MANIPAL INSTITUTE OF TECHNOLOGY

Manipal Academy of Higher Education

Manipal – 576 104

DEPARTMENT OF COMPUTER SCIENCE & ENGG.

CERTIFICATE

This is to certify that Ms./Mr. ……...…… Reg. No. …..…… Section: …… Roll No: ……... has satisfactorily completed the lab exercises prescribed for CSE 5163 INFORMATION SYSTEMS LAB-I [0 0 6 2] [CSIS] of First Year M. Tech. Degree at MIT, Manipal, in the academic year 2019-2020.

Date: …….....

Signature
Faculty in Charge

COURSE OBJECTIVES AND OUTCOMES
EVALUATION PLAN
INSTRUCTIONS TO THE STUDENTS
REVIEW OF PROGRAMMING CONCEPTS AND FAMILIRIZATION OF THE WORKING ENVIRONMENT
REVIEW OF PROGRAMMING CONCEPTS AND FAMILIRIZATION OF THE WORKING ENVIRONMENT, & TOOLS
EXPRIMENTS ON NUMBER THEORY TAUGHT IN ADVANCED CRYPTOGRAPHY AND ITS APPLICATIONS IN VARIOUS CRYPTOSYSTEMS

INDEX

Note:2A and 2B are the experiments to demonstrate number theory taught in Advanced Cryptography

2(C)	Diffiee-Hellman Algorithm			
2(D)	MD5			
2(E)	SHA-1			
3	Implement the Signature Scheme for Digital Signature Standard			
4	Demonstrate how to provide secure data storage, secure data transmission and for creating digital signatures (GnuPG)			
5	Setup a honey pot and monitor the honeypot on network (KF Sensor)			
6	Installation of rootkits and study about the variety of options			

Note: the Main emphasis is on the use of advanced tools

Course Objectives

The student should be made to:

Be exposed to the different cipher techniques(in the context of number theory)

Learn to implement the algorithms AES/DES and RSA(in the context of number theory), RSA varients, MD5, SHA-1

Learn to use network security tools like GnuPG, KF sensor, Net Strumbler

Course Outcomes

At the end of the course, the student should be able to:

Implement the cipher techniques• Develop the various security algorithms•

Use different open source tools for network security and analysis•

Evaluation plan

- Internal Assessment Marks: 60%

- End semester assessment of 2 hour duration: 40 %

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book to every lab session
2. Be in time and adhere to the institution rules and maintain the decorum
3. Must Sign in the log register provided
4. Make sure to occupy the allotted system and answer the attendance

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- Copy the program and results in the Lab record.
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Academic honesty is required in all your work. You must solve all programming assignments entirely on your own, except where group work is explicitly authorized. This means you must not take, neither show, give or otherwise allow others to take your program code, problem solutions, or other work.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Programs should perform input validation (Data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
    - Comments should be used to give the statement of the problem and every function should indicate the purpose of the function, inputs, and outputs.
    - Statements within the program should be properly indented.
    - Use meaningful names for variables and functions.
    - Make use of constants and type definitions wherever needed.
- The exercises for each week are divided into three sets:
    - Solved exercise
    - Lab exercises - to be completed during lab hours
    - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- Questions for lab tests and examination are not necessarily limited to the questions in the manual but may involve some variations and/or combinations of the questions.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

LAB NO: 1 **Date:**

REVIEW OF C PROGRAMMING CONCEPTS:

Objectives:

In this lab, the student will be able to:

- Familiarize with a sublime editor for writing C programs
- Compile, execute and debug C program using ddd

I. SUBLIME EDITOR QUICK HELP GUIDE

Creating a source file:

1. Login to the student account in Ubuntu
2. Press CNTRL + ALT+ T to open the *terminal.* Alternatively, choose a *terminal* from dash home by typing the query 'terminal'
   EX: **user@user:~$** subl
3. Open sublime editor, write a program and save it with .c extension.

Building C programs using sublime:

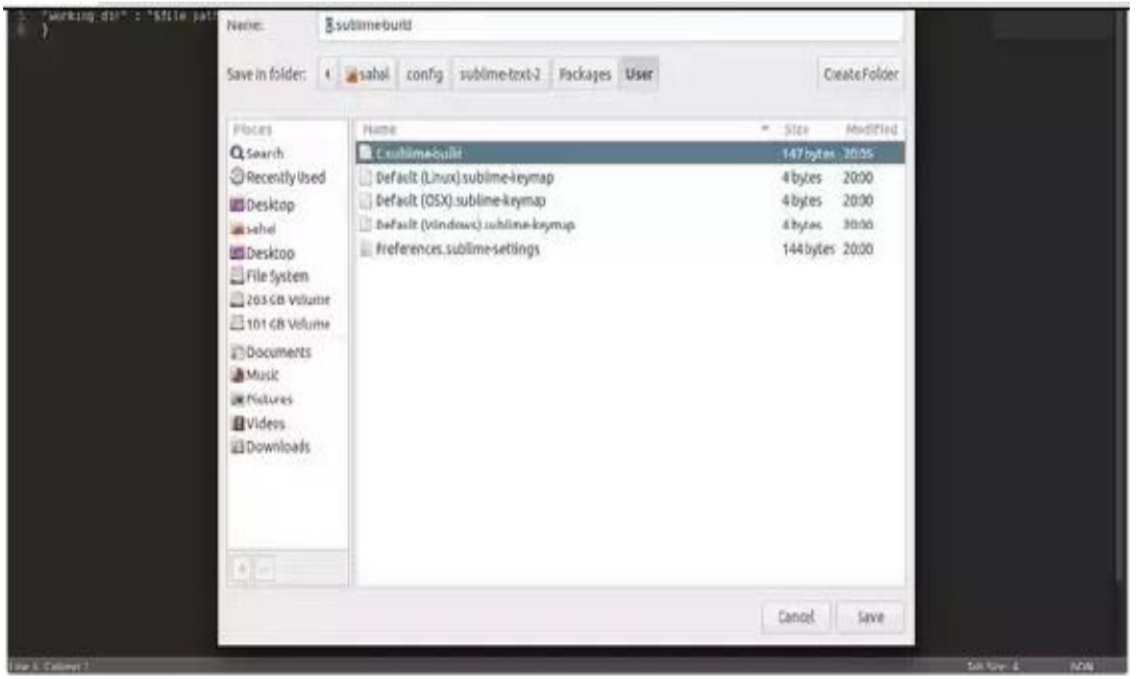
(i) Click **Tools -> Build System -> New Build System** and copy paste the following into curly braces:

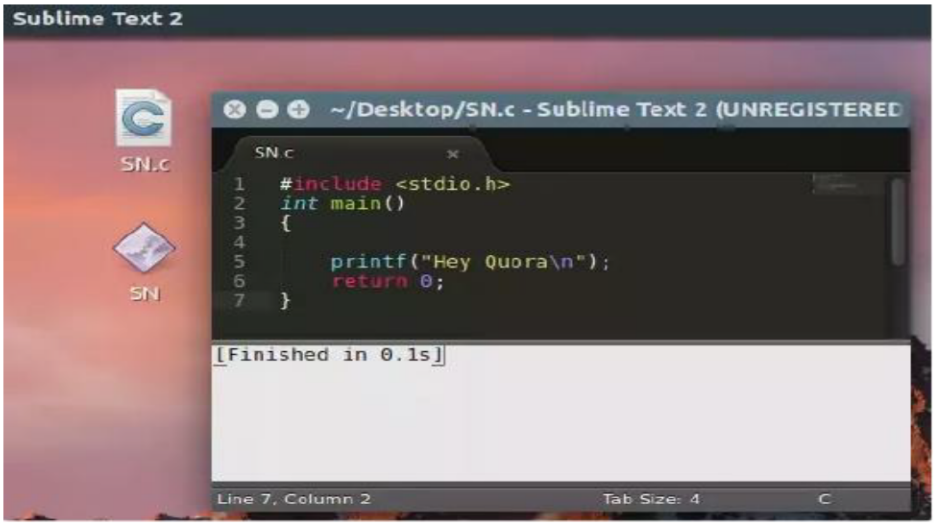
```
{
"cmd" : ["gcc  $file_name -o $file_base_name -lm -Wall"],
"selector" : "source.c",
"shell" : true,
 "working_dir" : "$file_path"
}
```

(ii) Select **File>Save** and name **c.sublime.build** without changing the directory.

(iii) Click **Tools>Build System** and select **c**.

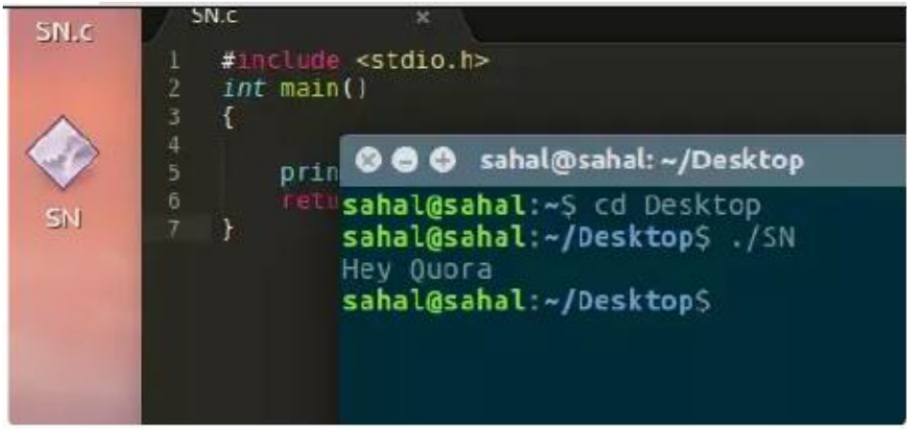
Now sublime is ready to build c programs [CNTRL+ B for compiling] in the editor.





Executing the Program:

1. Open a terminal and locate your C program.
   Ex: **user@user:~$** cd Desktop
2. Run build file by typing:
   user@user:~$ ./SN



Compiling and Executing through terminal:

- Type the Compile command as cc -g –o *outputfile inputcfile*
  - Ex:
    - cc –o *factorial factorial.c*

    In this case, the executable file is created as *factorial*. To run the executable go to terminal and type ./factorial. (Here ./ refers to the

current directory, alternatively we can specify the absolute or relative path of the executable file).

- cc *factorial.c*

In this case, the executable file is created as *a.out* by default. Remember this executable file will be over written when you run the same command again.

To run the executable go to terminal and type ./a.out

Debugging through data display debugger:

- Open the terminal and run the ddd commands (refer ddd video)

LAB NO: 2 **Date:**

C Program to Find GCD of Two Numbers Using Recursive Euclid Algorithm

Purpose: This is a C Program to find GCD of two numbers using Recursive Euclid Algorithm. In mathematics, the Euclidean algorithm, or Euclid's algorithm, is a method for computing the greatest common divisor (GCD) of two (usually positive) integers, also known as the greatest common factor (GCF) or highest common factor (HCF). It is named after the Greek mathematician Euclid, The GCD of two positive integers is the largest integer that divides both of them without leaving a remainder (the GCD of two integers, in general, is defined in a more subtle way).

In its simplest form, Euclid's algorithm starts with a pair of positive integers and forms a new pair that consists of the smaller number and the difference between the larger and smaller numbers. The process repeats until the numbers in the pair are equal. That number then is the greatest common divisor of the original pair of integers.

The main principle is that the GCD does not change if the smaller number is subtracted from the larger number. For example, the GCD of 252 and 105 is exactly the GCD of 147 (= 252 – 105) and 105. Since the larger of the two numbers is reduced, repeating this process gives successively smaller numbers, so this repetition will necessarily stop sooner or later — when the numbers are equal (if the process is attempted once more, one of the numbers will become 0).

```c
#include <stdio.h>

int gcd_algorithm(int x, int y)
{
    if (y == 0) {
        return x;
    } else if (x >= y && y > 0) {
        return gcd_algorithm(y, (x % y));
    }
}

int main(void)
{
    int num1, num2, gcd;
    printf("\nEnter two numbers to find gcd using Euclidean algorithm: ");
    scanf("%d%d", &num1, &num2);
    gcd = gcd_algorithm(num1, num2);
    if (gcd)
        printf("\nThe GCD of %d and %d is %d\n", num1, num2, gcd);
    else
        printf("\nInvalid input!!!\n");
    return 0;
}
```

```
$ gcc gcd.c -o gcd
$ ./gcd
```

Programming Problems:

1. Program to Find GCD of Two Numbers Using Extended Euclid Algorithm
2. Write a program to compute GCD of two given numbers using non recursive procedure.
3. **Complex Problem:** Write a program to comute Modular Inverse.

LAB NO: 3 **Date:**

EX. NO: 1(A)

IMPLEMENTATION OF CAESAR CIPHER

AIM:

To implement the simple substitution technique named Caesar cipher using C language.

DESCRIPTION:

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

ALGORITHM:

STEP-1: Read the plain text from the user.

STEP-2: Read the key value from the user.

STEP-3: If the key is positive then encrypt the text by adding the key with each character in the plain text.

STEP-4: Else subtract the key from the plain text.

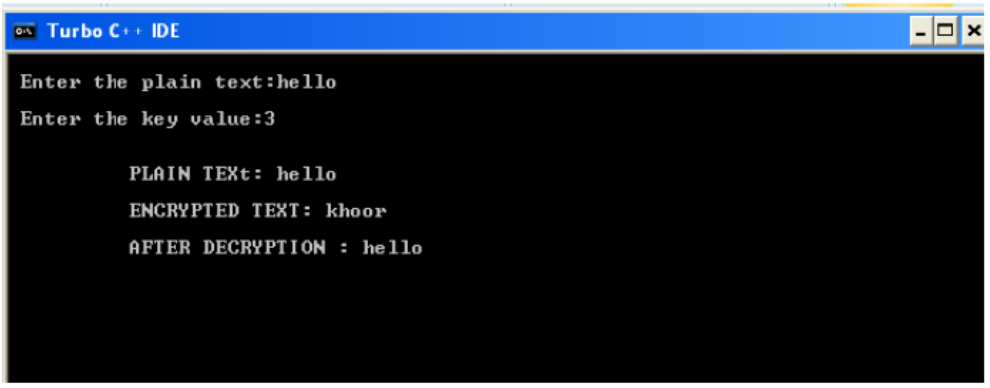
STEP-5: Display the cipher text obtained above.

