



**TRIBHUVAN UNIVERSITY  
INSTITUTE OF ENGINEERING  
PULCHOWK CAMPUS**

**A Project of C on  
“Cryptics”  
*‘Encryption via RSA’***

**Submitted to:**

**Department of Electronics and Computer Engineering, Pulchowk Campus  
IOE, TU**

**Submitted by:**

**Aavash Chhetri (077BCT004)**

**Amrit Sharma(077BCT009)**

**David Gurung (077BCT025)**

**Kushal Paudel(077BCT039)**

**Mandip Thapa(077BCT044)**



*“Encryption via RSA”*

# Acknowledgement

We would like to deeply thank the various people who provided us with the useful and helpful assistance for the preparation of this project. First, we would like to thank **Mr. Ganesh Gautam** sir, our teacher of C Programming Language who gave us the concepts of C programming language. Also, we would like to thank all our dear friends and seniors for their advice, encouragement, support and help. Also, we would like to thank authors of various books in C language which provided us great help for the preparation of this project.

- Mr.Yashwant Kanetkar-Let Us C.
- Mr. Venugopal-Programming With C.
- Mr. Byron S. Gotterfried-Programming With C.
- Mr.Balagurusamy-Programming In ANSI C.
- Krishna Kandel- Learning C by Examples

Furthermore, we would like to appreciate all the resources that are made available online free of cost as well as documentations of C Language made available by Microsoft at <https://docs.microsoft.com/en-us/cpp/c-language/> .

Lastly, we would like to thank everyone who has directly or indirectly helped and impacted in making this project possible.

# Table of Content

Chapter 1: INTRODUCTIONS.....	5
1.1 The C-Programming Language .....	7
1.2 Program Structure.....	8
1.3 Socket Coding .....	9
Chapter 2: Review of related lectures .....	13
Chapter 3: Algorithm and Flowchart.....	23
3.1 RSA.h .....	26
o Generation of keys .....	26
o Check Coprime .....	28
o Encryption Algorithm .....	30
o Decryption Algorithm .....	32
3.2 Main.c.....	34
3.3 Flowchart for TCP/IP implementation .....	37
Chapter 4: IMPLEMENTATION AND CODING.....	39
4.1. Implementation.....	39
4.2. Coding for the project.....	40
Chapter 5: RESULT AND DISCUSSION.....	42
Chapter 6: CONCLUSION.....	47
REFERENCES.....	47

# Chapter-1

## Introduction

Cyber Security has been a major concern for all the users on and off the internet as it affects almost all the sectors of today's world. In spite of the technological advancements, we are humans after all and make mistakes. There are possibilities of making such mistakes in a very critical stage of data transmission which can cause a big ruin in the world. This problem can be encountered by encrypting confidential data while storing and during transmission. Thereafter, even if the system is compromised the data will remain secure as they are encrypted via a trusted algorithm. Therefore, we decided to develop a prototype implementation of such secure encryption Algorithm (RSA):

### ***‘Cryptics’ – Encryption via RSA.***

**Cryptics** is a simple implementation of encryption Algorithm known as RSA (Rivest–Shamir–Adleman). RSA (Rivest-Shamir-Adleman) is a publickey cryptosystem that is widely used for secure data transmission. The acronym RSA comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman, who publicly described the algorithm in 1977.

This project is written in the C programming language and aims to provide a basic knowledge of high-level programming languages such as C. It incorporates the use of pure English syntaxes of function that are predefined in C library. The Program in “**Cryptics**” helps to safely transfer valuable information by encrypting the data using RSA.

Using this program, we have made a simple TCP/IP data transferring system which will demonstrate the functionality of the code. For the TCP/IP implementation we have used header files like *sys/socket.h*, *netdb.h* and *netinet.h* which can be only used in Linux Kernel.

Furthermore, A simple website is made online which is available at <https://cryptics.netlify.app/> . This website provides the demonstration of the project in real life scenario. For the source code of the project, you can see at: <https://github.com/A-atmos/RSAINC> .

# The C- Programming Language

C is a high-level general proposed, procedural computer programming language supporting structured programming, recursion, with a static type system. By design, it provides efficient construct that maps efficiently to machine code due to which it is also used in construction of various Operating systems and various application software for computer architecture which varies from super computer to embedded systems.

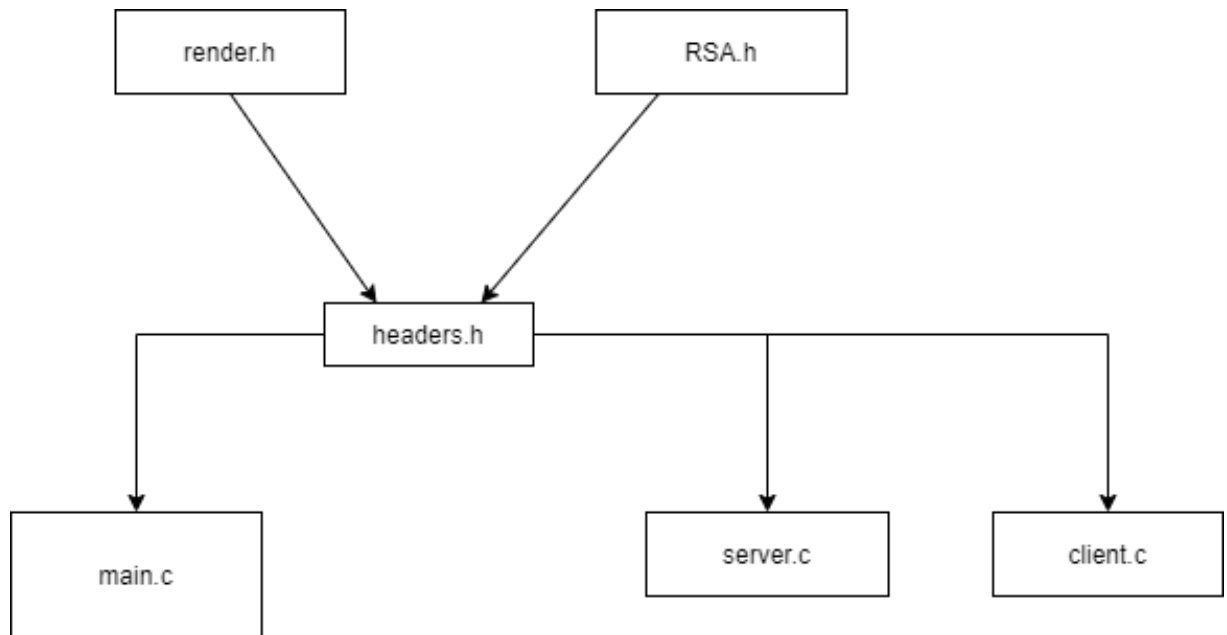
Actually, C has a close ties to the development of Unix systems, originally implemented in assembly code on a PDP-7. It was developed by Bell Laboratories in the early 1970s for conveniently writing the UNIX system codes which used to written in assembly code back then. The 4<sup>th</sup> iteration of the UNIX systems were extensively written in C which is probably the first kernel written in C.

After then, C has got various iterations. Some of the features of C includes:

1. Procedural Language.
2. Fast and Efficient.
3. Modularity.
4. Statically Type.
5. General-Propose.
6. Small size.
7. Pointer implementation which helps in memory management.

# Structure of the Code

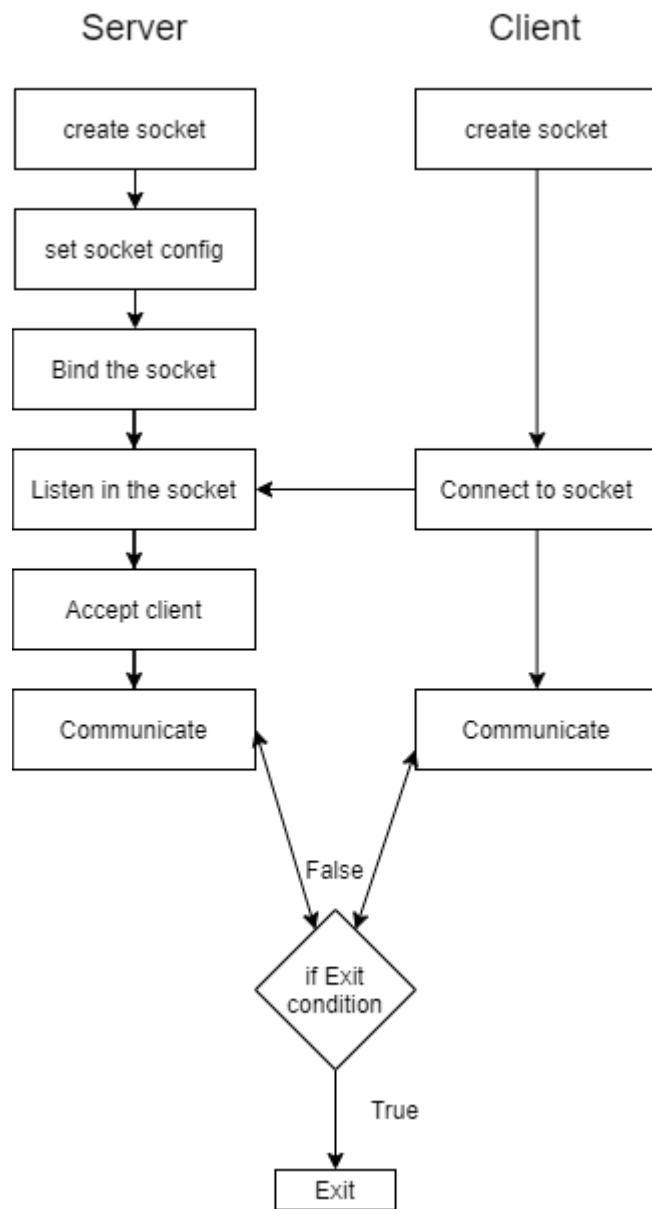
The code is modular and the main algorithm is written in the header file “RSA.h”. A “render.h” header file is made which gives a better look to the terminal. These header files are then included in “headers.h” which is then used by the main.c file. The main.c file is used to generate keys, encrypt and decrypt data and store it in text file. The same “headers.h” is used by “server.c” and “client.c” which is the socket implementation of the code. The following diagram shows the complete format of the code:





