SHILLONG COLLEGE

Shillong-793001, Meghalaya

Project on

"Steganography"

Submitted for the partial fulfilment for the award of the degree of Bachelor of Computer Applications

By

Banlamshai Lyngdoh Mawphlang

Roll No: P1500064

Regn No: 10463 of 2014-15

Department of Computer Science and Applications

Shillong College, Shillong-793001

NORTH EASTERN HILL UNIVERSITY

Certified that this is a bonafide of the project

Entitled

"Steganography"

Submitted for the partial fulfilment for the award of the degree of Bachelor of Computer Applications

Submitted

By

Banlamshai Lyngdoh Mawphlang

Roll No: P1500064

Regn No: 10463 of 2014-15

17/4/17

GUIDE

Mr Donald Thabah

Thabah 17/4/17

HEAD OF DEPARTMENT

Mrs A.Mitri

EXAMINER

17/4/17

Department of Computer Science and Applications

Shillong College, Shillong-793001

Table of Contents

ABSTRACT

Steganography is the art of hiding the fact that communication is taking place, by hiding information in other information. Many different carrier file formats can be used, but digital images are the most popular because of their frequency on the internet. For hiding secret information in images, there exists a large variety of steganography techniques some are more complex than others and all of them have respective strong and weak points. Different applications may require absolute invisibility of the secret information, while others require a large secret message to be hidden. This project report intends to give an overview of image steganography, its uses and techniques. It also attempts to identify the requirements of a good steganography algorithm and briefly reflects on which steganographic techniques are more suitable for which applications.

INTRODUCTION

One of the reasons that intruders can be successful is the most of the information they acquire from a system is in a form that they can read and comprehend. Intruders may reveal the information to others, modify it to misrepresent an individual or organization, or use it to launch an attack. One solution to this problem is, through the use of steganography. Steganography is a technique of hiding information in digital media. In contrast to cryptography, it is not to keep others from knowing the hidden information but it is to keep others from thinking that the information even exists.

Steganography become more important as more people join the cyberspace revolution. Steganography is the art of concealing information in ways that prevents the detection of hidden messages. Steganography include an array of secret communication methods that hide the message from being seen or discovered.

Due to advances in ICT, most of information is kept electronically. Consequently, the security of information has become a fundamental issue. Besides cryptography, steganography can be employed to secure information. In cryptography, the message or encrypted message is embedded in a digital host before passing it through the network, thus the existence of the message is unknown. Besides hiding data for confidentiality, this approach of information hiding can be extended to copyright protection for digital media: audio, video and images.

The growing possibilities of modern communications need the special means of security especially on computer network. The network security is becoming more important as the number of data being exchanged on the internet increases. Therefore, the confidentiality and data integrity are requires to protect against unauthorized access and use. This has resulted in an explosive growth of the field

of information hiding Information hiding is an emerging research area, which encompasses applications such as copyright protection for digital media, watermarking, fingerprinting, and steganography.

In watermarking applications, the message contains information such as owner identification and a digital time stamp, which usually applied for copyright protection.

Fingerprint, the owner of the data set embeds a serial number that uniquely identifies the user of the data set. This adds to copyright information to makes it possible to trace any unauthorized use of the data set back to the user.

Steganography hide the secrete message within the host data set and presence imperceptible and is to be reliably communicated to a receiver. The host data set is purposely corrupted, but in a covert way, designed to be invisible to an information analysis.

SYNOPSIS

Project Name: STEGANOGRAPHY

2. Objective of the project:

This project is developed for hiding information in any image file. The scope of the project is implementation of steganography tools for hiding information includes any type of information file and image files and the path where the user wants to save Image and extruded file.

The goal of steganography is covert communication. So, a fundamental requirement of this steganography system is that the hider message carried by stego-media should not be sensible to human beings.

The other goad of steganography is to avoid drawing suspicion to the existence of a hidden message. This approach of information hiding technique has recently became important in a number of application area

This project has following objectives:

- To product security tool based on steganography techniques.
- To explore techniques of hiding data using encryption module of this project
- To extract techniques of getting secret data using decryption module.

3. Project category:

The Project category is Application base developed in MS Visual Studio 2008 with Net Framework 3.5. Language used is C#.

4. Language and software tool used:

- Front End: C#
- Operating System: Window 7 and above
- Back End: Microsoft SQL Server 2008

6. Hardware requirement:

- Operating system: Window 7 and above.
- Hard disks: 40GB
- RAM: 256 MB

7. Software requirement:

- C#
- .Net Framework
- MS SQL server 2008

Overview

The word steganography comes from the Greek "Seganos", which mean covered or secret and — "graphy" mean writing or drawing. Therefore, steganography mean, literally, covered writing. It is the art and science of hiding information such its presence cannot be detected and a communication is happening. A secrete information is encoding in a manner such that the very existence of the information is concealed. Paired with existing communication methods, steganography can be used to carry out hidden exchanges.

The main goal of this projects it to communicate securely in a completely undetectable manner and to avoid drawing suspicion to the transmission of a hider data. There has been a rapid growth of interest in steganography for two reasons:

The publishing and broadcasting industries have become interested in techniques for hiding encrypted copyright marks and serial numbers in digital films, audio recordings, books and multimedia products

Moves by various governments to restrict the availability of encryption services have motivated people to study methods by which private messages can be embedded in seemingly innocuous cover messages.

The basic model of steganography consists of Carrier, Message and password. Carrier is also known as cover-object, which the message is embedded and serves to hide the presence of the message.

Methodology:

Two types of Encryption are present in this project.

1. Hiding text into an Image and
2. Hiding Files into an Image.

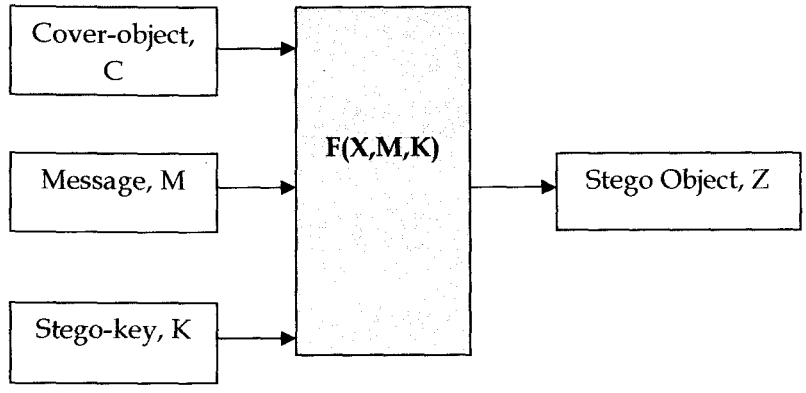
The first one is focus only on text based hiding. The user has three button i.e. Encrypt, Decrypt, Save. There is also a textbox to input the password, and the image preview picture box.

Steps:

1. File: is used to open the image the user want.
2. Text Area: is used to input the message to be encrypted.
3. Password: key to protect the encryption.
4. Encrypt: is used to encrypt and save the encrypted text and picture into a storage media.
5. Decrypt: is used to decrypt the encrypted image and text.

The Second one is focus in hiding files into an image i.e. .txt, .docx, .csv and so on. The user has two tab options — encrypt and decrypt. If user select encrypt, application give the screen to select image file, information file and option to save the image file. If user select decrypt, application gives the screen to select only image file and ask path where user want to save the secrete file. The Encryption Process is also protected with a Key (i.e. Password) for better security of the data and File. In encryption the secret information is hiding in with any type of image file. Decryption is getting the secret information from image file.

Basically, the model for steganography is shown on following figure:



Message is the data that the sender wishes to remain it confidential. It can be plain text, ciphertext, other image, or anything that can be embedded in a bit stream such as a copyright mark, a covert communication, or a serial number. Password is known as *stego-key*, which ensures that only recipient who know the corresponding decoding key will be able to extract the message from a *cover-object*. The *cover-object* with the secretly embedded message is then called the *Stego-object*.

Recovering message from a *stego-object* requires the *cover-object* itselt and a corresponding decoding key if a *stego-key* was used during the encoding process. The original image may or may not be required in most applications to extract the message.

There are several suitable carriers below to be the *cover-object:*

- Network protocols such as TCP, IP and UDP
- Audio that using digital audio formats such as wav, midi, avi, mpeg, mpi and voc

- File and Disk that can hides and append files by using the slack space
- Text such as null characters, just alike morse code including html and java
- Images file such as bmp, gif and jpg, where they can be both color and gray-scale.

In general, the information hiding process extracts redundant bits from *cover-object*. The process consists of two steps:

- Identification of redundant bits in a *cover-object*. Redundant bits are those bits that can be modified without corrupting the quality or destroying the integrity of the *cover-object*.
- Embedding process then selects the subset of the redundant bits to be replaced with data from a secret message. The *stego-object* is created by replacing the selected redundant bits with message bits

Steganography vs Cryptography:

Basically, the purpose of cryptography and steganography is to provide secret communication. However, steganography is not the same as cryptography. Cryptography hides the contents of a secrete message from a malicious people, whereas steganography even conceal the existence of the message. In cryptography, the system is broken when the attacker can read the secret message. Breaking a steganography system need the attacker to detect that steganography has been used.