PROGRAM: **(Caesar Cipher)**

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
#include <ctype.h>
void main()
```

```
{
 char plain[10], cipher[10];
 int key,i,length;
 int result;
 clrscr();
 printf("\n Enter the plain text:");
 scanf("%s", plain);
 printf("\n Enter the key value:");
 scanf("%d", &key);
 printf("\n \n \t  PLAIN TEXt: %s",plain);
 printf("\n \n \t  ENCRYPTED TEXT: ");
 for(i = 0, length = strlen(plain); i < length; i++)
     {
     cipher[i]=plain[i] + key;
     if (isupper(plain[i]) && (cipher[i] > 'Z'))
     cipher[i] = cipher[i] - 26;
     if (islower(plain[i]) && (cipher[i] > 'z'))
     cipher[i] = cipher[i] - 26;
     printf("%c", cipher[i]);
     }
     printf("\n \n \t  AFTER DECRYPTION : ");
 for(i=0;i<length;i++)
     {

      plain[i]=cipher[i]-key;
     if(isupper(cipher[i])&&(plain[i]<'A'))
     plain[i]=plain[i]+26;
     if(islower(cipher[i])&&(plain[i]<'a'))
     plain[i]=plain[i]+26;
     printf("%c",plain[i]);
 }
getch();
}
```

OUTPUT:



EX. NO: 1(B)

IMPLEMENTATION OF PLAYFAIR CIPHER

AIM:

To write a C program to implement the Playfair Substitution technique.

DESCRIPTION:

The Playfair cipher starts with creating a key table. The key table is a 5×5 grid of letters that will act as the key for encrypting your plaintext. Each of the 25 letters must be unique and one letter of the alphabet is omitted from the table (as there are 25 spots and 26 letters in the alphabet).

To encrypt a message, one would break the message into digrams (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD", and map them out on the key table. The two letters of the diagram are considered as the opposite corners of a rectangle in the key table. Note the relative position of the corners of this rectangle. Then apply the following 4 rules, in order, to each pair of letters in the plaintext:

1. If both letters are the same (or only one letter is left), add an "X" after the first letter
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively
3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively
4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair.

ALGORITHM:

STEP-1: Read the plain text from the user.

STEP-2: Read the keyword from the user.

STEP-3: Arrange the keyword without duplicates in a 5*5 matrix in the row order and fill the remaining cells with missed out letters in alphabetical order. Note that 'i' and 'j' takes the same cell.

STEP-4: Group the plain text in pairs and match the corresponding corner letters by forming a rectangular grid.

STEP-5: Display the obtained cipher text.

PROGRAM: (Playfair Cipher)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#define MX 5
void playfair(char ch1,char ch2, char key[MX][MX])
{
     int i,j,w,x,y,z;
     FILE *out;
if((out=fopen("cipher.txt","a+"))==NULL)
{
     printf("File Currupted.");
}
for(i=0;i<MX;i++)
{
     for(j=0;j<MX;j++)
     {
     if(ch1==key[i][j])
     {
     w=i;
     x=j;
     }
     else if(ch2==key[i][j])
     {
     y=i;
     z=j;
}}}
//printf("%d%d %d%d",w,x,y,z);
```

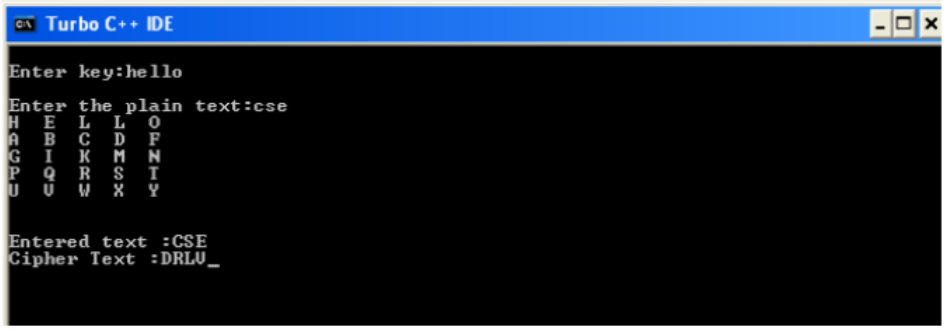
```
if(w==y)
{
     x=(x+1)%5;z=(z+1)%5;
     printf("%c%c",key[w][x],key[y][z]);
     fprintf(out, "%c%c",key[w][x],key[y][z]);
}
else if(x==z)
{

     w=(w+1)%5;y=(y+1)%5;
     printf("%c%c",key[w][x],key[y][z]);
     fprintf(out, "%c%c",key[w][x],key[y][z]);
}
else
{
     printf("%c%c",key[w][z],key[y][x]);
     fprintf(out, "%c%c",key[w][z],key[y][x]);
}
fclose(out);
}
void main()
{
     int i,j,k=0,l,m=0,n;
     char key[MX][MX],keyminus[25],keystr[10],str[25]={0};
     char
     alpa[26]={'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'}
     ;
     clrscr();

     printf("\nEnter key:");
     gets(keystr);
     printf("\nEnter the plain text:");
     gets(str);
     n=strlen(keystr);
     //convert the characters to uppertext
     for (i=0; i<n; i++)
     {
          if(keystr[i]=='j')keystr[i]='i';
          else if(keystr[i]=='J')keystr[i]='I';
          keystr[i] = toupper(keystr[i]);
     }
     //convert all the characters of plaintext  to uppertext
     for (i=0; i<strlen(str); i++)
     {
          if(str[i]=='j')str[i]='i';
          else if(str[i]=='J')str[i]='I';
          str[i] = toupper(str[i]);
     }
```

```
j=0;
for(i=0;i<26;i++)
{
      for(k=0;k<n;k++)
      {
      if(keystr[k]==alpa[i])
      break;
      else if(alpa[i]=='J')
      break;
      }
      if(k==n)
      {
      keyminus[j]=alpa[i];j++;
      }
}
```

```
//construct key keymatrix
k=0;
for(i=0;i<MX;i++)
{
      for(j=0;j<MX;j++)
      {
      if(k<n)
      {
      key[i][j]=keystr[k];
      k++;}
      else
      {
      key[i][j]=keyminus[m];m++;
      }
      printf("%c  ",key[i][j]);
      }
      printf("\n");
}
printf("\n\nEntered text :%s\nCipher Text :",str);
for(i=0;i<strlen(str);i++)
{
if(str[i]=='J')str[i]='I';
if(str[i+1]=='\0')
playfair(str[i],'X',key);
else
```

```
        {
                if(str[i+1]=='J')str[i+1]='I';
                if(str[i]==str[i+1])
                playfair(str[i],'X',key);
                else
                {
                playfair(str[i],str[i+1],key);i++;
                }}
        }
getch();
}
```

OUTPUT:



EX. NO: 1(C)

IMPLEMENTATION OF HILL CIPHER

AIM:

To write a C program to implement the hill cipher substitution techniques.

DESCRIPTION:

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

ALGORITHM:

STEP-1: Read the plain text and key from the user.

STEP-2: Split the plain text into groups of length three.

STEP-3: Arrange the keyword in a 3*3 matrix.

STEP-4: Multiply the two matrices to obtain the cipher text of length three.

STEP-5: Combine all these groups to get the complete cipher text.

PROGRAM: (Hill Cipher)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main(){
     unsigned int a[3][3]={{6,24,1},{13,16,10},{20,17,15}};
     unsigned int b[3][3]={{8,5,10},{21,8,21},{21,12,8}};
     int i,j, t=0;
     unsigned int c[20],d[20];
     char msg[20];
     clrscr();
     printf("Enter plain text\n ");
     scanf("%s",msg);

     for(i=0;i<strlen(msg);i++)
     {    c[i]=msg[i]-65;

          printf("%d ",c[i]);
     }
     for(i=0;i<3;i++)
     {    t=0;
     for(j=0;j<3;j++)
     {
          t=t+(a[i][j]*c[j]);
     }
          d[i]=t%26;
     }
printf("\nEncrypted Cipher Text :");
for(i=0;i<3;i++)
printf(" %c",d[i]+65);
for(i=0;i<3;i++)
{
     t=0;
     for(j=0;j<3;j++)
     {
          t=t+(b[i][j]*d[j]);
     }
     c[i]=t%26;
}
```

```
printf("\nDecrypted Cipher Text :");
for(i=0;i<3;i++)
printf(" %c",c[i]+65);
getch();
return 0;
}
```

OUTPUT:

EX. NO: 1(D)

IMPLEMENTATION OF VIGENERE CIPHER

AIM:

To implement the Vigenere Cipher substitution technique using C program.

DESCRIPTION:

To encrypt, a table of alphabets can be used, termed a tabula recta, Vigenère square, or Vigenère table. It consists of the alphabet 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.

Each row starts with a key letter. The remainder of the row holds the letters A to Z. Although there are 26 key rows shown, you will only use as many keys as there are unique letters in the key string, here just 5 keys, {L, E, M, O, N}. For successive letters of the message, we are going to take successive letters of the key string, and encipher each message letter using its corresponding key row. Choose the next letter of the key, go along that row to find the column heading that matches the message character; the letter at the intersection of [key-row, msg-col] is the enciphered letter.

ALGORITHM:

STEP-1: Arrange the alphabets in row and column of a 26*26 matrix.

STEP-2: Circulate the alphabets in each row to position left such that the first letter is attached to last.

STEP-3: Repeat this process for all 26 rows and construct the final key matrix.

STEP-4: The keyword and the plain text is read from the user.

STEP-5: The characters in the keyword are repeated sequentially so as to match with that of the plain text.

STEP-6: Pick the first letter of the plain text and that of the keyword as the row indices and column indices respectively.

STEP-7: The junction character where these two meet forms the cipher character.

STEP-8: Repeat the above steps to generate the entire cipher text.

PROGRAM: (Vigenere Cipher)

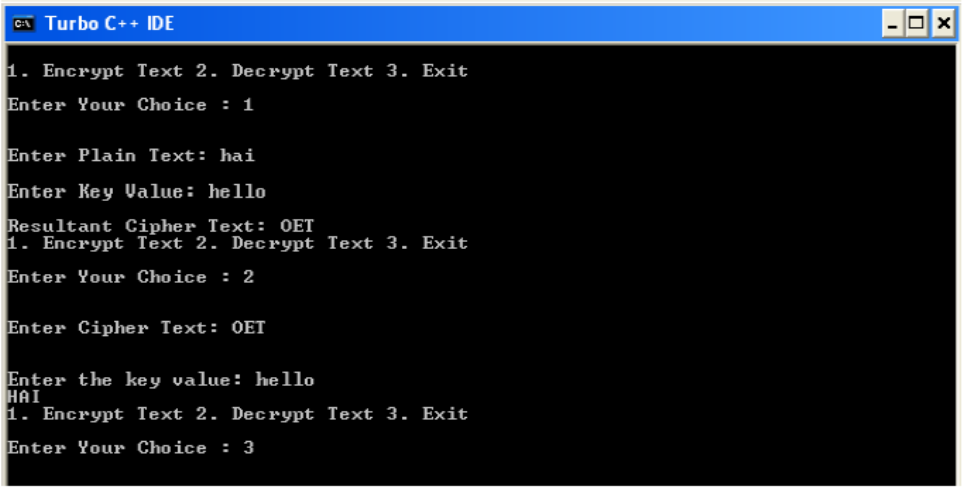
```
#include <stdio.h>
#include<conio.h>
#include <ctype.h>
#include <string.h>
void encipher();
void decipher();
void main()
{
     int choice;
     clrscr();
     while(1)
     {
          printf("\n1. Encrypt Text");
          printf("\t2. Decrypt Text");
          printf("\t3. Exit");
          printf("\n\nEnter Your Choice : ");
          scanf("%d",&choice);
          if(choice == 3)
          exit(0);
          else if(choice == 1)
          encipher();
          else if(choice == 2)
          decipher();
```

```
            else
            printf("Please Enter Valid Option.");
      }
}
void encipher()
{
      unsigned int i,j;
      char input[50],key[10];
      printf("\n\nEnter Plain Text: ");
```

```
      scanf("%s",input);
      printf("\nEnter Key Value: ");
      scanf("%s",key);
      printf("\nResultant Cipher Text: ");
      for(i=0,j=0;i<strlen(input);i++,j++)
      {
            if(j>=strlen(key))
                  {     j=0;
                  }
      printf("%c",65+(((toupper(input[i])-65)+(toupper(key[j])-
      65))%26));
      }}
```

```
void decipher()
{
      unsigned int i,j;
      char input[50],key[10];
      int value;
      printf("\n\nEnter Cipher Text: ");
      scanf("%s",input);
      printf("\n\nEnter the key value: ");
      scanf("%s",key);
for(i=0,j=0;i<strlen(input);i++,j++)
{
      if(j>=strlen(key))
      {     j=0;        }
      value = (toupper(input[i])-64)-(toupper(key[j])-64);
      if( value < 0)
      {  value = value * -1;
      }
      printf("%c",65 + (value % 26));
}}
```

OUTPUT:



EX. NO: 1(E)

IMPLEMENTATION OF RAIL FENCE – ROW & COLUMN TRANSFORMATION TECHNIQUE

AIM:

To write a C program to implement the rail fence transposition technique.

DESCRIPTION:

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

ALGORITHM:

STEP-1: Read the Plain text.

STEP-2: Arrange the plain text in row columnar matrix format.

STEP-3: Now read the keyword depending on the number of columns of the plain text.

STEP-4: Arrange the characters of the keyword in sorted order and the corresponding columns of the plain text.

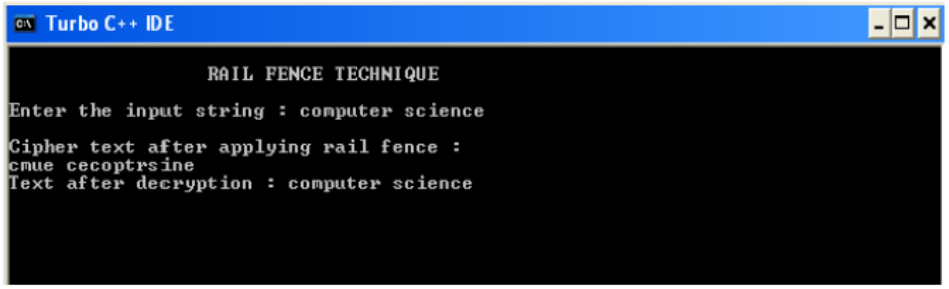
STEP-5: Read the characters row wise or column wise in the former order to get the cipher text.

PROGRAM: (Rail Fence)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
      int i,j,k,l;
      char a[20],c[20],d[20];
      clrscr();
      printf("\n\t\t RAIL FENCE TECHNIQUE");
      printf("\n\nEnter the input string : ");
      gets(a);
      l=strlen(a);

/*Ciphering*/
for(i=0,j=0;i<l;i++)
{
      if(i%2==0)
      c[j++]=a[i];
}
for(i=0;i<l;i++)
{
      if(i%2==1)
      c[j++]=a[i];
}
```

```
c[j]='\0';
printf("\nCipher text after applying rail fence :");
printf("\n%s",c);

/*Deciphering*/
if(l%2==0)
     k=l/2;
else
     k=(l/2)+1;
for(i=0,j=0;i<k;i++)
{
    d[j]=c[i];
    j=j+2;
}
for(i=k,j=1;i<l;i++)
{
    d[j]=c[i];
    j=j+2;
}
d[l]='\0';
printf("\nText after decryption : ");
printf("%s",d);
getch();
}
```

OUTPUT:



Programming Problem: Write a program that can encrypt and decrypt using the general Caesar cipher, also known as an additive cipher.

Complex Problem: Write a program that can perform a letter frequency attack on an additive cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify "give me the top 10 possible plaintexts".

EX. NO: 2(A) **IMPLEMENTATION OF DES**

AIM:

To write a C program to implement Data Encryption Standard (DES) using C Language.

