

# 1. INTRODUCTION

The techniques for secret hiding of messages in an otherwise innocent looking carrier message belong to the field of steganography. The purpose of steganography is to conceal the very presence of secret information. To make the communication more secure, the secret information can be compressed and encrypted before it is hidden in the carrier. This is important because in this way we minimize the amount of information that is to be sent, and it is also easier to hide a random looking message into the carrier than to hide a message with a high degree of regularity. Encrypting the compressed message before hiding is recommended and provides double protection. The main purpose of Steganography, which means 'writing in hiding' is to hide data in a cover media so that others will not be able to notice it. The applications of information hiding systems mainly range over a broad area from military, intelligence agencies, online elections, internet banking, medical-imaging and so on. These variety of applications make steganography a hot topic for study. The cover medium is usually chosen keeping in mind the type and the size of the secret message and many different carrier file formats can be used. In the current situation digital images are the most popular carrier / cover files that can be used to transmit secret information. Steganography works by replacing bits of useless or unused data in regular computer files with bits of different, invisible information. This hidden information can be plain text, cipher text, or even images. Steganography is a technology where modern data compression, information theory, spread spectrum, and cryptography technologies are brought together to satisfy the need for privacy on the Internet.

## 1.1 History:

The first recorded uses of steganography can be traced back to 440 BC when Herodotus mentions an example of steganography in the histories of Herodotus.

An ancient example is that of histories, who shaved the head of his most trusted slave and tattooed a message on it. After his hair had grown the message was hidden. The purpose was to instigate a revolt against the Persians.

1. During the "Cold War" period, US and USSR wanted to hide their sensors in the enemy's facilities. These devices had to send data to their nations, without being spotted.
2. In October 2001, the New York Times published an article claiming that al-Qaeda had used steganography to encode messages into images, and then transported these via e-mail and possibly via USENET to prepare and execute the September 11, 2001 terrorist attack.

## **1.2 Objective:**

The objective of this project is to limit unauthorized access and provide better security during message transmission. To meet the requirements, we use the simple and basic approach of steganography.

In this project, the proposed approach finds the suitable algorithm for embedding the data in an image using steganography which provides the better security pattern for sending messages through a network.

Requirement of this steganography system is that the hidden message carried by stego-media should not be sensible to human beings.

The other goal of steganography is to avoid drawing suspicion to the existence of a hidden message.

For practically implementing the function of the discussed algorithms, C programming language is used.

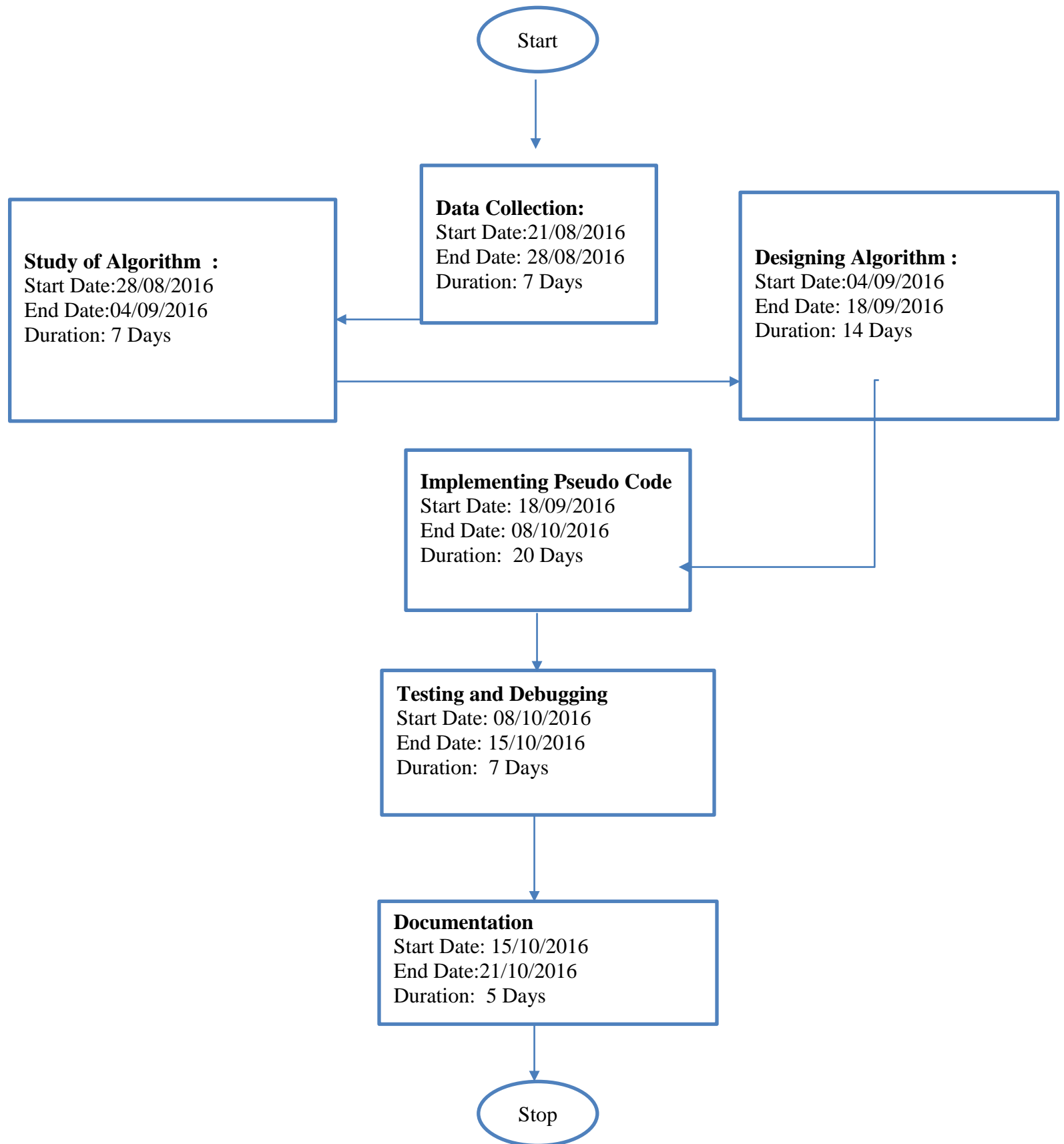
## **1.3 Problem Statement:**

How can we send a message secretly to the destination.

Using steganography, information can be hidden in carriers such as images, audio files, text files, videos and data transmissions.

In this study, I proposed a new framework of an image steganography system to hide a digital text of a secret message.

