PRACTICAL-1

AIM:

The "Caesar Box," or "Caesar Cipher," is one of the earliest known ciphers. Developed around 100 BC, it was used by Julius Caesar to send secret messages to his generals in the field. In the event that one of his messages got intercepted, his opponent could not read them. This obviously gave him a great strategic advantage. Caesar shifted each letter of his message few letters to the right to produce what could be called the ciphertext. The ciphertext is what the enemy would see instead of the true message. So, for example, if Caesar's messages were written in the English alphabet, and shift by 3 then each letter "A" in the message would become a "D," the "B's" would become "E's," and the "X's" become "A's." This type of cipher is appropriately called a "shift cipher." Implement the cipher in any programming language of your choice. Perform encryption, decryption. Discuss and try some possible attacks on traditional Caesar cipher.

THEORY:

The Caesar Cipher technique is one of the Oldest and simplest method of encryption technique. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

It's simply a type of substitution cipher where each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. For example , if Caesar's messages were written in the English alphabet, and shift by 3 then each letter "A" in the message would become a "D," the "B's" would become "E's," and the "X's" become "A's."

The Biggest disadvantage of Caesar Cipher is that it is way too simple for today. Consisting of 26 possibilities - it is very limited in its cipher capabilities and thus not very effective.

Due to this we can easily decode the text by Bruteforce approach.

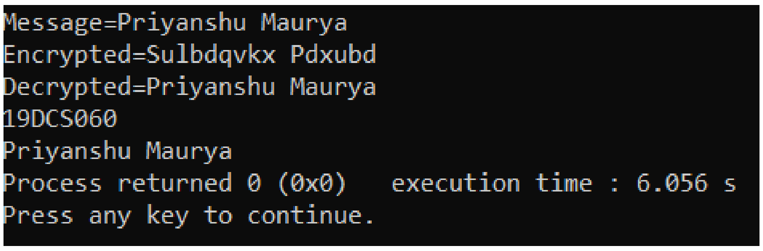
SOURCE CODE:

```
#include <iostream>


using namespace std;

void decrypt(char msg[]);

char* encrypt(char message[]){

    int i, key=3;

    char ch;

for(i = 0; message[i] != '\0'; ++i){ //traverse till eof

    ch = message[i];

    if(ch >= 'a' && ch <= 'z'){
```

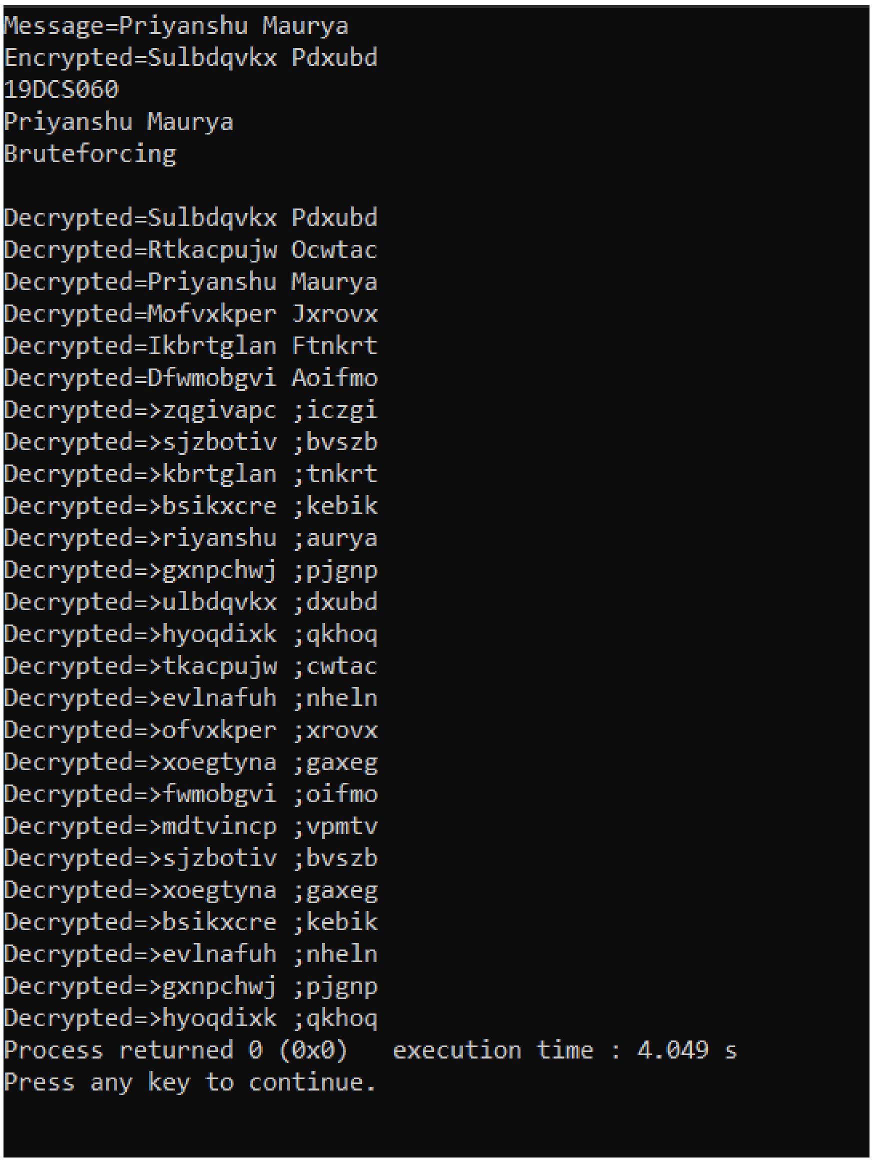
```
   ch = ch + key;
 //  cout<<"ch="<<ch<<endl;
   if(ch > 'z'){
       ch = ch - 'z' + 'a' - 1;
   //    cout<<"ch>z="<<ch<<endl;
   }
   message[i] = ch;
   }
   else if(ch >= 'A' && ch <= 'Z'){
   ch = ch + key;
   if(ch > 'Z'){
   ch = ch - 'Z' + 'A' - 1;
   }
   message[i] = ch;
}

}
cout << "Encrypted=" << message;
//decrypt(message);
return message;
}

char* decrypt(char message[],int key){
   char ch;
int i;

for(i = 0; message[i] != '\0'; ++i){
   ch = message[i];
   if(ch >= 'a' && ch <= 'z'){
   ch = ch - key;
   //cout<<endl<<"ch = ch - key;"<<int(ch);
   if(ch < 'a'){
   ch = ch + 'z' - 'a' + 1;
  // cout<<endl<<"ch<a="<<ch;
   }
   message[i] = ch;
   }
   else if(ch >= 'A' && ch <= 'Z'){
   ch = ch - key;
   if(ch > 'a'){
   ch = ch + 'Z' - 'A' + 1;
   }
   message[i] = ch;
}
}
cout <<endl<<"Decrypted=" << message;
   }
int main()
{
char message[100];
cout << "Message=";
cin.getline(message, 100);
char* x=encrypt(message);
//decrypt(x,3);
cout<<endl<<"19DCS060"<<endl<<"Priyanshu Maurya";
cout<<endl<<"Bruteforcing"<<endl;
for(int i=0;i<26;i++){
   char* y=decrypt(x,i);
   if(y==x){
       cout<<"Found key="<<i;
```

```
        break;
    }
}
return 0;
}
```

OUTPUT SCREENSHOT:



```
Message=Priyanshu Maurya
Encrypted=Sulbdqvkx Pdxubd
Decrypted=Priyanshu Maurya
19DCS060
Priyanshu Maurya
Process returned 0 (0x0)   execution time : 6.056 s
Press any key to continue.
```



```
Message=Priyanshu Maurya
Encrypted=Sulbdqvkx Pdxubd
19DCS060
Priyanshu Maurya
Bruteforcing

Decrypted=Sulbdqvkx Pdxubd
Decrypted=Rtkacpujw Ocwtac
Decrypted=Priyanshu Maurya
Decrypted=Mofvxkper Jxrovx
Decrypted=Ikbrtglan Ftnkrt
Decrypted=Dfwmobgvi Aoifmo
Decrypted=>zqgivapc ;iczgi
Decrypted=>sjzbotiv ;bvszb
Decrypted=>kbrtglan ;tnkrt
Decrypted=>bsikxcre ;kebik
Decrypted=>riyanshu ;aurya
Decrypted=>gxnpchwj ;pjgnp
Decrypted=>ulbdqvkx ;dxubd
Decrypted=>hyoqdixk ;qkhoq
Decrypted=>tkacpujw ;cwtac
Decrypted=>evlnafuh ;nheln
Decrypted=>ofvxkper ;xrovx
Decrypted=>xoegtyna ;gaxeg
Decrypted=>fwmobgvi ;oifmo
Decrypted=>mdtvincp ;vpmtv
Decrypted=>sjzbotiv ;bvszb
Decrypted=>xoegtyna ;gaxeg
Decrypted=>bsikxcre ;kebik
Decrypted=>evlnafuh ;nheln
Decrypted=>gxnpchwj ;pjgnp
Decrypted=>hyoqdixk ;qkhoq
Process returned 0 (0x0)   execution time : 4.049 s
Press any key to continue.
```

CONCLUSION:

In this practical I learned about a encryption technique called Ceaser Cipher which is one of the oldest encryption techniques and I also learned the flaws of the technique and how decoded the message by Bruteforcing.

PRACTICAL-2

AIM:

The Playfair cipher was predominantly used by British forces during the Second Boer War (1899-1902) and World War I (1914-1918). Soldier from field wants to send message to base. Implement the cipher to encrypt and decrypt message. Encrypt message : Hiroshima Use key : pearlharbour

THEORY:

The Playfair cipher was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.

For the encryption process we do the following:

1. Generate the key Square(5×5):
   In the square we have 25 squares but we have 26 alphabets so we consider I/J as one.
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.
3. 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)
   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).
   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.
4. For Decryption we do exact Vice versa of the Encryption rule i.e. if both letters are in same column we go up.
5. For e.g.
   For pearlharbour we get

   [p, e, a, r, l]

   [h, b, o, u, c]

   [d, f, g, i, k]

   [m, n, q, s, t]

   [v, w, x, y, z]

   Encrypted message: UDAUMUDSOA

SOURCE CODE:

```
import java.awt.Point;

import java.util.Scanner;

public class crnsp2
```

```
{

private int length = 0;

private String [][] table;

public static void main(String args[])
{
crnsp2 pf = new crnsp2();
}

private crnsp2()
{

System.out.print("Enter the key for playfair cipher: ");
Scanner sc = new Scanner(System.in);
String key = parseString(sc);
while(key.equals(""))
key = parseString(sc);
table = this.cipherTable(key);

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

String input = parseString(sc);
while(input.equals(""))
input = parseString(sc);

String output = cipher(input);
String decodedOutput = decode(output);
//output the results to user
```

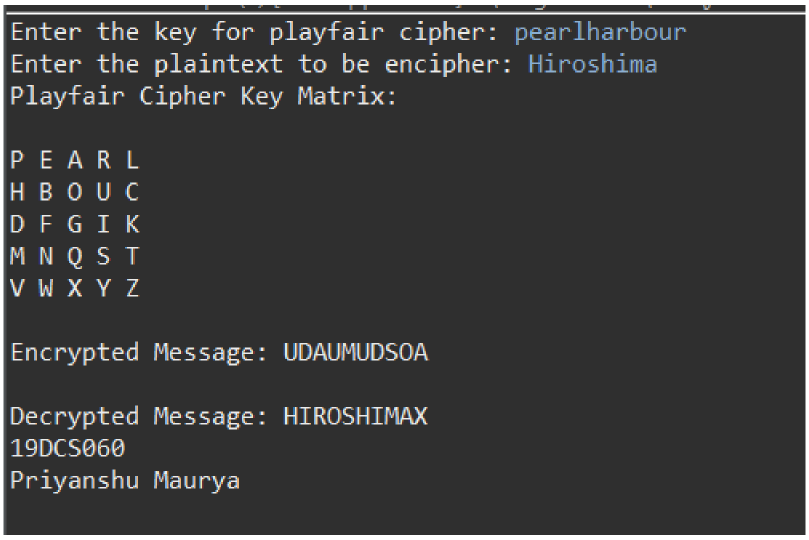
```
this.keyTable(table);
this.printResults(output,decodedOutput);
}

private String parseString(Scanner sc)
{
String parse = sc.nextLine();

parse = parse.toUpperCase();

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

parse = parse.replace("J", "I");
return parse;
}

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

String[][] playfairTable = new String[5][5];
String keyString = key + "ABCDEFGHIKLMNOPQRSTUVWXYZ";

for(int i = 0; i < 5; i++)
for(int j = 0; j < 5; j++)
playfairTable[i][j] = "";
for(int k = 0; k < keyString.length(); k++)
{
boolean repeat = false;
boolean used = false;
for(int i = 0; i < 5; i++)
{
for(int j = 0; j < 5; j++)
{
if(playfairTable[i][j].equals("" + keyString.charAt(k)))
{
repeat = true;
}
else if(playfairTable[i][j].equals("") && !repeat && !used)
{
playfairTable[i][j] = "" + keyString.charAt(k);
used = true;
}
}
}
}
return playfairTable;
}

private String cipher(String in)
{
length = (int) in.length() / 2 + in.length() % 2;

for(int i = 0; i < (length - 1); i++)
{
if(in.charAt(2 * i) == in.charAt(2 * i + 1))
{
in = new StringBuffer(in).insert(2 * i + 1, 'X').toString();
length = (int) in.length() / 2 + in.length() % 2;
}
}
```

```
String[] digraph = new String[length];
//loop iterates over the plaintext
for(int j = 0; j < length ; j++)
{

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

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

String out = "";
String[] encDigraphs = new String[length];
encDigraphs = encodeDigraph(digraph);
for(int k = 0; k < length; k++)
out = out + encDigraphs[k];
return out;
}

private String[] encodeDigraph(String di[])
{
String[] encipher = new String[length];
for(int i = 0; i < length; i++)
{
char a = di[i].charAt(0);
char b = di[i].charAt(1);
int r1 = (int) getPoint(a).getX();
int r2 = (int) getPoint(b).getX();
int c1 = (int) getPoint(a).getY();
int c2 = (int) getPoint(b).getY();

if(r1 == r2)
{
c1 = (c1 + 1) % 5;
c2 = (c2 + 1) % 5;
}

else if(c1 == c2)
{
r1 = (r1 + 1) % 5;
r2 = (r2 + 1) % 5;
}

else
{
int temp = c1;
c1 = c2;
c2 = temp;
}

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

private String decode(String out)
{
String decoded = "";
for(int i = 0; i < out.length() / 2; i++)
```

```
{
char a = out.charAt(2*i);
char b = out.charAt(2*i+1);
int r1 = (int) getPoint(a).getX();
int r2 = (int) getPoint(b).getX();
int c1 = (int) getPoint(a).getY();
int c2 = (int) getPoint(b).getY();
if(r1 == r2)
{
c1 = (c1 + 4) % 5;
c2 = (c2 + 4) % 5;
}
else if(c1 == c2)
{
r1 = (r1 + 4) % 5;
r2 = (r2 + 4) % 5;
}
else
{
int temp = c1;
c1 = c2;
c2 = temp;
}
decoded = decoded + table[r1][c1] + table[r2][c2];
}

return decoded;
}

private Point getPoint(char c)
```

```
{
Point pt = new Point(0,0);
for(int i = 0; i < 5; i++)
for(int j = 0; j < 5; j++)
if(c == table[i][j].charAt(0))
pt = new Point(i,j);
return pt;
}

private void keyTable(String[][] printTable)
{
System.out.println("Playfair Cipher Key Matrix: ");
System.out.println();

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

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

System.out.print(printTable[i][j]+" ");
}
System.out.println();
}
System.out.println();
}
//method that prints all the results
private void printResults(String encipher, String dec)
{
System.out.print("Encrypted Message: ");
//prints the encrypted message
System.out.println(encipher);
System.out.println();
System.out.print("Decrypted Message: ");
//prints the decryted message
System.out.println(dec);
System.out.println("19DCS060\nPriyanshu Maurya");
}
```

OUTPUT SCREENSHOT:



```
Enter the key for playfair cipher: pearlharbour
Enter the plaintext to be encipher: Hiroshima
Playfair Cipher Key Matrix:

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

Encrypted Message: UDAUMUDSOA

Decrypted Message: HIROSHIMAX
19DCS060
Priyanshu Maurya
```

CONCLUSION:

In this practical I learned about a encryption technique called Playfair Cipher.I also learned how to make key square and also three rules for encryption and decryption.