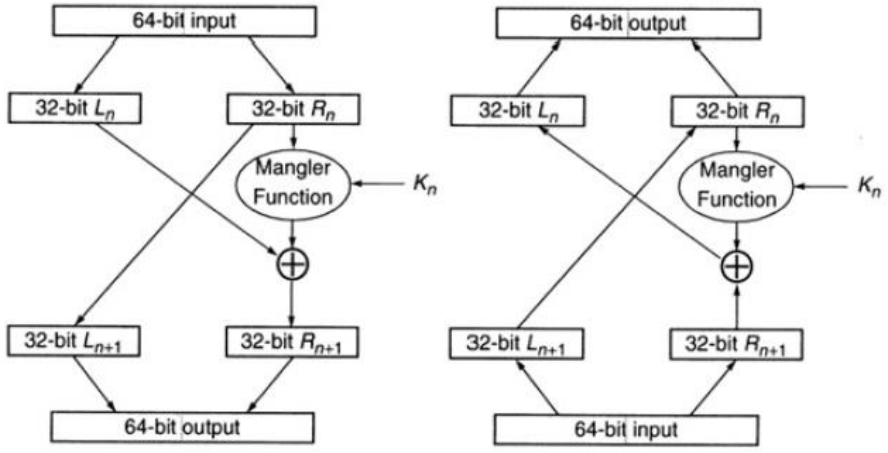
DESCRIPTION:

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted k_1 to k_{16}. Given that "only" 56 bits are actually used for encrypting, there can be 2^{56} different keys.

The main parts of the algorithm are as follows:

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

EXAMPLE:



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.

PROGRAM:

DES.java

```
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 ;
```

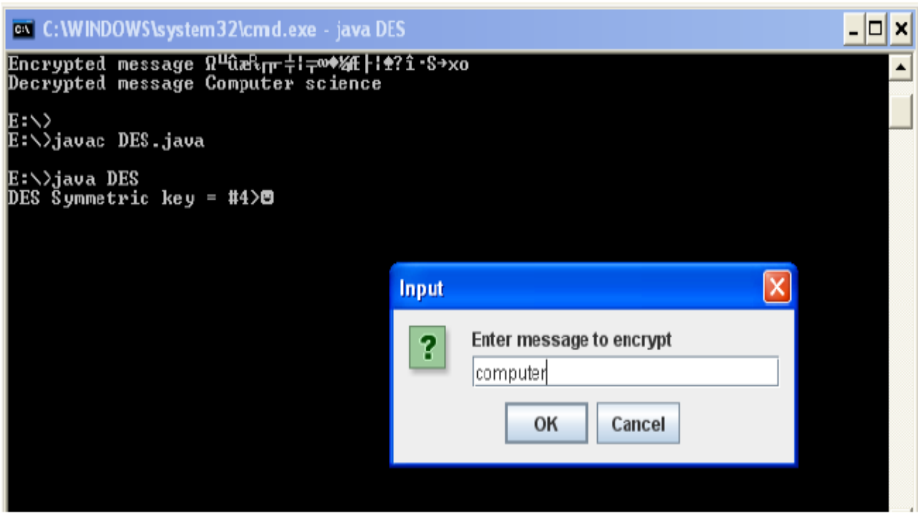
```
class DES {
     byte[] skey = new byte[1000];
     String skeyString;
     static byte[] raw;
     String inputMessage,encryptedData,decryptedMessage;
public DES()
{
try
{
     generateSymmetricKey();
     inputMessage=JOptionPane.showInputDialog(null,"Enter
     message to encrypt");
     byte[] 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);
}
```

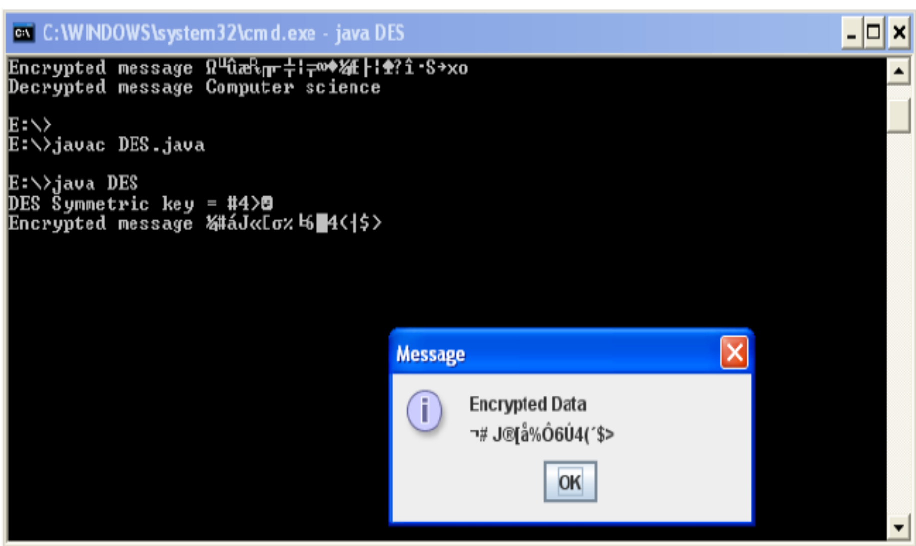
```
catch(Exception e)
{
     System.out.println(e);
}
}
```

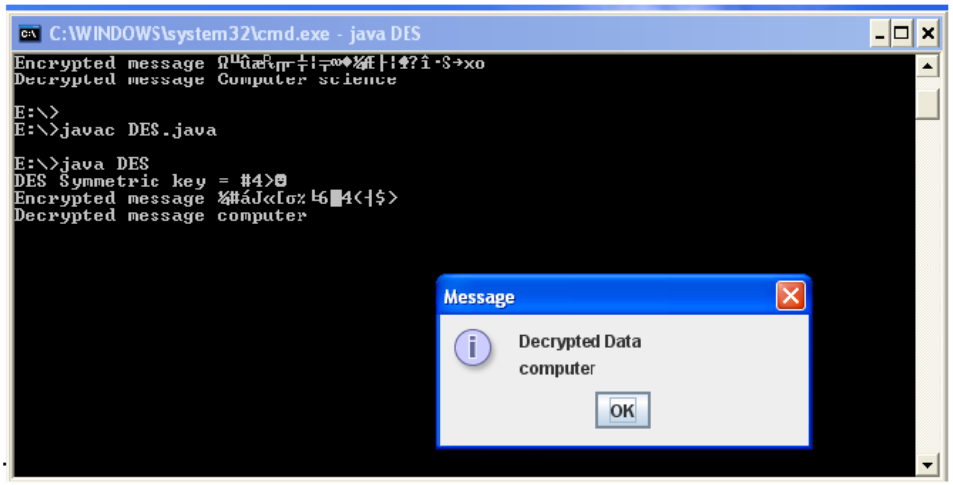
```
void generateSymmetricKey() {
try {
      Random r = new Random();
      int num = r.nextInt(10000);
      String knum = String.valueOf(num);
      byte[] knumb = knum.getBytes();
      skey=getRawKey(knumb);
      skeyString = new String(skey);
      System.out.println("DES Symmetric key = "+skeyString);
}
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();
}
}
```

OUTPUT:


C:\WINDOWS\system32\cmd.exe - java DES
Decrypted message Computer science
E:\>
E:\>javac DES.java
E:\>java DES
Input
Enter message to encrypt
computer
OK
Cancel


C:\WINDOWS\system32\cmd.exe - java DES
Decrypted message Computer science
E:\>
E:\>javac DES.java
E:\>java DES
Message
Encrypted Data
OK



Programming Problem: Create software that can encrypt and decrypt using a general substitution block cipher.

Complex Problem: Create software that can encrypt and decrypt using S-DES. Test data: Use plaintext, ciphertext, and key of Problem shown below.

For any block cipher, the fact that it is a nonlinear function is crucial to its security. To see this, suppose that we have a linear block cipher EL that encrypts 128-bit blocks of plaintext into 128-bit blocks of ciphertext. Let EL(k, m) denote the encryption of a 128- bit message m under a key k (the actual bit length of k is irrelevant). Thus EL(k, [m1 m2]) = EL(k, m1) EL(k, m1) for all 128-bit patterns m1, m2 Describe how, with 128 chosen ciphertexts, an adversary can decrypt any ciphertext without knowledge of the secret key k. (A "chosen ciphertext" means that an adversary has the ability to choose a ciphertext and then obtain its decryption. Here, you have 128 plaintext/ciphertext pairs to work with and you have the ability to chose the value of the ciphertexts.)

2. Write a simple four-function calculator in GF(24). You may use table lookups for the multiplicative inverses.

3. Write a simple four-function calculator in GF(28). You should compute the multiplicative inverses on the fly.

5.13 Create software that can encrypt and decrypt using S-AES. Test data: a binary plaintext of 0110 1111 0110 1011 encrypted with a binary key of 1010 0111 0011 1011 should give a binary ciphertext of 0000 0111 0011 1000 less ecb $$$). Decryption should work

correspondingly file:///D|/1/0131873164/ch05lev1sec4.html (4 von 5) [14.10.2007 09:40:31]
Section 5.4. Key Terms, Review Questions, and Problems 5.14 Implement a differential cryptanalysis attack on 1-round S-AES

EX. NO: 2(B) **IMPLEMENTATION OF RSA**

AIM:

To write a C program to implement the RSA encryption algorithm.

DESCRIPTION:

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integer m:

$$(m^e)^d = m \pmod n$$

The public key is represented by the integers n and e; and, the private key, by the integer d. m represents the message. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.

ALGORITHM:

STEP-1: Select two co-prime numbers as p and q.

STEP-2: Compute n as the product of p and q.

STEP-3: Compute (p-1)*(q-1) and store it in z.

STEP-4: Select a random prime number e that is less than that of z.

STEP-5: Compute the private key, d as e * mod^{-1}(z).

STEP-6: The cipher text is computed as $message^{e}$ * mod n.

STEP-7: Decryption is done as $cipher^{d}$mod n.

PROGRAM: (RSA)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
     long int
     p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
     char msg[100];
     int prime(long int);
     void ce();
     long int cd(long int);
     void encrypt();
     void decrypt();
void main()
{
     clrscr();
     printf("\nENTER FIRST PRIME NUMBER\n");
     scanf("%d",&p);
     flag=prime(p);
     if(flag==0)
      {
          printf("\nWRONG INPUT\n");
          getch();
      }
      printf("\nENTER ANOTHER PRIME NUMBER\n");
      scanf("%d",&q);
      flag=prime(q);
      if(flag==0||p==q)
      {
       printf("\nWRONG INPUT\n");
       getch();
      }
      printf("\nENTER MESSAGE\n");
```

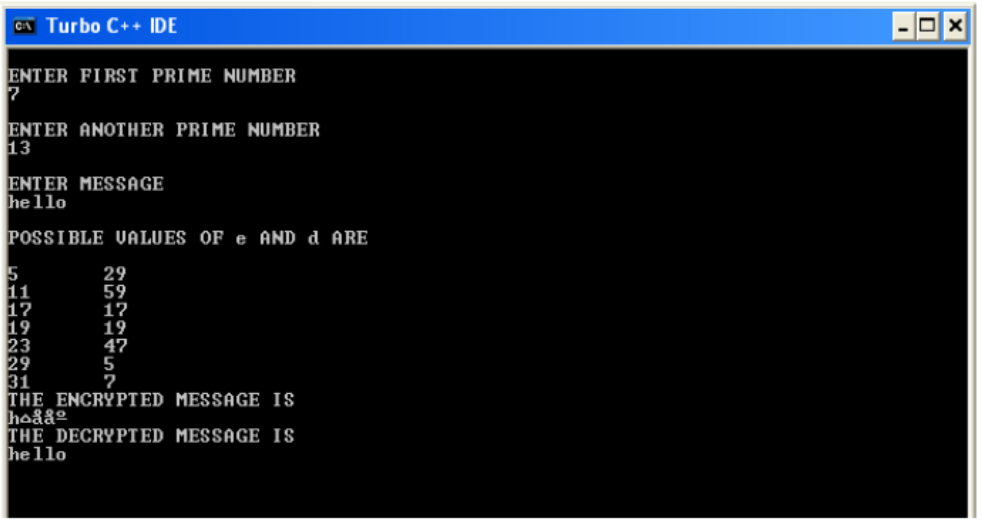
```
        fflush(stdin);
        scanf("%s",msg);
        for(i=0;msg[i]!=NULL;i++)
        m[i]=msg[i];
        n=p*q;

        t=(p-1)*(q-1);
        ce();
        printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
        for(i=0;i<j-1;i++)
        printf("\n%ld\t%ld",e[i],d[i]);
        encrypt();
        decrypt();
        getch();
}
int prime(long int pr)
{
int i;
j=sqrt(pr);
for(i=2;i<=j;i++)
{
if(pr%i==0)
return 0;
}
return 1;
}

void ce()
{
int k;
k=0;
for(i=2;i<t;i++)
{
 if(t%i==0)
 continue;
 flag=prime(i);
 if(flag==1&&i!=p&&i!=q)
 {
 e[k]=i;
 flag=cd(e[k]);
 if(flag>0)
 {
 d[k]=flag;
 k++;
 }
 if(k==99)
 break;
 } } }
 long int cd(long int x)
 {
 long int k=1;
```

```
while(1)
{
k=k+t;
if(k%x==0)
return(k/x);
} }
void encrypt() {
long int pt,ct,key=e[0],k,len;
i=0;
len=strlen(msg);
```

```
while(i!=len) {
pt=m[i];
pt=pt-96;
k=1;
for(j=0;j<key;j++)
{  k=k*pt;
k=k%n;
}
temp[i]=k;
ct=k+96;
en[i]=ct;
i++;
}
en[i]=-1;
printf("\nTHE ENCRYPTED MESSAGE IS\n");
for(i=0;en[i]!=-1;i++)
printf("%c",en[i]);
}
```

```
void decrypt()
{
long int pt,ct,key=d[0],k;
i=0;
while(en[i]!=-1)
{
ct=temp[i];
k=1;
for(j=0;j<key;j++)
{
k=k*ct;
k=k%n;
}
pt=k+96;
m[i]=pt;
i++;
}
m[i]=-1;
printf("\nTHE DECRYPTED MESSAGE IS\n");
for(i=0;m[i]!=-1;i++)
printf("%c",m[i]);
}
```

OUTPUT:



EX. NO: 2(C)

IMPLEMENTATION OF DIFFIE HELLMAN KEY EXCHANGE ALGORITHM

AIM:

To implement the Diffie-Hellman Key Exchange algorithm using C language.

DESCRIPTION:

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

ALGORITHM:

STEP-1: Both Alice and Bob shares the same public keys g and p.

STEP-2: Alice selects a random public key a.

STEP-3: Alice computes his secret key A as g^a mod p.

STEP-4: Then Alice sends A to Bob.

STEP-5: Similarly Bob also selects a public key b and computes his secret key as B and sends the same back to Alice.

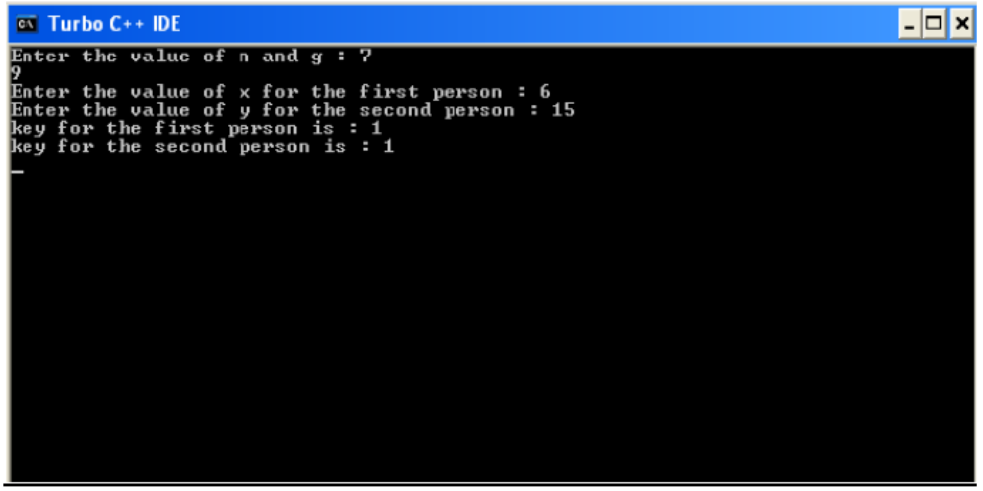
STEP-6: Now both of them compute their common secret key as the other one's secret key power of a mod p.

PROGRAM: (Diffie Hellman Key Exchange)