It is possible to combine the techniques by encrypting message using cryptography and then hiding the encrypted message using steganography. The resulting stego-image can be transmitted without revealing that secret information is being exchanged.

Steganography Techniques:

Over the past few years, numerous steganography techniques that embed hidden messages in multimedia objects have been proposed. There have been many techniques for hiding information or messages in images in such a manner that alteration made to the image is perceptually indiscernible. Commonly approaches are include LSB, Masking and filtering and Transform techniques.

Least significant bit (LSB) insertion is a simple approach to embedding information in image file. The simplest steganography techniques embed the bits of the message directly into least significant bit plane of the cover-image in a deterministic sequence. Modulating the least significant bit does not result in human perceptible difference because the amplitude of the change is small. In this technique, the embedding capacity can be increased by using two or more least significant bits. At the same time, not only the risk of making the embedded message statistically detectable increase but also the image fidelity degrades.

Hence a variable size LSB embedding schema is presented, in which the number of LSBs used for message embedding/extracting depends on the local characteristics of the pixel. The advantage of LSB-based method is easy to implement and high message pay-load.

Although LSB hides the message in such way that the humans do not perceive it, it is still possible for the opponent to retrieve the message due to the simplicity of the technique. Therefore, malicious people can easily try to extract the message from the beginning of the image if they are suspicious that there exists secret information that was embedded in the image.

Therefore, a system named Secure Information Hiding System (SIHS) is proposed to improve the LSB scheme. It overcomes the sequence-mapping problem by embedding the massage into a set of random pixels, which are scattered on the cover-image.

Masking and filtering techniques, usually restricted to 24 bits and gray scale image, hide information by marking an image, in a manner similar to paper watermarks. The technique perform analysis of the image, thus embed the information in significant areas so that the hidden message is more integral to cover image than just hiding it in the noise level.

Transform techniques embed the message by modulating coefficient in a transform domain, such as the Discrete Fourier Transform, or Wavelet Transform. These methods hide messages in significant areas of the cover image, which make them more robust to attack. Transformations can be applied over the entire image, to block throughout the image, or other variant.

Image Steganography and bitmap pictures:

Using bitmap pictures for hiding secret information is one of most popular choices for Steganography. Many types of software built for this purpose, some of these software use password protection to encrypting information on picture. To use these software you must have a 'BMP' format of a pictures to use it, but using other type of pictures like "JPEG", "GIF" or any other types is rather or never used, because of algorithm of "BMP" pictures for Steganography is simple. Also we know that in the web most popular of image types are "JPEG" and other types not "BPM", so we should have a solution for this problem.

This software provide the solution of this problem, it can accept any type of image to hide information file, but finally it give the only "BMP" image as an output that has hidden file inside it.

Bitmap Steganography:

Bitmap type is the simplest type of picture because that it doesn' t have any technology for decreasing file size. Structure of these files is that a bitmap image created from pixels that any pixel created from three colors (red, green and blue said RGB) each color of a pixel is one byte information that shows the density of that color. Merging these three color makes every color that we see in these pictures. We know that every byte in computer science is created from 8 bit that first bit is Most-Significant-Bit (MSB) and last bit Least-Significant-Bit (LSB), the idea of using Steganography science is in this place; we use LSB bit for writing our security information inside BMP pictures. So if we just use last layer (8st layar) of information, we should change the last bit of pixels, in other hands we have 3 bits in each pixel so we have 3*height*width bits memory to write our information. But before writing our data we must write name of data(file), size of name of data & size of data. We can do this by assigning some first bits of memory (8st layer).

(00101101 00011101 11011100)

(10100110 11000101 00001100)

(11010010 10101100 01100011)

Using each 3 pixel of picture to save a byte of data

System Analysis & Design

Steganography system requires any type of image file and the information or message that is to be hidden. It has two modules encrypt and decrypt.

Microsoft .Net framework prepares a huge amount of tool and options for programmers that they simples programming. One of .Net tools for pictures and images is auto-converting most types of pictures to BMP format. I used this tool in this software called "Steganography" that is written in C#.Net language and you can use this software to hide your information in any type of pictures without any converting its format to BMP (software converts inside it).

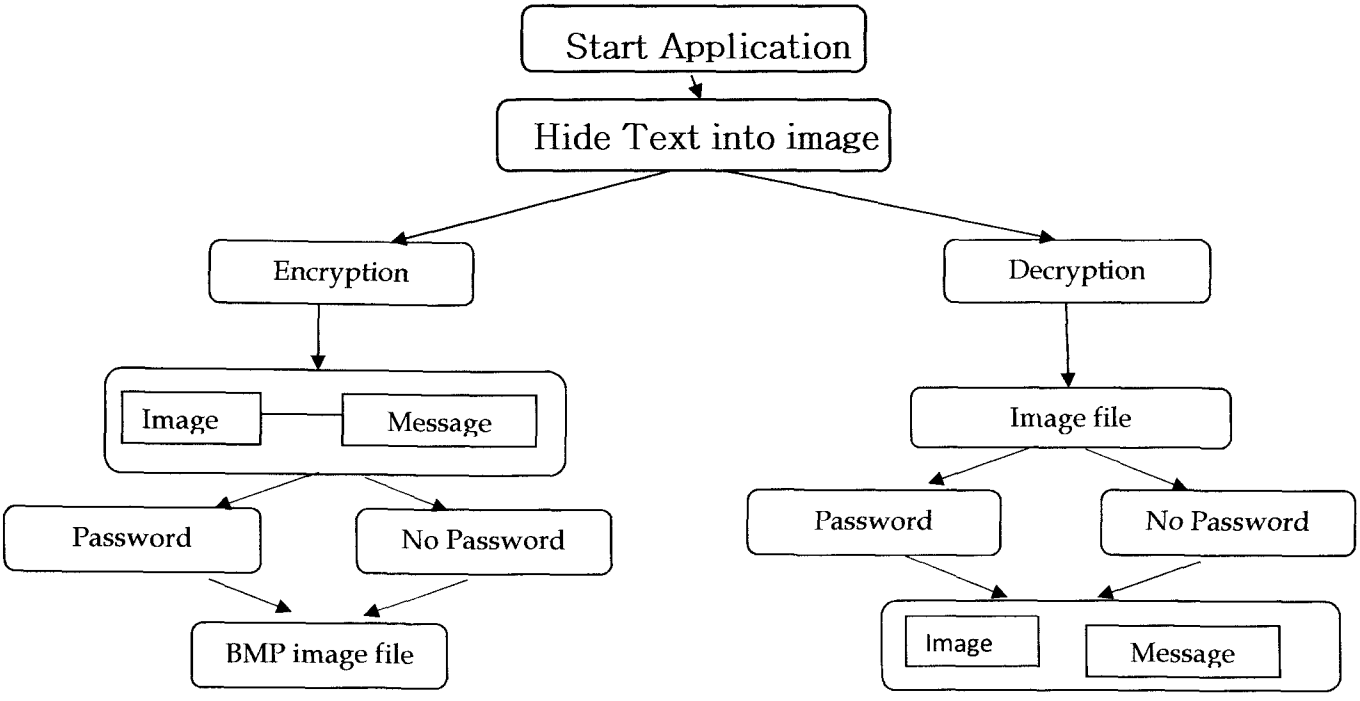
The algorithm used for Encryption and Decryption in this application provides using several layers lieu of using only LSB layer of image. Writing data starts from last layer (8st or LSB layer); because significant of this layer is least and every upper layer has doubled significant from its down layer. So every step we go to upper layer image quality decreases and image retouching transpires.