#### 1.4 Schedule: (PERT Chart)



## **2.System Analysis**

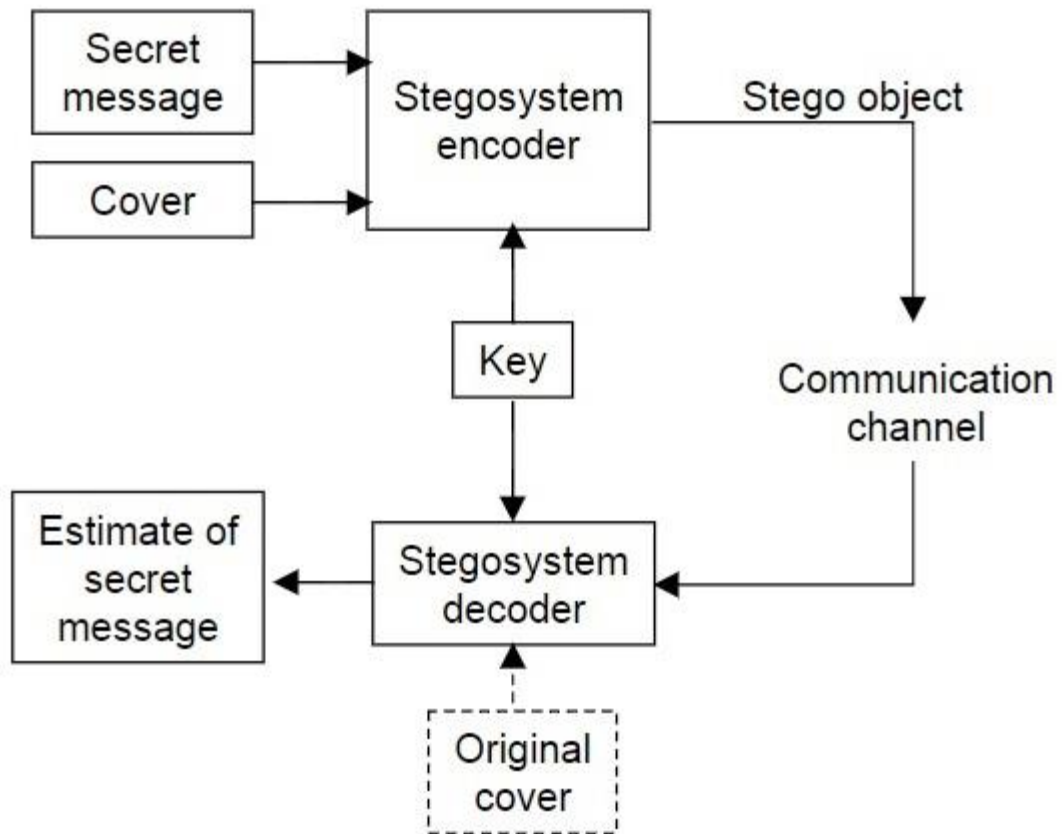
### **2.1 A Framework for Steganographic Algorithms:**

The steganographic algorithms are based on replacing a noise component of a digital object with a pseudo-random secret message. According to the terminology proposed by Pfitzmann to the First International Workshop on Information Hiding [BiPfi96], we call the noisy message cover and the bits carrying the noise cover bits. The bits embedded as pseudo-random noise are secret bits. The cover bits substituted with the secret bits are called hiding bits.

Typically, the cover bits are the least significant bits(LSB) of the digital object. These bits are the result of some series of measurements. For example in sounds the digital information can be viewed as a huge array of measurement results, each component having a random error component. The least significant bits may have some statistical properties. Changing some of them could result in the loss of those properties. Thus the message should be embedded mimicking the characteristics of the cover bits. One possibility is to generate a large number of cover messages in the same way and to choose the one having the secret embedded in it. This method is called selection method.

#### **2.1.1 The Scheme for Embedding Data**

A general scheme for embedding data is depicted in Figure8 below. A message is embedded in a file by the stegosystem encoder, which has as inputs the original cover, the secret message and a key. The resulting stego object is then transmitted over a communication channel to the recipient where the stegosystem decoder using the same key processes it and the message can be read. Secret message Cover Stegosystem encoder Key Communication channel Stegosystem decoder Estimate of secret Message Original cover Stego object.



*Figure 1 General schema of Steganography*

### 3. Steganography Technique

#### 3.1 Pure Steganography

Pure steganography is the process of embedding the data into the object without using any private keys. This type of steganography entirely depends upon the secrecy. This type of steganography uses a cover image in which data is to be embedded, personal information to be transmitted, and encryption decryption algorithms to embed the message into image.

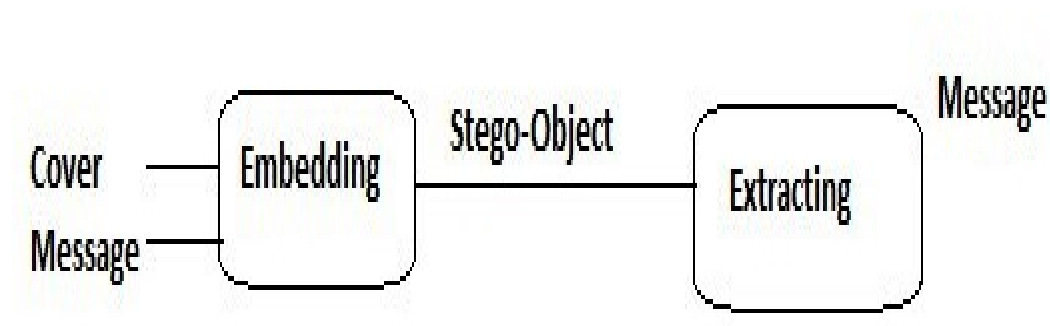


Figure 2 Scheme of steganography

#### 3.2 LSB Algorithm

- LSB (Least Significant Bit) substitution is the process of adjusting the least significant bit pixels of the carrier image.
- It is a simple approach for embedding message into the image.
- The Least Significant Bit insertion varies according to number of bits in an image.
- For an 8 bit image, the least significant bit i.e., the 8<sup>th</sup> bit of each byte of the image is changed to the bit of secret message.
- For 24 bit image, the colors of each component like RGB (red, green and blue) are changed.
- LSB is effective in using BMP images since the compression in BMP is lossless

### 3.3 How LSB Works:

Structure of **text file**:

Any **text file** consists of streams of characters , each character is 1 byte (ASCII code) each byte as all of us of course know consists of 8 bits .

Part of text file : H i a l l

Ascii Code : 72 105 32 97 108 108

Binary equivalent: 01001000 01101001 00100000 01100001 01101100 01101100

Figure 3 Text file format

Pixel	Byte 1 – Red	Byte 2 – Green	Byte 3 – Blue
1	0 1 0 1 0 1 1 0	0 1 1 1 1 0 1 1	0 0 0 1 1 1 1 0
2	0 1 0 1 0 1 1 0	0 1 1 1 1 0 1 0	0 0 0 1 1 1 1 0
3	0 1 0 1 0 1 1 0	0 1 1 1 1 0 1 1	0 0 0 1 1 1 1 0
4	0 1 0 1 0 1 1 1	0 1 1 1 1 0 1 1	0 0 0 1 1 1 1 0
5	0 1 0 1 0 1 1 1	0 1 1 1 1 0 1 0	0 0 0 1 1 1 1 0
6	0 1 0 1 0 1 1 0	0 1 1 1 1 0 1 0	0 0 0 1 1 1 1 1
7	0 1 0 1 0 1 1 1	0 1 1 1 1 0 1 0	0 0 0 1 1 1 1 0
8	0 1 0 1 0 1 1 0	0 1 1 1 1 0 1 0	0 0 0 1 1 1 1 1
9	0 1 0 1 0 1 1 0	0 1 1 1 1 0 1 0	0 0 0 1 1 1 1 1
10	0 1 0 1 0 1 1 0	0 1 1 1 1 0 1 0	0 0 0 1 1 1 1 0
11	0 1 0 1 0 1 1 0	0 1 1 1 1 0 1 1	0 0 0 1 1 1 1 0

Figure 4pixel view RGB

If we want to hide the data like “Aha!”

Then we convert the message “Aha!” into ASCII Code and then there equivalent binary code.

A=65 (01000001)

h=104(01101000)

a=97(01100001)

!=33(00100001)

## **4.Encryption and Decryption of text in C Language**

Steganography works by replacing bits of useless or unused data in regular computer files with bits of our important data. In our case, our data will be the plain text that we need to hide, and the unused data is the least significant bits (LSBs) in the image pixels.

The C programming language is used to extensively analyse the functions of the LSB algorithm in steganography. Texts and other file formats are encrypted and embedded into an image file which is then transferred to the destination.

### **4.1 Encryption of text**

Loop through the pixels of the image. In each iteration, get the RGB values separated each in a separate integer.

For each of R, G, and B, make the LSB equals to 0. These bits will be used in hiding characters.