```
#include<stdio.h>
#include<conio.h>
long long int power(int a, int b, int mod)
{
      long long int t;
      if(b==1)
      return a;
      t=power(a,b/2,mod);
      if(b%2==0)
      return (t*t)%mod;
      else
      return (((t*t)%mod)*a)%mod;
}
```

```
long int calculateKey(int a, int x, int n)
{
     return power(a,x,n);
}
void main()
{
     int n,g,x,a,y,b;
     clrscr();
     printf("Enter the value of n and g : ");
     scanf("%d%d",&n,&g);
     printf("Enter the value of x for the first person : ");
     scanf("%d",&x);
     a=power(g,x,n);
     printf("Enter the value of y for the second person : ");
     scanf("%d",&y);
     b=power(g,y,n);
     printf("key for the first person is :
     %lld\n",power(b,x,n));
     printf("key for the second person is :
     %lld\n",power(a,y,n));
     getch();
}
```

OUTPUT:



EX. NO: 2(D)

IMPLEMENTATION OF MD5

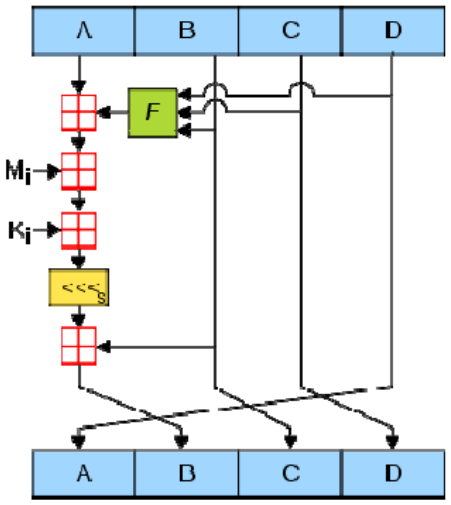
AIM:

To write a C program to implement the MD5 hashing technique.

DESCRIPTION:

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks. 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 less than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.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. The main algorithm then uses each 512-bit message block in turn to modify the state.

EXAMPLE:



ALGORITHM:

STEP-1: Read the 128-bit plain text.

STEP-2: Divide into four blocks of 32-bits named as A, B, C and D.

STEP-3: Compute the functions f, g, h and i with operations such as, rotations, permutations, etc,.

STEP-4: The output of these functions are combined together as F and performed circular shifting and then given to key round.

STEP-5: Finally, right shift of 's' times are performed and the results are combined together to produce the final output.

PROGRAM:(MD5)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include<conio.h>
typedef union uwb
{
    unsigned w;
    unsigned char b[4];
} MD5union;
     typedef unsigned DigestArray[4];
     unsigned func0( unsigned abcd[] ){

     return ( abcd[1] & abcd[2]) | (~abcd[1] & abcd[3]);}
     unsigned func1( unsigned abcd[] ){
     return ( abcd[3] & abcd[1]) | (~abcd[3] & abcd[2]);}
     unsigned func2( unsigned abcd[] ){
     return  abcd[1] ^ abcd[2] ^ abcd[3];}
     unsigned func3( unsigned abcd[] ){
     return abcd[2] ^ (abcd[1] |~ abcd[3]);}
     typedef unsigned (*DgstFctn)(unsigned a[]);
unsigned *calctable( unsigned *k)
{
     double s, pwr;
     int i;
     pwr = pow( 2, 32);
for (i=0; i<64; i++)
{
     s = fabs(sin(1+i));
     k[i] = (unsigned)( s * pwr );
}
     return k;
}
```

```
unsigned rol( unsigned r, short N )
{
    unsigned  mask1 = (1<<N) -1;
    return ((r>>(32-N)) & mask1) | ((r<<N) & ~mask1);
}
```

```
unsigned *md5( const char *msg, int mlen)
{
    static DigestArray h0 = { 0x67452301, 0xEFCDAB89,
    0x98BADCFE, 0x10325476 };
    static DgstFctn ff[] = { &func0, &func1, &func2, &func3};
    static short M[] = { 1, 5, 3, 7 };
    static short O[] = { 0, 1, 5, 0 };
    static short rot0[] = { 7,12,17,22};
    static short rot1[] = { 5, 9,14,20};
    static short rot2[] = { 4,11,16,23};
    static short rot3[] = { 6,10,15,21};
    static short *rots[] = {rot0, rot1, rot2, rot3 };
    static unsigned kspace[64];
    static unsigned *k;
    static DigestArray h;
    DigestArray abcd;
    DgstFctn fctn;
    short m, o, g;
    unsigned f;
    short *rotn;
    union
    {
```

```
        unsigned w[16];
        char     b[64];
}mm;
        int os = 0;
        int grp, grps, q, p;
        unsigned char *msg2;
        if (k==NULL) k= calctable(kspace);
for (q=0; q<4; q++) h[q] = h0[q];   // initialize
{
         grps  = 1 + (mlen+8)/64;
         msg2 = malloc( 64*grps);
         memcpy( msg2, msg, mlen);
         msg2[mlen] = (unsigned char)0x80;
         q = mlen + 1;
 while (q < 64*grps){ msg2[q] = 0; q++ ; }
 {
         MD5union u;
         u.w = 8*mlen;
         q -= 8;
         memcpy(msg2+q, &u.w, 4 );
 }
 }
```

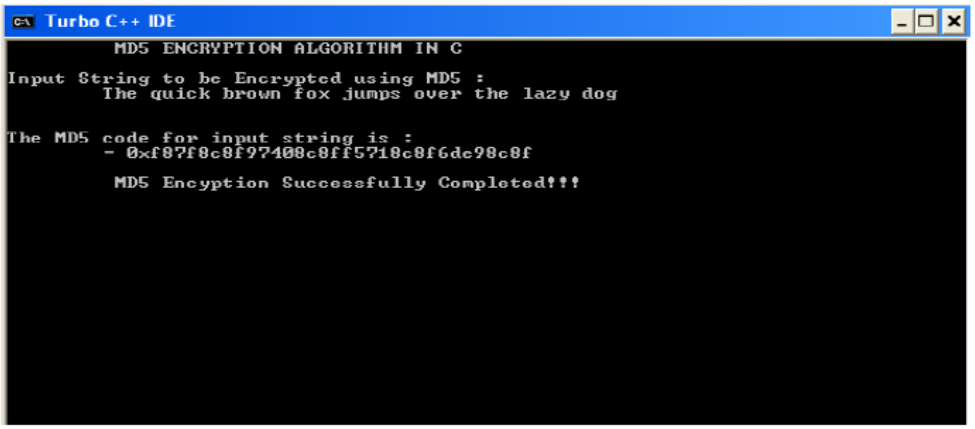
```
 for (grp=0; grp<grps; grp++)
 {
         memcpy( mm.b, msg2+os, 64);
```

```
        for(q=0;q<4;q++) abcd[q] = h[q];
        for (p = 0; p<4; p++)
        {
             fctn = ff[p];
             rotn = rots[p];
             m = M[p]; o= O[p];
        for (q=0; q<16; q++)
        {
             g = (m*q + o) % 16;
            f = abcd[1] + rol( abcd[0]+ fctn(abcd)+k[q+16*p]
            + mm.w[g], rotn[q%4]);
            abcd[0] = abcd[3];
             abcd[3] = abcd[2];
             abcd[2] = abcd[1];
            abcd[1] = f;
        }}
  for (p=0; p<4; p++)
  h[p] += abcd[p];
  os += 64;
  }
return h;}
```

```
void main()
{
     int j,k;
     const char *msg = "The quick brown fox jumps over
     the lazy dog";
     unsigned *d = md5(msg, strlen(msg));
     MD5union u;
     clrscr();
     printf("\t MD5 ENCRYPTION ALGORITHM IN C \n\n");
     printf("Input String to be Encrypted using MD5 :
     \n\t%s",msg);
     printf("\n\nThe MD5 code for input string is: \n");
     printf("\t= 0x");
     for (j=0;j<4; j++){
     u.w = d[j];
     for (k=0;k<4;k++) printf("%02x",u.b[k]);
}
printf("\n");
printf("\n\t MD5 Encyption Successfully
Completed!!!\n\n");
getch();
system("pause");
```

```
getch();}
```

OUTPUT:



EX. NO: 2(E) **IMPLEMENTATION OF SHA-I**

AIM:

To implement the SHA-I hashing technique using C program.

DESCRIPTION:

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

ALGORITHM:

STEP-1: Read the 256-bit key values.

STEP-2: Divide into five equal-sized blocks named A, B, C, D and E.

STEP-3: The blocks B, C and D are passed to the function F.

STEP-4: The resultant value is permuted with block E.

STEP-5: The block A is shifted right by 's' times and permuted with the result of step-4.

STEP-6: Then it is permuted with a weight value and then with some other key pair and taken as the first block.

STEP-7: Block A is taken as the second block and the block B is shifted by 's' times and taken as the third block.

STEP-8: The blocks C and D are taken as the block D and E for the final output.

PROGRAM: (Secure Hash Algorithm)

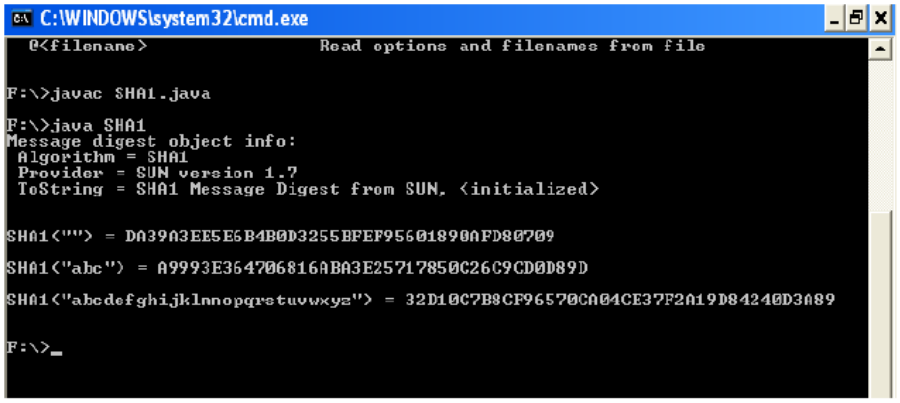
```
import java.security.*;
public class SHA1 {
     public static void main(String[] a) {
     try {
     MessageDigest md = MessageDigest.getInstance("SHA1");
     System.out.println("Message digest object info: ");
     System.out.println(" Algorithm = " +md.getAlgorithm());
     System.out.println(" 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+"\") =
        +bytesToHex(output));
        input = "abc";
        md.update(input.getBytes());
        output = md.digest();
        System.out.println();
        System.out.println("SHA1(\""+input+"\") = "
        +bytesToHex(output));
        input = "abcdefghijklmnopqrstuvwxyz";
        md.update(input.getBytes());
        output = md.digest();
        System.out.println();
        System.out.println("SHA1(\"" +input+"\") = "
        +bytesToHex(output));
        System.out.println(""); }
        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:



IMPLEMENTATION OF DIGITAL SIGNATURE STANDARD

AIM:

To write a C program to implement the signature scheme named digital signature standard (Euclidean Algorithm).

ALGORITHM:

STEP-1: Alice and Bob are investigating a forgery case of x and y.

STEP-2: X had document signed by him but he says he did not sign that document digitally.

STEP-3: Alice reads the two prime numbers p and a.

STEP-4: He chooses a random co-primes alpha and beta and the x's original signature x.

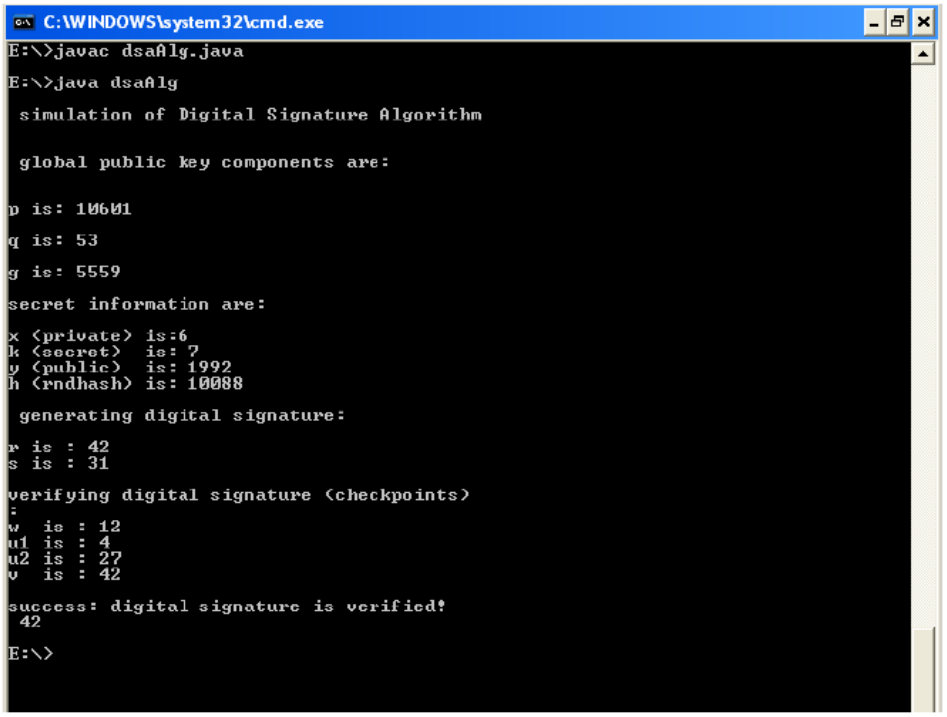
STEP-5: With these values, he applies it to the elliptic curve cryptographic equation to obtain y.

STEP-6: Comparing this 'y' with actual y's document, Alice concludes that y is a forgery.

PROGRAM: (Digital Signature Standard)