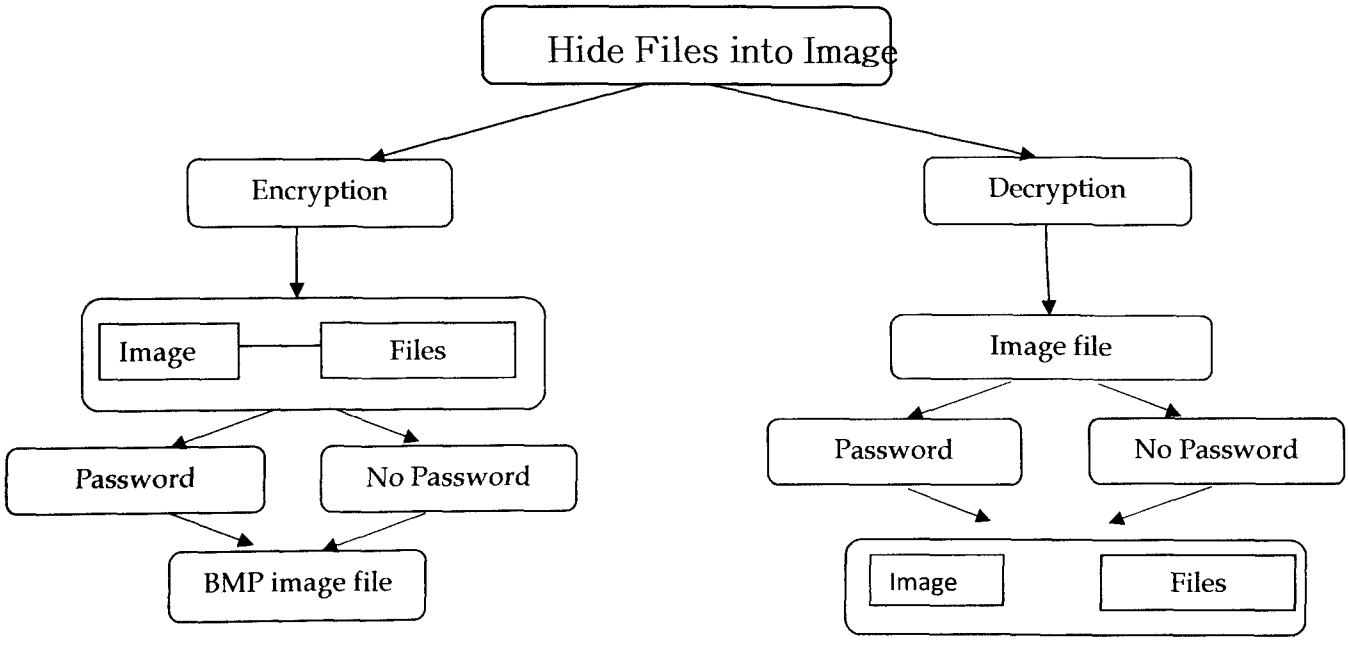
The encrypt module is used to hide information into the image; no one can see that information or file. This module requires any type of image and message and gives the only one image file in destination.

The decrypt module is used to get the hidden information in an image file. It take the image file as an output, and give two file at destination folder, one is the same image file and another is the message file that is hidden it that.

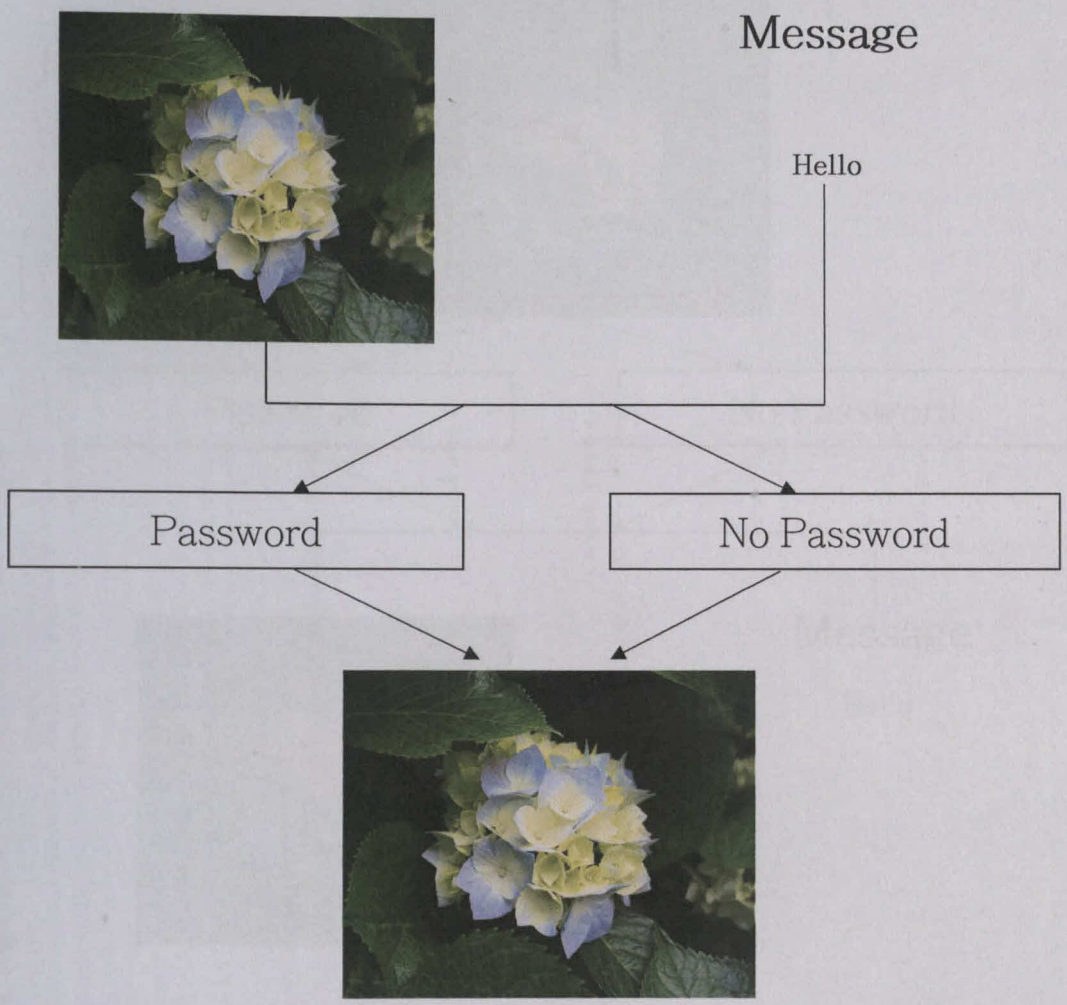
Before encrypting file inside image we must save name and size of file in a definite place of image. We could save file name before file information in LSB layer and save file size and file name size in most right-down pixels of image. Writing this information is needed to retrieve file from encrypted image in decryption state.

The graphical representation of this system is as follows:





Encryption Process in Hide Message into image



Decryption Process in Hide Message into image

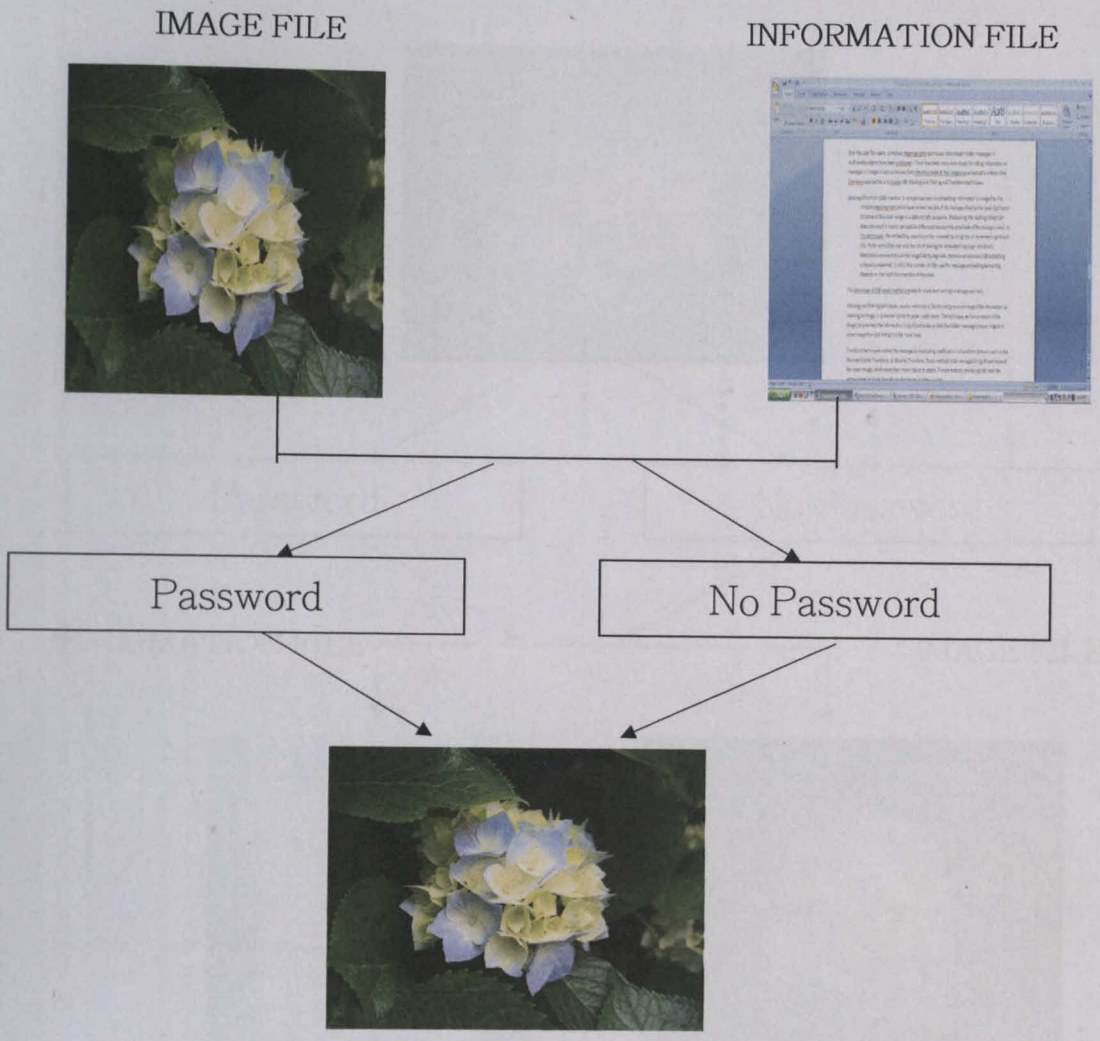
Password

No Password

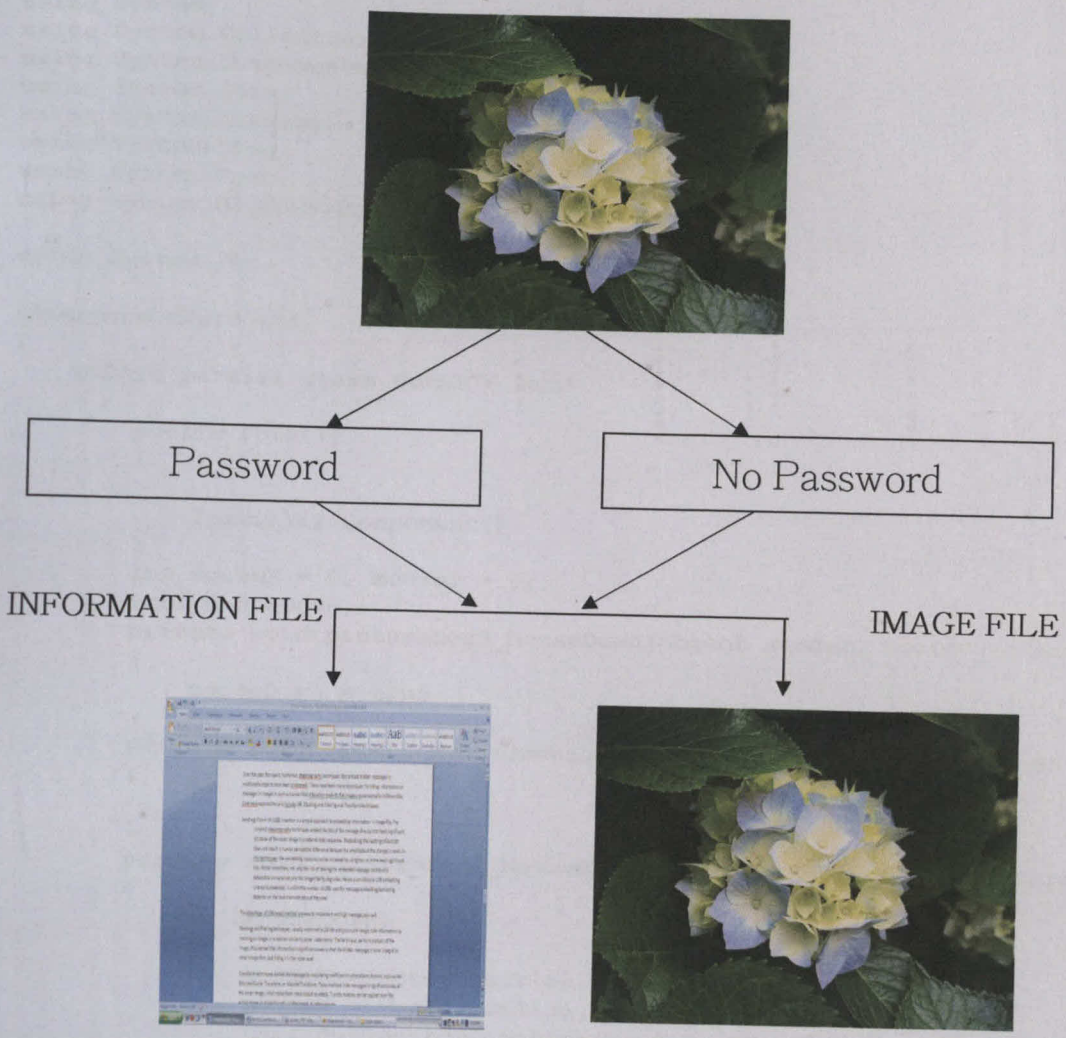
Message

Hello

Encryption Process in Hide Files into image



Decryption Process in Hide Files into image



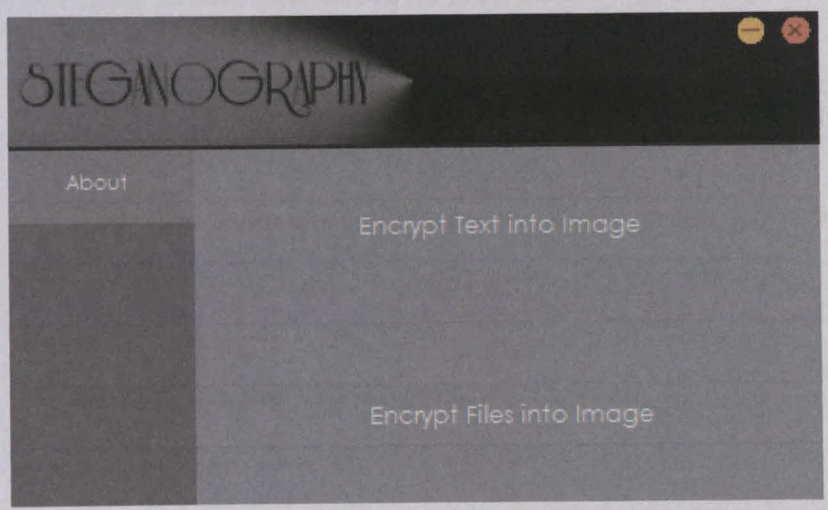
CODE ANALYSIS

Home Main Screen Code Analysis

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.IO;

namespace MYproject
{
    public partial class Form1 : Form
    {
        public Form1()
        {

            InitializeComponent();
        }
        int mouseX = 0, mouseY = 0;
        bool mouseDown;
        private void pictureBox3_MouseDown(object sender, MouseEventArgs e)
        {
            mouseDown = true;
        }
        private void panel1_MouseDown(object sender, MouseEventArgs e)
        {

        }
        private void pictureBox3_MouseMove(object sender, MouseEventArgs e)
        {
            if (mouseDown)
            {
                mouseX = MousePosition.X - 210;
                mouseY = MousePosition.Y - 10;
                this.SetDesktopLocation(mouseX, mouseY);
            }

        }
        private void pictureBox3_MouseUp(object sender, MouseEventArgs e)
        {
            mouseDown = false;
        }
        private void panel1_MouseMove(object sender, MouseEventArgs e)
        {

        }

        private void panel1_MouseUp(object sender, MouseEventArgs e)
        {

        }
```

```
private void button1_Click(object sender, EventArgs e)
{
    Texttoimageform t = new Texttoimageform();
    t.ShowDialog();


}
private void panel1_Paint(object sender, PaintEventArgs e)
{

}

private void button2_Click(object sender, EventArgs e)
{

}

private void pictureBox2_Click(object sender, EventArgs e)
{

}

private void Form1_Load(object sender, EventArgs e)
{

}

private void pictureBox1_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void pictureBox2_Click_1(object sender, EventArgs e)
{
    if (WindowState == FormWindowState.Normal)
    {
        WindowState = FormWindowState.Minimized;
    }
}

private void pictureBox3_Click(object sender, EventArgs e)
{

}

private void button2_Click_1(object sender, EventArgs e)
{

    FiletoimageForm ft = new FiletoimageForm();
    ft.ShowDialog();
}

private void button3_Click(object sender, EventArgs e)
{

}

private void pictureBox3_Click_1(object sender, EventArgs e)
{

}
```

```
        private void button5_Click(object sender, EventArgs e)
        {
            if (WindowState == FormWindowState.Normal)
            {
                WindowState = FormWindowState.Maximized;
            }
        }

        private void button4_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Files Hidden App");
        }