Get the current character and convert it to integer. Then hide its 8 bits in R1, G1, B1, R2, G2, B2, R3, G3, where the numbers refer to the numbers of the pixels. In each LSB of these elements (from R1 to G3), hide the bits of the character consecutively.

When the 8 bits of the character are processed, jump to the next character, and repeat the process until the whole text is processed.

The text can be hidden in a small part of the image according to the length of that text. So, there must be something to indicate that here we reached the end of the text. The indicator is simply 8 consecutive zeros. This will be needed when extracting the text from the image.

### **4.2 Decryption of text**

It's more simple than hiding. Just pass through the pixels of the image until you find 8 consecutive zeros. As you are passing, pick the LSB from each pixel element (R, G, B) and attach it into an empty value. When the 8 bits of this value are done, convert it back to character, then add that character to the result text you are seeking.



## 5.Design Phase

### 5.1Block Diagram for Steganography

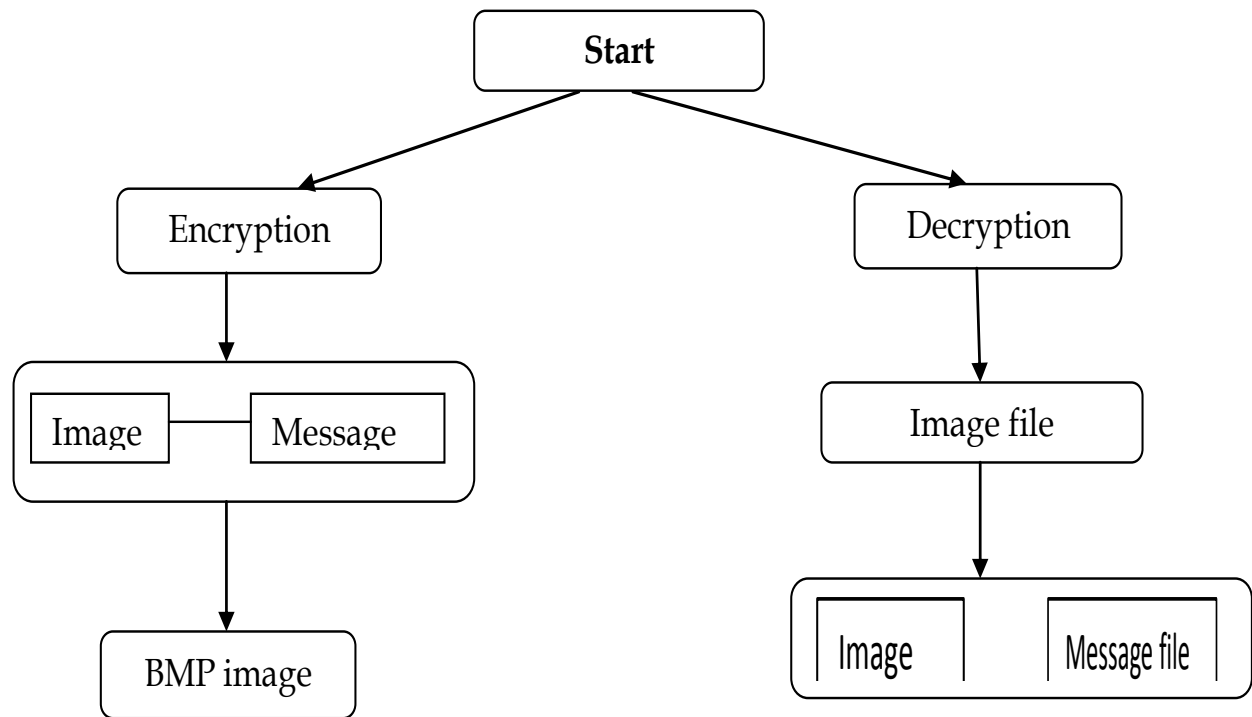
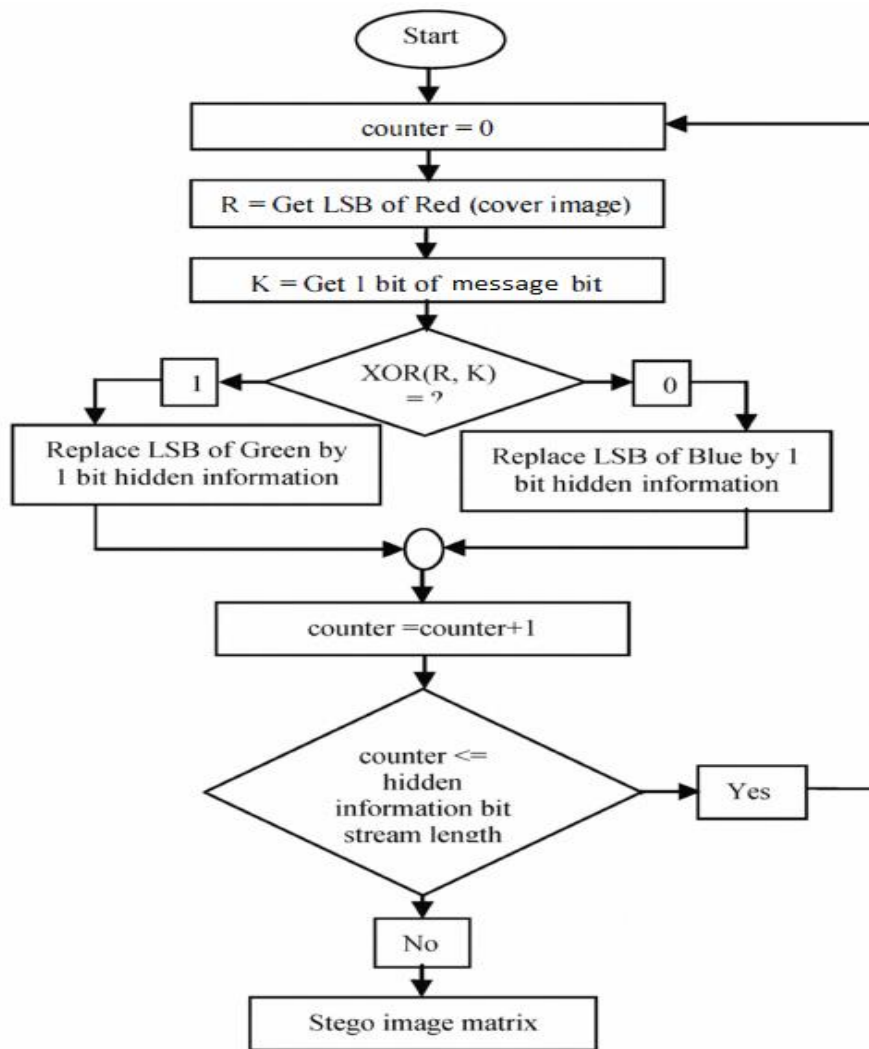