```
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 ");
    }
    }
    }
```



Programming Problem: Write a computer program that implements fast exponentiation (successive squaring) modulo n.

Complex Problem: Write a computer program that implements the Miller-Rabin algorithm for a userspecified n. The program should allow the user two choices: (1) specify a possible witness a to test using the Witness procedure, or (2) specify a number s of random witnesses for the Miller-Rabin test to check.

EX. NO: 04

SECURE DATA STORAGE, SECURE DATA TRANSMISSION AND FOR CREATING DIGITAL SIGNATURES (GNUPG)

AIM:

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

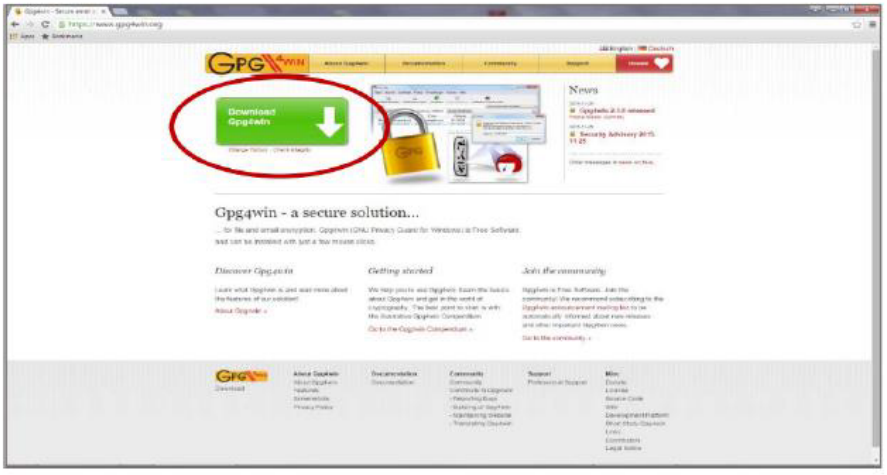
INTRODUCTION:

- 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:

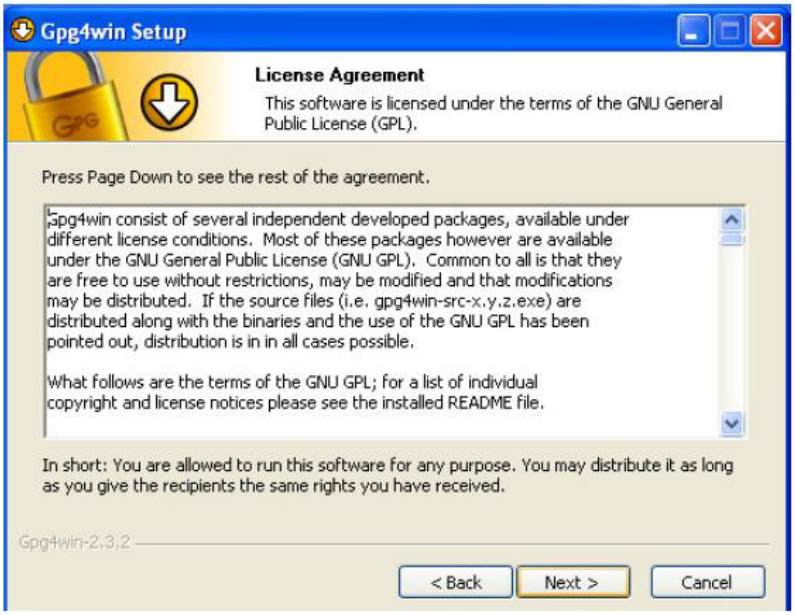
1. Visit www.gpg4win.org. Click on the "Gpg4win 2.3.0" button

2. On the following screen, click the "Download Gpg4win" button.

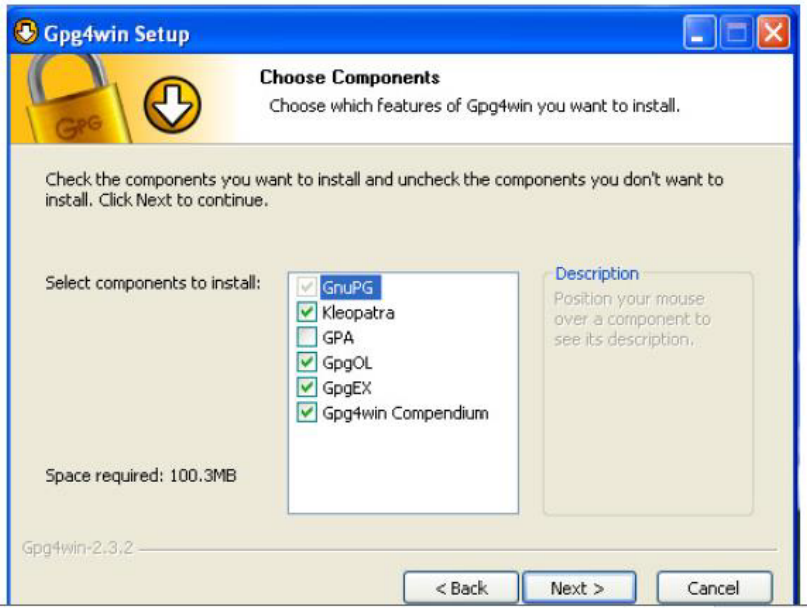


3. When the "Welcome" screen is displayed, click the "Next" button

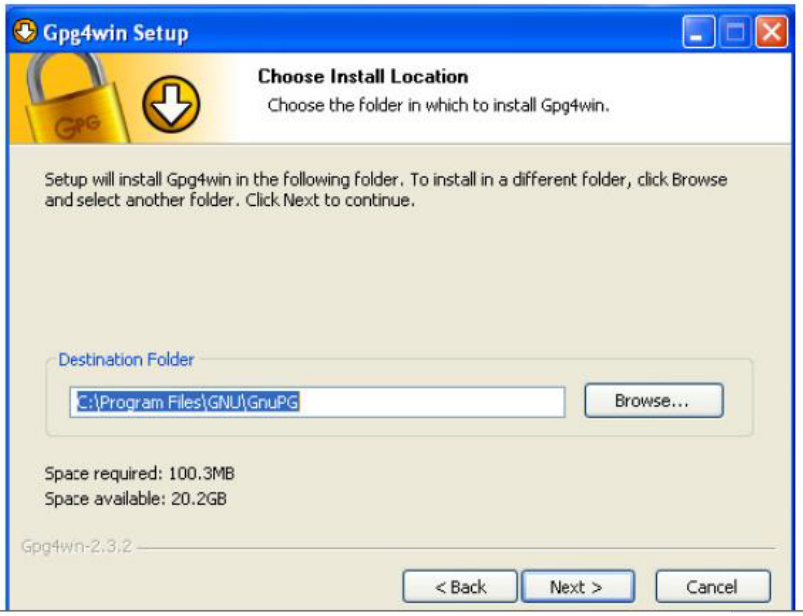
4. When the "License Agreement" page is displayed, click the "Next" button



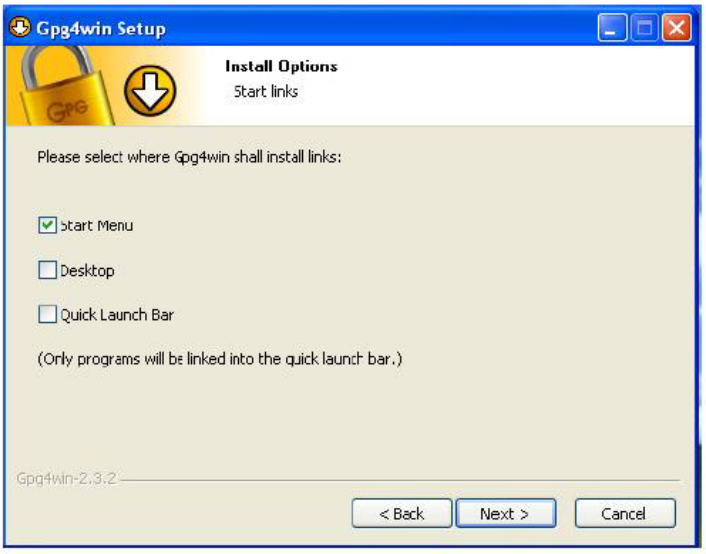
5. Set the check box values as specified below, then click the "Next" button



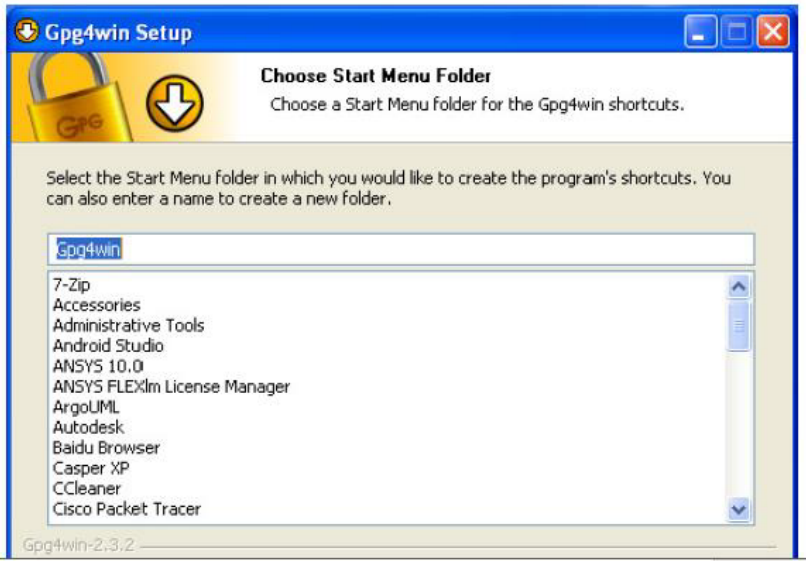
6. Set the location where you want the software to be installed. The default location is fine. Then, click the "Next" button.



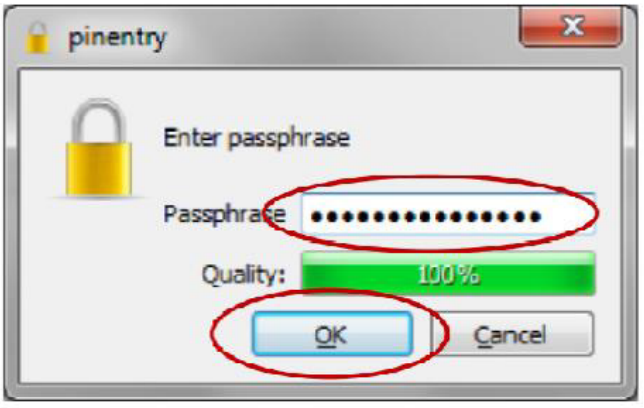
7. Specify where you want shortcuts to the software placed, then click the "Next" button.



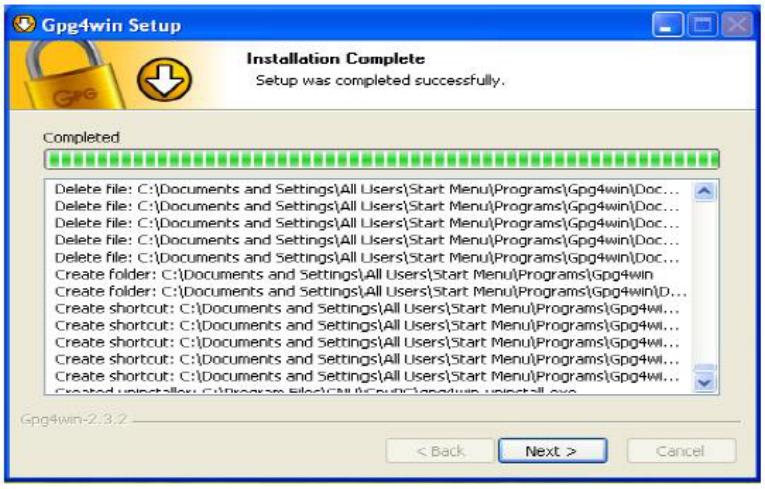
8. If you selected to have a GPG shortcut in your Start Menu, specify the folder in which it will be placed. The default "Gpg4win" is OK. Click the "Install" button to continue



9. A warning will be displayed if you have Outlook or Explorer opened. If this occurs, click the "OK" button.

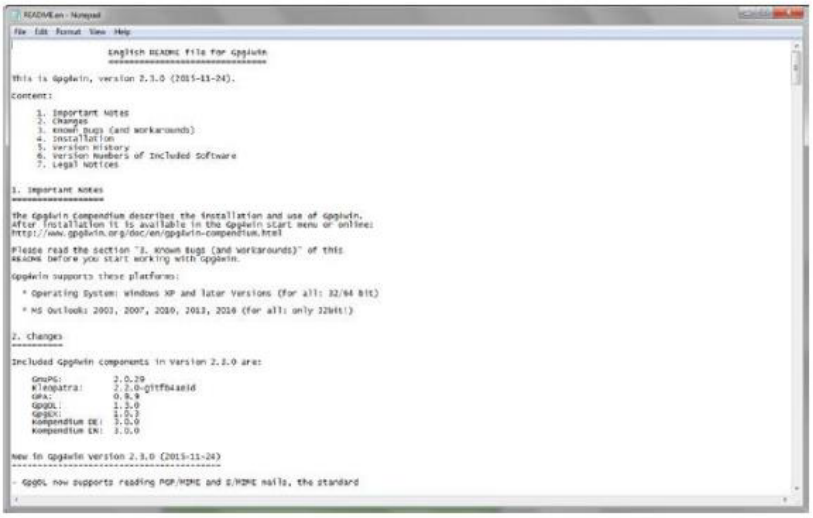


10. The installation process will tell you when it is complete. Click the "Next" button



11. Once the Gpg4win setup wizard is complete, the following screen will be displayed. Click the "Finish" button

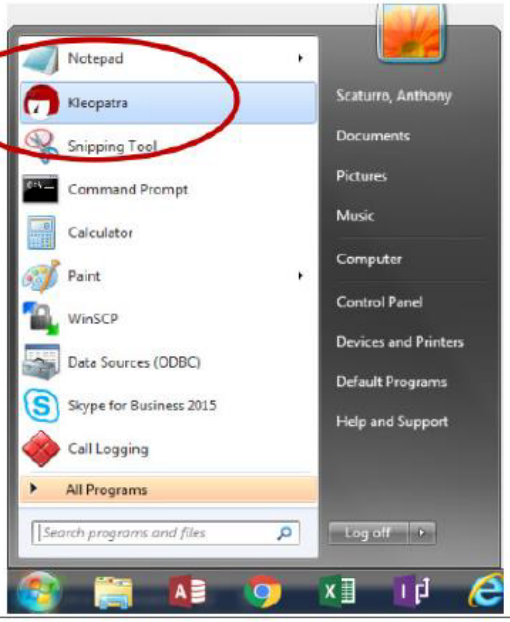
12. If you do not uncheck the "Show the README file" check box, the README file will be displayed. The window can be closed after you've reviewed it.



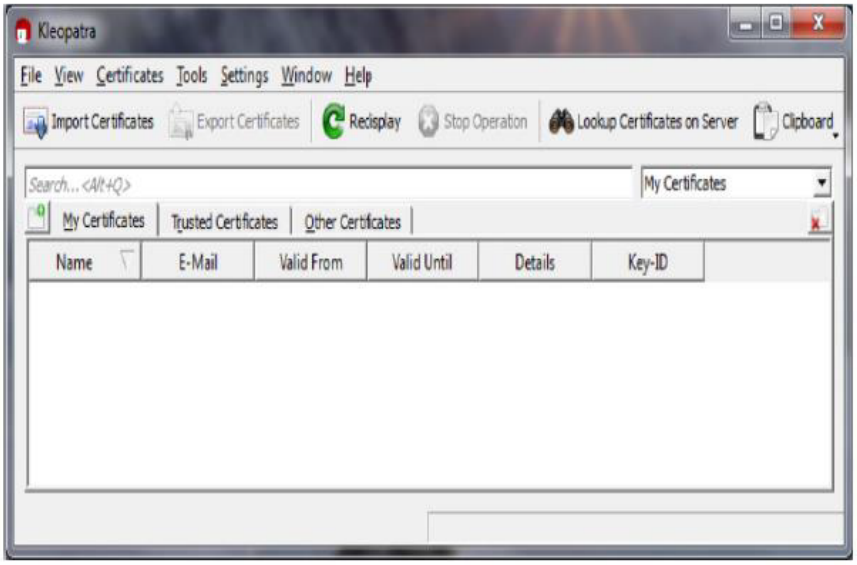