        private void pictureBox3_Click_2(object sender, EventArgs e)
        {

        }

    }
}
```



Hide text into image code Analysis

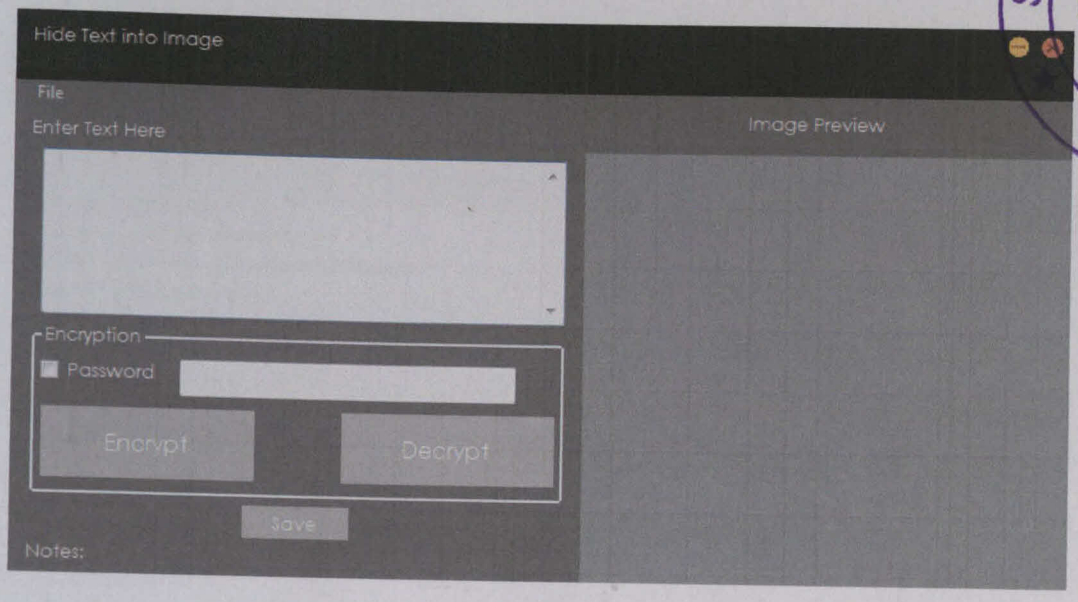
```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.IO;

namespace MYproject
{
    public partial class Texttoimageform : Form
    {
        private Bitmap bmp = null;
        private string extractedText = string.Empty;

        public Texttoimageform()
        {
            InitializeComponent();
        }

        private void hideButton_Click(object sender, EventArgs e)
        {
            bmp = (Bitmap)imagePictureBox.Image;

            string text = dataTextBox.Text;

            if (text.Equals(""))
            {
                MessageBox.Show("The text you want to hide can't be empty",
"Warning");

                return;
            }

            if (encryptCheckBox.Checked)
            {
                if (passwordTextBox.Text.Length < 6)
                {
                    MessageBox.Show("Please enter a password with at least
6 characters", "Warning");

                    return;
                }
                else
                {
                    text = Crypto.EncryptStringAES(text,
passwordTextBox.Text);
                }
                passwordTextBox.Text = "";
                dataTextBox.Text = "";
            }

            bmp = TexttoimageHelper.embedText(text, bmp);

            MessageBox.Show("Your text was hidden in the image
successfully!", "Done");

            notesLabel.Text = "Notes: don't forget to save your new
image.";
```

```
            notesLabel.ForeColor = Color.White;
        }

        private void extractButton_Click(object sender, EventArgs e)
        {
            bmp = (Bitmap)imagePictureBox.Image;

            string extractedText = TexttoimageHelper.extractText(bmp);

            if (encryptCheckBox.Checked)
            {
                try
                {
                    extractedText = Crypto.DecryptStringAES(extractedText,
passwordTextBox.Text);
                }
                catch
                {
                    MessageBox.Show("Wrong password", "Error");

                    return;
                }
            }

            dataTextBox.Text = extractedText;
        }

        private void imageToolStripMenuItem1_Click(object sender, EventArgs
e)
        {
            OpenFileDialog open_dialog = new OpenFileDialog();
            open_dialog.Filter = "Image Files (*.jpeg; *.png; *.bmp)|*.jpg;
*.png; *.bmp";

            if (open_dialog.ShowDialog() == DialogResult.OK)
            {
                imagePictureBox.Image =
Image.FromFile(open_dialog.FileName);
            }
        }

        private void imageToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            SaveFileDialog save_dialog = new SaveFileDialog();
            save_dialog.Filter = "Png Image|*.png|Bitmap Image|*.bmp";

            if (save_dialog.ShowDialog() == DialogResult.OK)
            {
                switch (save_dialog.FilterIndex)
                {
                    case 0:
                        {
                            bmp.Save(save_dialog.FileName,
ImageFormat.Png);
                        } break;
                    case 1:
                        {
                            bmp.Save(save_dialog.FileName,
ImageFormat.Bmp);
                        } break;
```

```
            }

            notesLabel.Text = "Notes:";
            notesLabel.ForeColor = Color.White;
        }
    }

    private void textToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        SaveFileDialog save_dialog = new SaveFileDialog();
        save_dialog.Filter = "Text Files|*.txt";

        if (save_dialog.ShowDialog() == DialogResult.OK)
        {
            File.WriteAllText(save_dialog.FileName, dataTextBox.Text);
        }
    }

    private void textToolStripMenuItem1_Click(object sender, EventArgs
e)
    {
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Filter = "Text Files|*.txt";

        if (open_dialog.ShowDialog() == DialogResult.OK)
        {
            dataTextBox.Text = File.ReadAllText(open_dialog.FileName);
        }
    }



    private void Texttoimageform_Load(object sender, EventArgs e)
    {

    }

    private void imagePictureBox_Click(object sender, EventArgs e)
    {

    }

    private void button1_Click(object sender, EventArgs e)
    {
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Filter = "Image Files (*.jpeg; *.png; *.bmp)|*.jpg;
*.png; *.bmp";

        if (open_dialog.ShowDialog() == DialogResult.OK)
        {
            imagePictureBox.Image =
Image.FromFile(open_dialog.FileName);
        }
    }
    int mouseX = 0, mouseY = 0;
    bool mouseDown;

    private void panel1_Paint(object sender, PaintEventArgs e)
    {
```

```
        }

        private void panel1_MouseDown(object sender, MouseEventArgs e)
        {
            mouseDown = true;
        }

        private void panel1_MouseMove(object sender, MouseEventArgs e)
        {
            if (mouseDown)
            {
                mouseX = MousePosition.X - 370;
                mouseY = MousePosition.Y - 20;
                this.SetDesktopLocation(mouseX, mouseY);
            }
        }

        private void panel1_MouseUp(object sender, MouseEventArgs e)
        {
            mouseDown = false;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            SaveFileDialog save_dialog = new SaveFileDialog();
            save_dialog.Filter = "Png Image|*.png|Bitmap Image|*.bmp";

            if (save_dialog.ShowDialog() == DialogResult.OK)
            {
                switch (save_dialog.FilterIndex)
                {
                    case 0:
                        {
                            bmp.Save(save_dialog.FileName,
ImageFormat.Png);
                        } break;
                    case 1:
                        {
                            bmp.Save(save_dialog.FileName,
ImageFormat.Bmp);
                        } break;
                }

                notesLabel.Text = "Notes:";
                notesLabel.ForeColor = Color.White;
            }
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {
            this.Close();

        }

        private void pictureBox2_Click(object sender, EventArgs e)
        {
            if (WindowState == FormWindowState.Normal)
            {
                WindowState = FormWindowState.Minimized;
            }
        }
```

```
        private void notesLabel_Click(object sender, EventArgs e)
        {

        }

        private void passwordTextBox_TextChanged(object sender, EventArgs
e)
        {

        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void encryptCheckBox_CheckedChanged(object sender,
EventArgs e)
        {

        }

        private void dataTextBox_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void openToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            OpenFileDialog open_dialog = new OpenFileDialog();
            open_dialog.Filter = "Image Files (*.jpeg; *.png; *.bmp)|*.jpg;
*.png; *.bmp";