Figure 5 Block Diagram

## 5.2Flow Chart

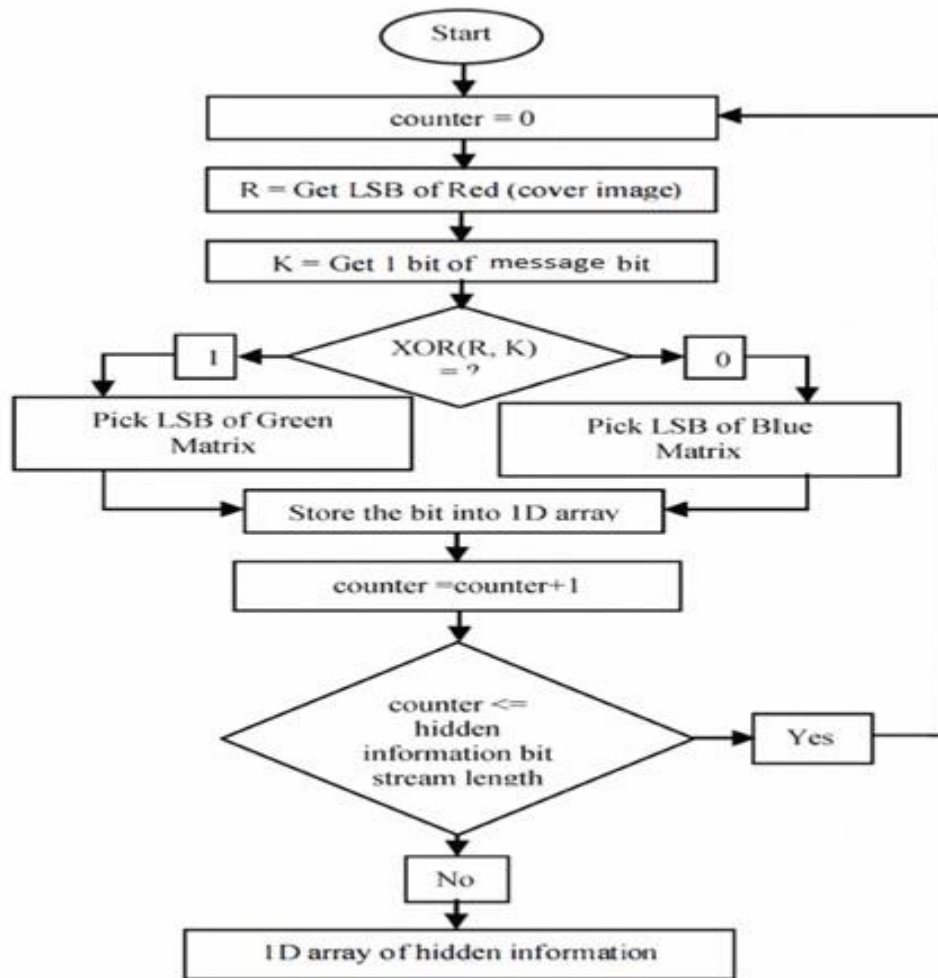
### 5.1.1 Encryption



Flow Chart to hidden information into cover image

*Figure 6 Flow Chart Encryption*

### 5.1.2Decryption



Flow Chart to recover hidden information from stego image

Figure 7 Flow Chart Decryption