# Socket Coding

For the implementation of TCP/IP server and client, we have used the sockets. Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



In C, Socket programming differs according to the kernel used, We have implemented the system as per the Linux kernel as it is widely used in server construction.

- *Sys/socket.h*
- *Netdb.h*
- *Netinet/in.h*

The above header files are used for the socket creation and communication in our project.

The socket is created as:

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
    printf("socket creation failed...\n");
    exit(0);
}
else
    printf("Socket successfully created..\n");
```

Here, AF\_INET and SOCK\_STREAM is predefined in the above-mentioned header files. The SOCK\_STREAM gives the protocol for the socket creation, which is TCP and AF\_INET refers to the address family, here IPV4, for the socket connection.

After the socket is created, it is bind to the required IPV4 address and port number, IPV4 address defined as INADDR\_ANY, localhost (127.0.0.1) and port is defined in in the code as 8080. Therefore, the server configures the socket to localhost and at the port 8080. Code is as below:

```
// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);
```

After successfully configuring the socket, the server binds to the socket. The code for binding to the socket is as below:

```
// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
    printf("socket bind failed...\n");
    exit(0);
}
else
    printf("Socket successfully binded..\n");
```

After successfully binding to the socket at localhost:8080, it listens to any client trying to connect to it. The maximum number of clients that can connect to the server is 5. In case of any exceptions, the server throws an error as “Listening Failed” and exits the program.

```
// Now server is ready to listen and verification
if ((listen(sockfd, 5)) != 0) {
    printf("Listen failed...\n");
    exit(0);
}
else
    printf("Server listening..\n");
```

After a client tries to connect to the server, it accepts the data from the client and verifies the data packet.

```
// Accept the data packet from client and verification
connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
    printf("server accept failed...\n");
    exit(0);
}
else
    printf("server accept the client...\n");
```

After successfully connecting to the client, then message transmission takes place between them, until an exit condition is reached.

Meanwhile, the server side creates the socket as mentioned above and configures to the desired IPV4 address, here localhost:8080. After creating the socket, the server listens to the client and the client sends a data packet which is then verified by the server. Thus they connect.

For transmission of data:

```
read(sockfd, &client_n, sizeof(client_n));  
write(sockfd, &n, sizeof(n));
```

read() and write() functions are used. Here, the sockfd is the socket details, &client\_n is pointing to the memory which receives the data from the other side, and sizeof(client\_n) is the size of the data that is to be received. Similarly, in the write function &n is the memory address which is to be sent to the other side and sizeof(n) is the size of data that is to be sent.

After the communication, the socket is closed as:

```
close(sockfd);
```

This closes the socket, and prevents further connection to the server from the client side.

# Chapter 2

## REVIEW OF RELATED LITERATURES

This project is based on high level language i.e., c programming. In this project we used important parts of c programming which are input output statements, control statement, looping, function, array, structure and file handling and tried out many other parts of c-programming.

### 3.1.1 INPUT/ OUTPUT:

In every C program, it is required to input the data into the variables and display the values of the variables. To accomplish such tasks input and output functions are available.

It is categorized into two types. They are:

- i. Formatted Input/ output functions
- ii. Unformatted Input/ output functions

**A. Formatted Input/ Output:** It means that the input has to be given in a specific input/ output can be obtained in a specific format. In formatted input/ output it is required to specify the format or control string with which input/ output has to be carried out.

There are two formatted I/O functions: `printf ( )` and `scanf ( )`

**a. Printf( ):** The `printf( )` function is used to display the value(s) of the variables onto the screen under the control of the control string.

Syntax:

```
printf (" control string" , arg1, arg2,.....arg n);
```

**b. Scanf ( ):** The `scanf( )` function is used to input data from the user and to store the entire data in the memory address of the variable used.

Syntax:

```
scanf(" control string", &list_of_arguments);
```

**B. Unformatted Input/ Output:** Unformatted I/O statements do not specify how the inputs and outputs are carried out. They do not contain any information about field width, format specifiers and space sequence. Different inputs characters

numbers etc are distinguished rather they are stored in the form of characters or string only.

Some of unformatted I/O statements are: gets(), puts(), getch(),putch(),getche(),putche(),getchar(),putchar() etc.

### **3.1.2. CONTROL STATEMENTS:**

Control statements are used to control the execution of statements facilitating the decision making process in programming. Control statements serve to control the flow of the program by means of branches and loops.

There are 2 types of control statements:

- i. Branching ( if, if-else, if-else-if, Nested if...else etc)
- ii. Looping ( for loop,while loop ,do while loop etc)

#### **A. Selective structure:**

Selective structures are used when we have a number of situations where we need to change the order of execution of statements based on certain condition. The selective statements make a decision to take the right path before changing the order of execution. C provides the following statements for selective structure:

- i. if statements
- ii. switch statements

#### **a. if statements:**

The if statement is a powerful decision making statement and it is used to control the flow of execution of statements. It is a two way statement and is used in conjunction with an expression.

If statement allows the computer to evaluate the expression first and then on depending whether the value of the expression is true or false it transfer the control to the particular statement. At this point of the program has two paths to follow: one for true condition and other for false condition.

The types of if statements are explained below:

**i. Simple if statement:** The simple if statement is used to conditionally excite a block of code based on whether a test condition is true or false. If the condition is true the block of code is executed, otherwise it is skipped.

The syntax of if statement is given below:

```
if (test expression)
{
    statement-block;
```

```
}  
statement-x;
```

**ii. if else statement:** The if else statement extends the idea of the if statement by specifying another section of code that should be executed only if the condition is false i.e. conditional branching. True- block statements are to be executed only if the test expression is true and false block statements to be executed only if the condition is false.

The syntax of if else statement is given below:

```
if (test expression)  
{  
    true block statement;  
}  
else  
{  
    false block statement;  
}
```

**iii. if...else if....else statements:**

This statement is used to check series of conditions for writing multi way decision. It is the type of nesting in which there is an if-else statement in every else part except the last else part.

The syntax of if...else if statement is given below:

```
if (expression 1)  
statement A;  
else if( expression 2)  
statement B;  
else if( expression 3)  
statement C;  
.....  
.....  
Else  
statement D;
```

**b. The switch statement:**

C has built in multi way decision statement known as switch. It successively test the value of an expression against a list of case values (integer or character consonants).when a match is found the statement associated with that case is executed.

The syntax of switch expression is given below:

```
switch (expression)  
{ case constant 1:
```

```

statement
.....
case constant 2:
Statement
.....
case constant N:
statement:
.....
default:
statement:
.....
}

```

Firstly, switch control expression is evaluated, if the value of expression matches with an case constant expression then control is transferred to the label. If none of case constant matches with the value of expression, then the control is transferred to the default label; which is optional.

**B. Looping:** The process of repeating a block of statements until some condition is satisfied is known as looping. It is known as iteration or repetitive structure.

**a. for loop:**

The for loop is used to execute a block of code for a fixed number of repetitions. Initialization is generally an assignment statement used to set loop control variable. Test expression is a relational expression that determines when loop exits. Update expression defines how the loop variable changes each time when the loop is repeated.

The syntax of for loop is given below:

```

for(initialization expression; test expression; update expression)
{
    body of loop;
}

```

**b. do while loop:**

do while loop is also used to repeat a certain block of code for a fixed number of times. Unlike the for loop the initialization is done outside the do loop where as the update expression still lies inside the body of the loop. In do while loop the test



condition is only checked after the statement in the body has been executed atleast ones.

The syntax of do while loop is given below:

```
do  
{
```

```
body of loop;  
update expression;  
}  
while(test expression)
```

### **c. while loop:**

While loop is similar to do while loop where the only difference lies in the chronology of execution of body of loop and test expression. In while loop the body of the loop will no execute until the test expression checked.

The syntax of while loop is given below:

```
while(test expression)  
{  
body of loop;  
update expression:  
}
```

### **d. Break statement:**

Break statement is used inside loops and switch statements. Sometimes, it becomes necessary to come out of loop even before the loop condition becomes false. Thus break statements is used to terminate the loop.

It is written as

```
break;
```

### **e. GOTO statements:**

This is an unconditional control statements that transfer the flow of control to another to another prt of program.

Syntax of goto is given below:

```
goto label;  
.....  
.....  
label:  
Statements  
.....  
.....
```

label is any valid C identifier followed by a colon.

**3.1.3. Function:** A function is a self contained program segment or module that carries out a very specific and well defined task. A program may contain one or more functions. It helps to divide a large program into several modules that it is easy to understand and maintain the program. A well written function can be reused and using of functions help in testing and debugging more easily.

The function is classified into two types. They are:

- i. library Function
- ii. User defined function

**A. Library Functions:**

Library functions are those functions, which are already defined in C library. Example printf( ), scanf ( ), strcat( ) etc. We just need to include appropriate header files to use these functions as they are already declared in and defined in C library.

**User defined Functions:** Those functions which are defined by the user at the time of writing program are called user defined functions.

**Return statement:** The return statement is used for immediate exit from the called function to the calling function and returns a value to a calling function. The ways in which it can be used are:

return;  
return(expression);

We classify functions into four types depending upon the return value and arguments.

- i. Functions with no arguments and no return value
- ii. Functions with no arguments and return value
- iii. Functions with arguments and no return value
- iv. Functions with arguments and return value

Here for our project we only used Function with no arguments and no return value. Functions with no arguments and no return value: These functions carry no arguments and do not return any value to main function. So there is no communication between calling and called function.

These functions are written as:

```
void function ( void);  
int main( )  
{  
.....  
function ( );  
.....  
}  
void function (void)  
{  
Statements or Block of statements  
.....  
}
```

### **3.1.4 Arrays:**

An array in C is a collection of similar types of data, stored at contiguous memory locations. Elements can be accessed randomly using indices of an array and they are used to store similar type of elements as in the data type must be the same for all elements. They can be used to store collection of primitive data types such as int, float, double, char, etc of any particular type.

#### **A. Array declaration**

Syntax for array declaration:

```
dataType arrayName[arraySize];  
float mark[5];
```

Here, array named mark have been declared of floating-point type. And its size is 5.

#### **B. Input and Output Array Elements**

Here's how input from the user can be taken and stored in an array element

Take input and store it in the 3rd element

```
scanf("%d", &mark[2]);
```

Take input and store it in the ith element

```
scanf("%d", &mark[i-1]);
```

Take input and store it in the nth element

```
scanf("%d", &mark[n-1]);
```

Here's how individual element of an array can be printed

Print the first element of the array

```
printf("%d", mark[0]);
```

Print the third element of the array

```
printf("%d", mark[2]);
```

Print ith element of the array  
`printf("%d", mark[i-1]);`  
Print nth element of the array  
`printf("%d", mark[n-1]);`

### **C. Strings**

Strings are the array of characters terminating at null character ('\0').

The syntax for declaring strings is given below:

`Char string_variable_name[size];`

### **D. String handling functions**

Some functions are predefined in the C library that helps in the manipulations of strings. The declaration of these functions are in the header file `string.h`. Here in the project a string handling function called `strcmp()` is used.

#### **a. strcmp()**

It is used to compare two strings using their ASCII values. The comparison is done on the basis of ASCII value of individual character of the strings and not the length of the strings.

The syntax for `strcmp()` function is given below:

`strcmp(string1, string2);`

The function returns 0 if `string1` and `string2` are the same

-1 if `string1 < string2`

+1 if `string1 > string2`

### **3.1.5 Structures:**

Structure is a collection of dissimilar types of data. Use of structure in programs makes the programs more readable and efficient.

The syntax for defining a structure is as follows:-

```
struct structure_name
{
Data_type variable_name 1;
Data_type variable_name 2;
Data_type variable_name 3;
.....
Data_type variable_name n;
};
```

#### **A. Accessing Structure Elements:**

Each element of structure can be accessed by using dot (.) operator. If structure has been defined called `student` and has been declared as follows:

```
struct student s;
```

Each element of s can be accessed by using dot operator as follows:  
s.name, s.roll, s.address etc. Each of these can be treated as a separate variable. We can input value into them using functions like scanf() and display their values using functions like printf() as follows:

```
scanf("%s", s.name);  
printf("%s", s.name);
```

### **B .Array of structures:**

An array of structure can be created like similar to array of basic data types.

For example:

```
struct student s[40];
```

The above statement declares/creates an array of structure called student with size 40, This means 40 structures ranging from s[0], s[1],.... up to s[39] has been created at once. The member can be accessed as s[0].name, s[0].roll, s[0].address, s[1].name, s[1].roll, s[1].address and so on.

### **3.1.6 Files:**

A file is a collection of related information. File programming is different from Console I/O. In console I/O data can be read and displayed onto the screen but are not stored for future use whereas in file I/O, data can be saved onto files so that the information can be retrieved for the future usage.

File Operations:

In every file I/O, one has to do three basic operations. They are as follows:

- i. Opening a file.
- ii. Performing read, write, and append operation etc on an opened file.
- iii. Closing a file.

#### **A.FILE Pointer:**

FILE pointer is a pointer to structure of FILE type.

A file pointer can be declared as follows:

```
FILE *file_pointer;
```

#### **B. Opening a file:**

The file pointer can be used to open and close the file as follows:

```
file_pointer = fopen("path_string_for_file", "mode_string");
```

#### **C. Different modes of File Operations:-**

Files can be opened in different modes depending upon the file operation(s) we wish to perform. Following are the different mode strings used in the project as follows:

- a. write mode ("w")
- b. read mode ("r")
- c. append mode ("a")

#### **D. Closing a file:**

A file can be closed by using `fclose()` function using the syntax as follows:

```
fclose(file_pointer);  
where, file_pointer is a file pointer to opened file.  
}
```

#### **E. Record IO operations in Files:**

In C programming language, record IO operations can be performed on a file using functions like `fwrite()` and `fread()` using following syntax:

1. `fwrite(&structure_or_array_variable, sizeof_structure_or_array_variable, number_of_structure_or_array, file_pointer);`

The function `fwrite()` writes the contents of the structure or array into the file pointed by `file_pointer`.

2. `fread(&structure_or_array_variable, sizeof_structure_or_array_variable, number_of_structure_or_array, file_pointer);`

The function `fread()` reads the contents of the file pointed by the `file_pointer` and saves the read values into the structure variable.

# Chapter-3

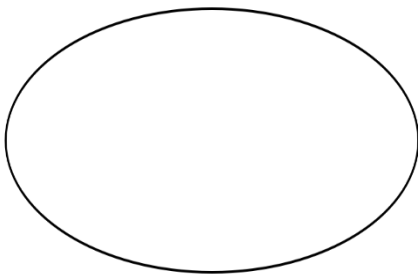
## Flowcharts

A set of ordered steps required or procedure necessary to solve is called algorithm. It is an effective procedure for solving a problem in a finite number of steps. Algorithm maintains sequence of computer instructions required to solve problem in such a way that if the instructions are executed in the specified sequence. The desired result is obtained. It is done before programming a computer in simple understandable language. Similarly, graphical representation of an algorithm is called flowchart. Here in flowchart different standard symbols are used for different work sequence. Flowchart plays a vital role in programming of a project and plays quite important role and are helpful for in understanding the logic of complicated and long problems. Once the flowchart is drawn it becomes easy to write and execute the program in any programming language. Different standard symbols used in flowcharts are below:-

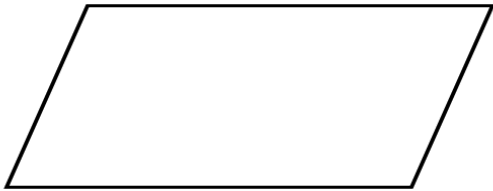
### 1.Flow lines:



### 2.Terminal Box(start /Stop)



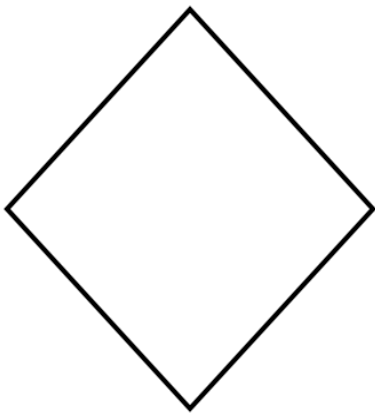
3.Input/Output



4. Processing/Instructions

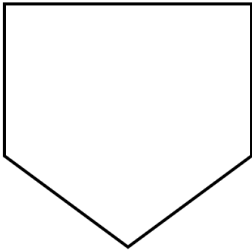


5.Decision making





6. Off page connector



7. Function call



The main algorithm is present in the RSA.h header file which has the algorithm and flowchart as below.

### **3.1.1. Generation of Keys:**

***Algorithm:***

Step 1: Start

Step 2: Input two numbers

Step 3: Let the numbers be p and q

Step 4: If the numbers are not prime go to Step 2

Step 5:  $n=p*q$  and  $\text{phifunction}=(p-1)*(q-1)$

Step 6: Initialize  $j=0$

Step 7: Check whether j and phifunction are co prime or not

Step 7.1: If no,  $j=j+1$  and go to Step 7

Step 7.2: If yes,  $e=j$

Step 8: Initialize  $i=1$

Step 9:  $d=(\text{phifunction}*i+1)/e$

Step 10: If d is an integer, go to Step 12

Step 11: If d is not an integer,  $i=i+1$  and go to Step 9

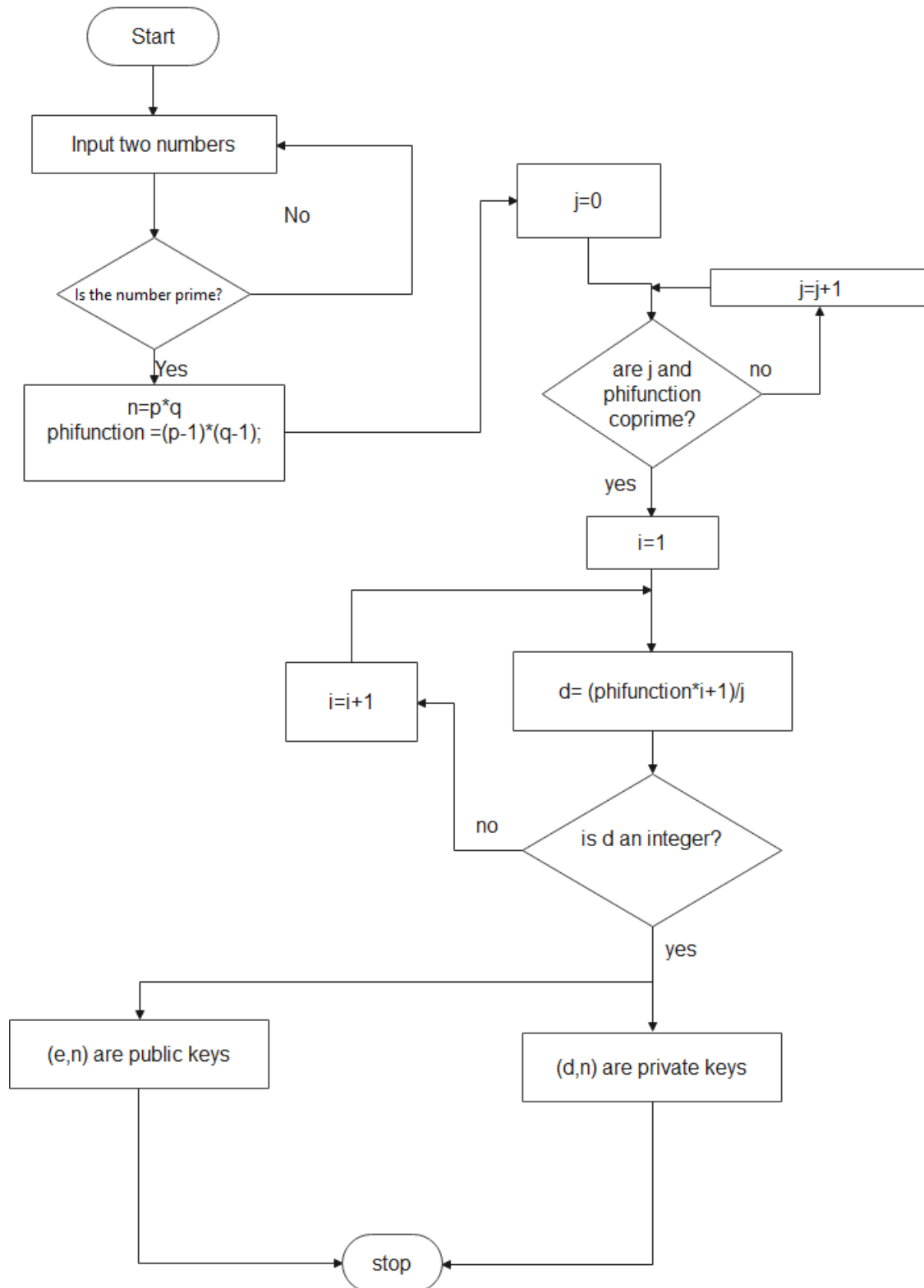
Step 12: Return e as public key

Step 13: Return d as private key

Step 14: Stop

### Flowchart:

#### Generating Keys



### 3.1.2. Check Co-prime:

***Algorithm:***

Step 1: Start

Step 2: Initialize  $i=2$

Step 3: If  $i < j$  (from Generate keys), go to Step 6

Step 4: If  $i=j$ , return  $j$  as coprime

Step 5: Stop

Step 6: If  $j \% i = 0$  and  $\text{phifunction} \% i = 0$ , go to Step 8

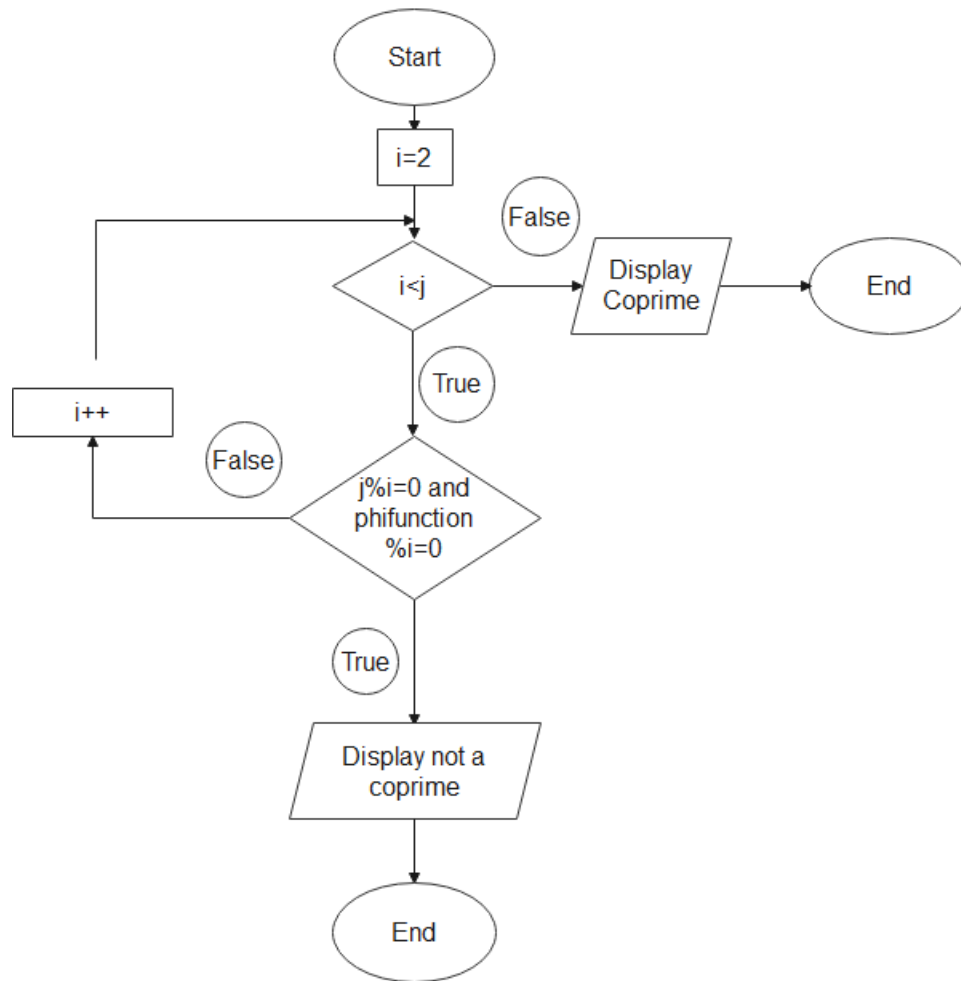
Step 7:  $i=i+1$  and go to Step 3

Step 8: Return  $j$  as not a coprime

Step 9: Stop

### Flowchart:

Check Coprime



### 3.1.3. Encryption Algorithm:

***Algorithm:***

Step 1: Start

Step 2: If the user wants to read from a file, call read (0) and go to Step 4

Step 3: Ask user to input a text to encrypt

Step 4: Initialize  $i=0$

Step 5: If input[i] is not equal to EOF, go to Step 7

Step 6: If input[i] is equal to EOF, go to Step

Step 7: Initialize  $j=0$