PRACTICAL-3

AIM:

The Rail Fence Cipher was invented in ancient times. It was used by the Greeks, who created a special tool, called scytale, to make message encryption and decryption easier. The letters are arranged in a way which is similar to the shape of the top edge of the rail fence. If king Leonidas want to send message to Sparta as “300 achieved glory at hot gate, unite for Greece” then what will be ciphertext when it is encrypted using 3 rows. Also implement decryption of message.

THEORY:

The rail fence cipher (also called a zigzag cipher) is a form of classical transposition cipher. It derives its name from the manner in which encryption is performed.

In the rail fence cipher, the plaintext is written downwards diagonally on successive "rails" of an imaginary fence, then moving up when the bottom rail is reached, down again when the top rail is reached, and so on until the whole plaintext is written out. The ciphertext is then read off in rows.

For example:

For “300 achieved glory at hot gate, unite for Greece” when we encrypt the message for 3 rows

3				C				V				L				A				T				E				I				O				E			
	0		A		H		E		E		G		O		Y		T		O		G		T		,		N		T		F		R		R		E		E
		0				I				D				R				H				A				U				E				G				C	

Encrypted=3ilheee0hegotot,tfre0cvdratauioGcaeygnre

Decrypted=300achievedgloryathotgate,uniteforGreece

The advantage of the Rail Fence cipher over other transposition ciphers like the sawtooth cipher is that there is a variable distance between consecutive letters. What we mean by variable distance is that the letters need not be arranged in fixed vertical columns that descends, but it can also be arranged in a zig zag manner. Therefore, this increases the difficulty of cracking the code.

One of the problems that the rail fence cipher face is that the security of the code is dependant on the fact that a cryptanalyst does not know the method of encryption. Hence, once the method of encryption is broken, the code is broken already.

SOURCE CODE:

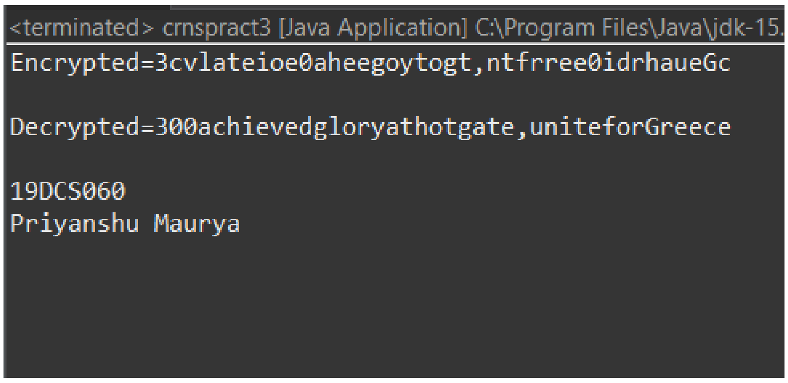
```
public class crnspract3 {


	public static void main(String[] args) {
		// TODO Auto-generated method stub
		crnspract3 p=new crnspract3();
		String str="300 achieved glory at hot gate, unite for Greece";
```

```
            str=str.replaceAll("\\s","");//removing white spaces
            System.out.println(str.length());
            String encrypted=p.encryptRailFence(str, 4);
            System.out.println("Encrypted="+encrypted);
            String decrypted=p.decryptRailFence(encrypted, 4);
            System.out.println("\nDecrypted="+decrypted);
            System.out.println("\n19DCS060\nPriyanshu Maurya");

    }
    String encryptRailFence(String text, int key)
    {
            char rail[][]=new char[key][(text.length())];

        // filling the rail matrix to distinguish filled
        // spaces from blank ones
        for (int i=0; i < key; i++)
            for (int j = 0; j < text.length(); j++)
                rail[i][j] = '\n';

        boolean dir_down = false;
        int row = 0, col = 0;

        for (int i=0; i < text.length(); i++)
        {
            // check the direction of flow
            // reverse the direction if we've just
            // filled the top or bottom rail
            if (row == 0 || row == key-1)
                dir_down = !dir_down;//we change dir down when either we are at end row of
matrix or first
                //System.out.println("I="+i+" Row="+row+" Dir="+dir_down);
            // fill the corresponding alphabet
//              rail[row][col++] = text[i];
            rail[row][col++] = text.charAt(i);
            //System.out.println("Row="+row+" COl="+col);
            // find the next row using direction flag
//              dir_down?row++ : row--;
            if(dir_down) {
                row++;
            }
            else {
                row--;
            }
        }

        //now we can construct the cipher using the rail matrix
        String result="";
        for (int i=0; i < key; i++)
            for (int j=0; j < text.length(); j++) {
                //System.out.print(rail[i][j]);
                if (rail[i][j]!='\n')
                    //result.push_back(rail[i][j]);
                                result=result+rail[i][j];}

        return result;
    }
//      String decryptRailFence(String text, int key) {
//
//      }
    String decryptRailFence(String cipher, int key)
```

```
    {
       // create the matrix to cipher plain text
       // key = rows , length(text) = columns
              char rail[][]=new char[key][(cipher.length())];

       // filling the rail matrix to distinguish filled
       // spaces from blank ones
       for (int i=0; i < key; i++)
           for (int j=0; j < cipher.length(); j++)
               rail[i][j] = '\n';

       // to find the direction
       boolean dir_down=true;

       int row = 0, col = 0;

       // mark the places with '*'
       for (int i=0; i < cipher.length(); i++)
       {
           // check the direction of flow
           if (row == 0)
               dir_down = true;
           if (row == key-1)
               dir_down = false;

           // place the marker
           rail[row][col++] = '*';

           // find the next row using direction flag
          // dir_down?row++ : row--;
           if(dir_down) {
               row++;
           }
           else {
               row--;
           }
       }

       // now we can construct the fill the rail matrix
       int index = 0;
       for (int i=0; i<key; i++)
           for (int j=0; j<cipher.length(); j++)
               if (rail[i][j] == '*' && index<cipher.length())
                   rail[i][j] = cipher.charAt(index++);



       // now read the matrix in zig-zag manner to construct
       // the resultant text
       String result="";

       row = 0;
       col = 0;
       for (int i=0; i< cipher.length(); i++)
       {
           // check the direction of flow
           if (row == 0)
               dir_down = true;
           if (row == key-1)
               dir_down = false;
```

```
                // place the marker
                if (rail[row][col] != '*') {
                    result=result+(rail[row][col++]);}

                // find the next row using direction flag
                //dir_down?row++: row--;
                if(dir_down) {
                    row++;
                }
                else {
                    row--;
                }
            }
            return result;
        }

}
```

OUTPUT SCREENSHOT:



```
<terminated> crnspract3 [Java Application] C:\Program Files\Java\jdk-15.
Encrypted=3cvlateioe0aheegoytogt,ntfrree0idrhaueGc

Decrypted=300achievedgloryathotgate,uniteforGreece

19DCS060
Priyanshu Maurya
```

CONCLUSION:

In this practical I learned about a encryption technique called Rail Fence cipher which writes plaintext diagonally and reads it rowwise.I also learned how to decrypt by following the reverse approach.

PRACTICAL-4

AIM:

RSA algorithm is used by Salim to transfer session key to Anarkali. He suspects that Akbar is performing man in middle attack he chose to use 1024 bit prime numbers. Hint: you may choose to use big integer in java.

THEORY:

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and Private key is kept private.

Here,

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Steps:

1. Generating the keys

1. Select two large prime numbers, x*x* and y*y*. The prime numbers need to be large so that they will be difficult for someone to figure out.
2. Calculate n =x*y.
3. Calculate the ***totient*** function; phi(n) = (x-1)(y-1)=$\phi(n)=(x-1)(y-1)$.
4. Select an integer *e*, such that *e* is ***co-prime*** to \phi(n)$\phi(n)$ and 1 < e < phi(n)$1<e<\phi(n)$. The pair of numbers (*n*,*e*) makes up the public key.
5. Calculate d*d* such that e.d = 1$e.d$=1 %phi(*n*).

2. Encryption

Given a plaintext *P*, represented as a number, the ciphertext *C* is calculated as:

C = P^{e}*%n*.

3. Decryption

Using the private key (*n*,*d*), the plaintext can be found using:

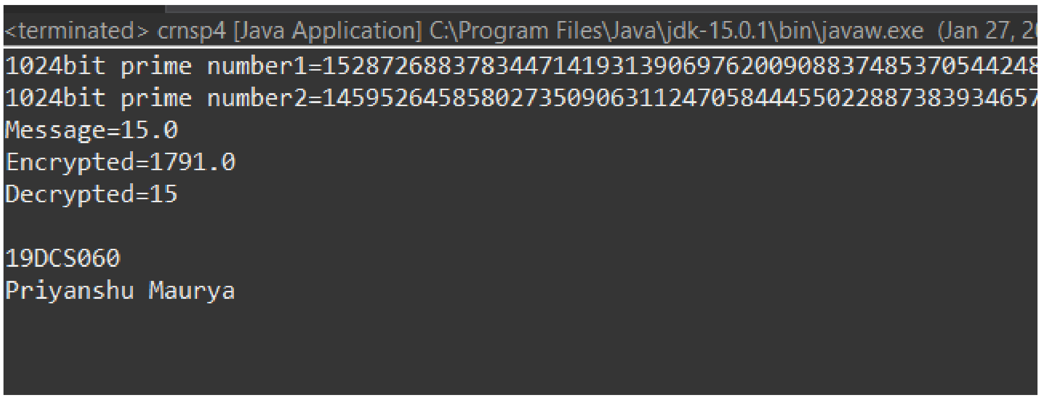
P = C^{d}*%n*.

SOURCE CODE:

```
import java.lang.Math;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Random;
public class crnsp4 {
        int gcd(int a, int b)
        {
            // Everything divides 0
            if (a == 0)
              return b;
            if (b == 0)
              return a;

            // base case
            if (a == b)
               return a;

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

        public static BigInteger largePrime(int bits) {
                Random randomInteger = new Random();
                BigInteger largePrime = BigInteger.probablePrime(bits, randomInteger);
                return largePrime;
        }

        public static void main(String[] args) {
                BigInteger p1=largePrime(1024);
            BigInteger p2=largePrime(1024);

            System.out.println("1024bit prime number1="+p1);
            System.out.println("1024bit prime number2="+p2);
//                 BigInteger a=largePrime(1024);
//          BigInteger b=largePrime(1024);
                // TODO Auto-generated method stub
                crnsp4 p4 =new crnsp4();
                double message=15;
//              double str1 = Double.parseDouble(message);

                System.out.println("Message="+message);
                int a=61;
                int b=53;
//              BigInteger m1=new BigInteger("-1");
//              BigInteger n=a.multiply(b);
//              BigInteger euler=(a.subtract(m1)).multiply((b.subtract(m1)));
```

```
			int n=a*b;
			int euler=(a-1)*(b-1);
			int enc=2,temp;
			while(enc<euler) {
				temp=p4.gcd(enc,euler);
				if(temp==1) {
					break;
				}
				else {
					enc++;
				}
			}
			//System.out.println("Enc="+enc);
			int d=0;
			for(int i=0;i<=9;i++) {
				int x=1+(i*euler);
				if(x%enc==0) {
					d=x/enc;
					break;
				}
			}
			double c=Math.pow(message, enc)%n;

			System.out.println("Encrypted="+c);

			//double m=c.pow(d).mod(n);
			 BigInteger C = BigDecimal.valueOf(c).toBigInteger();
			BigInteger N = BigInteger.valueOf(n);
			 BigInteger msgback = (C.pow(d)).mod(N);
			// k = 2;  // A constant value
		  //  double d1 = (1 + (k*euler))/enc;
		//		double m=Math.pow(c, d1)%n;
		    System.out.println("Decrypted="+msgback);

		    System.out.println("\n19DCS060\nPriyanshu Maurya");


		}

	}
```

OUTPUT SCREENSHOT:



```
<terminated> crnsp4 [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe  (Jan 27, 2
1024bit prime number1=15287268837834471419313906976200908837485370544248
1024bit prime number2=14595264585802735090631124705844455022887383934657
Message=15.0
Encrypted=1791.0
Decrypted=15

19DCS060
Priyanshu Maurya
```

CONCLUSION:

In this practical I learned about a encryption algorithm called RSA algorithm, which is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and Private key is kept private and Encrypted and decrypted message using it.

PRACTICAL-5

AIM:

The transmission of information needs to be secure over the communication channel and the data has to be confidential. Study and implement the practical approach for Steganography.