### 5.3 Class Diagram

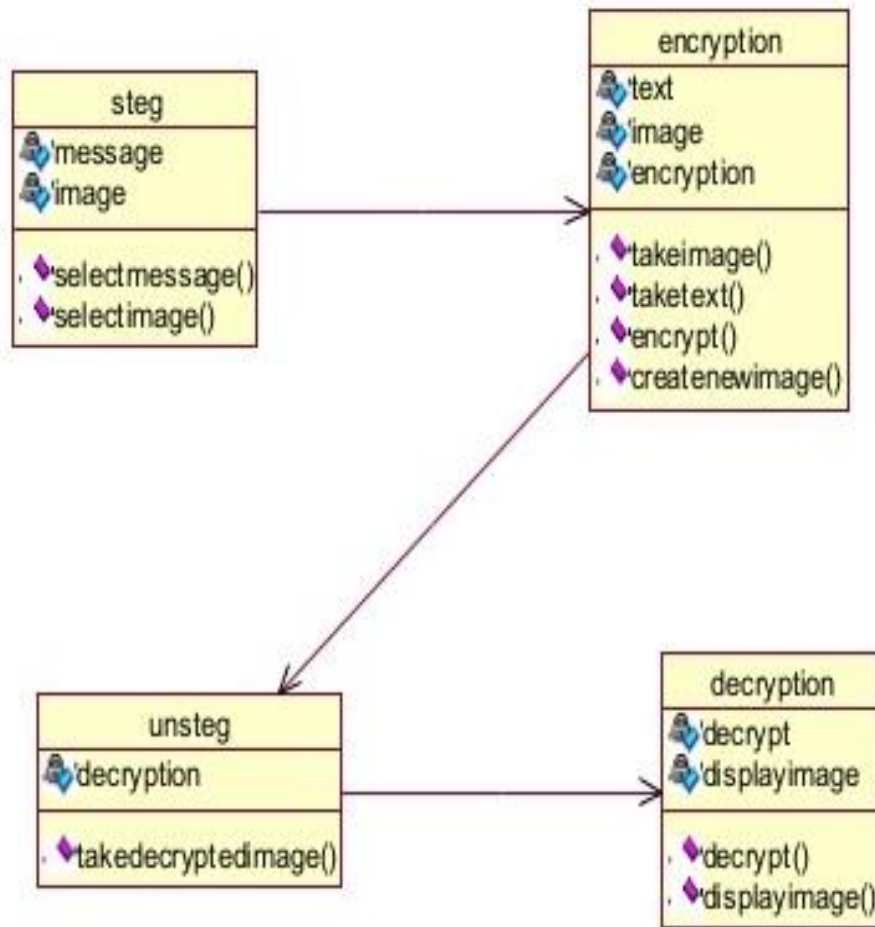


Figure 8 Class Diagram

## 5.4 Use Case Diagram

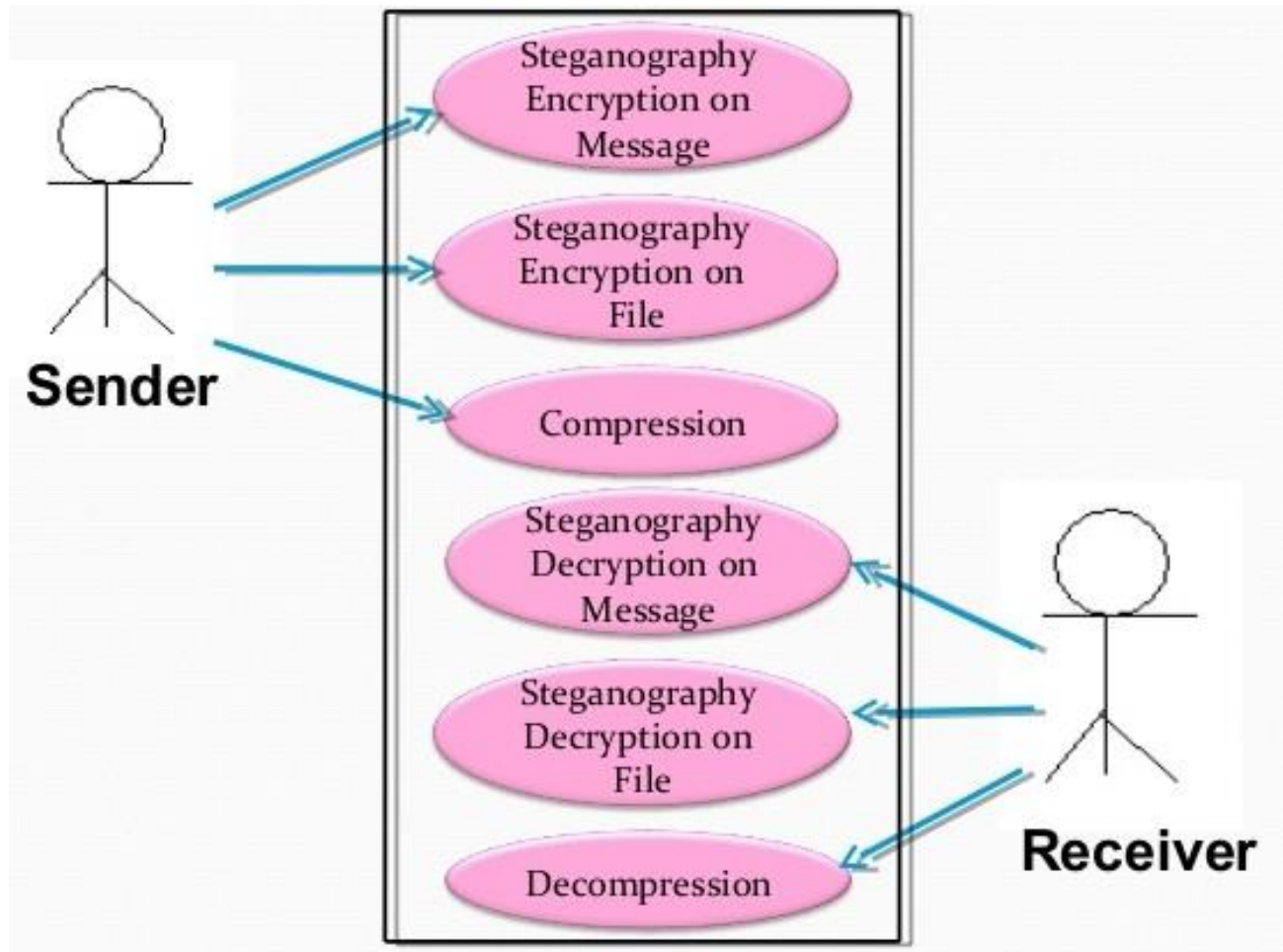


Figure 9 Use Case Diagram

## 5.5 Activity Diagram

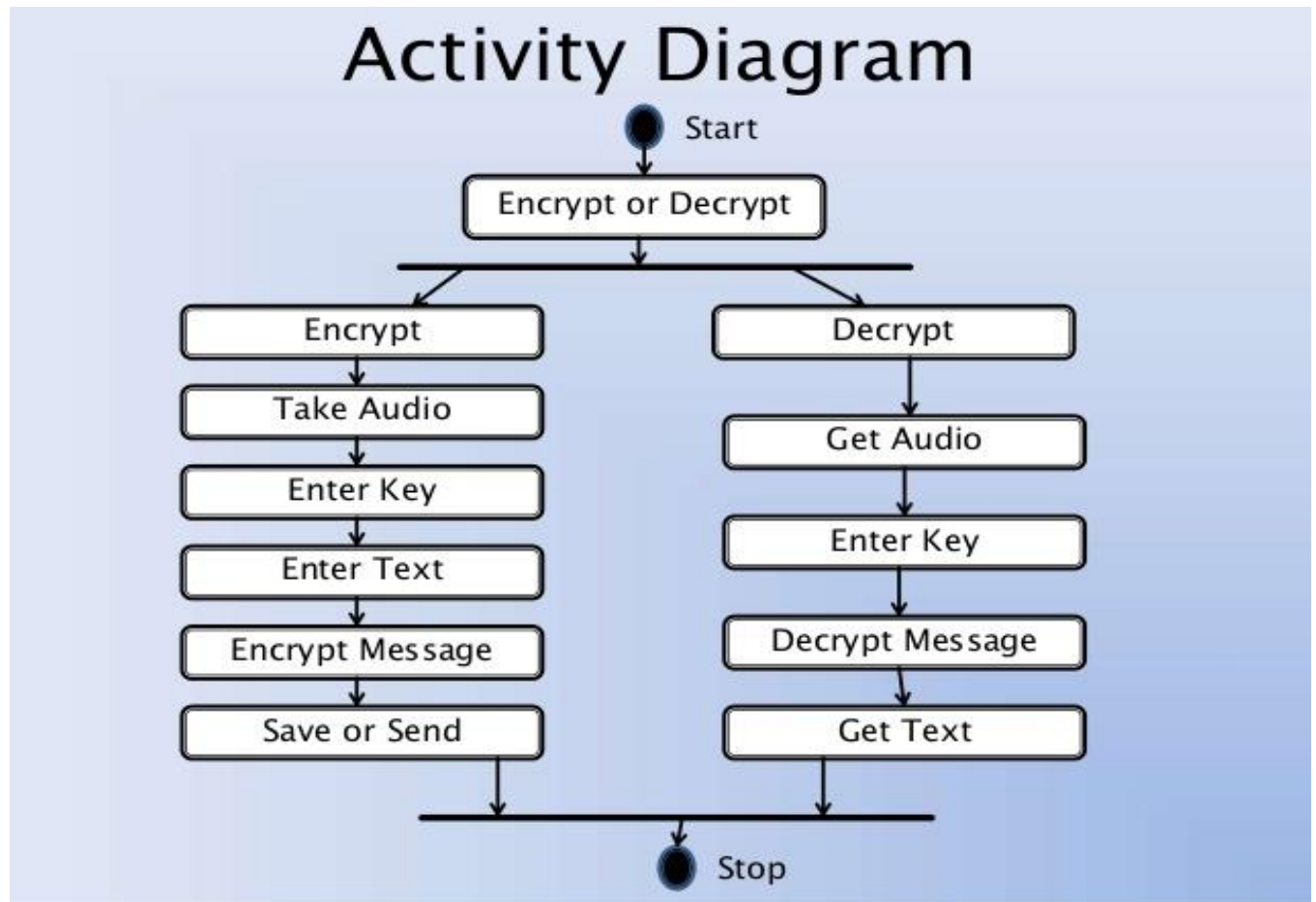
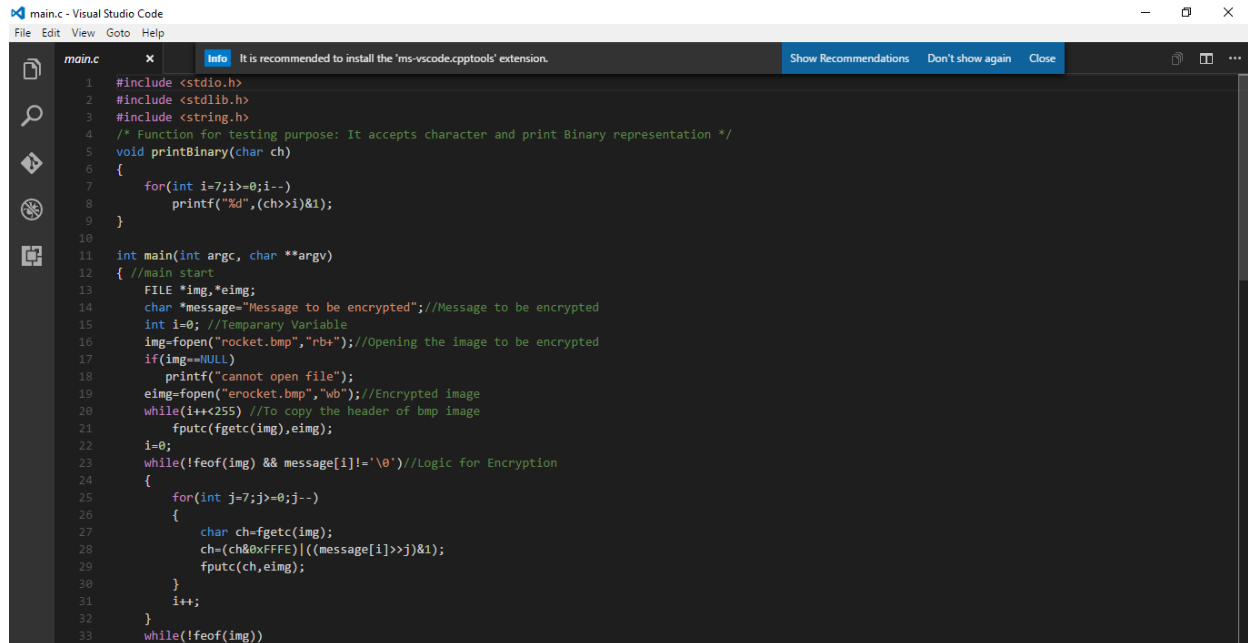