CREATING YOUR PUBLIC AND PRIVATE KEYS

GPG encryption and decryption is based upon the keys of the person who will be receiving the encrypted file or message. Any individual who wants to send the person an encrypted file or message must possess the recipient's public key certificate to encrypt the message. The recipient must have the associated private key, which is different than the public key, to be able to decrypt the file. The public and private key pair for an individual is usually generated by the individual on his or her computer using the installed GPG program, called "Kleopatra" and the following procedure:

1. From your start bar, select the "Kleopatra" icon to start the Kleopatra certificate management software

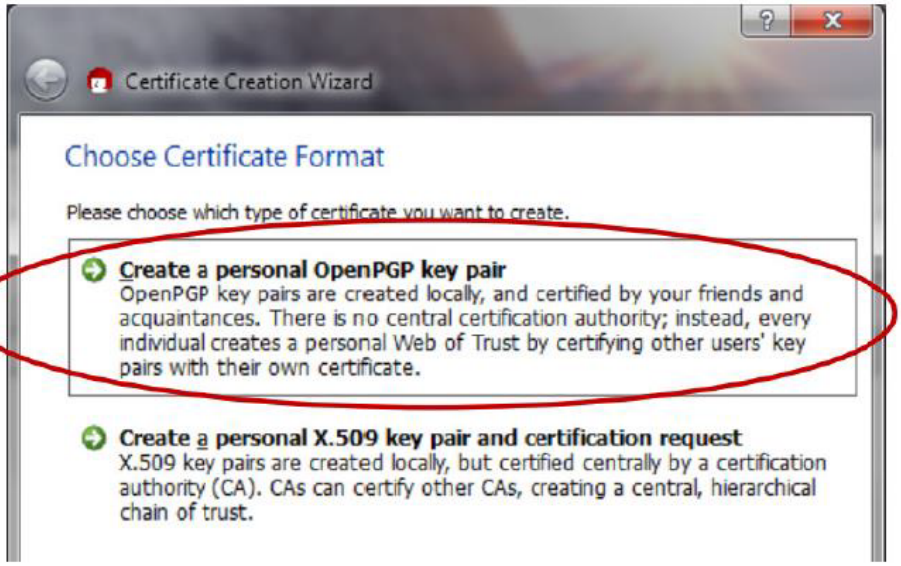


2. The following screen will be displayed

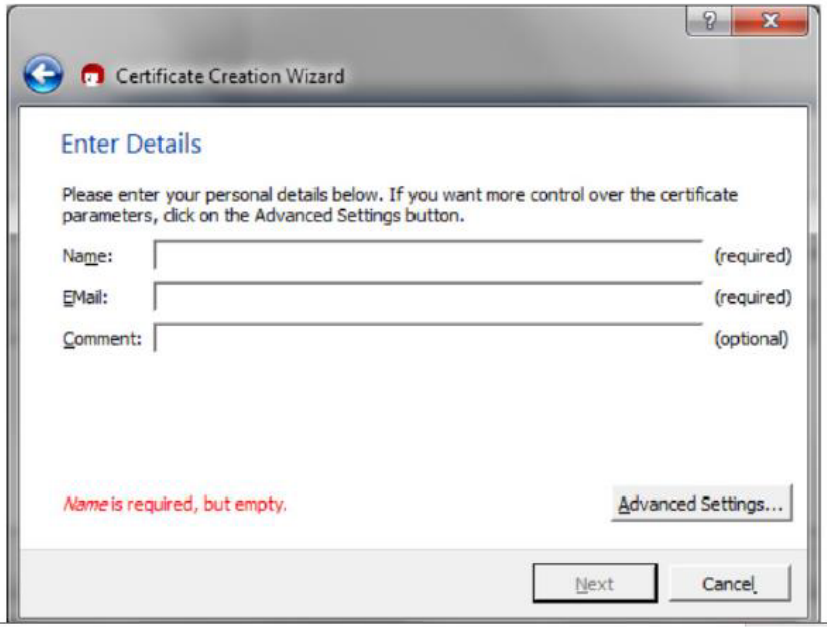


3. From the "File" dropdown, click on the "New Certificate" option

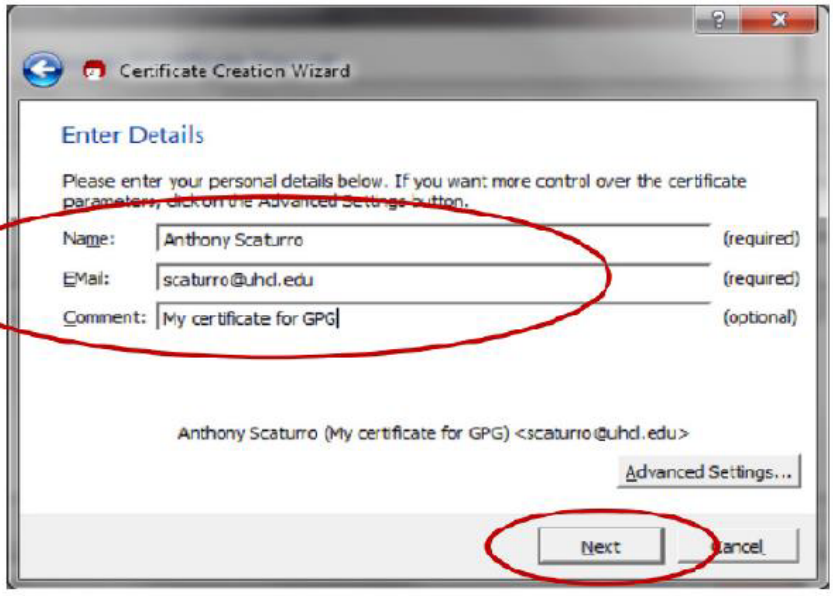
4. The following screen will be displayed. Click on "Create a personal OpenGPG key pair" and the "Next" button



5. The Certificate Creation Wizard will start and display the following:

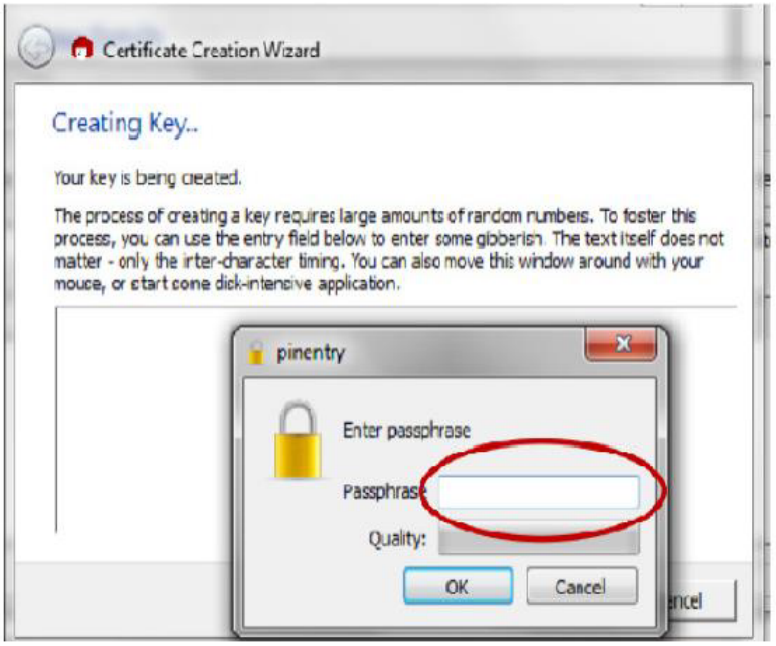


6. Enter your name and e-mail address. You may also enter an optional comment. Then, click the "Next" button

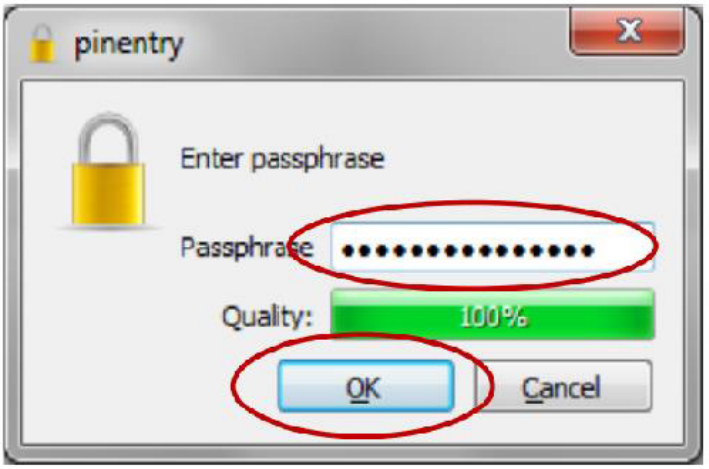


7. Review your entered values. If OK, click the "Create Key" button

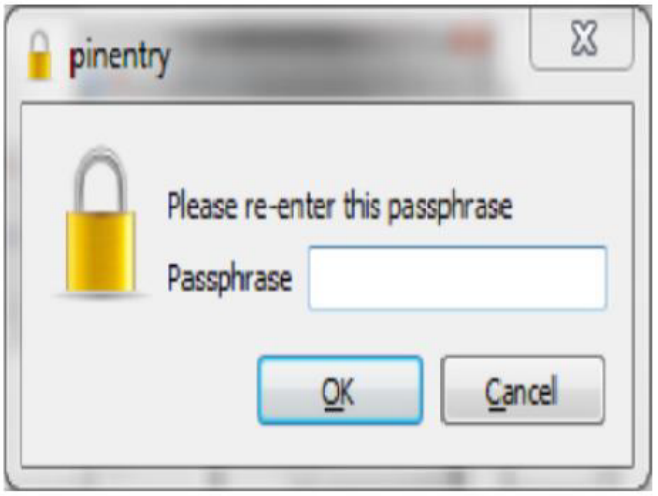
8. You will be asked to enter a passphrase



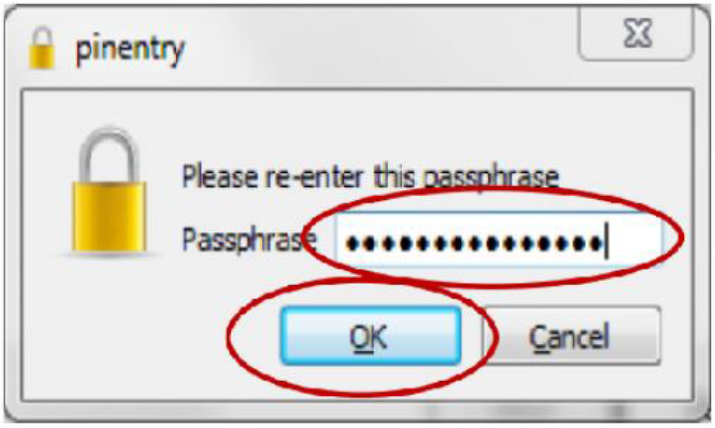
9. The passphrase should follow strong password standards. After you've entered your passphrase, click the "OK" button.



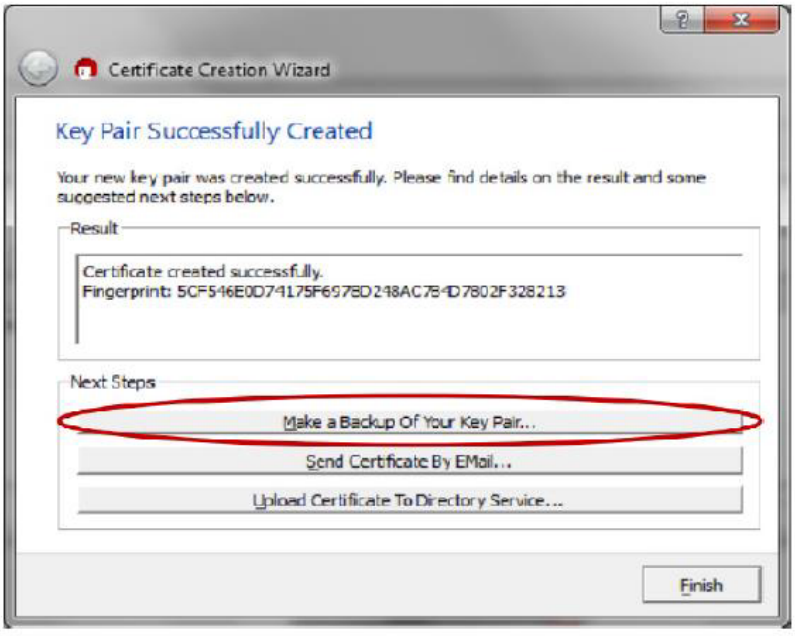
10. You will be asked to re-enter the passphrase



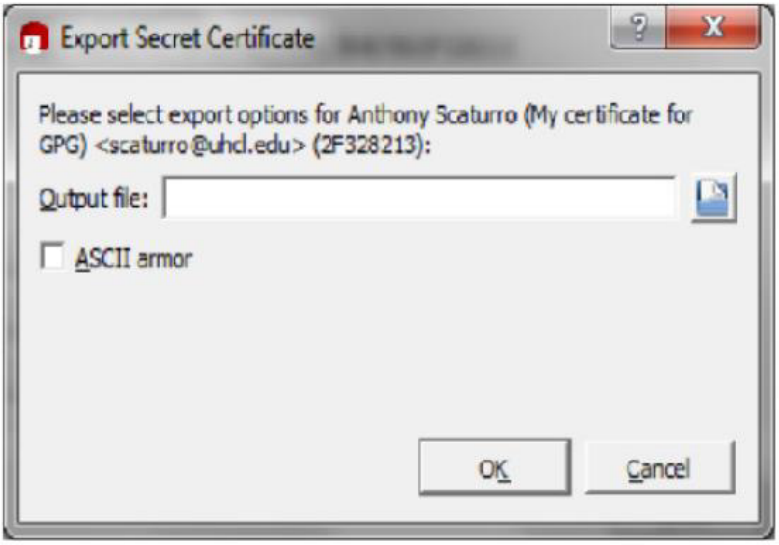
11. Re-enter the passphrase value. Then click the "OK" button. If the passphrases match, the certificate will be created.



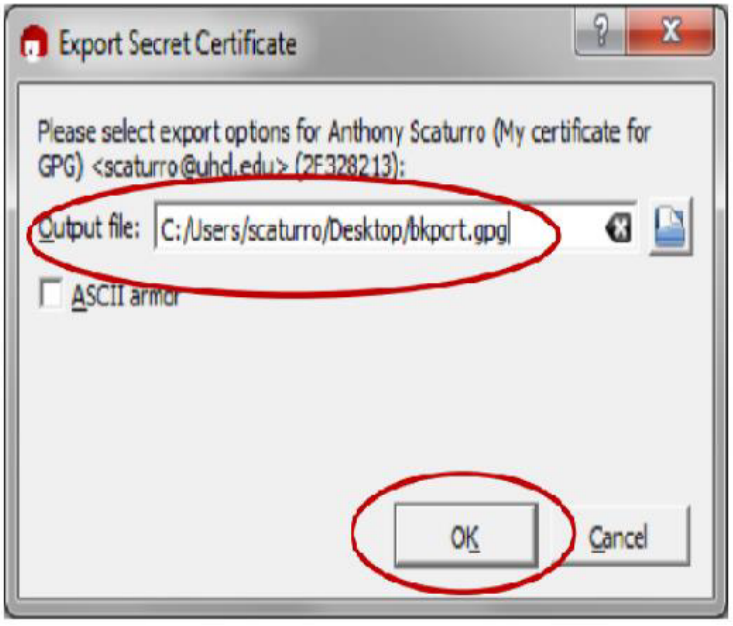
12. Once the certificate is created, the following screen will be displayed. You can save a backup of your public and private keys by clicking the "Make a backup Of Your Key Pair" button. This backup can be used to copy certificates onto other authorized computers.



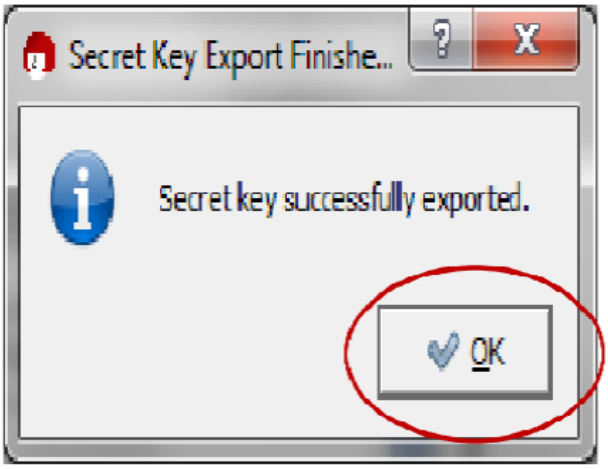
13. If you choose to backup your key pair, you will be presented with the following screen:



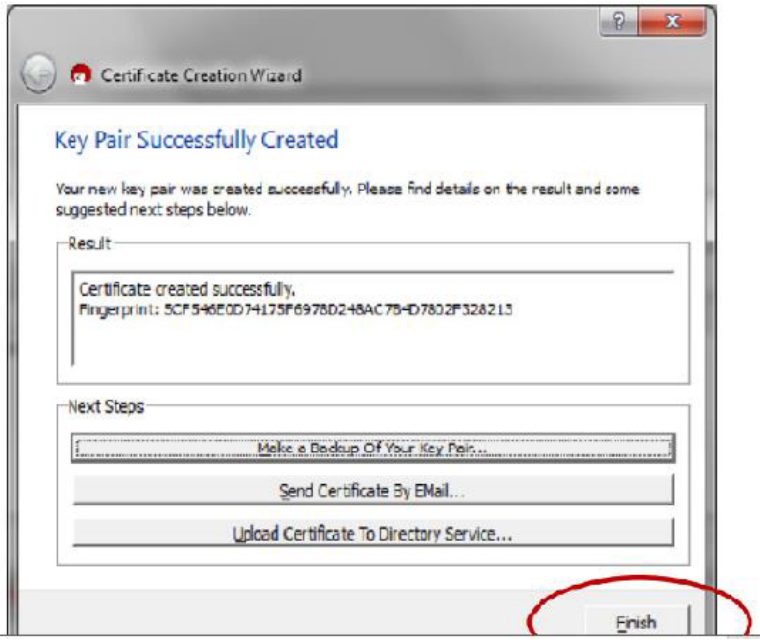