-Using DOS commands

-Using OpenPuff Tool

THEORY:

Steganography is the practice of concealing a message within another message or a physical object. In computing/electronic contexts, a computer file, message, image, or video is concealed within another file, message, image, or video.

The word *steganography* comes from Greek *steganographia*, which combines the words *steganós* (στεγανός), meaning "covered or concealed", and *-graphia* (γραφή) meaning "writing".

Steganography has been used for centuries,for example if a friend of mine sent me a steganographic message – a secret message embedded within an image. With my own copy of Steghide I used the command sequence to extract that secret message. I then read can it using the cat command.

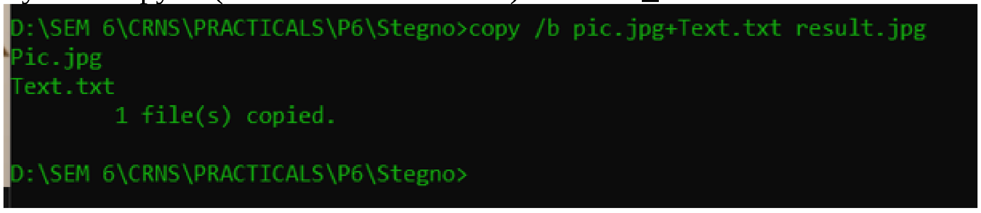
There are different ways to hide a message. When a file or image is created, some bytes in the file or image are not necessary and can be replaced with a message without destroying the original message. In this way the secret message is hidden. There are different types of steganography like.

1. Steganography in Images
2. Steganography in Audio
3. Steganography in Video
4. Steganography in Documents

OUTPUT SCREENSHOT:

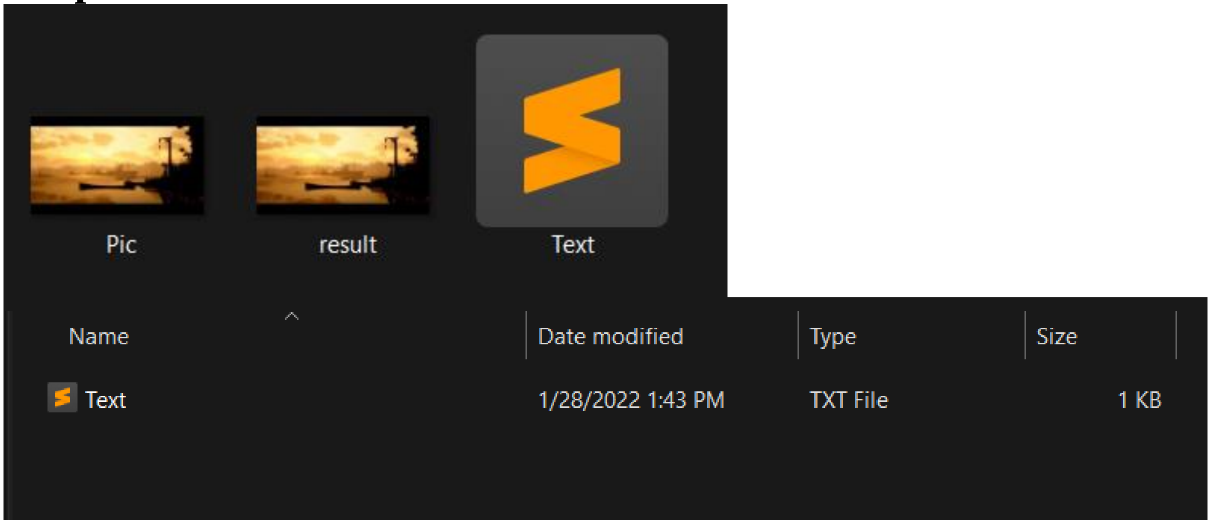
1> Using DOS commands:

We can use DOS command to encrypt the image
Syntex: copy /b (filename+encfilename) resultfile_name



```
D:\SEM 6\CRNS\PRACTICALS\P6\Stegno>copy /b pic.jpg+Text.txt result.jpg
Pic.jpg
Text.txt
        1 file(s) copied.

D:\SEM 6\CRNS\PRACTICALS\P6\Stegno>
```

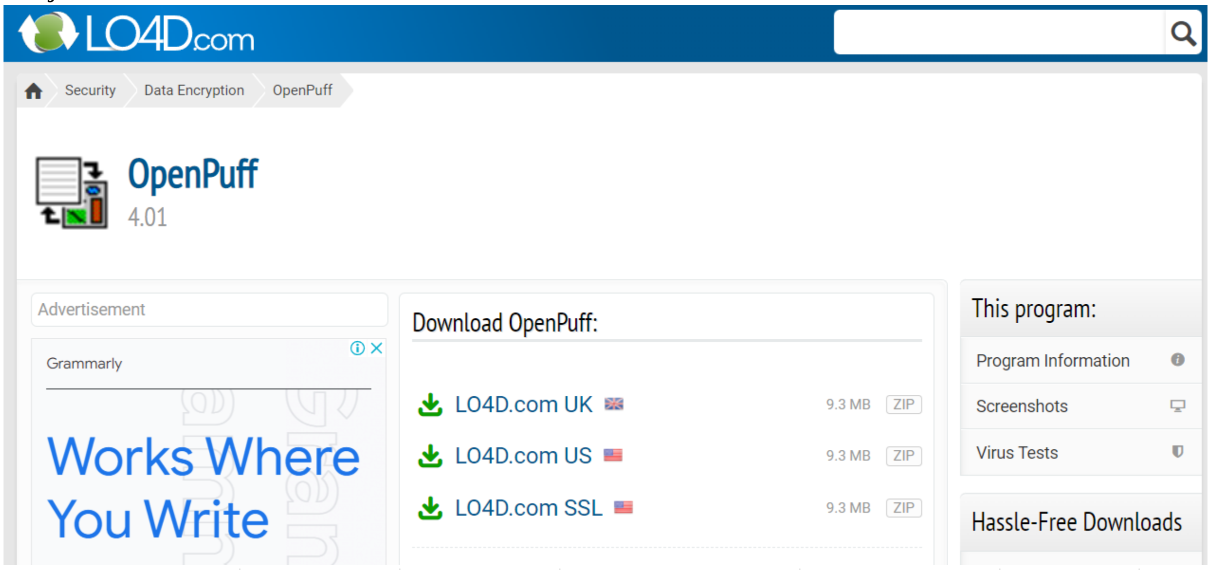
Output:

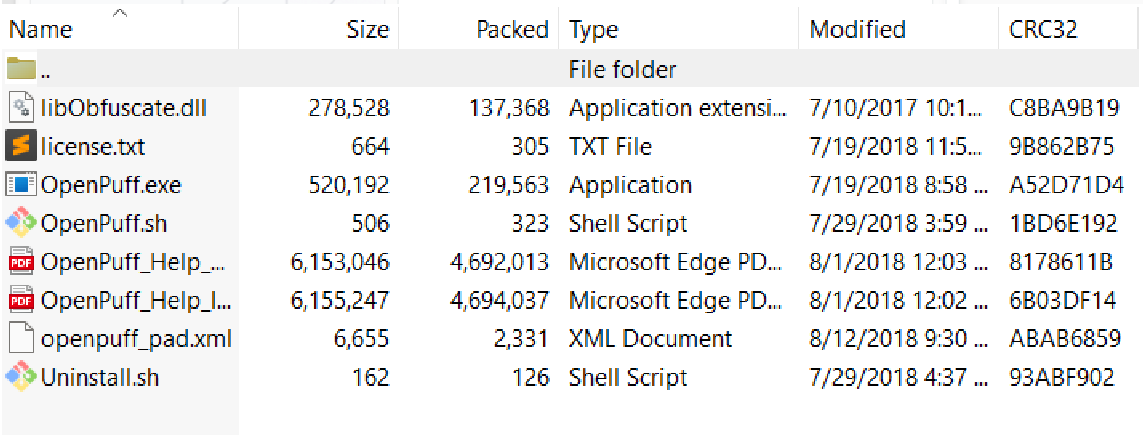


Name	Date modified	Type	Size
Text	1/28/2022 1:43 PM	TXT File	1 KB

2>Using OpenPuff tool

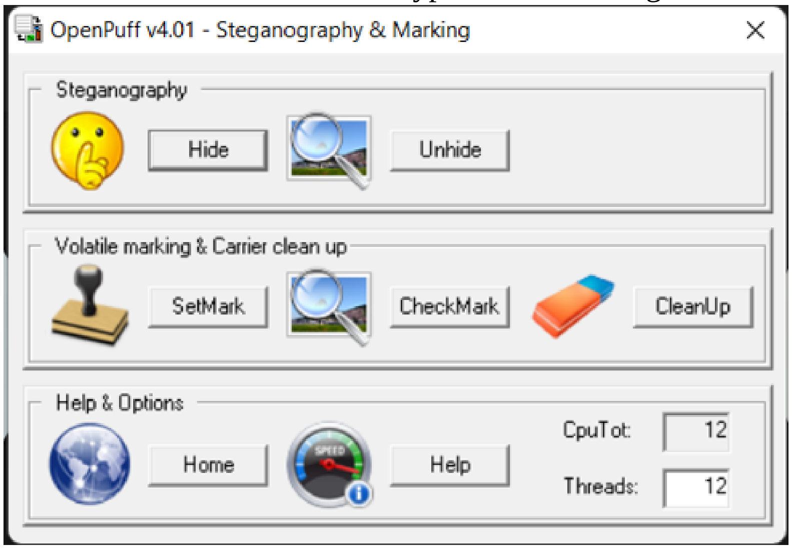
1. We download and the openpuff tool from the internet it does not require any installion we just extract files.





Name	Size	Packed	Type	Modified	CRC32
..			File folder		
libObfuscate.dll	278,528	137,368	Application extensi...	7/10/2017 10:1...	C8BA9B19
license.txt	664	305	TXT File	7/19/2018 11:5...	9B862B75
OpenPuff.exe	520,192	219,563	Application	7/19/2018 8:58 ...	A52D71D4
OpenPuff.sh	506	323	Shell Script	7/29/2018 3:59 ...	1BD6E192
OpenPuff_Help_...	6,153,046	4,692,013	Microsoft Edge PD...	8/1/2018 12:03 ...	8178611B
OpenPuff_Help_I...	6,155,247	4,694,037	Microsoft Edge PD...	8/1/2018 12:02 ...	6B03DF14
openpuff_pad.xml	6,655	2,331	XML Document	8/12/2018 9:30 ...	ABAB6859
Uninstall.sh	162	126	Shell Script	7/29/2018 4:37 ...	93ABF902

2. We click on hide button to Encrypt the text in image


OpenPuff v4.01 - Steganography & Marking

Steganography: Hide | Unhide

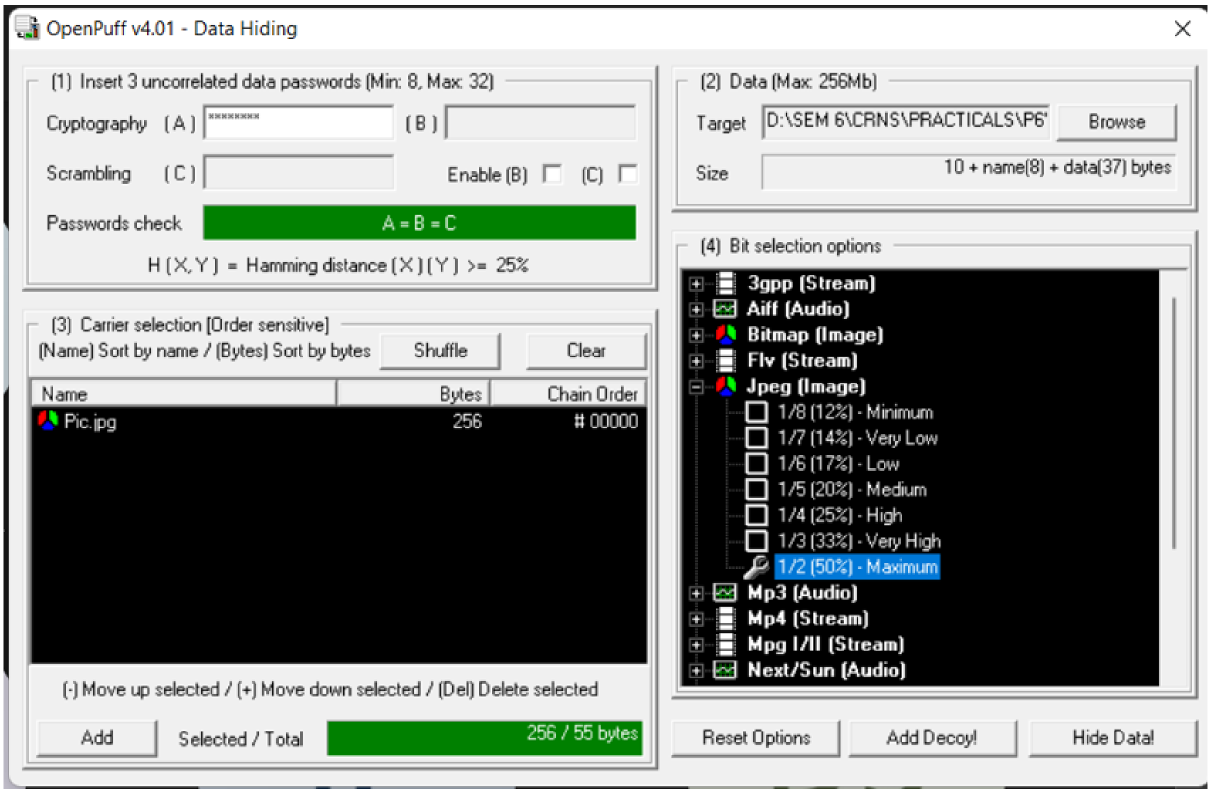
Volatile marking & Carrier clean up: SetMark | CheckMark | CleanUp

Help & Options: Home | Help

CpuTot: 12

Threads: 12

3. We have to give password to encrypt and select the Respective file we want to apply steganography to and select the format of file.