Figure 10 Activity Diagram

## 6.Implementation

### 6.1 Code Snippets

#### 6.1.1 First Phase Coding

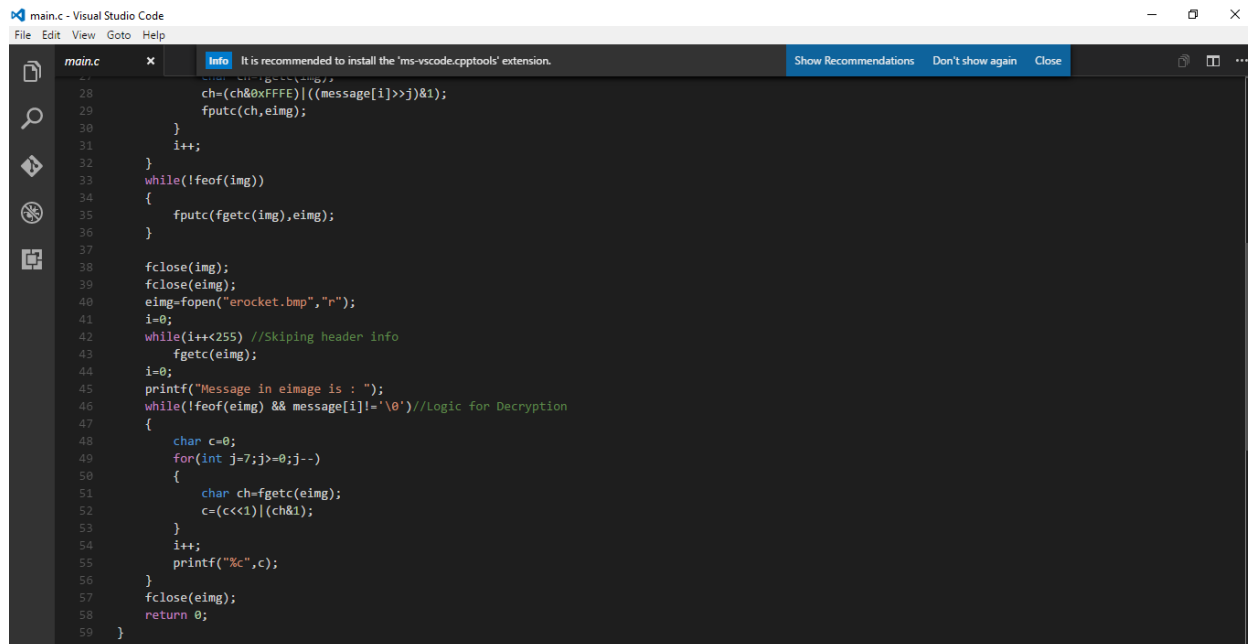


```
main.c - Visual Studio Code
File Edit View Goto Help

main.c x Info It is recommended to install the 'ms-vscode.cpptools' extension. Show Recommendations Don't show again Close

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 /* Function for testing purpose: It accepts character and print Binary representation */
5 void printBinary(char ch)
6 {
7     for(int i=7;i>=0;i--)
8         printf("%d", (ch>>i)&1);
9 }
10
11 int main(int argc, char **argv)
12 { //main start
13     FILE *img,*eimg;
14     char *message="Message to be encrypted";//Message to be encrypted
15     int i=0; //Temporary Variable
16     img=fopen("rocket.bmp","rb");//Opening the image to be encrypted
17     if(img==NULL)
18         printf("cannot open file");
19     eimg=fopen("erocket.bmp","wb");//Encrypted image
20     while(i++<255) //To copy the header of bmp image
21         fputc(fgetc(img),eimg);
22     i=0;
23     while(!feof(img) && message[i]!='\0')//Logic for Encryption
24     {
25         for(int j=7;j>=0;j--)
26         {
27             char ch=fgetc(img);
28             ch=(ch&0xFFFE)|((message[i]>>j)&1);
29             fputc(ch,eimg);
30         }
31         i++;
32     }
33     while(!feof(img))
```

Figure 11 Test Code 1



```
main.c - Visual Studio Code
File Edit View Goto Help

main.c x Info It is recommended to install the 'ms-vscode.cpptools' extension. Show Recommendations Don't show again Close

28     ch=(ch&0xFFFE)|((message[i]>>j)&1);
29     fputc(ch,eimg);
30 }
31 i++;
32 }
33 while(!feof(img))
34 {
35     fputc(fgetc(img),eimg);
36 }
37
38 fclose(img);
39 fclose(eimg);
40 eimg=fopen("erocket.bmp","r");
41 i=0;
42 while(i++<255) //Skipping header info
43     fgetc(eimg);
44 i=0;
45 printf("Message in eimage is : ");
46 while(!feof(eimg) && message[i]!='\0')//Logic for Decryption
47 {
48     char c=0;
49     for(int j=7;j>=0;j--)
50     {
51         char ch=fgetc(eimg);
52         c=(c<<1)|(ch&1);
53     }
54     i++;
55     printf("%c",c);
56 }
57 fclose(eimg);
58 return 0;
59 }
```

Figure 12 Test Code 2

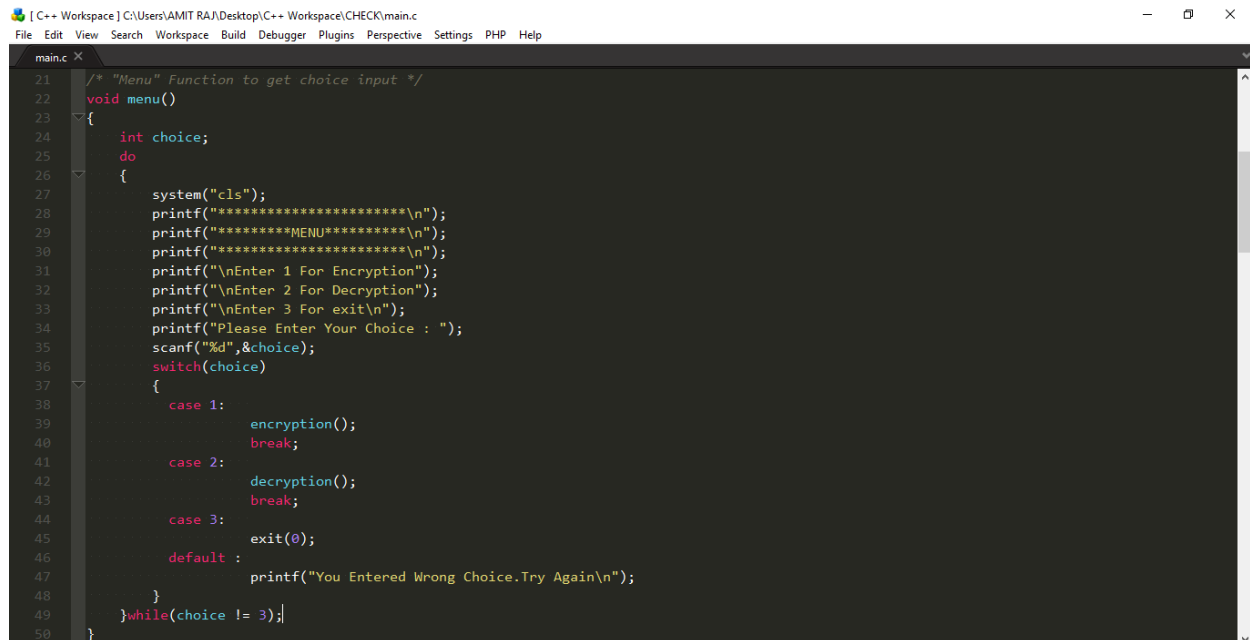
## 6.1.2 Final Phase



```
[ C++ Workspace ] C:\Users\AMIT RAJ\Desktop\C++ Workspace\CHECK\main.c
File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help

main.c X
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  /* Function for testing purpose: It accepts character and print Binary representation */
5  void printBinary(char ch)
6  {
7      for(int i=7;i>=0;i--)
8          printf("%d", (ch>>i)&1);
9  }
10
11 void menu();
12 void encryption();
13 void decryption();
14
15 int main(int argc, char **argv) //main start
16 {
17     menu();
18     return 0;
19 } //main end
20
21 /* "Menu" Function to get choice input */
22 void menu()
23 {
24     int choice;
25     do
26     {
27         system("cls");
28         printf("*****\n");
29         printf("*****MENU*****\n");
30         printf("*****\n");
```