14. Specify the folder and name the file. Then click the "OK" button.



15. After the key is exported, the following will be displayed. Click the "OK" button.



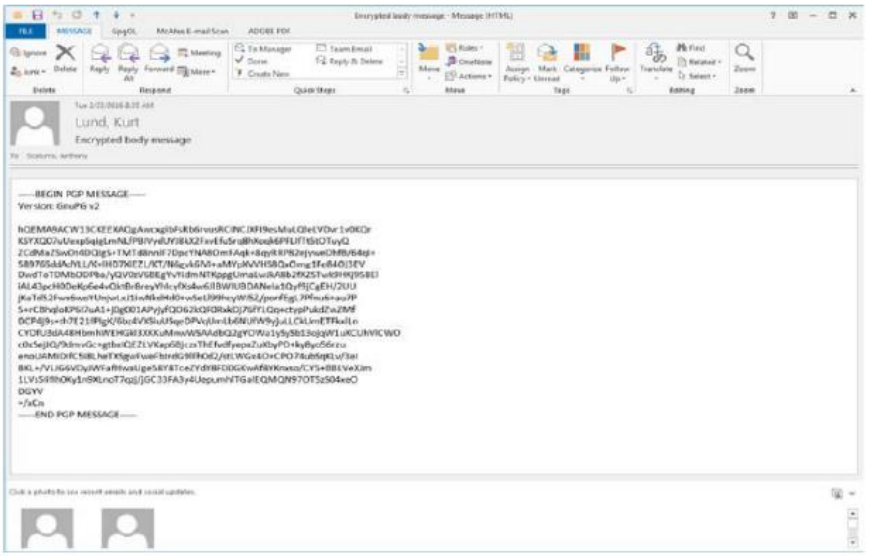
16. You will be returned to the "Key Pair Successfully Created" screen. Click the "Finish" button.



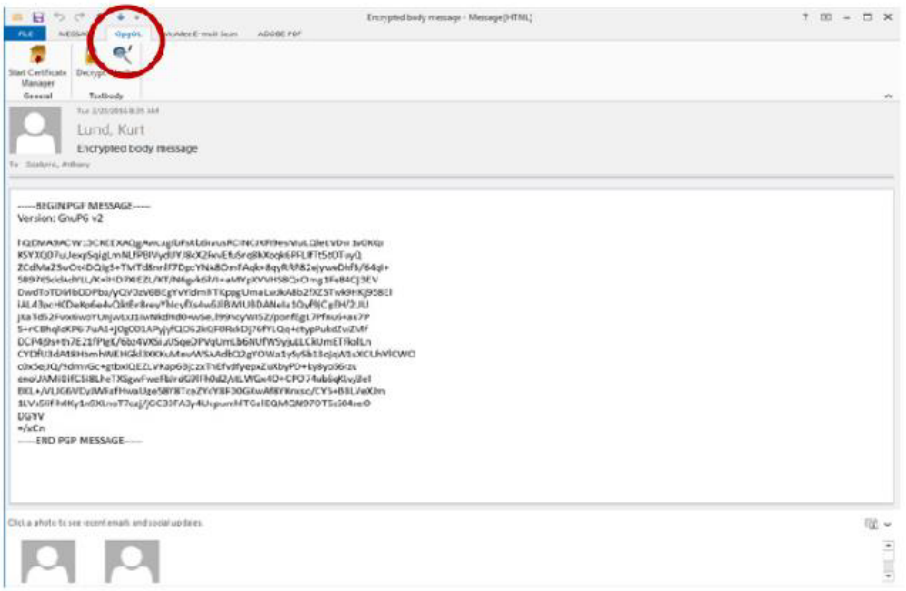
17. Before the program closes, you will need to confirm that you want to close the program by clicking on the "Quit Kleopatra" button

DECRYPTING AN ENCRYPTED E-MAIL THAT HAS BEEN SENT TO YOU:

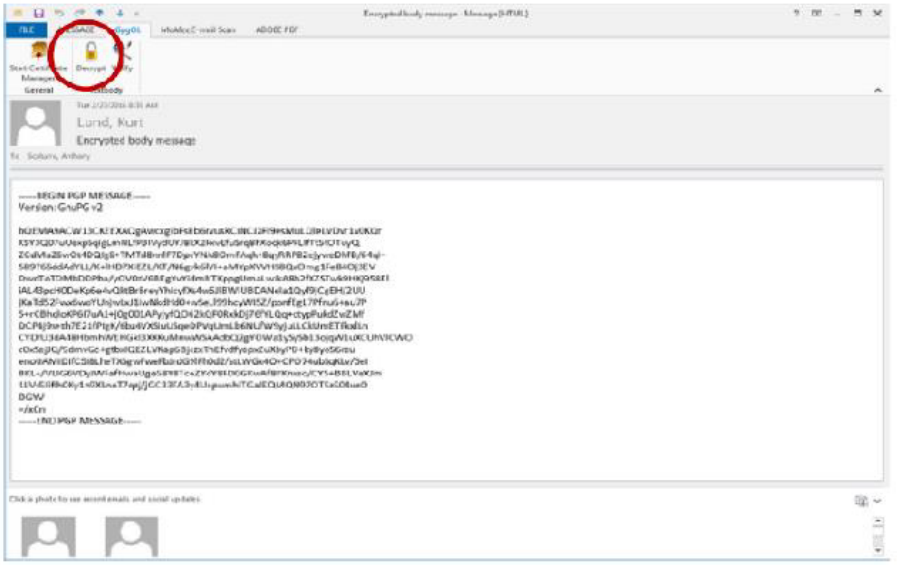
1. Open the e-mail message



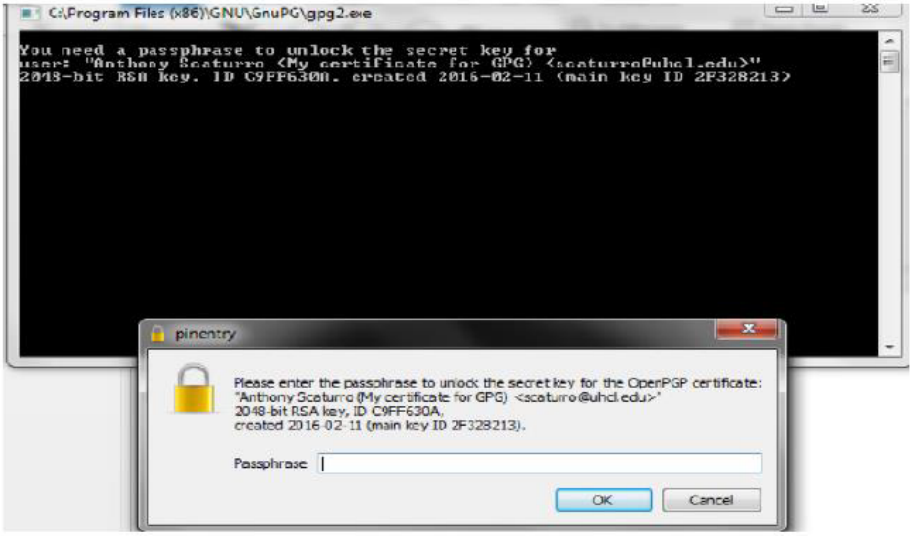
2. Select the GpgOL tab



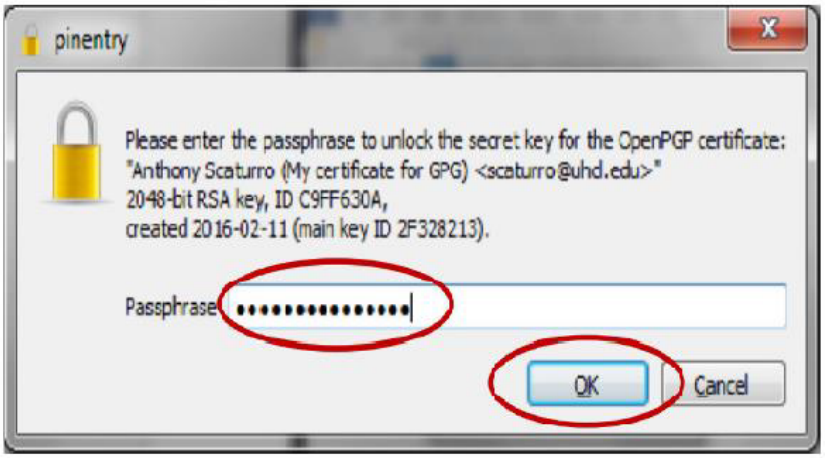
3. Click the "Decrypt" button



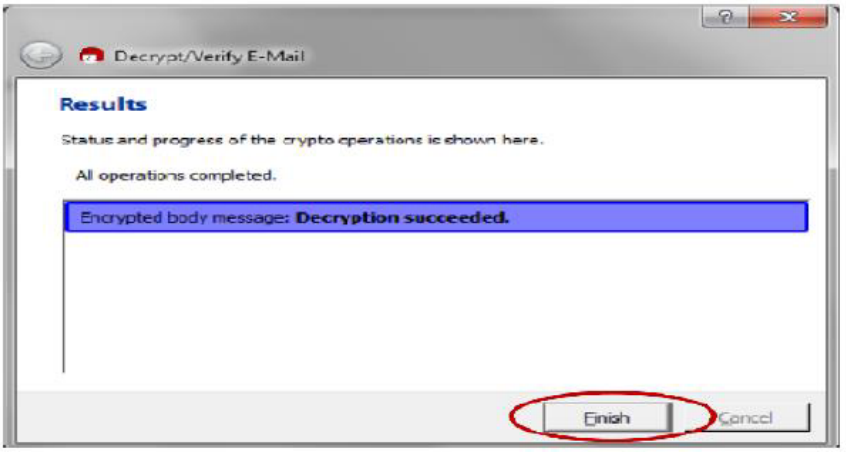
4. A command window will open along with a window that asks for the Passphrase to your private key that will be used to decrypt the incoming message.



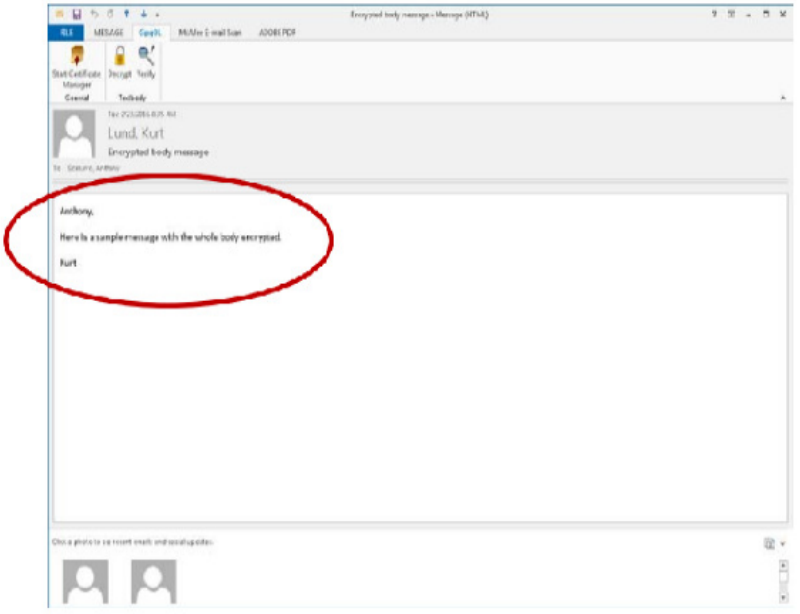
5. Enter your passphrase and click the "OK" button



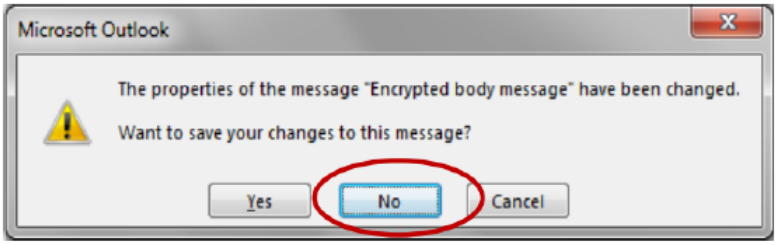
6. The results window will tell you if the decryption succeeded. Click the "Finish" button top close the window



7. Your unencrypted e-mail message body will be displayed.



8. When you close the e-mail you will be asked if you want to save the e-mail message in its unencrypted form. For maximum security, click the "No" button. This will keep the message encrypted within the e-mail system and will require you to enter your passphrase each time you reopen the e-mail message



RESULT:

Thus the secure data storage, secure data transmission and for creating digital signatures (GnuPG) was developed successfully.

Programming Problem: Write a program that can encrypt and decrypt using the general Caesar cipher, also known as an additive cipher.

Complex Problem: Write a program that can perform a letter frequency attack on an additive cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify "give me the top 10 possible plaintexts".

Complex Problems:

1. Write a program of knapsack cryptosystem to implement
a)Encryption
b)Decryption

2. Write a program of RSA varient cryptosystem to implement
a)Encryption
b)Decryption

3.Write a program of ECC cryptosystem to implement
a)Encryption
b)Decryption

DEBUGGING A SAMPLE C PROGRAM WITH ERRORS:

Create the following C program using commands explained earlier.

