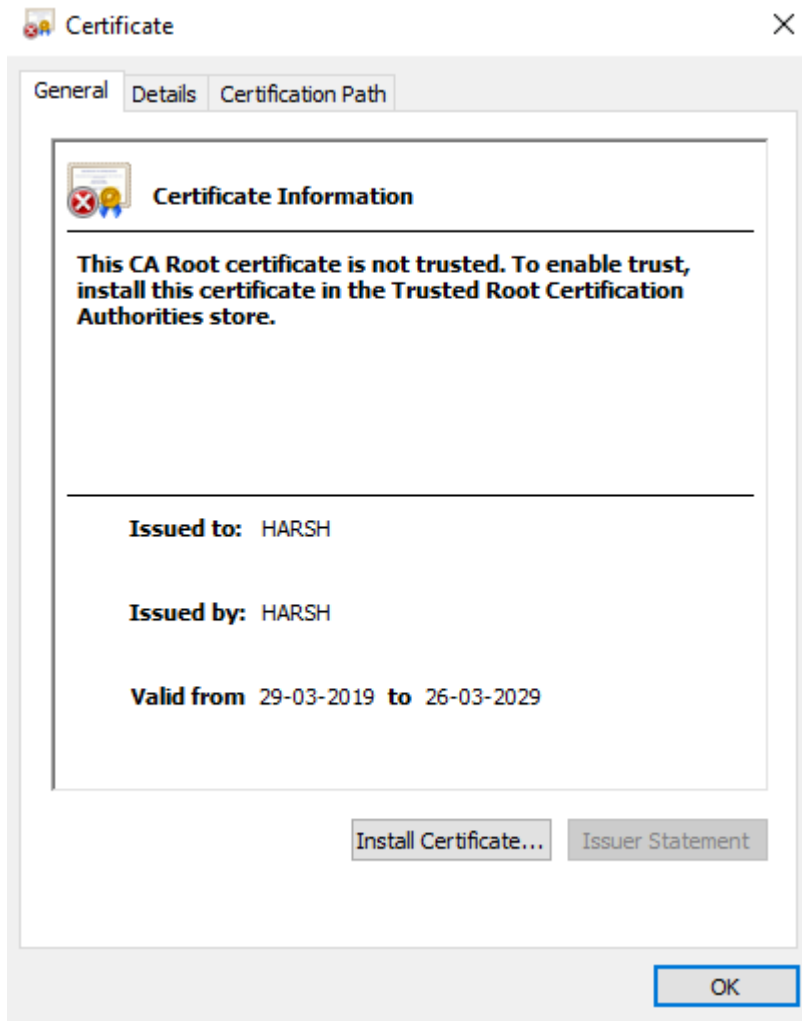
```

```
open(join(cert_dir, CERT_FILE), "wb").write(crypto.dump_certificate(crypto.FILETYPE_PEM, cert))

open(join(cert_dir, KEY_FILE), "wb").write( crypto.dump_privatekey(crypto.FILETYPE_PEM, k))