Figure 13 Final Phase Code 1



```
21 /* "Menu" Function to get choice input */
22 void menu()
23 {
24     int choice;
25     do
26     {
27         system("cls");
28         printf("*****\n");
29         printf("*****MENU*****\n");
30         printf("*****\n");
31         printf("\nEnter 1 For Encryption");
32         printf("\nEnter 2 For Decryption");
33         printf("\nEnter 3 For exit\n");
34         printf("Please Enter Your Choice : ");
35         scanf("%d",&choice);
36         switch(choice)
37         {
38             case 1:
39                 encryption();
40                 break;
41             case 2:
42                 decryption();
43                 break;
44             case 3:
45                 exit(0);
46             default :
47                 printf("You Entered Wrong Choice.Try Again\n");
48         }
49     }while(choice != 3);
50 }
```

Figure 14 Final Phase Code2



```

103 FILE *img,*eimg;
104 int i;
105 eimg=fopen("eabcd.bmp","r");
106 i=0;
107 while(i++<255) //Skipping header info
108     fgetc(eimg);
109 i=0;
110 int messageLength=0;
111 for(int j=31;j>=0;j--)
112 {
113     char ch=fgetc(eimg);
114     messageLength=(messageLength<<1)|(ch&1);
115 }
116 printf("Message in encrypted image is : ");
117 while(!feof(eimg) && i<messageLength)//Logic for Decryption
118 {
119     char c=0;
120     for(int j=7;j>=0;j--)
121     {
122         char ch=fgetc(eimg);
123         c=(c<<1)|(ch&1);
124     }
125     i++;
126     printf("%c",c);
127 }
128 printf("\n");
129 fclose(eimg);
130 printf("Press any key to continue...");
131 getch();
132 }

```

Figure 15 Final Phase Code3

```

52 /* encryption() for encrypting the image with required message */
53 void encryption()
54 {
55     FILE *img,*eimg;
56     char *message;
57     int i,messageLength;
58     printf("Enter message to be encrypted:-");
59     fseek(stdin,0,SEEK_END);//To flush stdin
60     gets(message);
61     for(i=0;message[i]!='\0';i++);//To get the length of message to be encrypted
62     messageLength = i;
63     img=fopen("abcd.bmp","rb+");//Opening the image to be encrypted
64     if(img==NULL)
65         printf("cannot open file");
66     eimg=fopen("eabcd.bmp","wb");//Encrypted image
67     i=0;
68     while(i++<255) //To copy the header of bmp image
69     {
70         fputc(fgetc(img),eimg);
71     }
72     for(i=31;i>=0;i--)
73     {
74         char ch=fgetc(img);
75         ch=(ch&0xFFE)|((messageLength>>i)&1);
76         fputc(ch,eimg);
77     }
78     i=0;
79     while(!feof(img) && i<messageLength)//Logic for Encryption
80     {
81         for(int j=7;j>=0;j--)

```

Figure 14 Final Phase Code4

```

70     fputc(fgetc(img),eimg);
71 }
72 for(i=31;i>=0;i--)
73 {
74     char ch=fgetc(img);
75     ch=(ch&0xFFE)|((messageLength>>i)&1);
76     fputc(ch,eimg);
77 }
78 i=0;
79 while(!feof(img) && i<messageLength)//Logic for Encryption
80 {
81     for(int j=7;j>=0;j--)
82     {
83         char ch=fgetc(img);
84         ch=(ch&0xFFE)|((message[i]>>j)&1);
85         fputc(ch,eimg);
86     }
87     i++;
88 }
89
90 while(!feof(img))
91 {
92     fputc(fgetc(img),eimg);
93 }
94 fclose(img);
95 fclose(eimg);
96 printf("\n\nYour Message is successfully Encrypted...\n\nPress any key to continue");
97 getch();
98 }
99

```

Figure 1516 Final Phase Code5

```

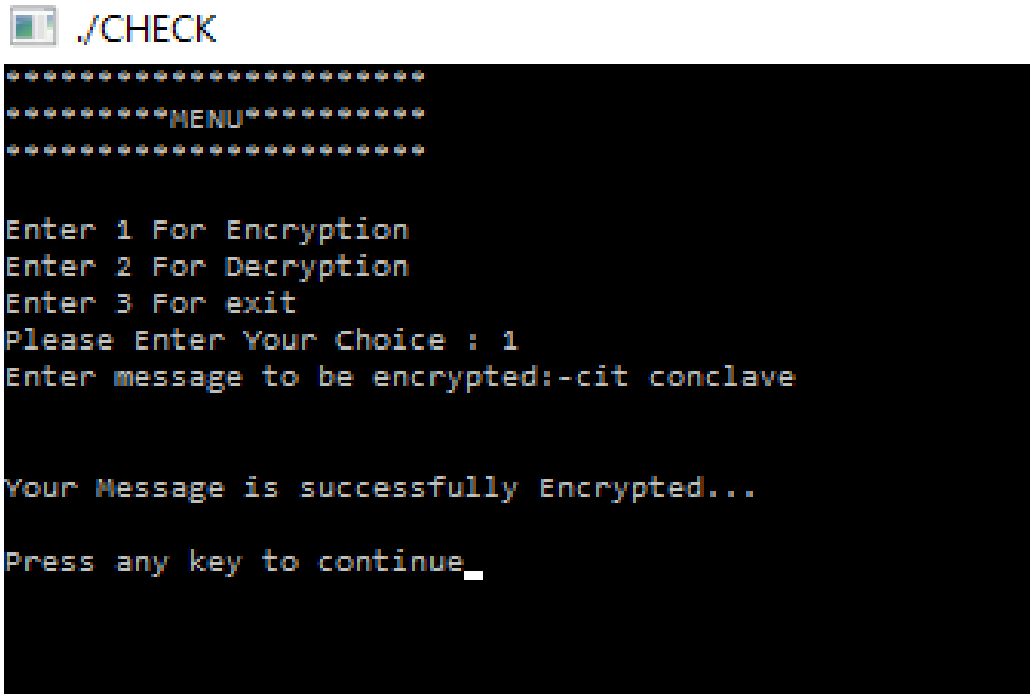
100 /* decryption() for decrypting the message from the image */
101 void decryption()
102 {
103     FILE *img,*eimg;
104     int i;
105     eimg=fopen("eabcd.bmp","r");
106     i=0;
107     while(i++<255) //Skipping header info
108         fgetc(eimg);
109     i=0;
110     int messageLength=0;
111     for(int j=31;j>=0;j--)
112     {
113         char ch=fgetc(eimg);
114         messageLength=(messageLength<<1)|(ch&1);
115     }
116     printf("Message in encrypted image is : ");
117     while(!feof(eimg) && i<messageLength)//Logic for Decryption
118     {
119         char c=0;
120         for(int j=7;j>=0;j--)
121         {
122             char ch=fgetc(eimg);
123             c=(c<<1)|(ch&1);
124         }
125         i++;
126         printf("%c",c);
127     }
128     printf("\n");
129     fclose(eimg);
130 }
131