OpenPuff v4.01 - Data Hiding

(1) Insert 3 uncorrelated data passwords (Min: 8, Max: 32)

Cryptography (A) ******** (B)

Scrambling (C) Enable (B) ☐ (C) ☐

Passwords check: A = B = C

H(X,Y) = Hamming distance (X)(Y) >= 25%

(2) Data (Max: 256Mb)

Target: D:\SEM 6\CRNS\PRACTICALS\P6' Browse

Size: 10 + name(8) + data(37) bytes

(3) Carrier selection [Order sensitive]

(Name) Sort by name / (Bytes) Sort by bytes Shuffle Clear

Name	Bytes	Chain Order
Pic.jpg	256	# 00000

(-) Move up selected / (+) Move down selected / (Del) Delete selected

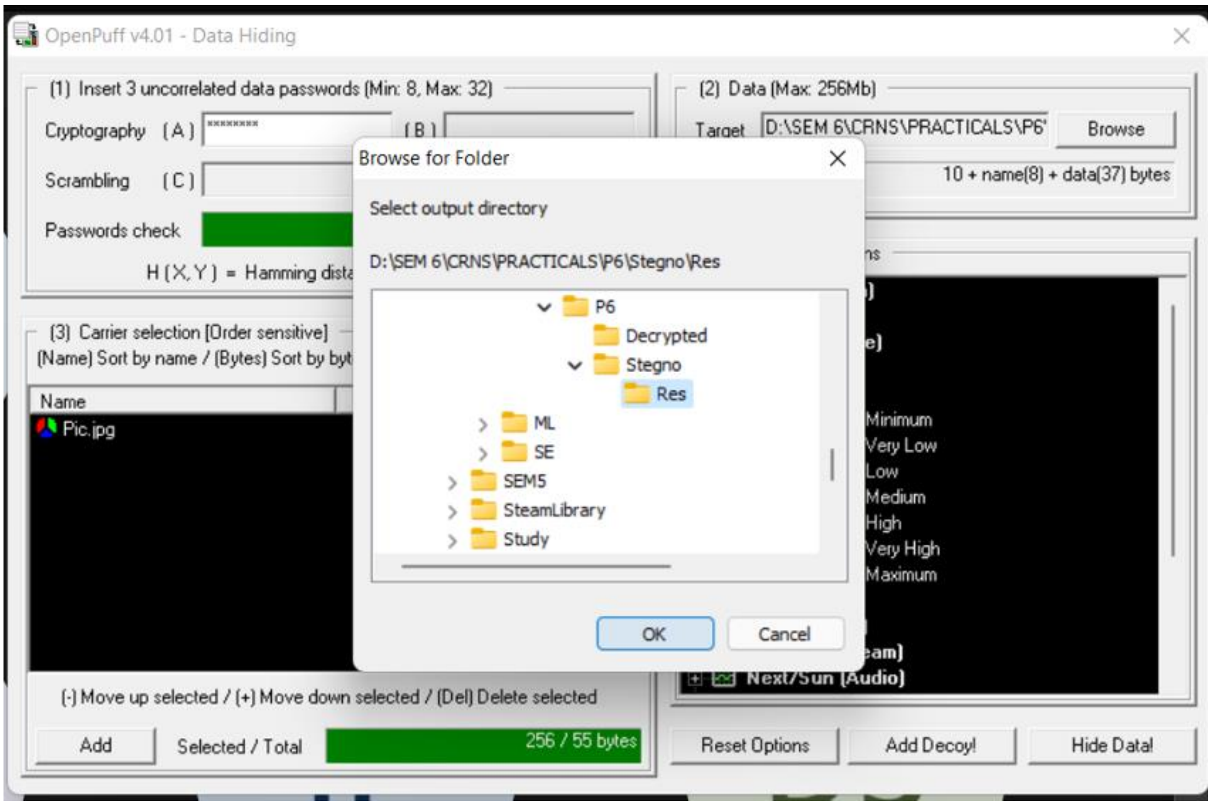
Add Selected / Total 256 / 55 bytes

(4) Bit selection options

- 3gpp (Stream)
- Aiff (Audio)
- Bitmap (Image)
- Flv (Stream)
- Jpeg (Image)
  - 1/8 (12%) - Minimum
  - 1/7 (14%) - Very Low
  - 1/6 (17%) - Low
  - 1/5 (20%) - Medium
  - 1/4 (25%) - High
  - 1/3 (33%) - Very High
  - 1/2 (50%) - Maximum
- Mp3 (Audio)
- Mp4 (Stream)
- Mpg I/II (Stream)
- Next/Sun (Audio)

Reset Options | Add Decoy! | Hide Data!

4.


OpenPuff v4.01 - Data Hiding

(1) Insert 3 uncorrelated data passwords (Min: 8, Max: 32)
Cryptography (A) ********* (B)
Scrambling (C)
Passwords check
H(X,Y) = Hamming dista

(2) Data (Max: 256Mb)
Target D:\SEM 6\CRNS\PRACTICALS\P6' Browse
10 + name(8) + data(37) bytes

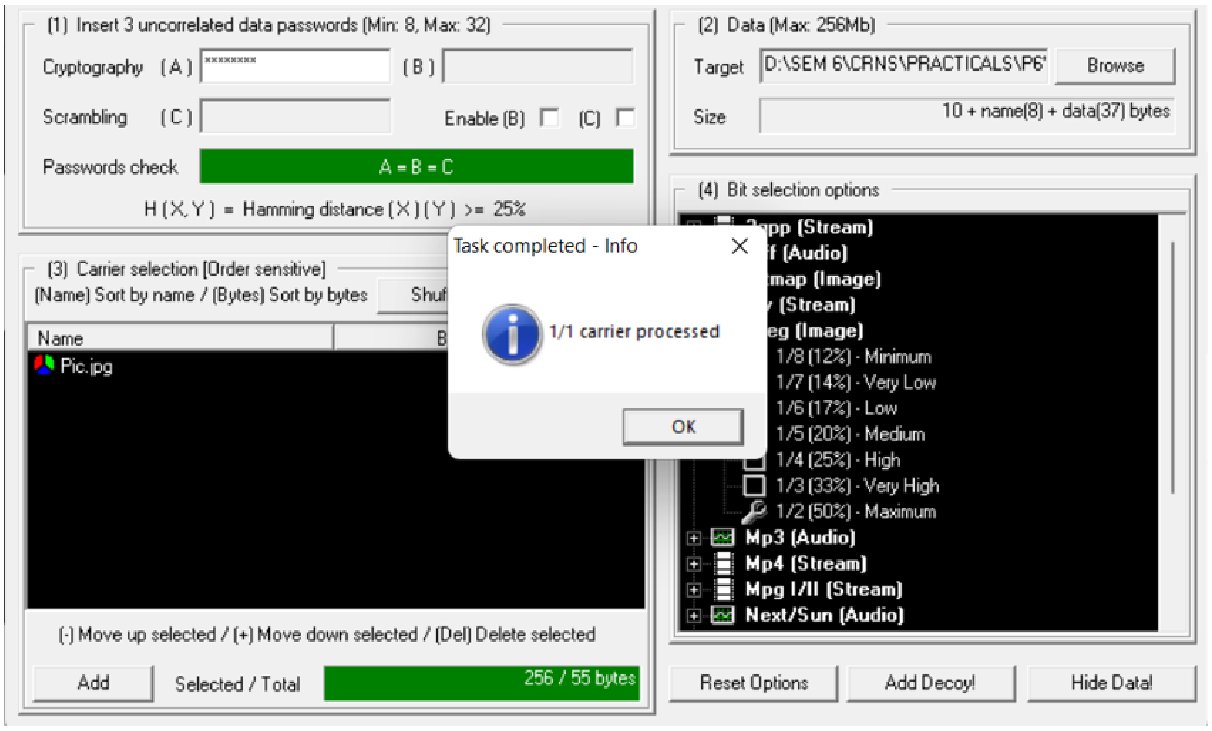
(3) Carrier selection [Order sensitive]
(Name) Sort by name / (Bytes) Sort by byt
Name
Pic.jpg

Browse for Folder
Select output directory
D:\SEM 6\CRNS\PRACTICALS\P6\Stegno\Res
P6
Decrypted
Stegno
Res
ML
SE
SEM5
SteamLibrary
Study
OK Cancel

Minimum
Very Low
Low
Medium
High
Very High
Maximum
Next/Sun (Audio)

(-) Move up selected / (+) Move down selected / (Del) Delete selected
Add Selected / Total 256 / 55 bytes
Reset Options Add Decoy! Hide Data!

5.


(1) Insert 3 uncorrelated data passwords (Min: 8, Max: 32)
Cryptography (A) ********* (B)
Scrambling (C) Enable (B) ☐ (C) ☐
Passwords check A = B = C
H(X,Y) = Hamming distance (X)(Y) >= 25%

(2) Data (Max: 256Mb)
Target D:\SEM 6\CRNS\PRACTICALS\P6' Browse
Size 10 + name(8) + data(37) bytes

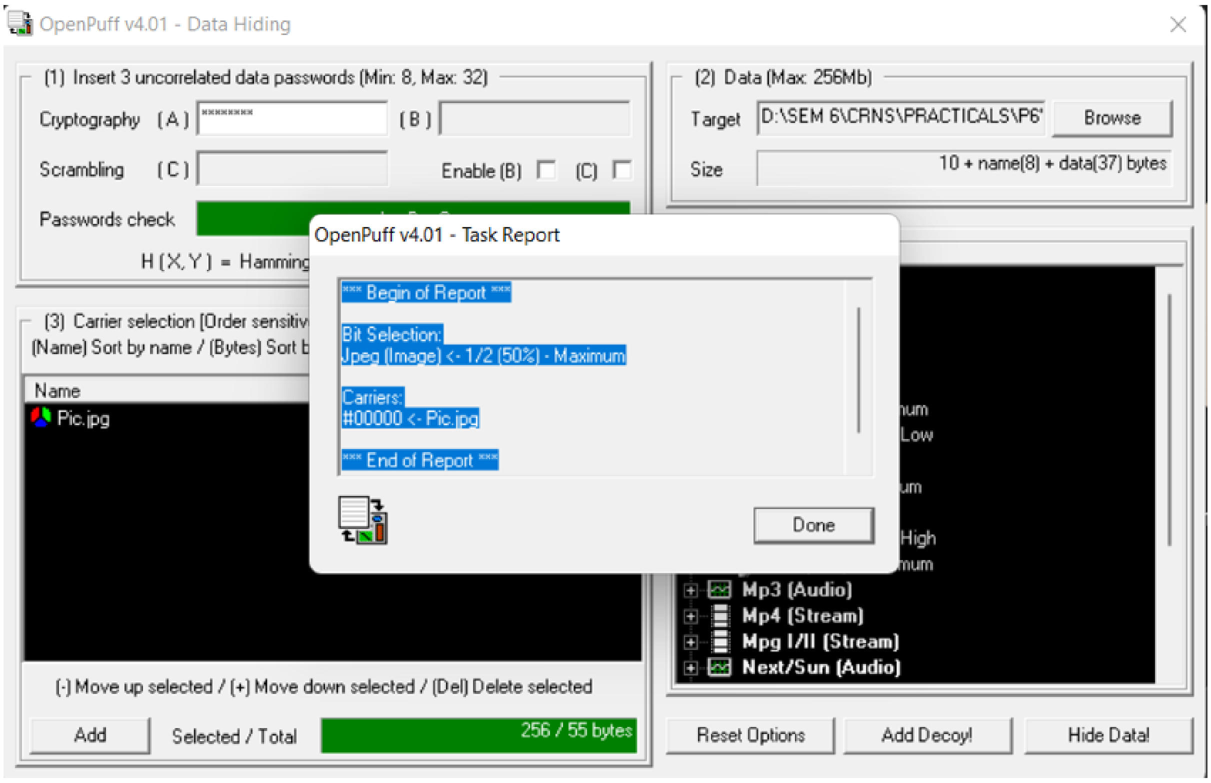
(3) Carrier selection [Order sensitive]
(Name) Sort by name / (Bytes) Sort by bytes Shuf
Name B
Pic.jpg

Task completed - Info
1/1 carrier processed
OK

(4) Bit selection options
pp (Stream)
f (Audio)
map (Image)
(Stream)
eg (Image)
1/8 (12%) - Minimum
1/7 (14%) - Very Low
1/6 (17%) - Low
1/5 (20%) - Medium
1/4 (25%) - High
1/3 (33%) - Very High
1/2 (50%) - Maximum
Mp3 (Audio)
Mp4 (Stream)
Mpg I/II (Stream)
Next/Sun (Audio)

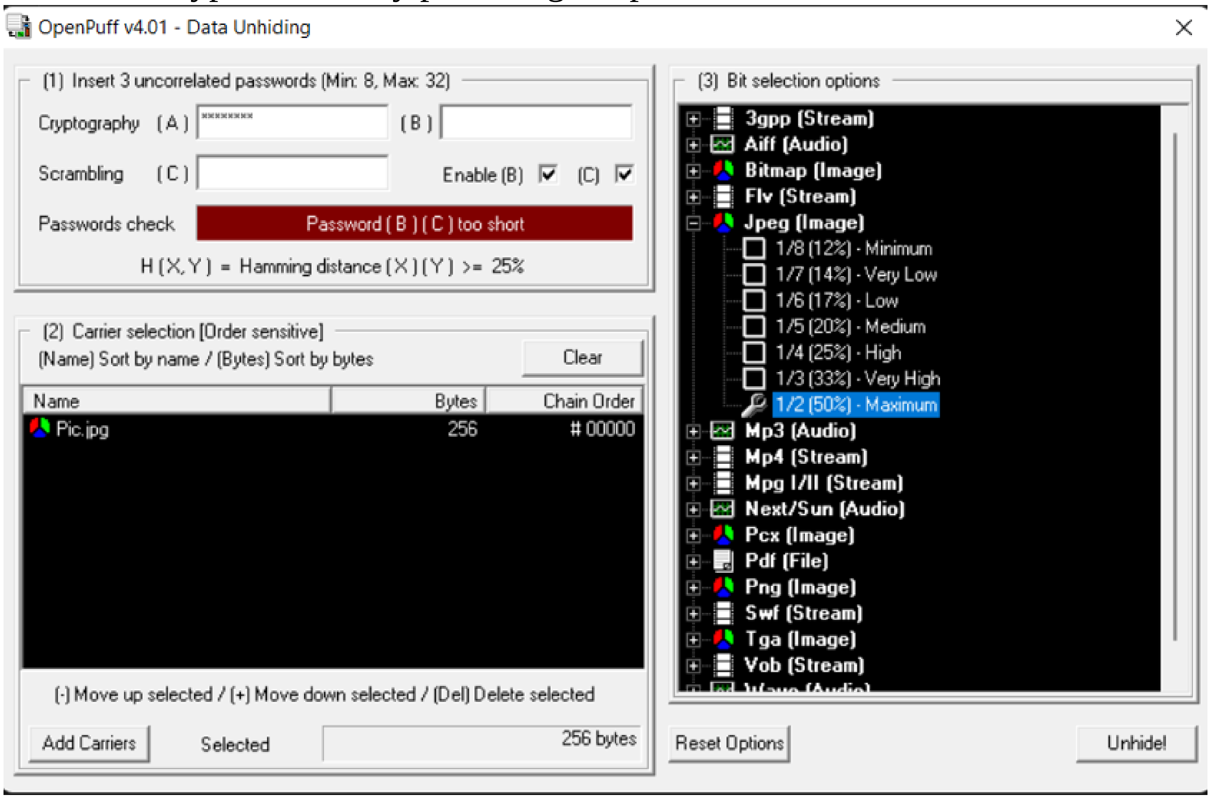
(-) Move up selected / (+) Move down selected / (Del) Delete selected
Add Selected / Total 256 / 55 bytes
Reset Options Add Decoy! Hide Data!