            if (open_dialog.ShowDialog() == DialogResult.OK)
            {
                imagePictureBox.Image =
Image.FromFile(open_dialog.FileName);
            }
        }
    }
}
```



Files to image code analysis

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.IO;

namespace MYproject
{

    public partial class FiletoimageForm : Form
    {
        public FiletoimageForm()
        {
            InitializeComponent();
        }

        //public values:
        string loadedTrueImagePath, loadedFilePath, saveToImage,
DLoadImagePath, DSaveFilePath;
        int height, width;
        long fileSize, fileNameSize;
        Image loadedTrueImage, DecryptedImage, AfterEncryption;
        Bitmap loadedTrueBitmap, DecryptedBitmap;
        Rectangle previewImage = new Rectangle(380, 75, 363, 295);
        bool canPaint = false, EncriptionDone = false;
        byte[] fileContainer;
        private Bitmap bmp = null;

        private void panel5_Paint(object sender, PaintEventArgs e)
        {
        }
        private void EnImageBrowse_btn_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                loadedTrueImagePath = openFileDialog1.FileName;
                EnImage_tbx.Text = loadedTrueImagePath;
                loadedTrueImage = Image.FromFile(loadedTrueImagePath);
                height = loadedTrueImage.Height;
                width = loadedTrueImage.Width;
                loadedTrueBitmap = new Bitmap(loadedTrueImage);

                FileInfo imginf = new FileInfo(loadedTrueImagePath);
                float fs = (float)imginf.Length / 1024;
                ImageSize_lbl.Text = smalldecimal(fs.ToString(), 2) + "
KB";
                ImageHeight_lbl.Text = loadedTrueImage.Height.ToString() +
" Pixel";
                ImageWidth_lbl.Text = loadedTrueImage.Width.ToString() + "
Pixel";
                double cansave = (8.0 * ((height * (width / 3) * 3) / 3 -
1)) / 1024;
                CanSave_lbl.Text = smalldecimal(cansave.ToString(), 2) + "
KB";

                canPaint = true;
```

```
            this.Invalidate();
        }
    }

    private string smalldecimal(string inp, int dec)
    {
        int i;
        for (i = inp.Length - 1; i > 0; i--)
            if (inp[i] == '.')
                break;
        try
        {
            return inp.Substring(0, i + dec + 1);
        }
        catch
        {
            return inp;
        }
    }

    private void EnFileBrowse_btn_Click(object sender, EventArgs e)
    {
        if (openFileDialog2.ShowDialog() == DialogResult.OK)
        {
            loadedFilePath = openFileDialog2.FileName;
            EnFile_tbx.Text = loadedFilePath;
            FileInfo finfo = new FileInfo(loadedFilePath);
            fileSize = finfo.Length;
            fileNameSize = justFName(loadedFilePath).Length;
        }
    }
    String a;
    private void Encrypt_btn_Click(object sender, EventArgs e)
    {
        String text = PasswordText.Text;

        if (checkBox1.Checked)
        {
            if (PasswordText.Text.Length < 6)
            {
                MessageBox.Show("Please enter a password with at least
6 characters", "Warning");

                return;
            }
            else
            {
                text = Crypto.EncryptStringAES(text,
PasswordText.Text);
            }


            a = text;
            PasswordText.Text = "" ;
        }


     if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            saveToImage = saveFileDialog1.FileName;
        }
```

```
            else
                return;
            if (EnImage_tbx.Text == String.Empty || EnFile_tbx.Text ==
String.Empty)
            {
                MessageBox.Show("Encrypton information is
incomplete!\nPlease complete them frist.", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            if (8 * ((height * (width / 3) * 3) / 3 - 1) < fileSize +
fileNameSize)
            {
                MessageBox.Show("File size is too large!\nPlease use a
larger image to hide this file.", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }
            fileContainer = File.ReadAllBytes(loadedFilePath);
            EncryptLayer();
        }


        private void EncryptLayer()
        {
            toolStripStatusLabel1.Text = "Encrypting... Please wait";
            Application.DoEvents();
            long FSize = fileSize;
            Bitmap changedBitmap = EncryptLayer(8, loadedTrueBitmap, 0,
(height * (width / 3) * 3) / 3 - fileNameSize - 1, true);
            FSize -= (height * (width / 3) * 3) / 3 - fileNameSize - 1;
            if (FSize > 0)
            {
                for (int i = 7; i >= 0 && FSize > 0; i--)
                {
                    changedBitmap = EncryptLayer(i, changedBitmap, (((8 -
i) * height * (width / 3) * 3) / 3 - fileNameSize - (8 - i)), (((9 - i) *
height * (width / 3) * 3) / 3 - fileNameSize - (9 - i)), false);
                    FSize -= (height * (width / 3) * 3) / 3 - 1;
                }
            }
            changedBitmap.Save(saveToImage);
            toolStripStatusLabel1.Text = "Encrypted image has been
successfully saved.";
            EncriptionDone = true;
            AfterEncryption = Image.FromFile(saveToImage);
            this.Invalidate();
        }


        private Bitmap EncryptLayer(int layer, Bitmap inputBitmap, long
startPosition, long endPosition, bool writeFileName)
        {
            Bitmap outputBitmap = inputBitmap;
            layer--;
            int i = 0, j = 0;
            long FNSize = 0;
            bool[] t = new bool[8];
            bool[] rb = new bool[8];
            bool[] gb = new bool[8];
            bool[] bb = new bool[8];
            Color pixel = new Color();
```

```
            byte r, g, b;

            if (writeFileName)
            {
                FNSize = fileNameSize;
                string fileName = justFName(loadedFilePath);

                //write fileName:
                for (i = 0; i < height && i * (height / 3) < fileNameSize;
i++)
                    for (j = 0; j < (width / 3) * 3 && i * (height / 3) +
(j / 3) < fileNameSize; j++)
                    {
                        byte2bool((byte)fileName[i * (height / 3) + j / 3],
ref t);
                        pixel = inputBitmap.GetPixel(j, i);
                        r = pixel.R;
                        g = pixel.G;
                        b = pixel.B;
                        byte2bool(r, ref rb);
                        byte2bool(g, ref gb);
                        byte2bool(b, ref bb);
                        if (j % 3 == 0)
                        {
                            rb[7] = t[0];
                            gb[7] = t[1];
                            bb[7] = t[2];
                        }
                        else if (j % 3 == 1)
                        {
                            rb[7] = t[3];
                            gb[7] = t[4];
                            bb[7] = t[5];
                        }
                        else
                        {
                            rb[7] = t[6];
                            gb[7] = t[7];
                        }
                        Color result = Color.FromArgb((int)bool2byte(rb),
(int)bool2byte(gb), (int)bool2byte(bb));
                        outputBitmap.SetPixel(j, i, result);
                    }
                i--;
            }
            //write file (after file name):
            int tempj = j;

            for (; i < height && i * (height / 3) < endPosition -
startPosition + FNSize && startPosition + i * (height / 3) < fileSize +
FNSize; i++)
                for (j = 0; j < (width / 3) * 3 && i * (height / 3) + (j /
3) < endPosition - startPosition + FNSize && startPosition + i * (height /
3) + (j / 3) < fileSize + FNSize; j++)
                {
                    if (tempj != 0)
                    {
                        j = tempj;
                        tempj = 0;
                    }
```

```
                    byte2bool((byte)fileContainer[startPosition + i *
(height / 3) + j / 3 - FNSize], ref t);
                    pixel = inputBitmap.GetPixel(j, i);
                    r = pixel.R;
                    g = pixel.G;
                    b = pixel.B;
                    byte2bool(r, ref rb);
                    byte2bool(g, ref gb);
                    byte2bool(b, ref bb);
                    if (j % 3 == 0)
                    {
                        rb[layer] = t[0];
                        gb[layer] = t[1];
                        bb[layer] = t[2];
                    }
                    else if (j % 3 == 1)
                    {
                        rb[layer] = t[3];
                        gb[layer] = t[4];
                        bb[layer] = t[5];
                    }
                    else
                    {
                        rb[layer] = t[6];
                        gb[layer] = t[7];
                    }
                    Color result = Color.FromArgb((int)bool2byte(rb),
(int)bool2byte(gb), (int)bool2byte(bb));
                    outputBitmap.SetPixel(j, i, result);

                }
            long tempFS = fileSize, tempFNS = fileNameSize;
            r = (byte)(tempFS % 100);
            tempFS /= 100;
            g = (byte)(tempFS % 100);
            tempFS /= 100;
            b = (byte)(tempFS % 100);
            Color flenColor = Color.FromArgb(r, g, b);
            outputBitmap.SetPixel(width - 1, height - 1, flenColor);

            r = (byte)(tempFNS % 100);
            tempFNS /= 100;
            g = (byte)(tempFNS % 100);
            tempFNS /= 100;
            b = (byte)(tempFNS % 100);
            Color fnlenColor = Color.FromArgb(r, g, b);
            outputBitmap.SetPixel(width - 2, height - 1, fnlenColor);

            return outputBitmap;
        }


        private void DecryptLayer()
        {
            toolStripStatusLabel1.Text = "Decrypting... Please wait";
            Application.DoEvents();
            int i, j = 0;
            bool[] t = new bool[8];
            bool[] rb = new bool[8];
            bool[] gb = new bool[8];
            bool[] bb = new bool[8];
```

```
            Color pixel = new Color();
            byte r, g, b;
            pixel = DecryptedBitmap.GetPixel(width - 1, height - 1);
            long fSize = pixel.R + pixel.G * 100 + pixel.B * 10000;
            pixel = DecryptedBitmap.GetPixel(width - 2, height - 1);
            long fNameSize = pixel.R + pixel.G * 100 + pixel.B * 10000;
            byte[] res = new byte[fSize];
            string resFName = "";
            byte temp;

            //Read file name:
            for (i = 0; i < height && i * (height / 3) < fNameSize; i++)
                for (j = 0; j < (width / 3) * 3 && i * (height / 3) + (j /
3) < fNameSize; j++)
                {
                    pixel = DecryptedBitmap.GetPixel(j, i);
                    r = pixel.R;
                    g = pixel.G;
                    b = pixel.B;
                    byte2bool(r, ref rb);
                    byte2bool(g, ref gb);
                    byte2bool(b, ref bb);
                    if (j % 3 == 0)
                    {
                        t[0] = rb[7];
                        t[1] = gb[7];
                        t[2] = bb[7];
                    }
                    else if (j % 3 == 1)
                    {
                        t[3] = rb[7];
                        t[4] = gb[7];
                        t[5] = bb[7];
                    }
                    else
                    {
                        t[6] = rb[7];
                        t[7] = gb[7];
                        temp = bool2byte(t);
                        resFName += (char)temp;
                    }
                }

            //Read file on layer 8 (after file name):
            int tempj = j;
            i--;

            for (; i < height && i * (height / 3) < fSize + fNameSize; i++)
                for (j = 0; j < (width / 3) * 3 && i * (height / 3) + (j /
3) < (height * (width / 3) * 3) / 3 - 1 && i * (height / 3) + (j / 3) <
fSize + fNameSize; j++)
                {
                    if (tempj != 0)
                    {
                        j = tempj;
                        tempj = 0;
                    }
                    pixel = DecryptedBitmap.GetPixel(j, i);
                    r = pixel.R;
                    g = pixel.G;
                    b = pixel.B;
```

```
                    byte2bool(r, ref rb);
                    byte2bool(g, ref gb);
                    byte2bool(b, ref bb);
                    if (j % 3 == 0)
                    {
                        t[0] = rb[7];
                        t[1] = gb[7];
                        t[2] = bb[7];
                    }
                    else if (j % 3 == 1)
                    {
                        t[3] = rb[7];
                        t[4] = gb[7];
                        t[5] = bb[7];
                    }
                    else
                    {
                        t[6] = rb[7];
                        t[7] = gb[7];
                        temp = bool2byte(t);
                        res[i * (height / 3) + j / 3 - fNameSize] = temp;
                    }
                }

            //Read file on other layers:
            long readedOnL8 = (height * (width / 3) * 3) / 3 - fNameSize -
1;

            for (int layer = 6; layer >= 0 && readedOnL8 + (6 - layer) *
((height * (width / 3) * 3) / 3 - 1) < fSize; layer--)
                for (i = 0; i < height && i * (height / 3) + readedOnL8 +
(6 - layer) * ((height * (width / 3) * 3) / 3 - 1) < fSize; i++)
                    for (j = 0; j < (width / 3) * 3 && i * (height / 3) +
(j / 3) + readedOnL8 + (6 - layer) * ((height * (width / 3) * 3) / 3 - 1) <
fSize; j++)
                    {
                        pixel = DecryptedBitmap.GetPixel(j, i);
                        r = pixel.R;
                        g = pixel.G;
                        b = pixel.B;
                        byte2bool(r, ref rb);
                        byte2bool(g, ref gb);
                        byte2bool(b, ref bb);
                        if (j % 3 == 0)
                        {
                            t[0] = rb[layer];
                            t[1] = gb[layer];
                            t[2] = bb[layer];
                        }
                        else if (j % 3 == 1)
                        {
                            t[3] = rb[layer];
                            t[4] = gb[layer];
                            t[5] = bb[layer];
                        }
                        else
                        {
                            t[6] = rb[layer];
                            t[7] = gb[layer];
                            temp = bool2byte(t);
```

```
                            res[i * (height / 3) + j / 3 + (6 - layer) *
((height * (width / 3) * 3) / 3 - 1) + readedOnL8] = temp;
                        }
                    }

            if (File.Exists(DSaveFilePath + "\\" + resFName))
            {
                MessageBox.Show("File \"" + resFName + "\" already exist
please choose another path to save file", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }
            else
                File.WriteAllBytes(DSaveFilePath + "\\" + resFName, res);
            toolStripStatusLabel1.Text = "Decrypted file has been
successfully saved.";
            Application.DoEvents();
        }

        private void byte2bool(byte inp, ref bool[] outp)
        {
            if (inp >= 0 && inp <= 255)
                for (short i = 7; i >= 0; i--)
                {
                    if (inp % 2 == 1)
                        outp[i] = true;
                    else
                        outp[i] = false;
                    inp /= 2;
                }
            else
                throw new Exception("Input number is illegal.");
        }

        private byte bool2byte(bool[] inp)
        {
            byte outp = 0;
            for (short i = 7; i >= 0; i--)
            {
                if (inp[i])
                    outp += (byte)Math.Pow(2.0, (double)(7 - i));
            }
            return outp;
        }

        private void Decrypt_btn_Click(object sender, EventArgs e)
        {
            String extractedText = a;
            if (checkBox1.Checked)
            {
                try
                {
                    extractedText = Crypto.DecryptStringAES(extractedText,
PasswordText.Text);
                }
                catch
                {
                    MessageBox.Show("Wrong password", "Error");

                    return;
                }
```