create_self_signed_cert(r"C:\Users\aniru\OneDrive\Desktop\New folder\BE 44 2ND SEM\infosec")
```

OUTPUT:



AIM: Write the Procedure to create a digital certificate of your own by using the

DESCRIPTION:

Java key tool

Generate digital certificate using keytool

1. Keytool is a utility for generating and managing cryptographic keys and certificates. Keytool is a part of Java installation, so you need to have Java on your computer to be able to use keytool.

2.

2. keytool" -genkeypair -alias my_certificate -keystore my_keystore.pfx -storepass my_password -validity 365 -keyalg RSA -keysize 2048 -storetype pkcs

Some information about parameters:

my_certificate is the alias for your certificate in the key store. Normally you will never use it, but every new certificate in your key store must have its own alias.

my_keystore.pfx is the key store file, which will be generated as the result of the process. It will contain your certificate and a corresponding private key. You will be able to reuse this key store for next certificates you maybe will generate. One key store can contain many certificates.

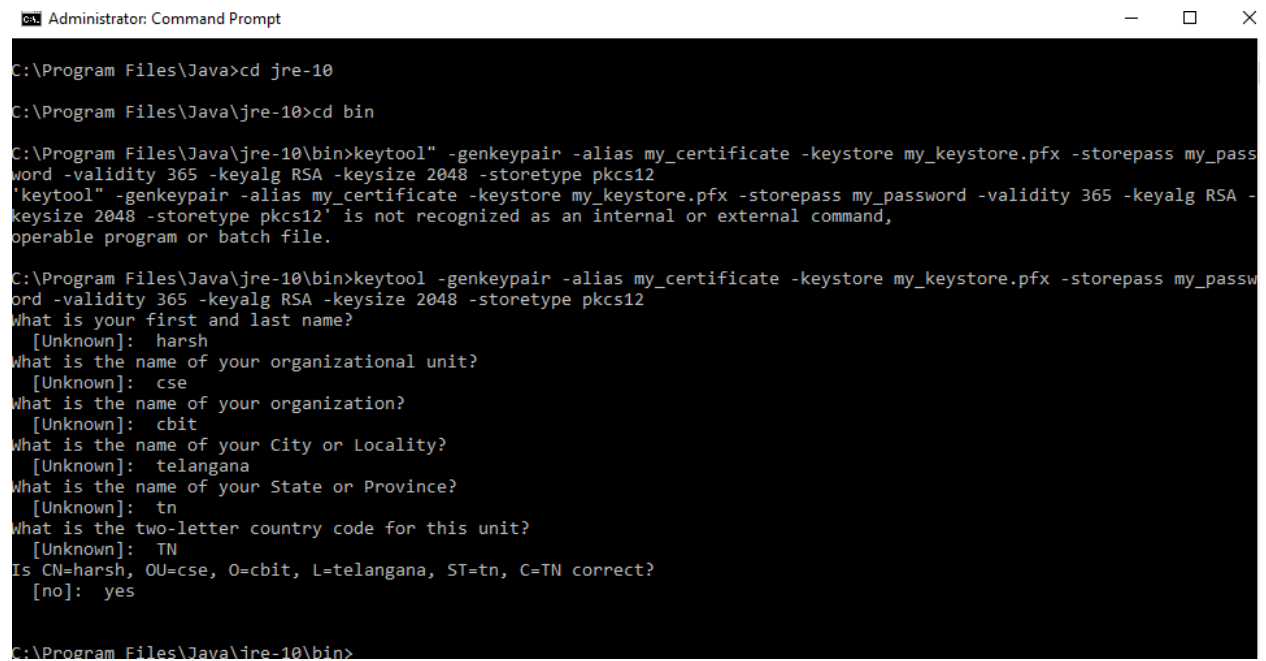
my_password is the password, that protects your key store file. You will have to enter it every time you want to sign a document. validity is the number of days your certificate will stay valid. You can enter more than 365.

RSA is the algorithm used to generate the cryptographic keys, corresponding to your certificate. 2048 is the length of the cryptographic keys. The more the length the stronger the signature.

pkcs12 is the format of the key store file. PKCS#12 (a.k.a PFX) key stores can be understood by a lot of different programs and you can also import a PKCS#12 file in your Windows key store (just double click it and follow the instructions). During the certificate generation process you will be prompted to enter some information about you. This information will be saved in your certificate. At the end you have to confirm the entered information.

At the end you will find the new key store file my_keystore.pfx in your current directory. You can register this key store with DigiSigner and use your certificate to sign PDF documents.

OUTPUT:



```
Administrator: Command Prompt

C:\Program Files\Java>cd jre-10

C:\Program Files\Java\jre-10>cd bin

C:\Program Files\Java\jre-10\bin>keytool" -genkeypair -alias my_certificate -keystore my_keystore.pfx -storepass my_password -validity 365 -keyalg RSA -keysize 2048 -storetype pkcs12
'keytool" -genkeypair -alias my_certificate -keystore my_keystore.pfx -storepass my_password -validity 365 -keyalg RSA -keysize 2048 -storetype pkcs12' is not recognized as an internal or external command,
operable program or batch file.

C:\Program Files\Java\jre-10\bin>keytool -genkeypair -alias my_certificate -keystore my_keystore.pfx -storepass my_password -validity 365 -keyalg RSA -keysize 2048 -storetype pkcs12
What is your first and last name?
[Unknown]: harsh
What is the name of your organizational unit?
[Unknown]: cse
What is the name of your organization?
[Unknown]: cbit
What is the name of your City or Locality?
[Unknown]: telangana
What is the name of your State or Province?
[Unknown]: tn
What is the two-letter country code for this unit?
[Unknown]: TN
Is CN=harsh, OU=cse, O=cbit, L=telangana, ST=tn, C=TN correct?
[no]: yes

C:\Program Files\Java\jre-10\bin>
```

AIM: Program to implement AES Algorithm

DESCRIPTION:

AES is an iterative rather than Feistel cipher. It is based on ‘substitution–permutation network’. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs and others involve shuffling bits around.

AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix.

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

ALGORITHM:

1. Algorithm:
KeyExpansion—round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more.
2. Initial round key addition :
 1. AddRoundKey—each byte of the state is combined with a block of the round key using bitwise XOR.
3. 9, 11 or 13 rounds:
 1. SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.
 2. ShiftRows—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
 3. MixColumns—a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
 4. AddRoundKey
4. Final round (making 10, 12 or 14 rounds in total):
 1. SubBytes
 2. ShiftRows
 3. AddRoundKey

Byte Substitution (SubBytes):

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.

Shiftrows:

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. Shift is carried out as follows –

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns:

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.

Addroundkey:

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

PROGRAM:

```
from Crypto.Cipher import AES
cipher = AES.new('This is a key123', AES.MODE_CBC, 'This is an IV456')
cipher_text = cipher.encrypt(input("Enter multiples of 16-bit"))
print(cipher_text)
cipher1 = AES.new('This is a key123', AES.MODE_CBC, 'This is an IV456')
plain_text = cipher1.decrypt(cipher_text)
print(plain_text)
```

OUTPUT:

```
Enter multiples of 16-bitataack on anish1
b'\xaa\xa9\x98\xb5\xecTm$\xc5\xc7{\xdb\x9b\x02\xba\x82'
b'ataack on anish1'
```