6. When we click Hide data it will encrypt our text data in the respective file and save the file.

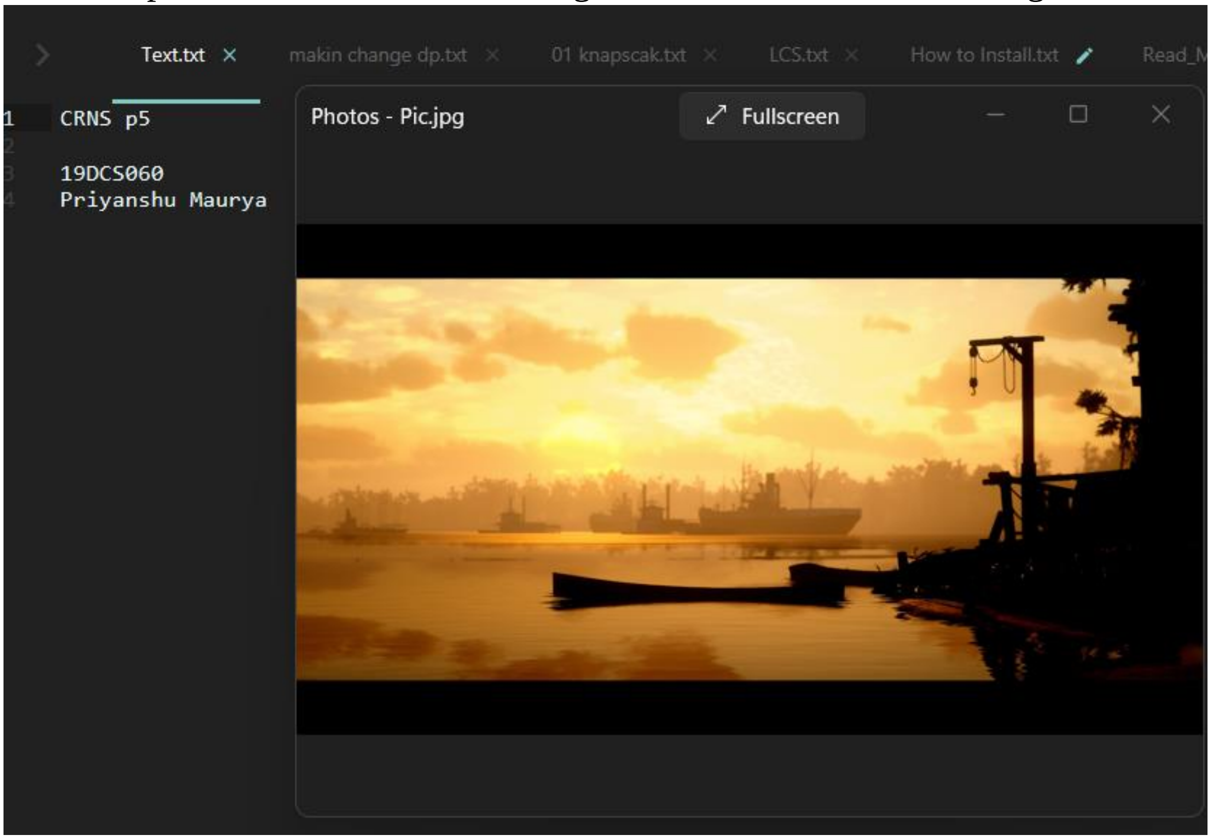

OpenPuff v4.01 - Data Hiding
(1) Insert 3 uncorrelated data passwords (Min: 8, Max: 32)
Cryptography (A) ********
(B)
Scrambling (C)
Enable (B) (C)
Passwords check
H(X,Y) = Hamming
(2) Data (Max: 256Mb)
Target D:\SEM 6\CRNS\PRACTICALS\P6
Browse
Size 10 + name(8) + data(37) bytes
(3) Carrier selection [Order sensitiv
(Name) Sort by name / (Bytes) Sort b
Name
Pic.jpg
(-) Move up selected / (+) Move down selected / (Del) Delete selected
Add
Selected / Total 256 / 55 bytes
Mp3 (Audio)
Mp4 (Stream)
Mpg I/II (Stream)
Next/Sun (Audio)
Reset Options
Add Decoy!
Hide Data!
OpenPuff v4.01 - Task Report
Done

```
*** Begin of Report ***

Bit Selection:
Jpeg (Image) <- 1/2 (50%) - Maximum

Carriers:
#00000 <- Pic.jpg

*** End of Report ***
```

7. We can decrypt the data by providing the password and file.



8. In the output we can see that we have got text file back from the image.



```
CRNS p5

19DCS060
Priyanshu Maurya
```

CONCLUSION:

In this practical I learned about a encryption technique called steganography and its use since old times. I also learned how to embed some message in file using DOS command and steganography.

PRACTICAL-6

AIM:

Implement GPG for windows.

THEORY:

GPG, or GNU Privacy Guard, is a public key cryptography implementation. This allows for the secure transmission of information between parties and can be used to verify that the origin of a message is genuine.

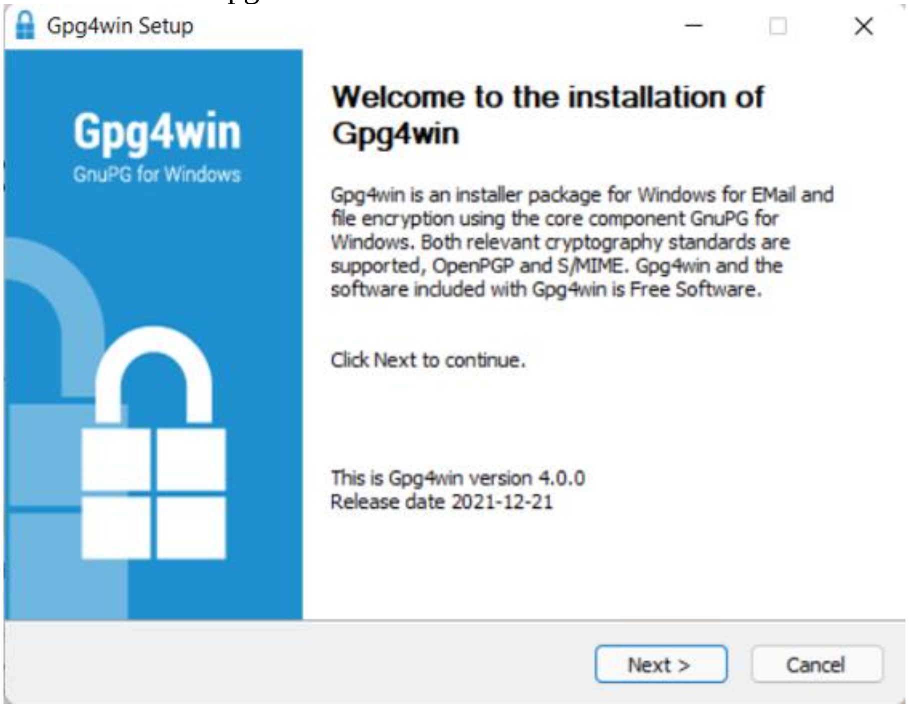
GPG relies on a security concept known as public key encryption. The idea is that you can split the encrypting and decrypting stages of the transmission into two separate pieces. That way, you can freely distribute the encrypting portion, as long as you secure the decrypting portion.

This would allow for a one-way message transfer that can be created and encrypted by anyone, but only be decrypted by the designated user (the one with the private decrypting key). If both of the parties create public/private key pairs and give each other their public encrypting keys, they can both encrypt messages to each other.

OUTPUT SCREENSHOT:

GPG installation

1. First download Gpg4win from website and run the installation.



Gpg4win Setup

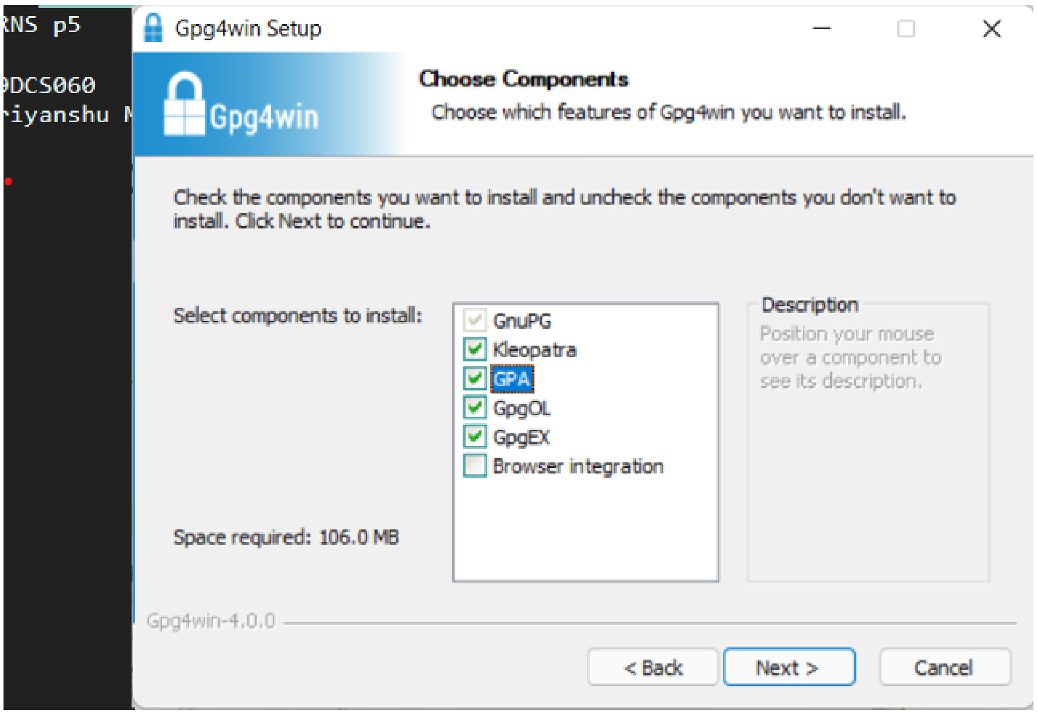
Welcome to the installation of Gpg4win

Gpg4win is an installer package for Windows for EMail and file encryption using the core component GnuPG for Windows. Both relevant cryptography standards are supported, OpenPGP and S/MIME. Gpg4win and the software included with Gpg4win is Free Software.

Click Next to continue.

This is Gpg4win version 4.0.0
Release date 2021-12-21

2. Select to install GPA in the selection menu and click next.



```
RNS p5

9DCS060
riyanshu M
```

Gpg4win Setup

Choose Components

Choose which features of Gpg4win you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [x] GnuPG
- [x] Kleopatra
- [x] GPA
- [x] GpgOL
- [x] GpgEX
- [ ] Browser integration

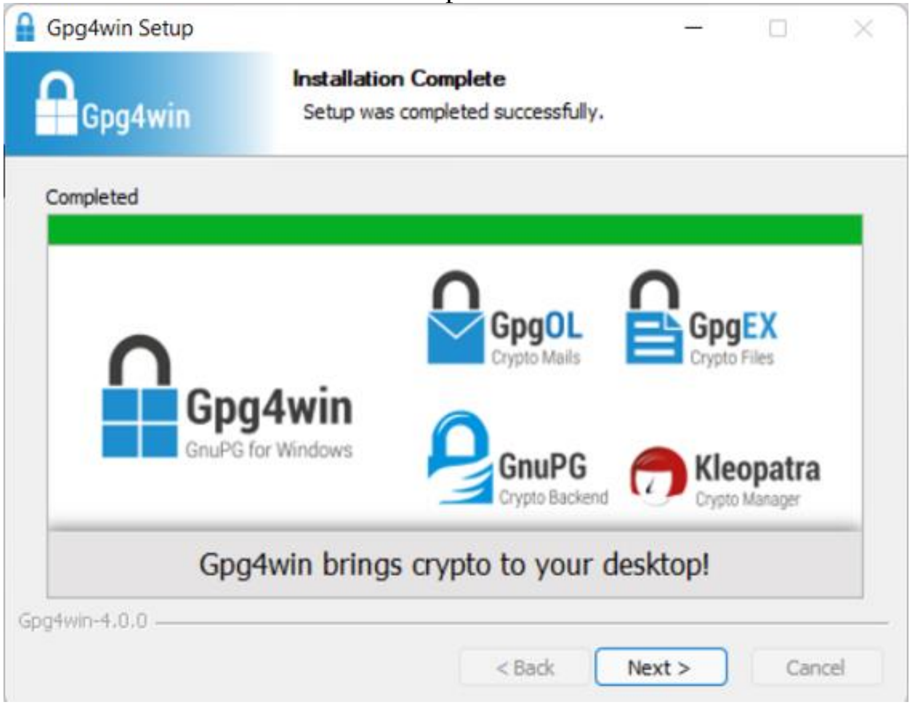
Space required: 106.0 MB

Description

Position your mouse over a component to see its description.