```

Figure 17 Final Phase Code6

## 7.Output Screens

### 7.1 Output Window for Encryption



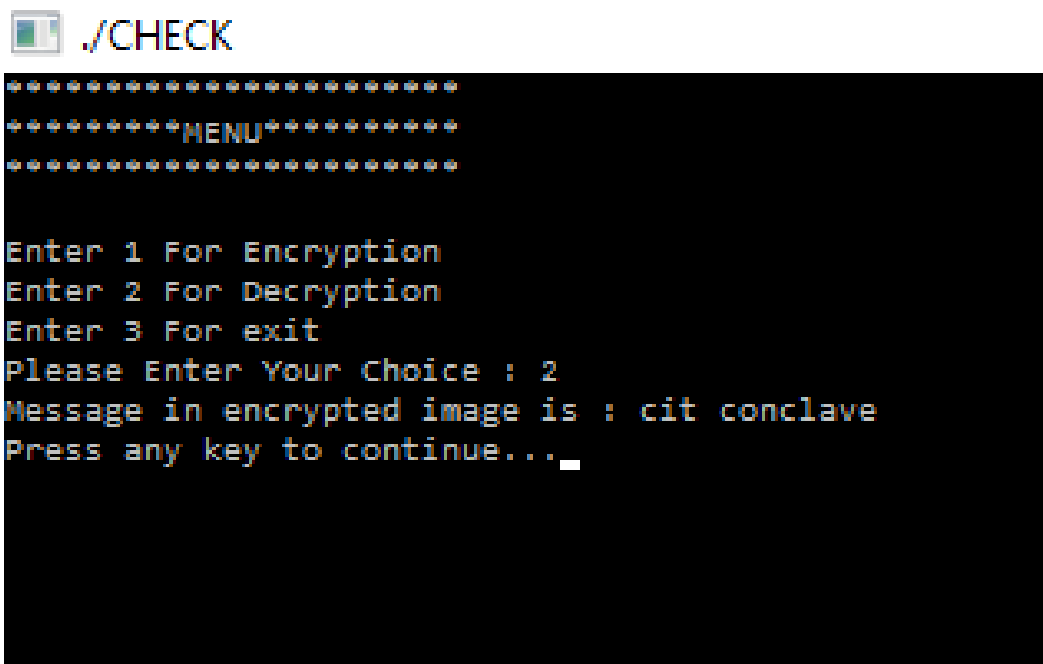
```
./CHECK
*****
*****MENU*****
*****

Enter 1 For Encryption
Enter 2 For Decryption
Enter 3 For exit
Please Enter Your Choice : 1
Enter message to be encrypted:-cit conclave

Your Message is successfully Encrypted...
Press any key to continue_
```

*Figure 18 Output Console1*

### 7.2 Output Window for Decryption



```
./CHECK
*****
*****MENU*****
*****

Enter 1 For Encryption
Enter 2 For Decryption
Enter 3 For exit
Please Enter Your Choice : 2
Message in encrypted image is : cit conclave
Press any key to continue..._
```

*Figure 19 Output Console2*

7.3 Output Images



Figure 20Original Image



Figure 21Stego Image

7.4 Output Image Window

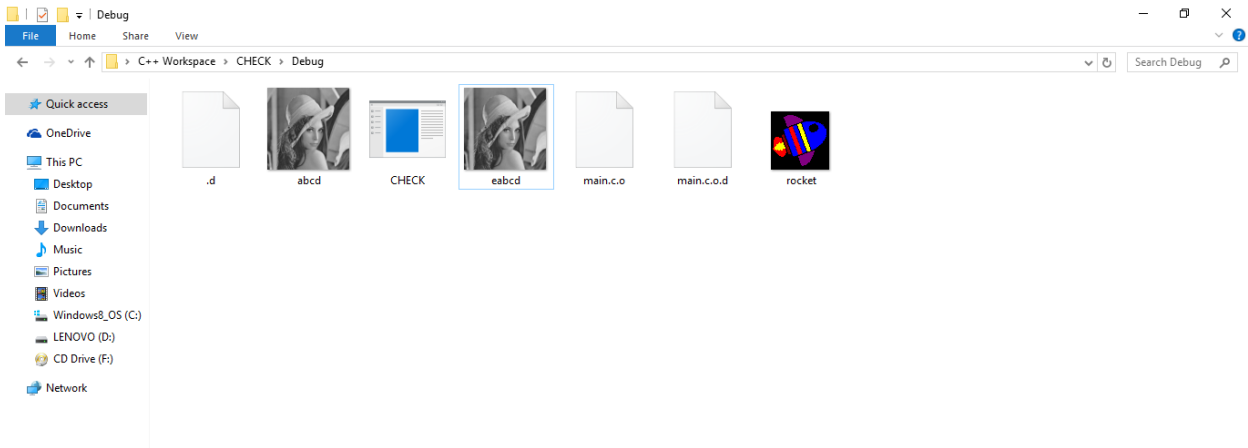


Figure 22 Output Folder

## **8.Result and Discussion**

In this project we mainly concentrated on embedding the data into an image. We have designed the steganographic application which embedded the data into the image.

Normally, after embedding the data into the image, the image may lose its resolution. In the proposed approach, the image remains unchanged in its resolution as well in size.

The speed of embedding the data into the image is also high in the proposed approach such that the image is protected and the data to the destination is sent securely.

For the decryption phase, we have used the same C programming language for the purpose of designing.

There are many steganographic algorithms available like JSteg, F5 and LSB algorithms. We have used the Least Significant Bit algorithm in designing the steganographic application because LSB algorithm works efficiently when we consider bit map images .bmp files. The speed of embedding is also high when using LSB compared to the JSteg algorithm.

## 9.Limitations

- Ultimately, image *understanding* is important for secure adaptive steganography. A human can easily recognize that a pixel is actually a dot above the letter "i" and must not be changed. However, it would be very hard to write a computer program capable of making such intelligent decisions in all possible cases.
- The format of the image is BMP i.e., BITMAP only and no other format is tested.
- The image is 24 bit BITMAP file.
- The image has to be input in the program manually in the specified location for the operations of image processing.

## 10.Conclusion and Further Enhancement

In the present world, the data transfers using internet is rapidly growing because it is so easier as well as faster to transfer the data to destination. So, many individuals and business people use to transfer business documents, important information using internet.

Security is an important issue while transferring the data using internet because any unauthorized individual can hack the data and make it useless or obtain information unintended to him. The future work on this project is to improve the compression ratio of the image to the text. This project can be extended to a level such that it can be used for the different types of image formats like .bmp, .jpeg, .tif etc., in the future. The security using Least Significant Bit Algorithm is good but we can improve the level to a certain extent by varying the carriers as well as using different keys for encryption and decryption.

## 11.References

- [1] Amirthanjan,R. Akila,R&Deepikachowdavarapu, P., 2010. A Comparative Analysis of Image Steganography, International Journal of Computer Application.
- [2] Bandyopadhyay, S.K., 2010. An Alternative Approach of Steganography Using Reference Image.International Journal of Advancements in Technology.
- [3] Cox, I. Miller, M. Bloom, J. Fridrich, J &Kalker, T. 2008.Digital watermarking and Steganography.2ndEd.Elsevier.
- [4]Obaida Mohammad Awad Al-Hazaimeh Hiding Data in Images Using New Random Technique Department of Information Technology, AL-BALQA Applied University/Al-Huson University College, Irbid, Al-Huson, 50, Jordan.
- [5]An Overview of Steganography by Shawn D. Dickman Computer Forensics Term Paper, James Madison University.
- [6] A Tutorial Review on Steganography by Samir K Bandyopadhyay, Debnath Bhattacharyya, DebashisGanguly, SwarnenduMukherjeet and Poulami Das University of Calcutta.
- [7] Exploring Steganography: Seeing the Unseen by Neil F. Johnson SushilJajodia George Mason University.