```
1   # include <stdio.h>
2   int main()
3  {
4           int i, num, j;
5           printf ("Enter the number: ");
6           scanf ("%d", &num );
7           for (i=1; i<num; i++)
8                       j=j*i;
9           printf("The factorial of %d is %d\n",num,j);
10      }

$ cc factorial.c
$ ./a.out
Enter the number: 3
The factorial of 3 is 12548672
```

Now debug it while reviewing the most useful commands in gdb.

Step 1: Compile the C program with debugging option -g
Compile your C program with -g option. This allows the compiler to collect the debugging information.

```
$ cc -g factorial.c
```

Note: The above command creates a.out file which will be used for debugging as shown below.

Step 2: Launch gdb
Launch the C debugger (gdb) as shown below.

```
$ gdb a.out
```

Step 3: Set up a break point inside C program

Syntax:

```
break line_number
```

Other formats:

- break [file_name]:line_number
- break [file_name]:func_name

Places break point in the C program, where you suspect errors. While executing the program, the debugger will stop at the break point, and gives you the prompt to debug. So before starting up the program, let us place the following break point in our program.

```
break 8
Breakpoint 1 at 0x804846f: file factorial.c, line 8.
```

Step 4: Execute the C program in gdb debugger

```
run [args]
```

You can start running the program using the run command in the gdb debugger. You can also give command line arguments to the program via run args. The example program we used here does not requires any command line arguments so let us give run, and start the program execution.

```
run
Starting program: /home/student/Debugging/c/a.out
```

Once you executed the C program, it would execute until the first break point, and give you the prompt for debugging.

```
Breakpoint 1, main () at factorial.c:8
8                                   j=j*i;
```

You can use various gdb commands to debug the C program as explained in the sections below.

Step 5: Printing the variable values inside gdb debugger

```
Syntax: print {variable}
Examples:
        print i
```

```
print j
print num
(gdb) p i
$1 = 1
(gdb) p j
$2 = 3042592
(gdb) p num
$3 = 3
(gdb)
```

As you see above, in the factorial.c, we have not initialized the variable j. So, it gets garbage value resulting in a big numbers as factorial values.

Fix this issue by initializing variable j with 1, compile the C program and execute it again.

Even after this fix there seems to be some problem in the factorial.c program, as it still gives wrong factorial value.

So, place the break point in 8th line, and continue as explained in the next section.

Step 6. Continue, stepping over and in – gdb commands

There are three kind of gdb operations you can choose when the program stops at a break point. They are continuing until the next break point, stepping in, or stepping over the next program lines.

- c or continue: Debugger will continue executing until the next break point.
- n or next: Debugger will execute the next line as single instruction.
- s or step: Same as next, but does not treats function as a single instruction, instead goes into the function and executes it line by line.

By continuing or stepping through you could have found that the issue is because we have not used the <= in the 'for loop' condition checking. So changing that from < to <= will solve the issue.

gdb command shortcuts

Use following shortcuts for most of the frequent gdb operations.

- l – list
- p – print
- c – continue
- s – step

- ENTER: pressing enter key would execute the previously executed command again.

Miscellaneous gdb commands

- **l command:** Use gdb command l or list to print the source code in the debug mode. Use l line-number to view a specific line number (or) l function to view a specific function.
- **bt: backtrack** – Print backtrace of all stack frames, or innermost COUNT frames.
- **help** – View help for a particular gdb topic — help TOPICNAME.
- **quit** – Exit from the gdb debugger.

C QUICK REFERENCE

PREPROCESSOR

```
// Comment to end of line
/* Multi-line
   Comment
 */
#include <stdio.h>              // Insert standard header file
#include "myfile.h"             // Insert file in current directory
#define X some text             // Replace X with some text
#define F(a,b) a+b              // Replace F(1,2) with 1+2
#define X \
        some text               // Line continuation
#undef X                        // Remove definition
#if defined(X)                  // Condional compilation (#ifdef X)
#else                           // Optional (#ifndef X or #if !defined(X))
#endif                          // Required after #if, #ifdef
```

LITERALS

```
255, 0377, 0xff                 // Integers (decimal, octal, hex)
2147463647L, 0x7fffffff1        // Long (32-bit) integers
123.0, 1.23e2                   // double (real) numbers
'a', '\141', '\x61'             // Character (literal, octal, hex)
'\n', '\\', '\'', '\"',         // Newline, backslash, single quote, double quote
"string\n"                      // Array of characters ending with newline and \0
"hello" "world"                 // Concatenated strings
```

DECLARATIONS

```
int x;                          // Declare x to be an integer (value undefined)
int x=255;                      // Declare and initialize x to 255
short s; long l;                // Usually 16 or 32 bit integer (int may be either)
char c= 'a';                    // Usually 8 bit character
unsigned char u=255; signed char m=-1; // char might be either
unsigned long x=0xffffffffL; // short, int, long are signed
float f; double d;              // Single or double precision real (never unsigned)
int a, b, c;                    // Multiple declarations
int a[10];                      // Array of 10 ints (a[0] through a[9])
int a[]={0,1,2};                // Initialized array (or a[3]={0,1,2}; )
int a[2][3]={{1,2,3},{4,5,6}};  // Array of array of ints
char s[]= "hello";              // String (6 elements including '\0')
int* p;                         // p is a pointer to (address of) int
char* s= "hello";               // s points to unnamed array containing "hello"
```

```
void* p=NULL;                  // Address of untyped memory (NULL is 0)
int& r=x;                      // r is a reference to (alias of) int x
enum weekend {SAT, SUN};// weekend is a type with values SAT and SUN
enum weekend day;              // day is a variable of type weekend
enum weekend {SAT=0,SUN=1}; // Explicit representation as int
enum {SAT,SUN} day;            // Anonymous enum
typedef String char*;          // String s; means char* s;
const int c=3;                 // Constants must be initialized, cannot assign
const int* p=a;                // Contents of p (elements of a) are constant
int* const p=a;                // p (but not contents) are constant
const int* const p=a;          // Both p and its contents are constant
const int& cr=x;               // cr cannot be assigned to change x
```

STORAGE CLASSES

```
int x;                         // Auto (memory exists only while in scope)
static int x;                  // Global lifetime even if local scope
extern int x;                  // Information only, declared elsewhere
```

STATEMENTS

```
x=y;                           // Every expression is a statement
int x;                         // Declarations are statements
;                              // Empty statement
{                              // A block is a single statement
   int x;                      // Scope of x is from declaration to end of
//block
   a;                          // In C, declarations must precede statements
}
if (x) a;                      // If x is true (not 0), evaluate a
else if (y) b;                 // If not x and y (optional, may be repeated)
else c;                        // If not x and not y (optional)
while (x) a;                   // Repeat 0 or more times while x is true
for (x; y; z) a;               // Equivalent to:  x; while(y) {a; z;}
do a; while (x);               // Equivalent to:  a; while(x) a;
switch (x) {                   // x must be int
        case X1: a;            // If x == X1 (must be a const), jump here
        case X2: b;            // Else if x == X2, jump here
        default: c;            // Else jump here (optional)
}
break;                         // Jump out of while, do, for loop, or switch
continue;                      // Jump to bottom of while, do, or for loop
return x;                      // Return x from function to caller
```

FUNCTIONS

```
int f(int x, int);          // f is a function taking 2 ints and returning int
void f();                   // f is a procedure taking no arguments
void f(int a=0);            // f() is equivalent to f(0)
f();                        // Default return type is int
inline f();                 // Optimize for speed
f( ) { statements; }        // Function definition (must be global)
```

Function parameters and return values may be of any type. A function must either be declared or defined before it is used. It may be declared first and defined later. Every program consists of a set of global variable declarations and a set of function definitions (possibly in separate files), one of which must be:

```
int main() { statements... }     or
int main(int argc, char* argv[]) { statements... }
```

argv is an array of argc strings from the command line. By convention, main returns status 0 if successful, 1 or higher for errors.

EXPRESSIONS

Operators are grouped by precedence, highest first.
Unary operators and assignment evaluate right to left.
All others are left to right.
Precedence does not affect order of evaluation which is undefined. There are no runtime checks for arrays out of bounds, invalid pointers etc.

```
t.x             // Member x of struct or class t
p → x           // Member x of struct or class pointed to by p
a[i]            // i'th element of array a
f(x, y)         // Call to function f with arguments x and y
x++             // Add 1 to x, evaluates to original x (postfix)
x--             // Subtract 1 from x, evaluates to original x
sizeof x        // Number of bytes used to represent object x
sizeof(T)       // Number of bytes to represent type T
++x             // Add 1 to x, evaluates to new value (prefix)
--x             // Subtract 1 from x, evaluates to new value
~x              // Bitwise complement of x
!x              // true if x is 0, else false (1 or 0 in C)
-x              // Unary minus
+x              // Unary plus (default)
&x              // Address of x
```

```
*p                  // Contents of address p (*&x equals x)
x * y               // Multiply
x / y               // Divide (integers round toward 0)
x % y               // Modulo (result has sign of x)
x + y               // Add, or &x[y]
x – y               // Subtract, or number of elements from *x to *y
x << y              // x shifted y bits to left (x * pow(2, y))
x >> y              // x shifted y bits to right (x / pow(2, y))
x < y               // Less than
x <= y              // Less than or equal to
x > y               // Greater than
x >= y              // Greater than or equal to
x == y              // Equals
x != y              // Not equals
x & y               // Bitwise and (3 & 6 is 2)
x ^ y               // Bitwise exclusive or (3 ^ 6 is 5)
x | y               // Bitwise or (3 | 6 is 7)
x && y              // x and then y (evaluates y only if x (not 0))
x || r              // x or else y (evaluates y only if x is false(0))
x = y               // Assign y to x, returns new value of x
x += y              // x = x + y, also -= *= /= <<= >>= &= |= ^=
x ? y : z           // y if x is true (nonzero), else z
x, y                // evaluates x and y, returns y (seldom used)
```

STDIO.H

```
prinf("Format specifers",value1,value2,..);
printf("user defined message");
scanf("Format specifiers",&value1,&value2,.....);
```

Format specifier:

Format specifier	Type of value

%d	Integer
%f	Float
%lf	Double
%c	Single character
%s	String
%u	Unsigned int
%ld	Long int
%lf	Long double

STRING Handling functions

```
strcat(s1,s2)              // Concatenates s2 to s1
strcmp(s1,s2)              // Comparison of two strings and returns 0 if equal
strstr(mainstr, substr)    // Search for the Substring in mainstr
strlen(s1)                 // string length of s1
gets(s1)                   // Read line ending in '\n'
```

Other functions

```
asin(x); acos(x); atan(x);      // Inverses
atan2(y, x);                    // atan(y/x)
sinh(x); cosh(x); tanh(x);      // Hyperbolic
exp(x); log(x); log10(x);       // e to the x, log base e, log base 10
pow(x, y); sqrt(x);             // x to the y, square root
ceil(x); floor(x);              // Round up or down (as a double)
fabs(x); fmod(x, y);            // Absolute value, x mod y
```

FILE HANDLING IN C

Before we read (or write) information from (to file) on a disk, we must open a file in particular mode. To open a file we use fopen() function. fopen() is a high level IO function. There are many high level IO function we use in file handling.

Syntax:

FILE *fp;
fp= fopen("filename","mode");

FILE is a type supported in 'C'.
fp = this is a file pointer of type FILE data structure.
fopen = High level IO function
filename = valid filename which you want to open
mode = r for read, w for write, a for append

I. **Binary Files:** To read, write contents of a binary file we use fread and fwrite respectively. Examples of binary files are .exe files .bmp or image files. More generally, any file which cannot be read using any normal text editor.

Functions to read contents: fread() and fwrite()

Syntax: size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

The functions fread/fwrite are used for reading/writing data from/to the file opened by fopen function. These functions accept three arguments. The first argument is a pointer to buffer used for reading/writing the data. The data read/written is in the form of 'nmemb' elements each 'size' bytes long.

Reading a particular byte in a file: fseek()

Syntax: int fseek(FILE *stream, long offset, int whence);

The fseek() function is used to set the file position indicator for the stream to a new position. This function accepts three arguments. The first argument is the FILE stream pointer returned by the fopen() function. The second argument 'offset' tells the amount of bytes to seek. The third argument 'whence' tells from where the seek of 'offset' number of bytes is to be done. The available values for whence are SEEK_SET, SEEK_CUR, or SEEK_END. These three values (in order) depict the start of the file, the current position and the end of the file. Upon success, this function returns 0, otherwise it returns -1.

Closing a file: fclose()

Syntax: int fclose(FILE *fp);

The fclose() function first flushes the stream opened by fopen() and then closes the underlying descriptor. Upon successful completion this function returns 0 else end of file (eof) is returned. In case of failure, if the stream is accessed further then the behavior remains undefined.

Writing into binary file:

Example: To write an employ's record to a file. Record has three fields name age and basic salary.

```
struct emp{
  char name[20];
  int age;
  float bs;
};
struct emp e;
void main(){
        fp = fopen("EMP.DAT","wb");
        if(fp == NULL){
        printf("Cannot open file");
        }
        printf("Enter name, age and basic salary");
        scanf("%s%d%f",e.name,&e.age,&e.bs);
        fwrite(&e,sizeof(e),1,fp);
        fclose(fp);
}
```

Reading from the binary file:

The following code segment illustrates how to read data from file in binary format.

```
struct emp{
  char name[20];
  int age;
  float bs;
};
struct emp e;
void main(){
        fp = fopen("EMP.DAT","rb");
        /* read all the records one by one */
        while(fread(&e,sizeof(e),1,fp)==1){
          printf("%s %d %f\n",e.name,e.age,e.bs);
        }
```

```
        fclose(fp);
}
```

II. **Text File:** File contents are written in ascii or unicode format. Generally, these files can be read using a text editor.

Writing to text file: fprintf()

Syntax: fprintf(FILE*,format,variable_list);

The fprintf statement should look very familiar to you. It can be almost used in the same way as printf. The only new thing is that it uses the file pointer as its first parameter.

```
#include<stdio.h>
int main()
{
        FILE *ptr_file;
        int x;

        ptr_file =fopen("output.txt", "w");

        if (ptr_file == NULL)
                return -1;

        for (x=1; x<=10; x++)
                fprintf(ptr_file,"%d\n", x);

        fclose(ptr_file);
}
```

Reading a text file: fscanf()

Syntax: int fscanf(FILE *stream, const char *format, variable_list)
The C library function int fscanf(FILE *stream, const char *format, ...) reads formatted input from a stream.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
```

```
        char str1[10], str2[10], str3[10];
        int year;
        FILE * fp;
        fp = fopen ("file.txt", "w+");
        fputs("We are in 2012", fp);
        rewind(fp);
        fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
        printf("Read String1 |%s|\n", str1 );
        printf("Read String2 |%s|\n", str2 );
        printf("Read String3 |%s|\n", str3 );
        printf("Read Integer |%d|\n", year );
        fclose(fp);
        return(0);
}
```

Similar to fprintf, fscanf, we can use fputc, fgetc to handle character data.