Gpg4win-4.0.0

3. Click next after installation is complete to finish the installation.



Gpg4win Setup

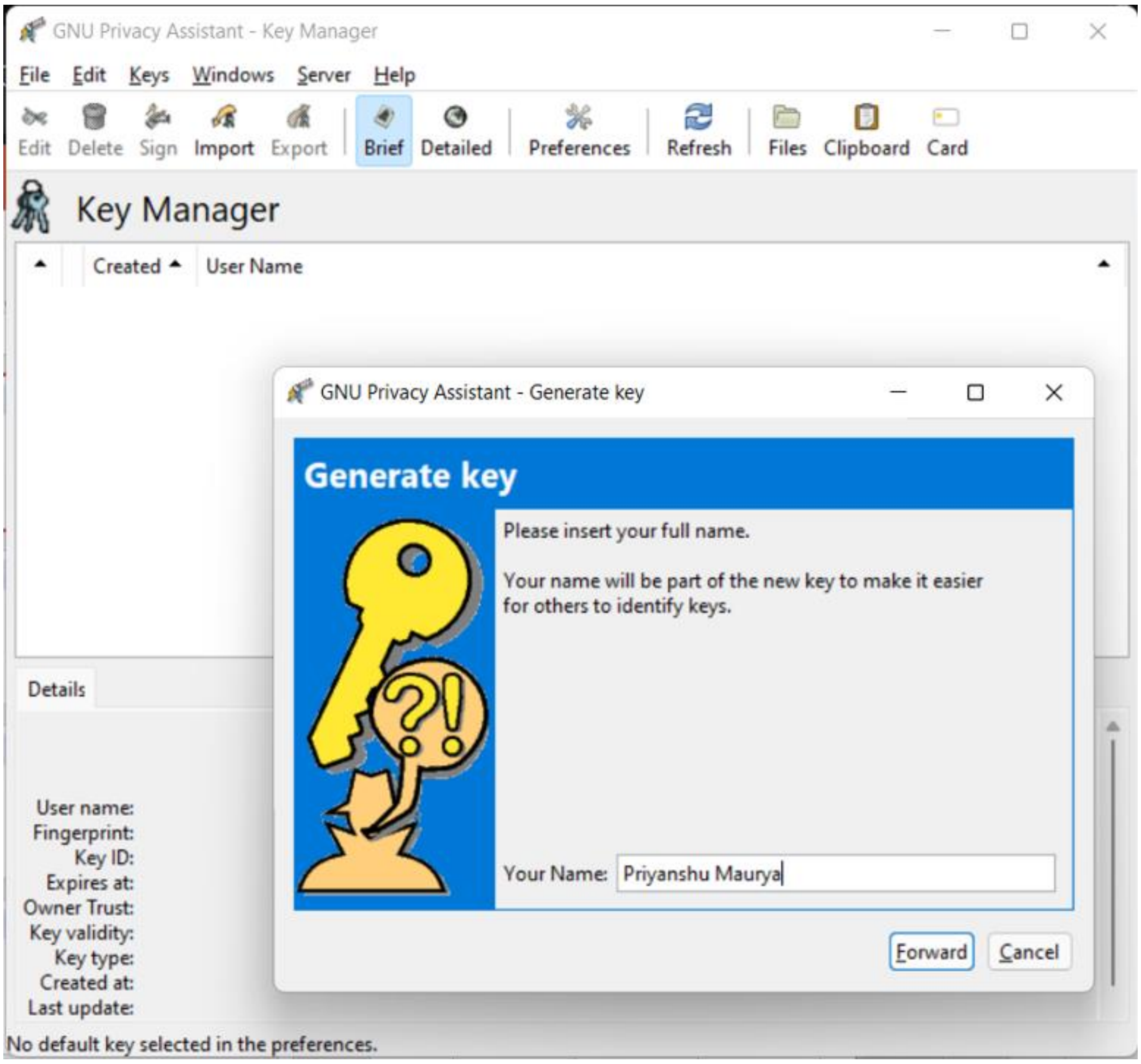
Installation Complete

Setup was completed successfully.

Completed

GPG4WIN

4. Now we have installed the software. Open it and click on New->New Key to create a new key and input Name, Email for the key creation this follows asymmetric cryptosystem so the Receiver has both public and private key.


GNU Privacy Assistant - Key Manager
File Edit Keys Windows Server Help
Edit Delete Sign Import Export Brief Detailed Preferences Refresh Files Clipboard Card
Key Manager
Created User Name
Details
User name:
Fingerprint:
Key ID:
Expires at:
Owner Trust:
Key validity:
Key type:
Created at:
Last update:
GNU Privacy Assistant - Generate key
Generate key
Forward
Cancel

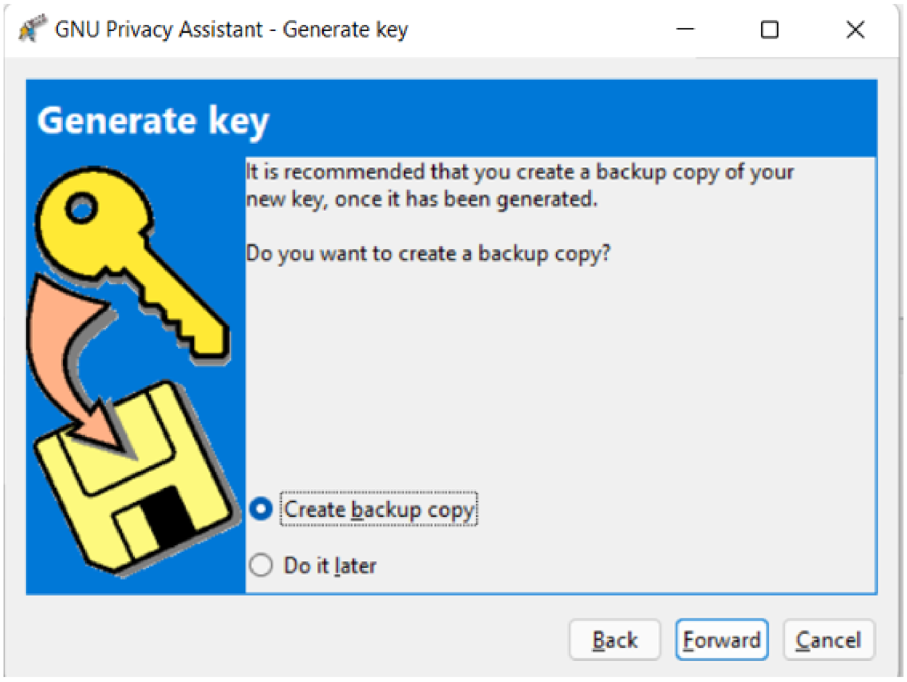
Please insert your full name.

Your name will be part of the new key to make it easier for others to identify keys.

Your Name: Priyanshu Maurya

No default key selected in the preferences.

5.


GNU Privacy Assistant - Generate key

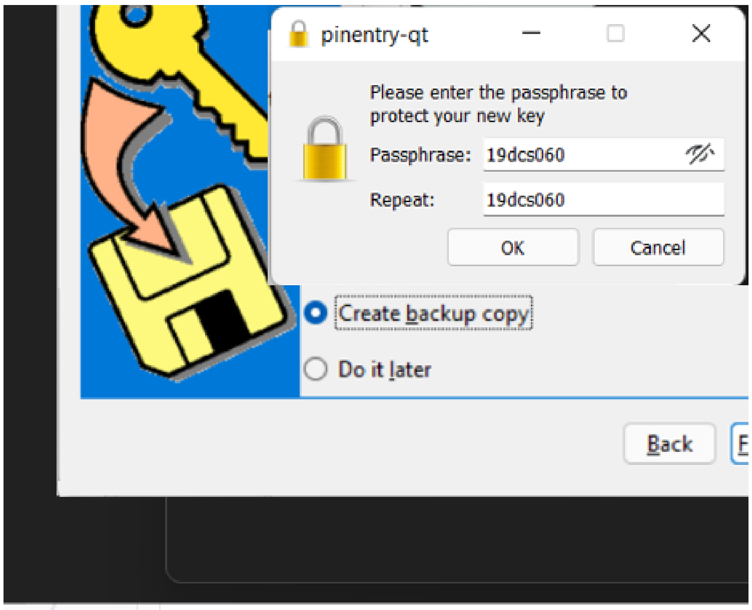
Generate key

It is recommended that you create a backup copy of your new key, once it has been generated.

Do you want to create a backup copy?

[✓ Create backup copy] Do it later

When we will create key it will ask for a passphrase,we can input whatever password we like.

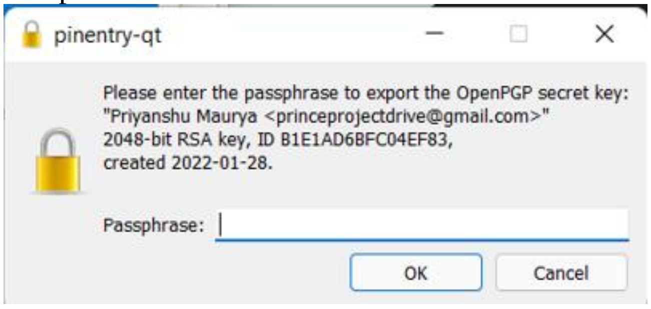

pinentry-qt

Please enter the passphrase to protect your new key

Passphrase: 19dcs060

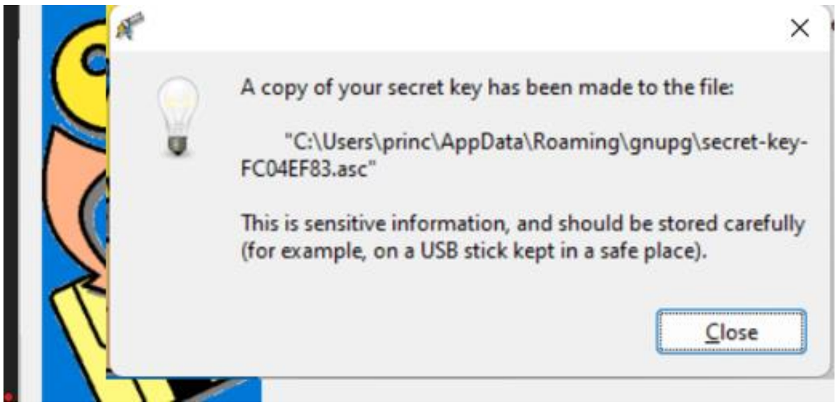
Repeat: 19dcs060

6. When we will try to finally click the create the key it will ask for passphrase, when we correctly input the passphrase popup will appear to save the key locally on the computer.


pinentry-qt

Please enter the passphrase to export the OpenPGP secret key:
"Priyanshu Maurya <princeprojectdrive@gmail.com>"
2048-bit RSA key, ID B1E1AD6BFC04EF83,
created 2022-01-28.

Passphrase:

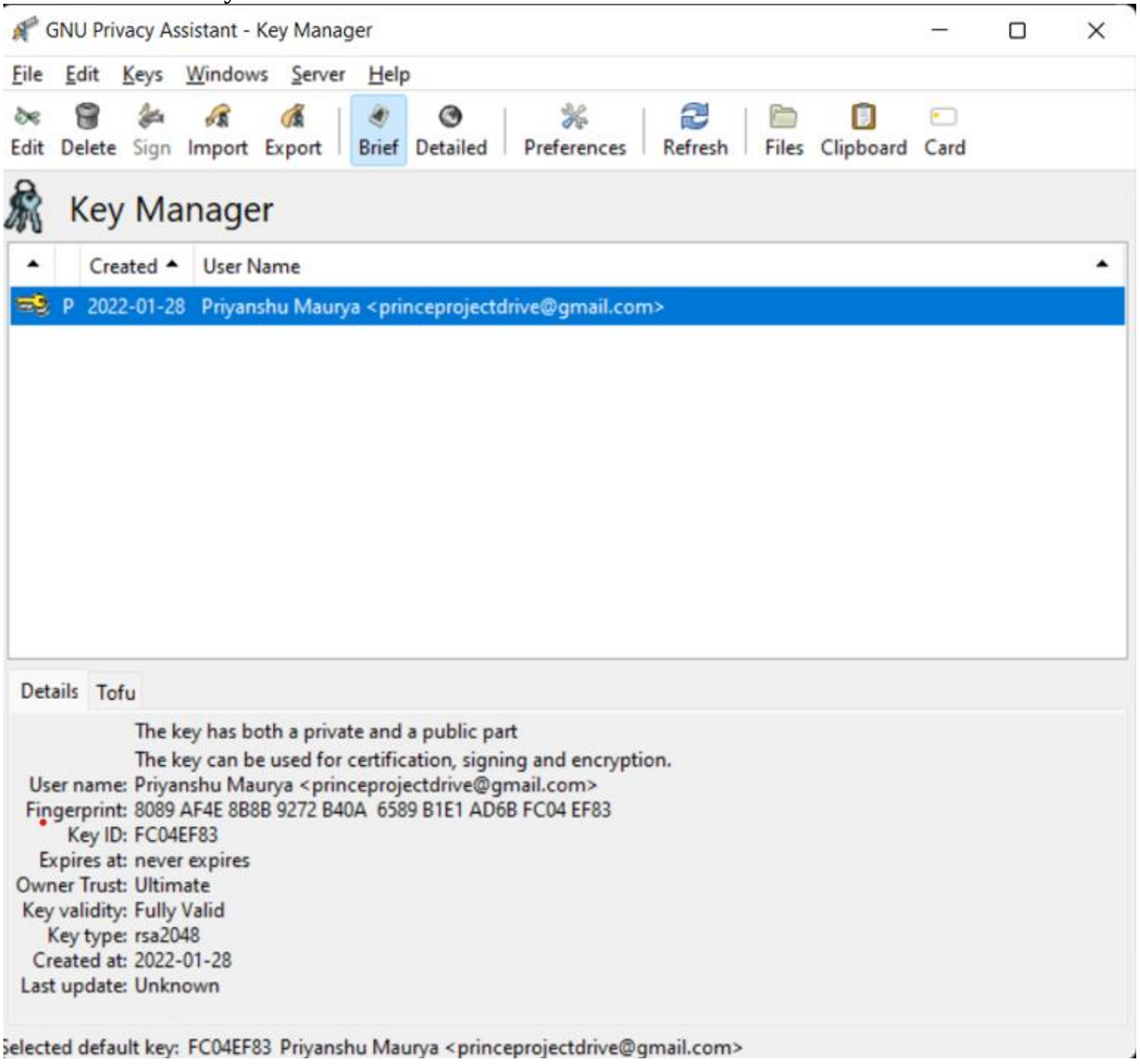


A copy of your secret key has been made to the file:

"C:\Users\princ\AppData\Roaming\gnupg\secret-key-FC04EF83.asc"

This is sensitive information, and should be stored carefully (for example, on a USB stick kept in a safe place).