Step 8: If  $j < \text{public key}$ , go to Step 10

Step 9: If  $j = \text{public key}$ , go to Step 13

Step 10: Initialize  $\text{temp}=1$

Step 11:  $\text{temp} = \text{temp} * (\text{ascii value of}) \text{input}[i]$

Step 12:  $\text{temp} = \text{temp} \% n$

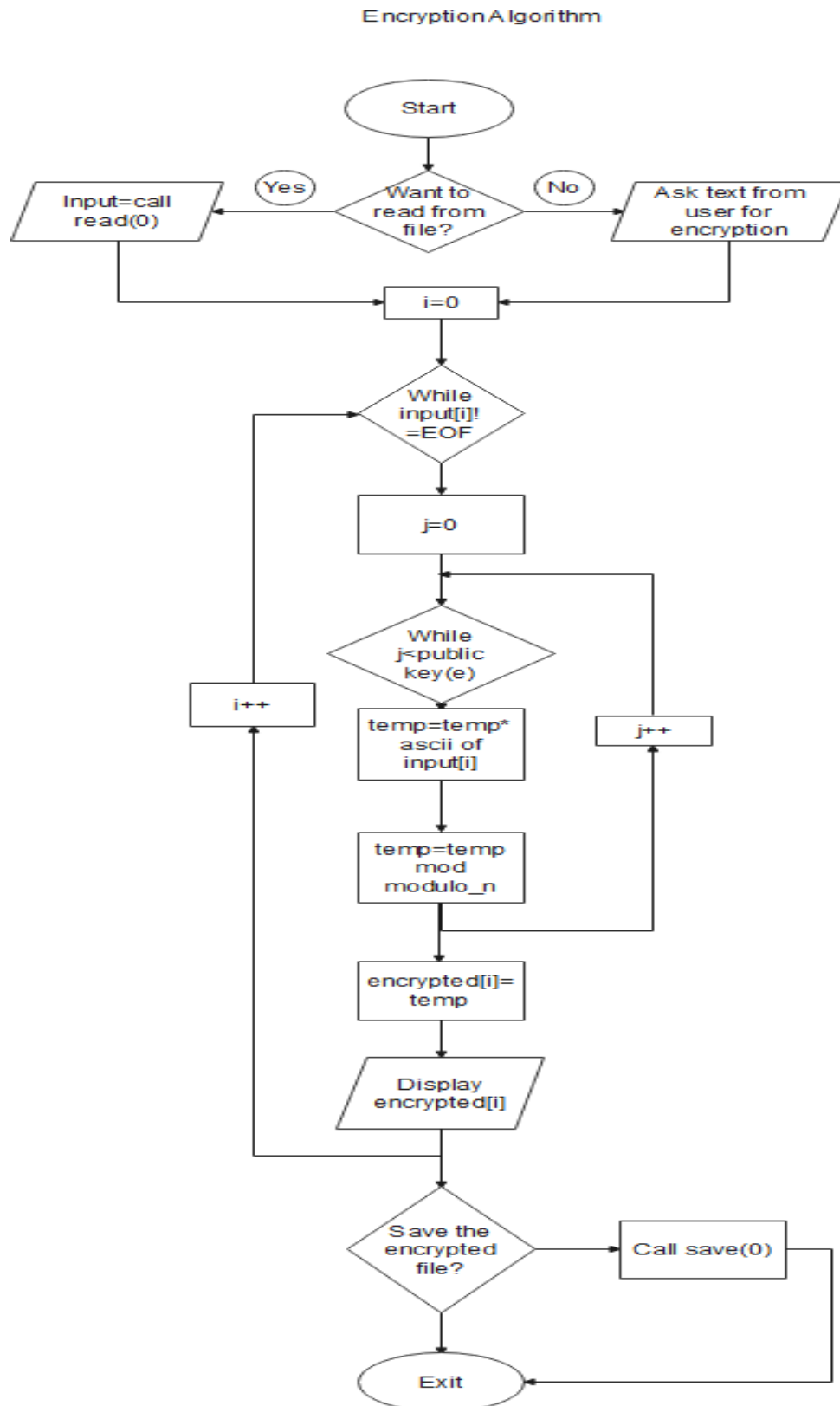
Step 13:  $\text{encrypted}[i] = \text{temp}$

Step 14:  $i = i + 1$  and go to Step 5

Step 15: If the user wants to save the encrypted text, call write(1)

Step 16: Stop

**Flowchart:**



### 3.1.4. Decryption Algorithm:

***Algorithm:***

Step 1: Start

Step 2: If user wants to read from a file, Call read (1) and go to Step 4

Step 3: Ask user to enter encrypted message

Step 4: Initialize  $i=0$

Step 5: If  $\text{input}[i]$  is not equal to EOF, go to Step 7

Step 6: If  $\text{input}[i]$  is equal to EOF, go to Step 15

Step 7: Initialize  $j=0$

Step 8: If  $j < d$ , go to Step 10

Step 9: If  $j = d$ , go to Step 12

Step 10: Initialize  $\text{temp}=1$

Step 11: Apply  $\text{temp}=\text{temp}*\text{input}[i]$  and  $\text{temp}=\text{temp}/n$

Step 12:  $\text{decrypted}[i]=\text{character corresponding to ascii value of temp}$

Step 13: Display  $\text{decrypted}[i]$

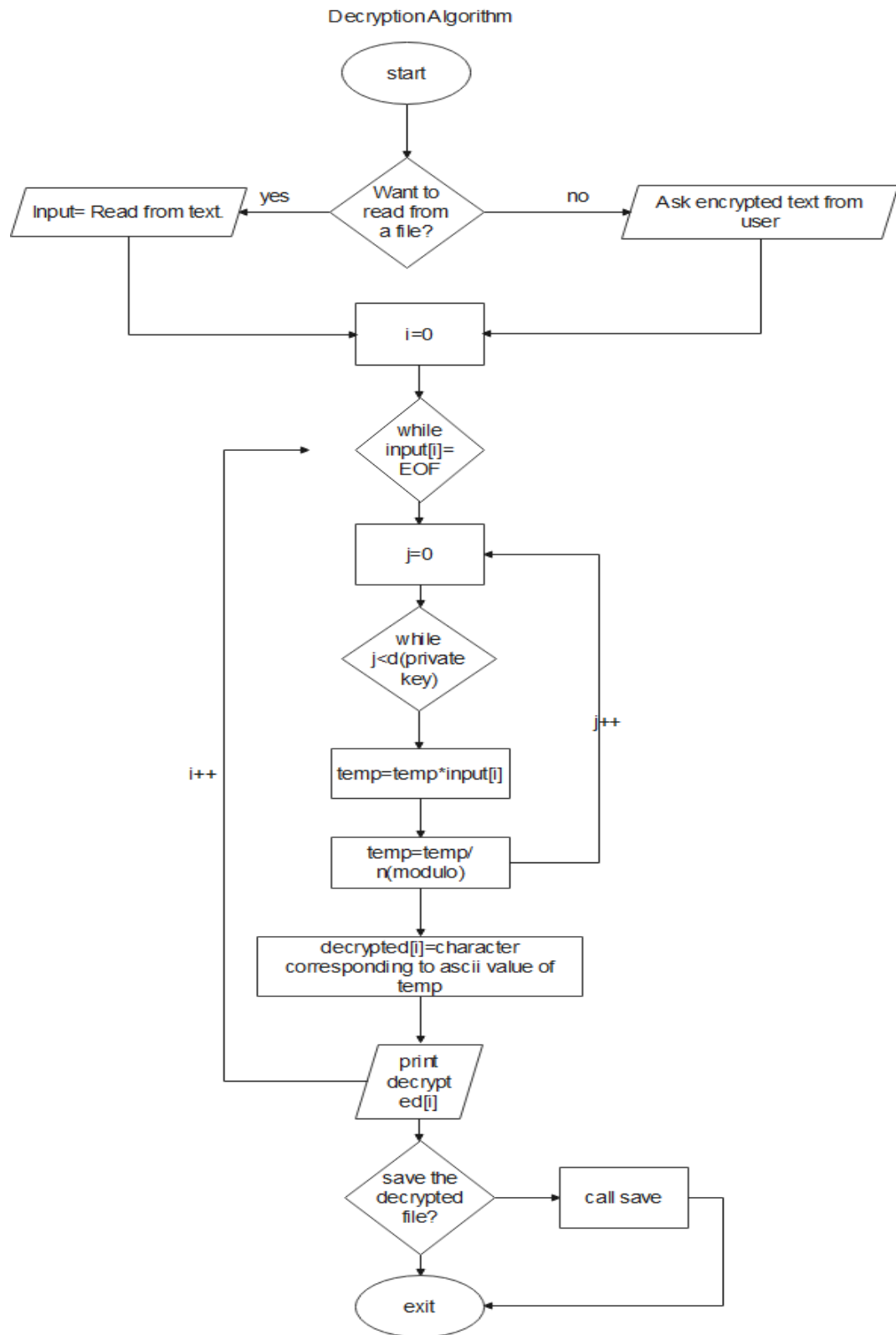
Step 14:  $i=i+1$  and go to Step 5

Step 15: If user wants to save the decrypted file, call  $\text{save}(1)$

Step 16: Stop



### Flowchart:



The above functions are present in 'RSA.h' header file. These are the main functions responsible for the functioning of the below codes. The flowchart of the main function 'main.c' is displayed below:

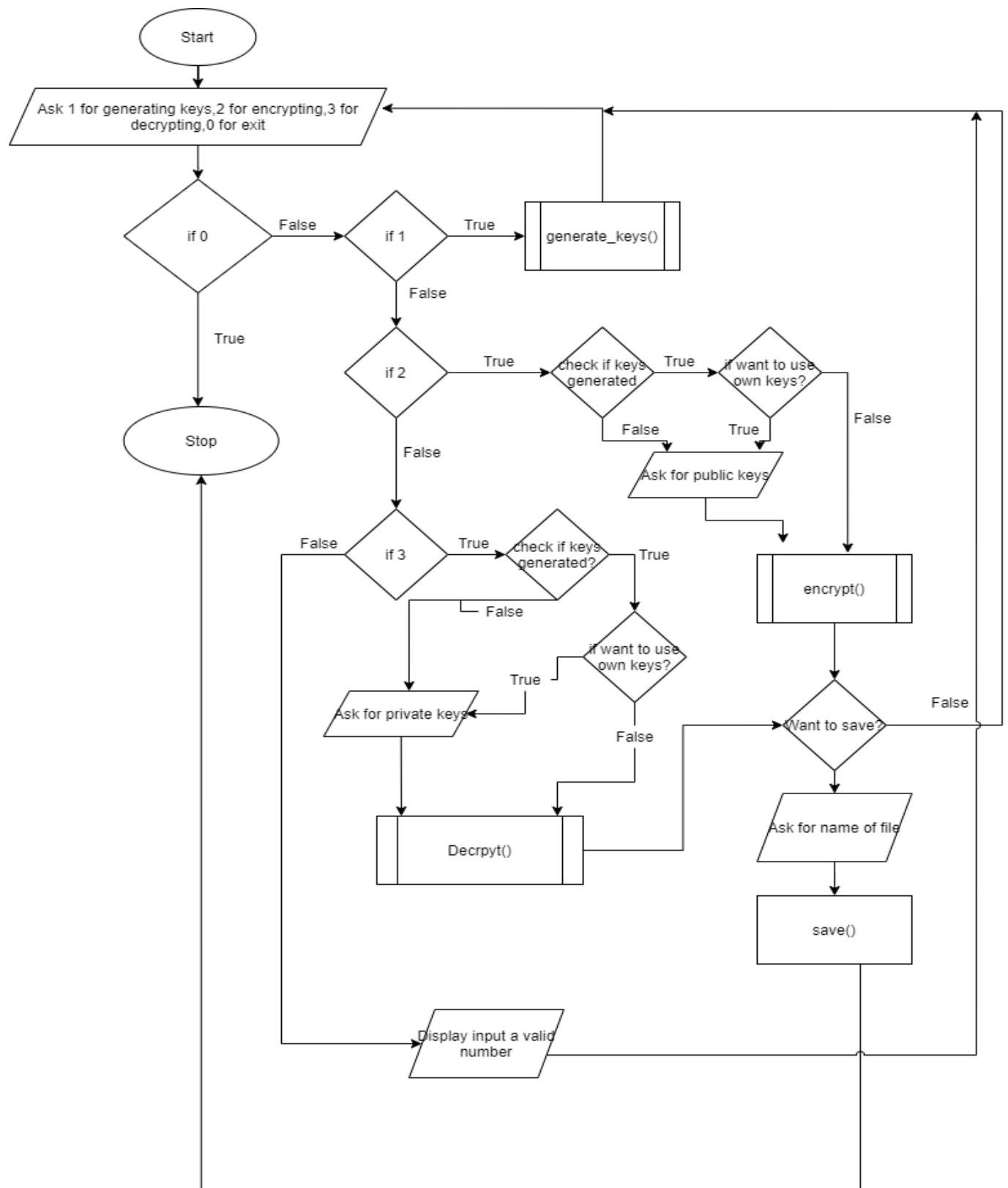
### **3.2. Main.c**

#### **Algorithm:**

- Step 1: Start
- Step 2: Ask user to input a number 1, 2, 3 or 0
- Step 3: If number is 1, go to Step 8
- Step 4: If number is 2, go to Step 10
- Step 5: If number is 3, go to Step 18
- Step 6: If number is 0, go to Step 26
- Step 7: If number is not 1, 2, 3 or 0, go to Step 20
- Step 8: Call Generate Keys
- Step 9: Go to Step 2
- Step 10: If public key is generated, go to Step 12
- Step 11: If public key is not generated, go to Step 15
- Step 12: Ask if user would like to enter custom keys
- Step 13: If yes, go to Step 15
- Step 14: If no, go to Step 16
- Step 15: Input public key and modulo n from user
- Step 16: Call Encryption Algorithm
- Step 17: Go to Step 2
- Step 18: If private key is generated, go to Step 20
- Step 19: If private key is not generated, go to Step 23
- Step 20: Ask if user would like to enter custom keys

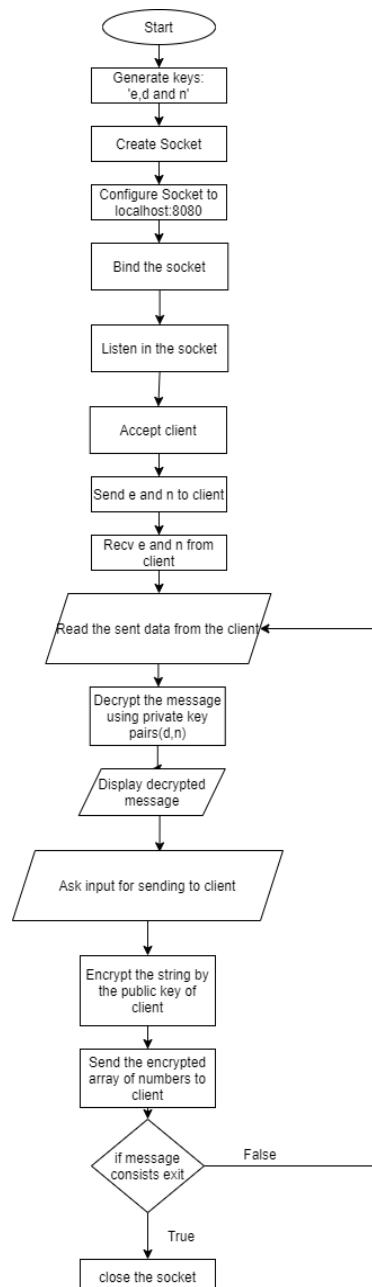
Step 21: If yes, go to Step 23  
Step 22: If no, go to Step 24  
Step 23: Input private key and modulo  $n$  from user  
Step 24: Call Decryption Algorithm  
Step 25: Go to Step 2  
Step 26: Display "Please enter a valid number"  
Step 27: Go to Step 2  
Step 28: Stop

## Flowchart:

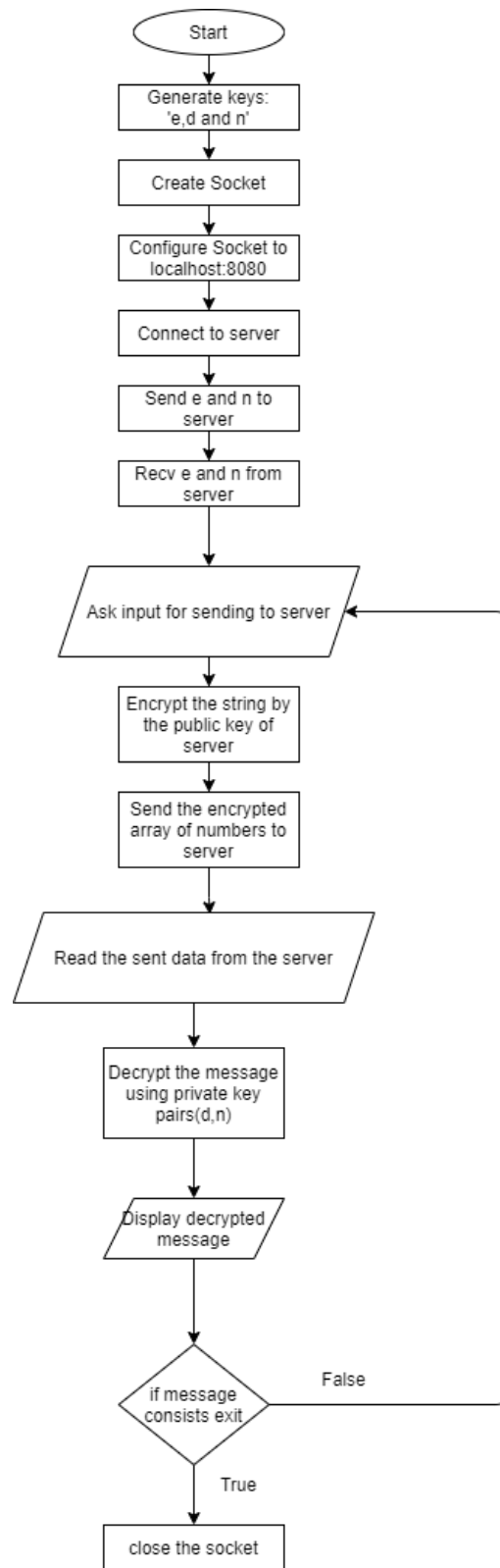


The flowchart of the server and client implementation is shown as below:  
Key generation, encryption and decryption algorithms are used from the 'RSA.h' header file.

### 3.3. Flowchart for Server.c:



### 3.4. Flowchart for Client.c:



# Chapter-4

## Implementation and Coding

### 4.1 Implementation:

As the main algorithm is in 'RSA.h' header file, we have used the header file as 2 different application. One of the implementations is in 'main.c' file which encrypts and decrypts the data locally. It generates the keys, also can use previously generated keys, to encrypt the data from a \*.txt file. Basically, it reads the txt file and encrypts the text. Then, it displays the text as ciphered numbers and also the user can save the ciphered numbers into a file. The following is displayed when the code is executed in main.c in the front display:

```
*//////////@@@@@@@@@@@@@@//////////@@@@#////////&@@@@@@@@@@@@@*****
*//////////@@@@@@@@@@@@@@@@//////////@@@@//*****@@@@*****
*//////////@@@@@@@@@@@@@@@@//////////&@@*,,,,,,,,,***@@/*****
*//////////&@@@@@@@@@@@@@@@@//,,@@@@,,,,,,,,,,@@@@*****
*//////////@@@@@@@@@@@@@@@@,@@@&,,@@@@,@@*****
*//////////@@@@@@@@@@@@@@@@#,,,,,@@@@@@@@&,,,,,@@@@@@@@@(*)*****
*//////////@@@@@@@@@@@@@@@@@@@@#,,,,,@@@@@@@@@@@@@@@@@@@@*****
*//////////*,,,#@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@,****
*////,,,,,,,,,,&@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@,****
,,,,,,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@,
,,,,,,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@/,,,,,,,,
,,,,,,,,,,,,,,,,,@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@(,,,,,,,,
,,,,,,,,,,,,,,,,,@@@@,,,,,,,,,,,,,,,,,@@@@,,,,,,,,,,,,,,,,
-,,,,,,,,@,*@,@@,@@,&,@,@@,*(@,##(@,
,,,,,,,,@*,&,@,@@,/*@@/,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,
Welcome to
Project RSA
By Cryptics
Enter:
1 to generate keys.
2 to encrypt.
3 to decrypt
0 to exit
```

From here, the navigation takes place whether to generate keys, encrypt data or decrypt the data.

## 4.2 Coding:

The code section is divided into 3 major parts. They are included in RSA.h header file, main.c and client.c and server.c. The RSA.h header file consists of all the algorithmic part of the RSA Algorithm, main.c is the implementation of RSA.h as a encryption system in local machine. It asks the input from user as string or file input and encrypts the input via the functions of RSA.h header file. Similarly, server.c and client.c is the socket implementation of RSA.h header file. They encrypt and decrypt the data as they are being transferred from one another.

The functions used in RSA.h are:

```
int check_gcd(long int n1, long int n2)
struct keys generate_keys(long int phifunction)
long int encrypt(long int in_num, long int e, long int n)
long int decrypt(long int encrypted_number, long int d, long int n)
int check_prime(int n)
```

### 1. int check\_gcd(long int n1, long int n2):

Here check\_gcd() is a function to check given numbers (n1,n2) are co-prime or not.

### 2. struct keys generate\_keys(long int phifunction):

In function, generate\_keys(), the process of key generation takes place. It takes the phifunction of n which is  $\phi(n)$ , is the number of coprime of n, where  $n = \text{product of input primes}$ . It returns a structure of keys which is given as:

```
struct keys{
    long int e;
    long int d;
}keys;
```

The process of key generation is mentioned below:

1. At first select two large prime numbers (more than 100 digits) `p1` and `p2`.
2. Calculate modulo(n):  $n = p1 * p2$



3. Calculate the totient function:  $\phi(n) = (p_1-1)(p_2-1)$ .

The totient function gives the number of numbers that are coprime to  $n$ .

Two numbers are `coprime` if their HCF is 1. Example : 5 and 6 are `coprime`.

4. Select an integer `e` such that is coprime to totient function ( $\phi(n)$ ) and  $1 < e < \phi(n)$ .

The pair `(e,n)` now serves as public key.

5. The private key `(d)` should be calculated such that  $(d.e) \bmod \phi(n) = 1$ .

Mathematically, `d` can be calculated using the formula :

$$d = (i * \phi(n) + 1) / e$$

where we increase the value of i till d comes out as integer.

The pair `(d,n)` now serves as private key.

The value of `d` can also be found using extended Euclidean algorithm.

### 3. long int encrypt(long int in\_num, long int e, long int n):

Now the public key(e,n) is used for encryption. Given any text, we first convert the text into ASCII and each ASCII code is encrypted using public key(e,n) as below:

Given a plaintext P, represented as an ASCII, the ciphertext C is calculated as:

$$C = P^e \bmod n$$

then the encrypted number C is converted into character.

### 4. long int decrypt(long int encrypted\_number, long int d, long int n):

Now the private key(d,n) is used for decryption. Given the encrypted number, we first decrypt the number into equivalent ASCII and then convert into equivalent character (which is the required character).

Given a ciphertext C, represented as an ASCII, the plaintext P is calculated as:

$$P = C^d \bmod n$$

then the encrypted number C is converted into character.

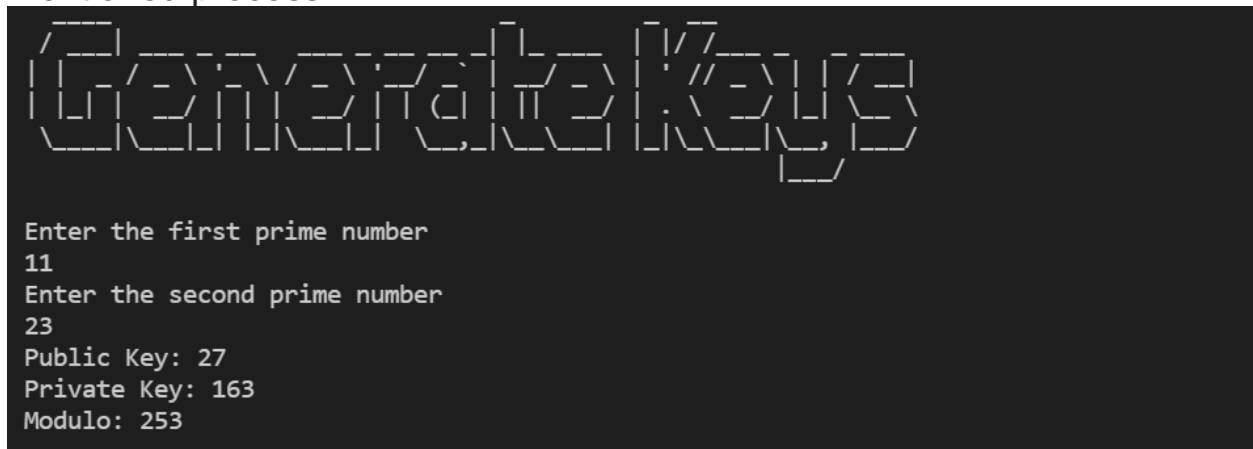
### 5. int check\_prime(int n):

This is a function which is used to check if the number(n) is prime or not.

# Chapter-5

## Results and Discussion

As the main algorithm is in 'RSA.h' header file, we have used the header file as 2 different application. One of the implementations is in 'main.c' file which encrypts and decrypts the data locally. It generates the keys, also can use previously generated keys, to encrypt the data from a \*.txt file. Basically, it reads the txt file and encrypts the text. Then, it displays the text as ciphered numbers and also the user can save the ciphered numbers into a file. Below are some of the screenshots of the mentioned process:



```
Generate Keys
Enter the first prime number
11
Enter the second prime number
23
Public Key: 27
Private Key: 163
Modulo: 253
```

**Fig 4.1: Generation of keys using the prime numbers provided**

```

[Encrypted]

Would you like to encrypt using custom encryption keys?
  If yes, press 1
Else to encrypt using recently generated keys, Enter any key
2
Press r to read from a text file, else press any other key to continue:
2
Enter the text to be encrypted:
hello mate
104    101    108    108    111    32    109    97    116    101