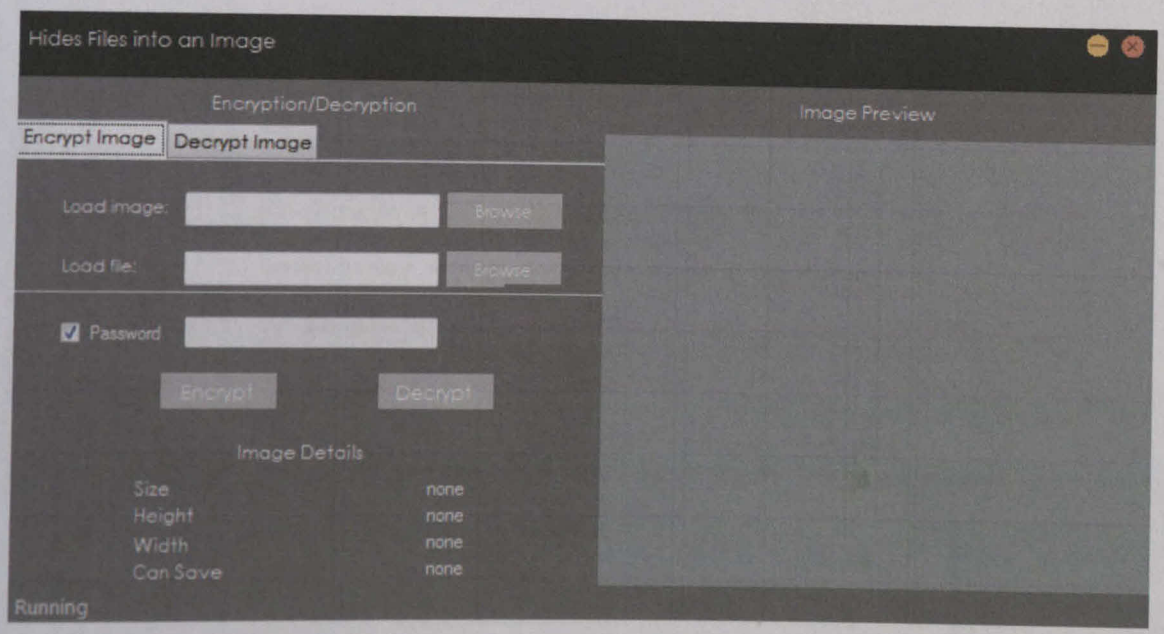
```
            }


            if (DeSaveFile_tbx.Text == String.Empty || DeLoadImage_tbx.Text
== String.Empty)
            {
                MessageBox.Show("Text boxes must not be empty!", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                return;
            }

            if (System.IO.File.Exists(DeLoadImage_tbx.Text) == false)
            {
                MessageBox.Show("Select image file.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                DeLoadImage_tbx.Focus();
                return;
            }



            DecryptLayer();
        }

        private void DeLoadImageBrowse_btn_Click(object sender, EventArgs
e)
        {
            if (openFileDialog3.ShowDialog() == DialogResult.OK)
            {
                DLoadImagePath = openFileDialog3.FileName;
                DeLoadImage_tbx.Text = DLoadImagePath;
                DecryptedImage = Image.FromFile(DLoadImagePath);
                height = DecryptedImage.Height;
                width = DecryptedImage.Width;
                DecryptedBitmap = new Bitmap(DecryptedImage);

                FileInfo imginf = new FileInfo(DLoadImagePath);
                float fs = (float)imginf.Length / 1024;
                ImageSize_lbl.Text = smalldecimal(fs.ToString(), 2) + "
KB";
                ImageHeight_lbl.Text = DecryptedImage.Height.ToString() + "
Pixel";
                ImageWidth_lbl.Text = DecryptedImage.Width.ToString() + "
Pixel";
                double cansave = (8.0 * ((height * (width / 3) * 3) / 3 -
1)) / 1024;
                CanSave_lbl.Text = smalldecimal(cansave.ToString(), 2) + "
KB";

                canPaint = true;
                this.Invalidate();
            }
        }

        private void DeSaveFileBrowse_btn_Click(object sender, EventArgs e)
        {
```

```
            if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
            {
                DSaveFilePath = folderBrowserDialog1.SelectedPath;
                DeSaveFile_tbx.Text = DSaveFilePath;
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            if (canPaint)
                try
                {
                    if (!EncriptionDone)
                        e.Graphics.DrawImage(loadedTrueImage,
previewImage);
                    else
                        e.Graphics.DrawImage(AfterEncryption,
previewImage);
                }
                catch
                {
                    e.Graphics.DrawImage(DecryptedImage, previewImage);
                }
        }

        private string justFName(string path)
        {
            string output;
            int i;
            if (path.Length == 3)    // i.e: "C:\\"
                return path.Substring(0, 1);
            for (i = path.Length - 1; i > 0; i--)
                if (path[i] == '\\')
                    break;
            output = path.Substring(i + 1);
            return output;
        }


        private string justEx(string fName)
        {
            string output;
            int i;
            for (i = fName.Length - 1; i > 0; i--)
                if (fName[i] == '.')
                    break;
            output = fName.Substring(i + 1);
            return output;
        }

        private void Close_btn_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void linkLabel1_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
        {
            System.Diagnostics.Process.Start("Design By Banlam");

        }
```

```
        private void FrmSteganography_Load(object sender, EventArgs e)
        {

        }

        private void label3_Click(object sender, EventArgs e)
        {

        }

        private void statusStrip1_ItemClicked(object sender,
ToolStripItemClickedEventArgs e)
        {

        }


        private void button1_Click_1(object sender, EventArgs e)
        {
            MessageBox.Show("Files Hidden Windows Application");
        }

        private void button2_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Load Image : Load any image you want(eg. jpg,
bmp, png). \nLoad Files : Load a files you want to hide into the selected
image(eg .txt,.doc,.csv,.exe etc)");
        }

        private void ImageWidth_lbl_Click(object sender, EventArgs e)
        {

        }

        private void CanSave_lbl_Click(object sender, EventArgs e)
        {

        }

        private void label9_Click(object sender, EventArgs e)
        {

        }

        private void label2_Click(object sender, EventArgs e)
        {

        }

        private void label8_Click(object sender, EventArgs e)
        {

        }

        private void label12_Click(object sender, EventArgs e)
        {

        }

        private void pictureBox1_Click(object sender, EventArgs e)
```

```
{
    this.Close();

}

private void pictureBox3_Click(object sender, EventArgs e)
{
    if(WindowState == FormWindowState.Normal)
    {
        WindowState = FormWindowState.Minimized;
    }
}
int mouseX = 0, mouseY = 0;
bool mouseDown;
private void panel1_MouseDown(object sender, MouseEventArgs e)
{
    mouseDown = true;
}

private void panel1_MouseMove(object sender, MouseEventArgs e)
{
    if (mouseDown)
    {
        mouseX = MousePosition.X - 370;
        mouseY = MousePosition.Y - 10;
        this.SetDesktopLocation(mouseX, mouseY);
    }
}

private void panel1_MouseUp(object sender, MouseEventArgs e)
{
    mouseDown = false;
}

private void panel1_Paint(object sender, PaintEventArgs e)
{

}

private void EnImage_tbx_TextChanged(object sender, EventArgs e)
{

}

private void tabPage1_Click(object sender, EventArgs e)
{

}

private void label8_Click_1(object sender, EventArgs e)
{

}

private void panel2_Paint(object sender, PaintEventArgs e)
{

}
```

```
        private void EnFile_tbx_TextChanged(object sender, EventArgs e)
        {

        }

        private void label10_Click(object sender, EventArgs e)
        {

        }

        private void label7_Click(object sender, EventArgs e)
        {

        }

        private void ImageSize_lbl_Click(object sender, EventArgs e)
        {

        }

        private void PasswordText_TextChanged(object sender, EventArgs e)
        {

        }

        private void passwordTextBox_TextChanged(object sender, EventArgs
e)
        {

        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {

        }

    }
}
```