7. We can see our key has been created



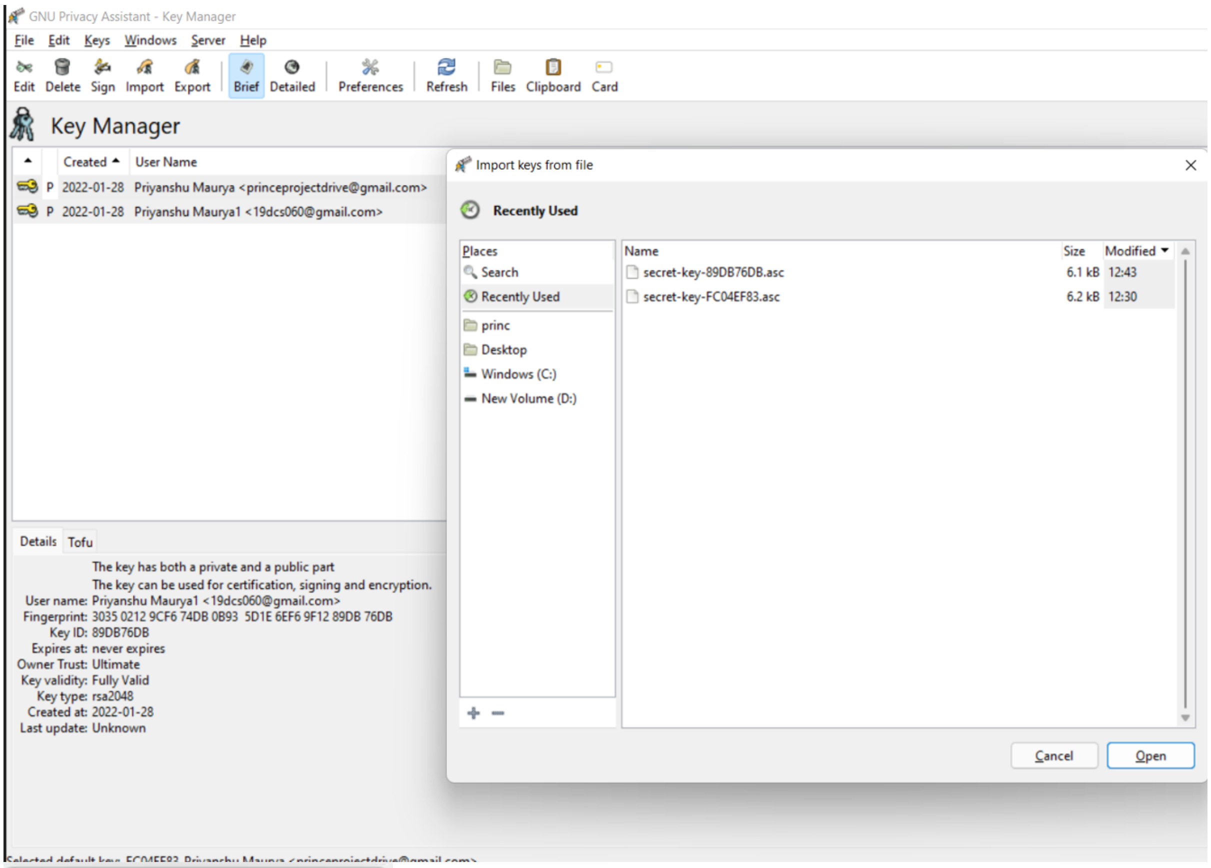
GNU Privacy Assistant - Key Manager
File Edit Keys Windows Server Help
Edit Delete Sign Import Export Brief Detailed Preferences Refresh Files Clipboard Card
Key Manager
Created User Name
Details Tofu

P 2022-01-28 Priyanshu Maurya <princeprojectdrive@gmail.com>

```
               The key has both a private and a public part
               The key can be used for certification, signing and encryption.
   User name:  Priyanshu Maurya <princeprojectdrive@gmail.com>
 Fingerprint:  8089 AF4E 8B8B 9272 B40A  6589 B1E1 AD6B FC04 EF83
      Key ID:  FC04EF83
  Expires at:  never expires
 Owner Trust:  Ultimate
Key validity:  Fully Valid
    Key type:  rsa2048
  Created at:  2022-01-28
 Last update:  Unknown
```

Selected default key: FC04EF83 Priyanshu Maurya <princeprojectdrive@gmail.com>

8. When we want to use the software to encrypt we must import the key using import button as we know we are using asymmetric cryptosystem we import the public key of the receiver.

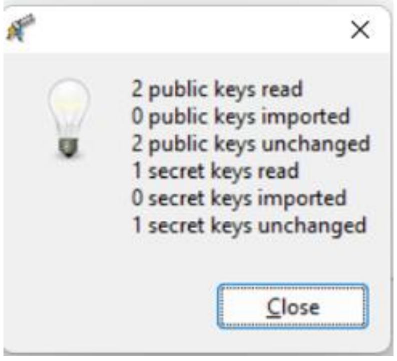

GNU Privacy Assistant - Key Manager
File Edit Keys Windows Server Help
Edit Delete Sign Import Export Brief Detailed Preferences Refresh Files Clipboard Card
Key Manager
Import keys from file
Recently Used
Places
Search
Recently Used
princ
Desktop
Windows (C:)
New Volume (D:)
Cancel
Open

	Created	User Name
P	2022-01-28	Priyanshu Maurya <princeprojectdrive@gmail.com>
P	2022-01-28	Priyanshu Maurya1 <19dcs060@gmail.com>

Name	Size	Modified
secret-key-89DB76DB.asc	6.1 kB	12:43
secret-key-FC04EF83.asc	6.2 kB	12:30

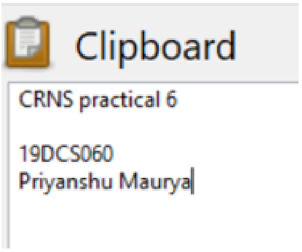
Details Tofu

```
               The key has both a private and a public part
               The key can be used for certification, signing and encryption.
   User name: Priyanshu Maurya1 <19dcs060@gmail.com>
 Fingerprint: 3035 0212 9CF6 74DB 0B93  5D1E 6EF6 9F12 89DB 76DB
      Key ID: 89DB76DB
  Expires at: never expires
 Owner Trust: Ultimate
Key validity: Fully Valid
    Key type: rsa2048
  Created at: 2022-01-28
 Last update: Unknown
```



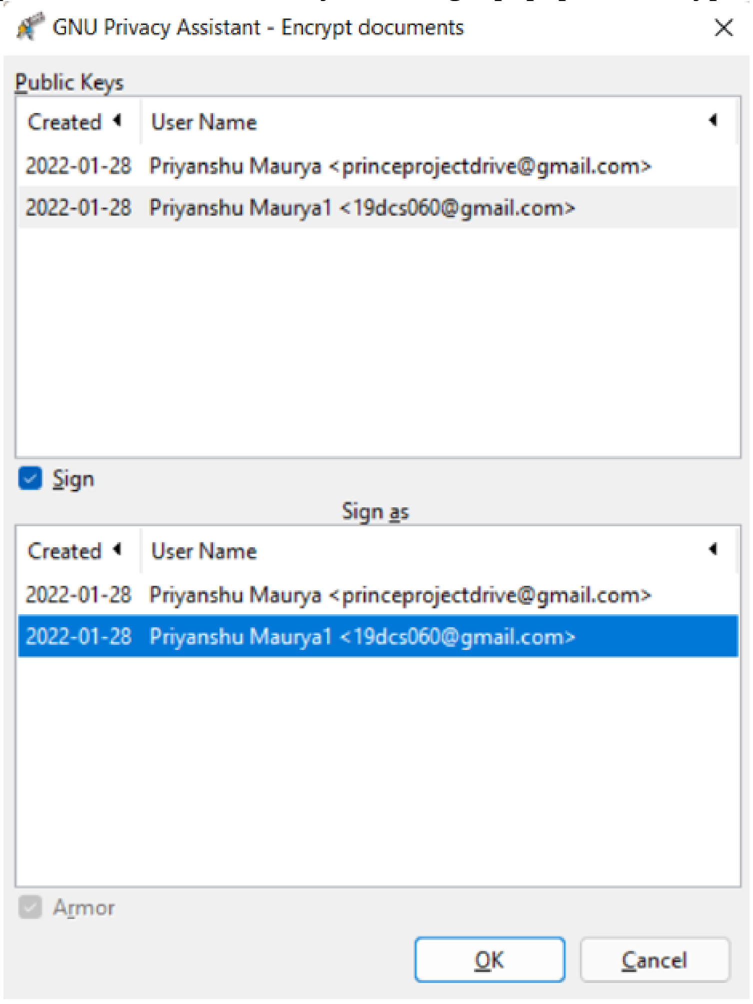
```
2 public keys read
0 public keys imported
2 public keys unchanged
1 secret keys read
0 secret keys imported
1 secret keys unchanged
```

9. Now we can click on clipboard to type the message.



```
CRNS practical 6

19DCS060
Priyanshu Maurya
```

10. After we click on encrypt option it will show us the key to select and tell us to input the password if done correctly we will get popup that encryption is successful.


GNU Privacy Assistant - Encrypt documents

Public Keys

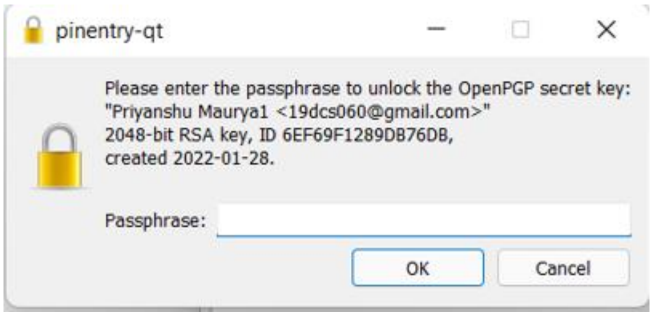
Created	User Name
2022-01-28	Priyanshu Maurya <princeprojectdrive@gmail.com>
2022-01-28	Priyanshu Maurya1 <19dcs060@gmail.com>

☑ Sign

Sign as

Created	User Name
2022-01-28	Priyanshu Maurya <princeprojectdrive@gmail.com>
2022-01-28	Priyanshu Maurya1 <19dcs060@gmail.com>

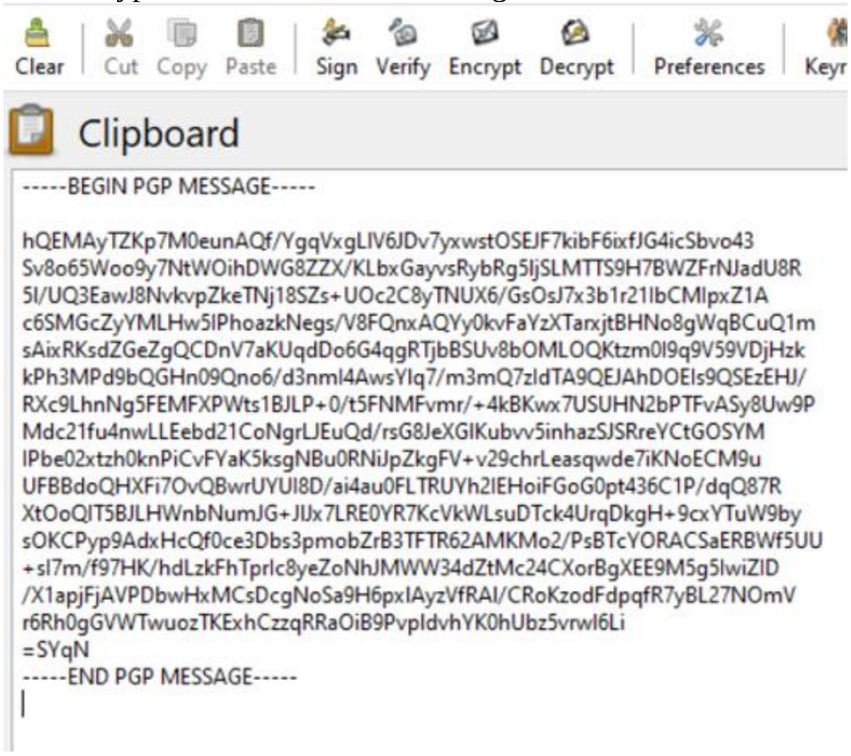
☑ Armor


pinentry-qt

Please enter the passphrase to unlock the OpenPGP secret key:
"Priyanshu Maurya1 <19dcs060@gmail.com>"
2048-bit RSA key, ID 6EF69F1289DB76DB,
created 2022-01-28.

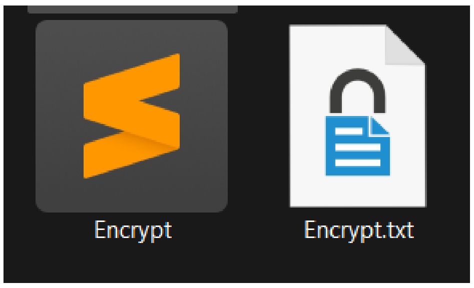
Passphrase:

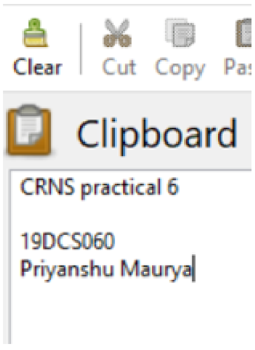
11. Our encrypted text will look something like this.



```
-----BEGIN PGP MESSAGE-----

hQEMAyTZKp7M0eunAQf/YgqVxgLIV6JDv7yxwstOSEJF7kibF6ixfJG4icSbvo43
Sv8o65Woo9y7NtWOihDWG8ZZX/KLbxGayvsRybRg5IjSLMTTS9H7BWZFrNJadU8R
5I/UQ3EawJ8NvkvpZkeTNj18SZs+UOc2C8yTNUX6/GsOsJ7x3b1r21IbCMIpxZ1A
c6SMGcZyYMLHw5IPhoazkNegs/V8FQnxAQYy0kvFaYzXTarxjtBHNo8gWqBCuQ1m
sAixRKsdZGeZgQCDnV7aKUqdDo6G4qgRTjbBSUv8bOMLOQKtzm0I9q9V59VDjHzk
kPh3MPd9bQGHn09Qno6/d3nml4AwsYlq7/m3mQ7zldTA9QEJAhDOEls9QSEzEHJ/
RXc9LhnNg5FEMFXPWts1BJLP+0/t5FNMFvmr/+4kBKwx7USUHN2bPTFvASy8Uw9P
Mdc21fu4nwLLEebd21CoNgrLJEuQd/rsG8JeXGIKubvv5inhazSJSRreYCtGOSYM
IPbe02xtzh0knPiCvFYaK5ksgNBu0RNiJpZkgFV+v29chrLeasqwde7iKNoECM9u
UFBBdoQHXFi7OvQBwrUYUI8D/ai4au0FLTRUYh2lEHoiFGoG0pt436C1P/dqQ87R
XtOoQIT5BJLHWnbNumJG+JIJx7LRE0YR7KcVkWLsuDTck4UrqDkgH+9cxYTuW9by
sOKCPyp9AdxHcQf0ce3Dbs3pmobZrB3TFTR62AMKMo2/PsBTcYORACSaERBWf5UU
+sl7m/f97HK/hdLzkFhTprlc8yeZoNhJMWW34dZtMc24CXorBgXEE9M5g5lwiZID
/X1apjFjAVPDbwHxMCsDcgNoSa9H6pxIAyzVfRAI/CRoKzodFdpqfR7yBL27NOmV
r6Rh0gGVWTwuozTKExhCzzqRRaOiB9PvpldvhYK0hUbz5vrwl6Li
=SYqN
-----END PGP MESSAGE-----
```