The ciphered text is:
179
238
213
213
34
54
21
158
162
238

Would you like to save the encrypted data in a file?
Press y for yes or any other character for No!
y
Name of file:test.txt

```

Fig 4.2: Encrypting string using the public key pairs

```

RSAinC > test.txt
1  179 238 213 213 34 54 21 158 162 238 |

```

Fig 4.3: Saved Encrypted data in text file as ciphered numbers

```
Decrypting

Enter private key(d) and modulo(n):
163
253
Press r to read from a text file, else press any other key to continue:
r
Enter the name of file to open:test.txt
The decrypted text is:
hello mate

Would you like to save the derypted data in a file?
Press y for yes or any other character for No!
y
Name of file:decrypted.txt
```

**Fig 4.4: Decrypting the encrypted data from the private key pairs**

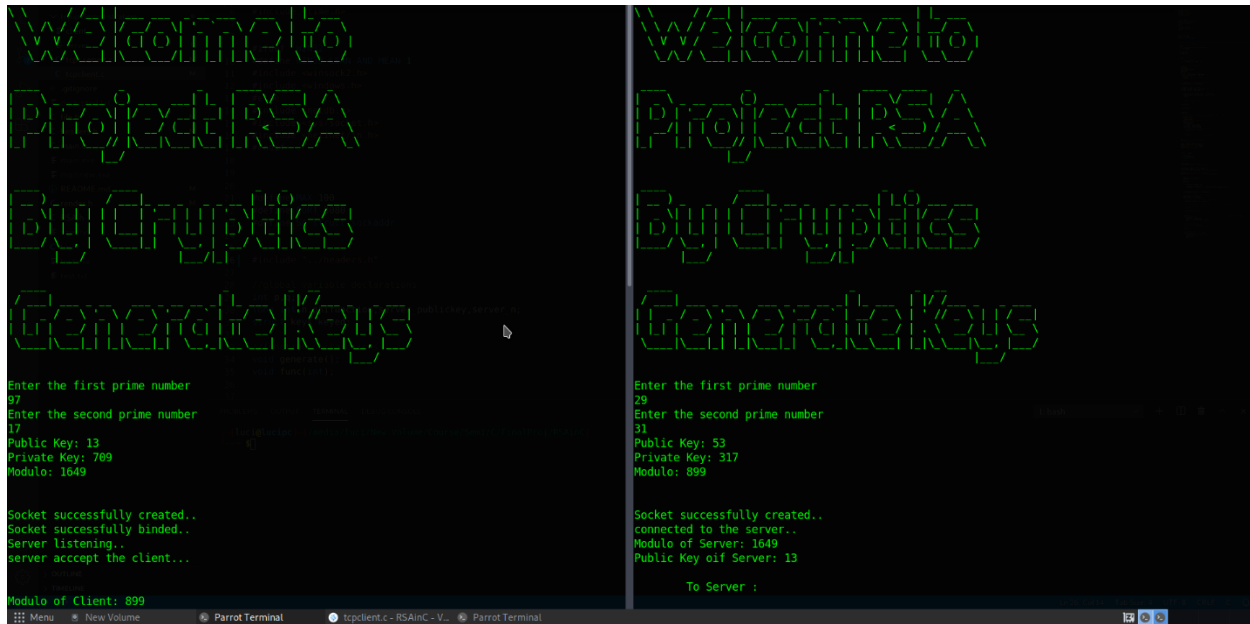
```
RSAC > ≡ decrypted.txt
1 hello mate
```

**Fig 4.5: Saved Decrypted Text**

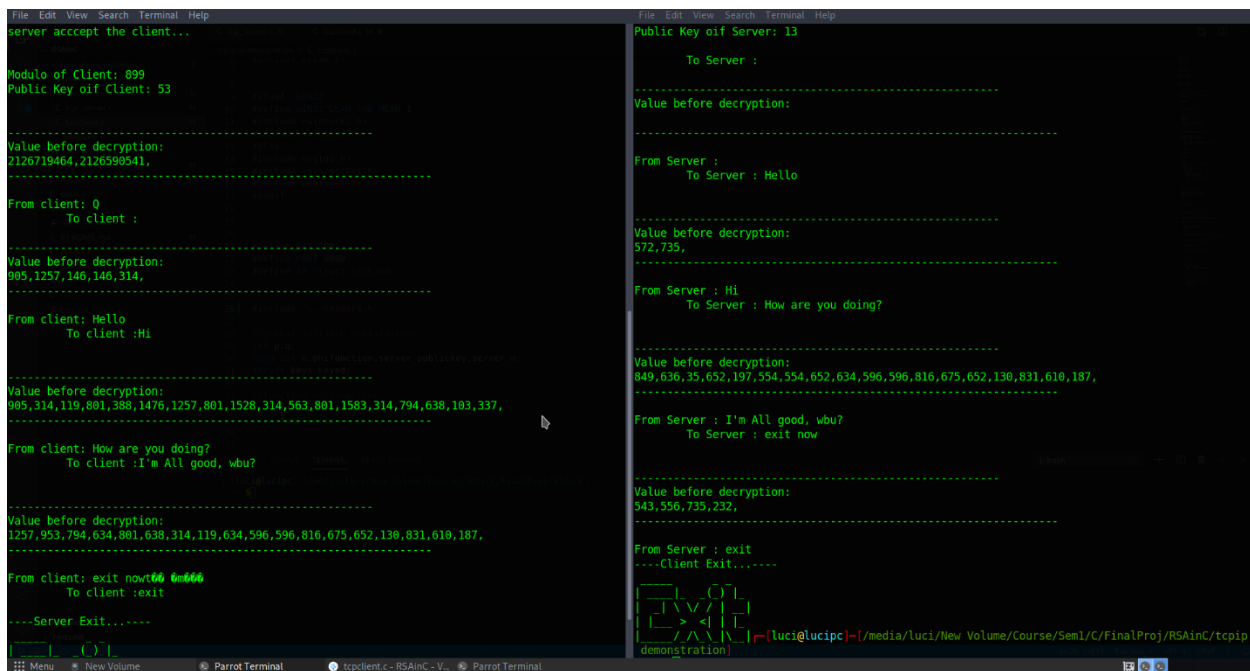
Above are some screenshots of the implementation of the project in main.c, where the user generates the key pairs(public key and private key) as in Fig 4.1, which is then used to encrypt some data which can be stored in a text file as ciphered numbers. We can also read some text file and encrypt it using the program and then save the ciphered data in file.

The ciphered data can only be decrypted using the generated private key pairs. This is demonstrated in fig 4.4 above. In the figure, The text file which was saved as in fig 4.2 was decrypted using the private key pairs generated in fig 4.1. This was then saved as 'decrypted.txt' as shown in fig 4.5 .

### ***TCP/IP implementation of the RSA Algorithm:***



#### Fig 4.6: Key generation and socket creation in server/client



**Fig 4.7: Transfer of messages using the socket in encrypted form**

Fig 4.6 is the TCP/IP implementation of RSA which shows the key generation before the socket creation. Then the socket is created by server which then listens to the localhost:8080. After the keys of the client is generated, the client tries to connect to the localhost:8080 and server accepts the client request to connect to it. After the connection, public key pair of client and server is exchanged.

Fig 4.7 shows the transfer of data via the socket in encrypted form. The encryption is done in the client side using the public keys of server, and vice versa. Therefore, the transferred data is in encrypted form and when intercepted by any means, need to be decrypted to make a sense out of it.

# Chapter-6

## Conclusion

After building and running this project “*Cryptics*”, we are more confident with the general coding concepts and more got more familiar with the concepts of C- Programming Language. This project has given experience of working as a team. Tools like git were not mandatory in the coding phase of the project, but learning and working with git made it far more reliable and easy working with the team.

After carefully analyzing the project we made, we drew the following conclusions from the project as a whole:

- 1) The TCP/IP implementation of *Cryptics* can be compared with end-to-end encrypted chat systems like WhatsApp.
- 2) In this modern era where cyber space is required to be secure, *Cryptics* like projects can make a huge impact in the security of data via encryption.
- 3) We were exposed to new concepts like socket programming which helped build a better foundation and provided new way of thinking to the team.
- 4) In a nutshell, the project gave a feel of how real-life implementations can be made by programming as a whole.

Despite of making our best efforts in the project, some modification is still required which will be made in the next iteration of the project. Some of the things include:

- 1) The TCP/IP implementation is limited only to the Linux Kernel and will be soon made available in UNIX and windows systems.
- 2) The terminal could be made more user-friendly.
- 3) More efficient code could be written with better memory management.

## References

1. Learning C by Examples – Mr. Krishna Kandel
2. <https://www.geeksforgeeks.org/>
3. <https://www.wikipedia.org/>
4. <https://stackoverflow.com/>
5. <https://docs.microsoft.com/en-us/cpp/c-language/>