12. When we want to decrypt the message we will click on decrypt option and select the decryption file.



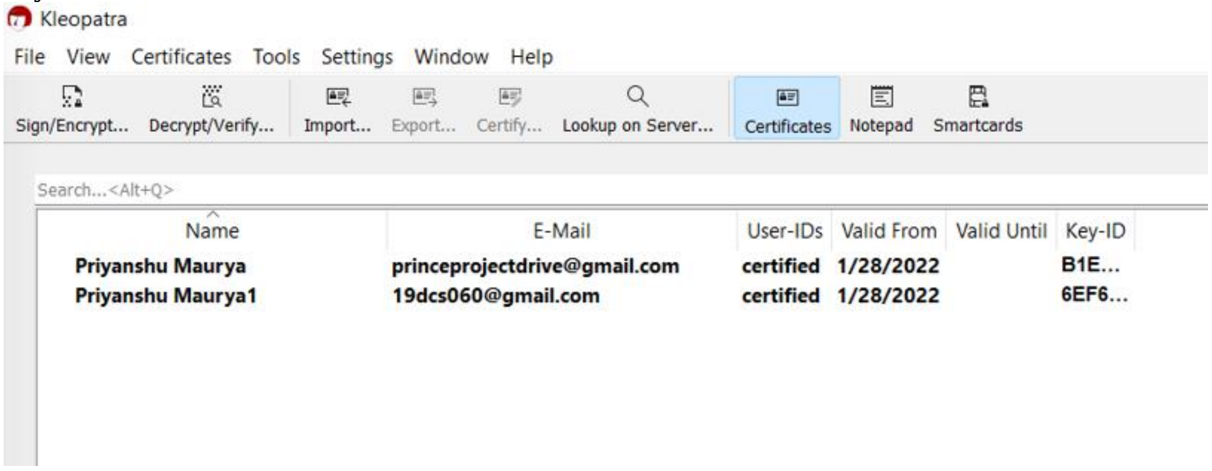


CRNS practical 6

19DCS060
Priyanshu Maurya

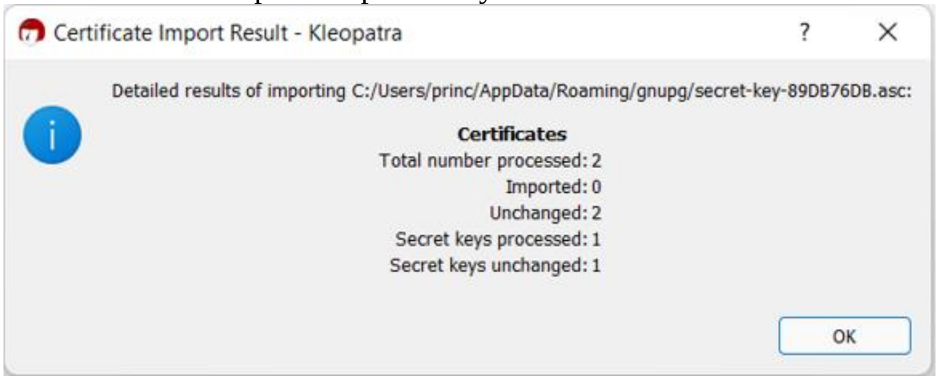
Kleopetra

13. Now we will use a tool called Kleopetra for encryption and decryption we can see the key created in the main screen.



Name	E-Mail	User-IDs	Valid From	Valid Until	Key-ID
Priyanshu Maurya	princeprojectdrive@gmail.com	certified	1/28/2022		B1E...
Priyanshu Maurya1	19dcs060@gmail.com	certified	1/28/2022		6EF6...

14. We now have to import the public key of Receiver.



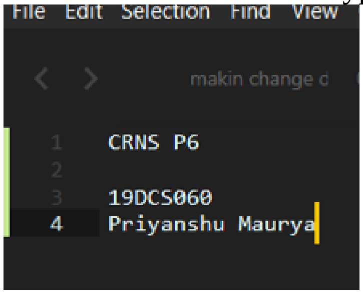
Certificate Import Result - Kleopatra

Detailed results of importing C:/Users/princ/AppData/Roaming/gnupg/secret-key-89DB76DB.asc:

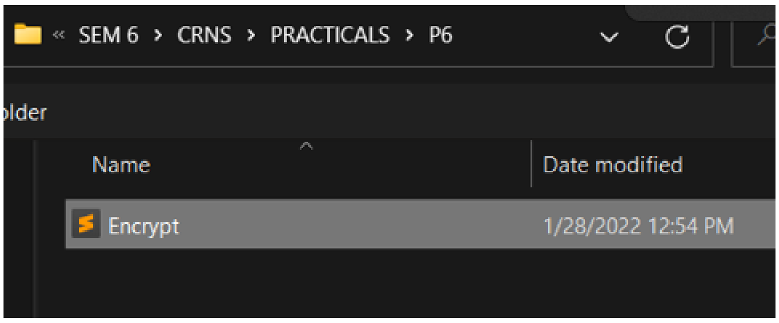
Certificates

```
Total number processed: 2
              Imported: 0
             Unchanged: 2
 Secret keys processed: 1
 Secret keys unchanged: 1
```

15. We have a file called Encrypt.txt to encrypt

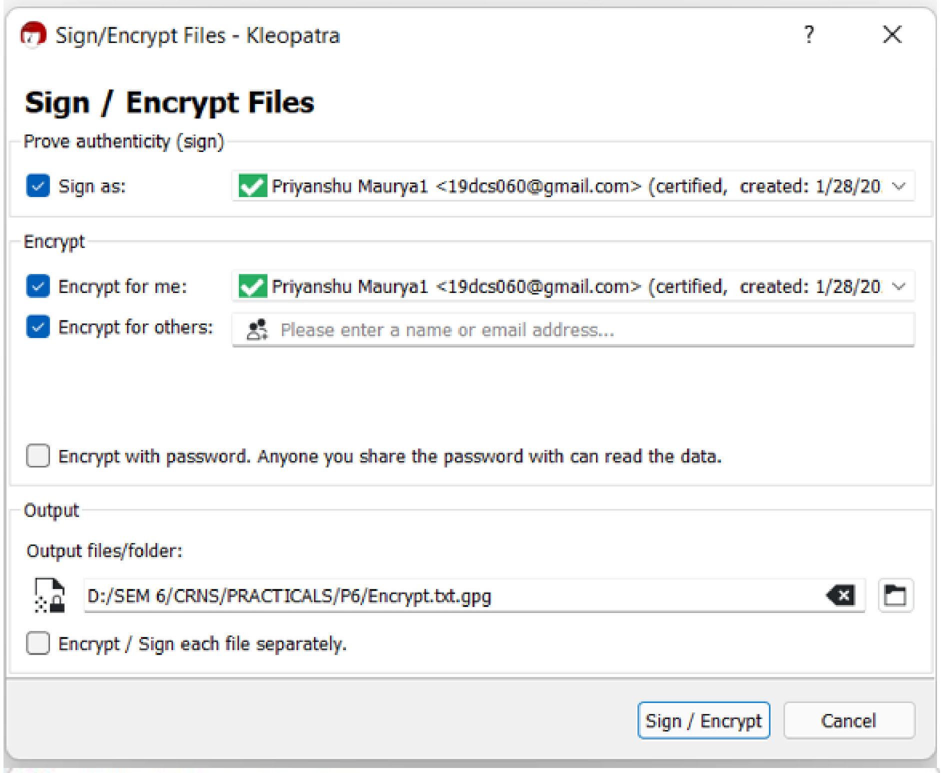


```
CRNS P6

19DCS060
Priyanshu Maurya
```



Name	Date modified
Encrypt	1/28/2022 12:54 PM

16. Now select the input file for encryption and also give location of the output file. On successful encryption we will get popup that encryption was successful.



Sign/Encrypt Files - Kleopatra

Sign / Encrypt Files

Prove authenticity (sign)

☑ Sign as: Priyanshu Maurya1 <19dcs060@gmail.com> (certified, created: 1/28/20

Encrypt

☑ Encrypt for me: Priyanshu Maurya1 <19dcs060@gmail.com> (certified, created: 1/28/20

☑ Encrypt for others: Please enter a name or email address...

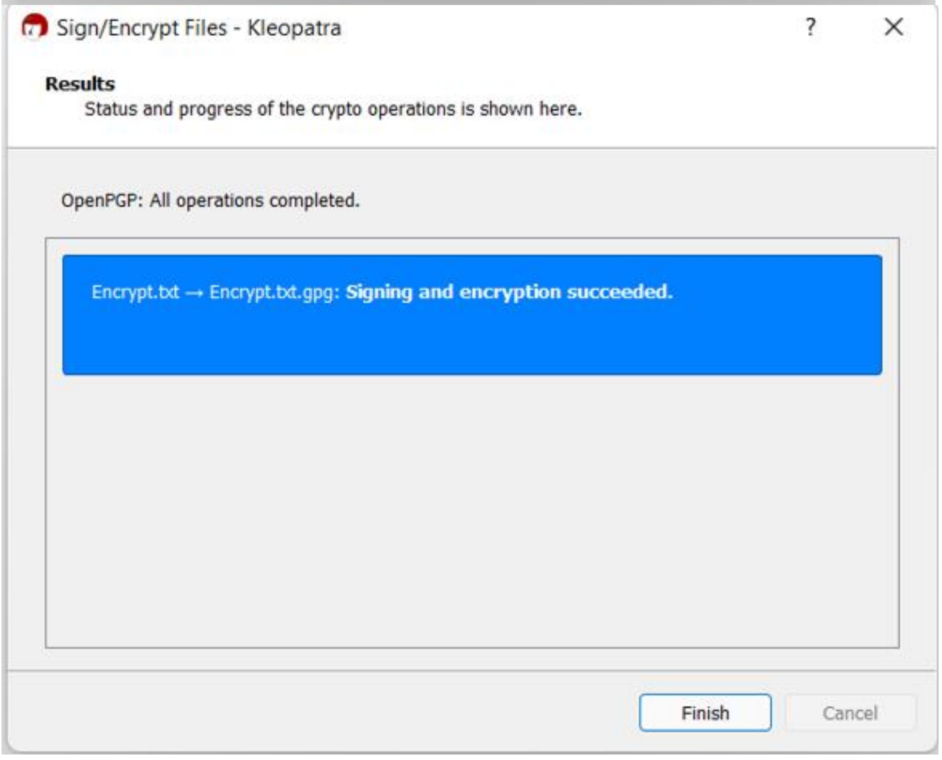
☐ Encrypt with password. Anyone you share the password with can read the data.

Output

Output files/folder:

D:/SEM 6/CRNS/PRACTICALS/P6/Encrypt.txt.gpg

☐ Encrypt / Sign each file separately.



Sign/Encrypt Files - Kleopatra

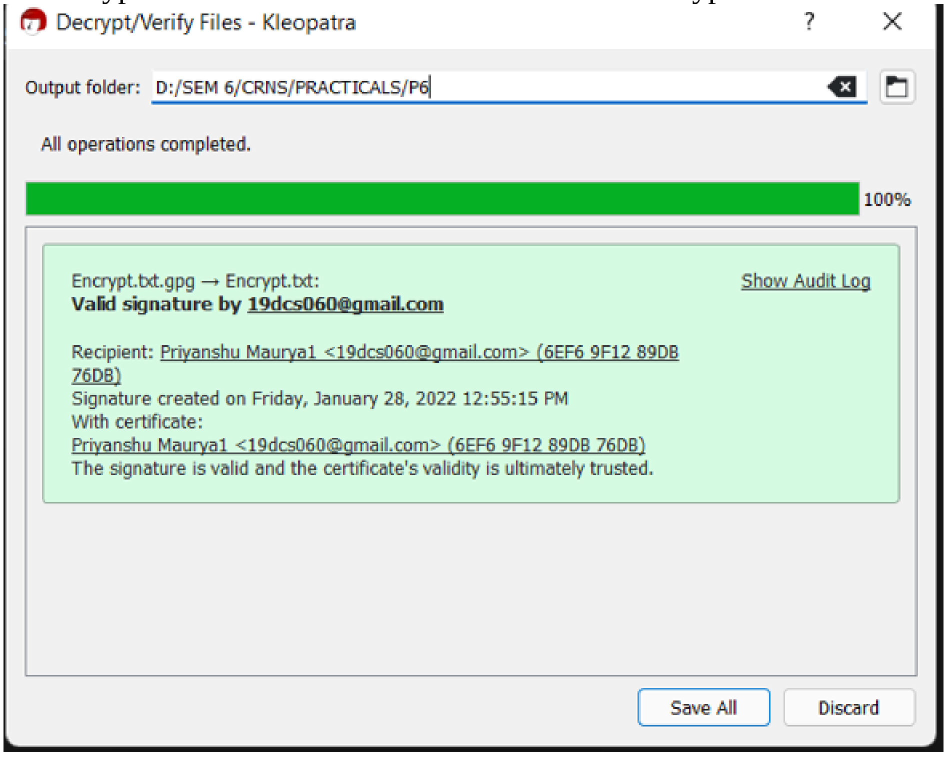
Results

Status and progress of the crypto operations is shown here.

OpenPGP: All operations completed.

Encrypt.txt → Encrypt.txt.gpg: **Signing and encryption succeeded.**

17. To decrypt the file we select the file and click on decrypt.



Output folder: D:/SEM 6/CRNS/PRACTICALS/P6

All operations completed.

100%

Encrypt.txt.gpg → Encrypt.txt:
Valid signature by 19dcs060@gmail.com

Recipient: Priyanshu Maurya1 <19dcs060@gmail.com> (6EF6 9F12 89DB 76DB)
Signature created on Friday, January 28, 2022 12:55:15 PM
With certificate:
Priyanshu Maurya1 <19dcs060@gmail.com> (6EF6 9F12 89DB 76DB)
The signature is valid and the certificate's validity is ultimately trusted.

CONCLUSION:

In this practical I learned about a encryption tool called GPG which is used to encrypt and decrypt the data and encrypted and decrypted the messages using GPG4WIN and Kleopetra tool.