Cryptoraphy Code

```
using System;
using System.Text;
using System.Security.Cryptography;
using System.IO;


namespace MYproject
{
    public class Crypto
    {
        private static byte[] _salt =
Encoding.ASCII.GetBytes("jasdh7834y8hfeur73rsharks214");

        /// <summary>
        /// Encrypt the given string using AES.  The string can be
decrypted using
        /// DecryptStringAES().  The sharedSecret parameters must match.
        /// </summary>
        /// <param name="plainText">The text to encrypt.</param>
        /// <param name="sharedSecret">A password used to generate a key
for encryption.</param>
        public static string EncryptStringAES(string plainText, string
sharedSecret)
        {
            if (string.IsNullOrEmpty(plainText))
                throw new ArgumentNullException("plainText");
            if (string.IsNullOrEmpty(sharedSecret))
                throw new ArgumentNullException("sharedSecret");

            string outStr = null;                       // Encrypted string
to return
            RijndaelManaged aesAlg = null;              // RijndaelManaged
object used to encrypt the data.

            try
            {
                // generate the key from the shared secret and the salt
                Rfc2898DeriveBytes key = new
Rfc2898DeriveBytes(sharedSecret, _salt);

                // Create a RijndaelManaged object
                aesAlg = new RijndaelManaged();
                aesAlg.Key = key.GetBytes(aesAlg.KeySize / 8);

                // Create a decryptor to perform the stream transform.
                ICryptoTransform encryptor =
aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

                // Create the streams used for encryption.
                using (MemoryStream msEncrypt = new MemoryStream())
                {
                    // prepend the IV

msEncrypt.Write(BitConverter.GetBytes(aesAlg.IV.Length), 0, sizeof(int));
                    msEncrypt.Write(aesAlg.IV, 0, aesAlg.IV.Length);
```

```
                    using (CryptoStream csEncrypt = new
CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
                    {
                        using (StreamWriter swEncrypt = new
StreamWriter(csEncrypt))
                        {
                            //Write all data to the stream.
                            swEncrypt.Write(plainText);
                        }
                    }
                    outStr = Convert.ToBase64String(msEncrypt.ToArray());
                }
            }
            finally
            {
                // Clear the RijndaelManaged object.
                if (aesAlg != null)
                    aesAlg.Clear();
            }

            // Return the encrypted bytes from the memory stream.
            return outStr;
        }

        /// <summary>
        /// Decrypt the given string.  Assumes the string was encrypted
using
        /// EncryptStringAES(), using an identical sharedSecret.
        /// </summary>
        /// <param name="cipherText">The text to decrypt.</param>
        /// <param name="sharedSecret">A password used to generate a key
for decryption.</param>
        public static string DecryptStringAES(string cipherText, string
sharedSecret)
        {
            if (string.IsNullOrEmpty(cipherText))
                throw new ArgumentNullException("cipherText");
            if (string.IsNullOrEmpty(sharedSecret))
                throw new ArgumentNullException("sharedSecret");

            // Declare the RijndaelManaged object
            // used to decrypt the data.
            RijndaelManaged aesAlg = null;

            // Declare the string used to hold
            // the decrypted text.
            string plaintext = null;

            try
            {
                // generate the key from the shared secret and the salt
                Rfc2898DeriveBytes key = new
Rfc2898DeriveBytes(sharedSecret, _salt);

                // Create the streams used for decryption.
                byte[] bytes = Convert.FromBase64String(cipherText);
                using (MemoryStream msDecrypt = new MemoryStream(bytes))
                {
                    // Create a RijndaelManaged object
                    // with the specified key and IV.
                    aesAlg = new RijndaelManaged();
```

```
                    aesAlg.Key = key.GetBytes(aesAlg.KeySize / 8);
                    // Get the initialization vector from the encrypted
stream
                    aesAlg.IV = ReadByteArray(msDecrypt);
                    // Create a decrytor to perform the stream transform.
                    ICryptoTransform decryptor =
aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);
                    using (CryptoStream csDecrypt = new
CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))
                    {
                        using (StreamReader srDecrypt = new
StreamReader(csDecrypt))

                            // Read the decrypted bytes from the decrypting
stream
                            // and place them in a string.
                            plaintext = srDecrypt.ReadToEnd();
                    }
                }
            }
            finally
            {
                // Clear the RijndaelManaged object.
                if (aesAlg != null)
                    aesAlg.Clear();
            }

            return plaintext;
        }

        private static byte[] ReadByteArray(Stream s)
        {
            byte[] rawLength = new byte[sizeof(int)];
            if (s.Read(rawLength, 0, rawLength.Length) != rawLength.Length)
            {
                throw new SystemException("Stream did not contain properly
formatted byte array");
            }

            byte[] buffer = new byte[BitConverter.ToInt32(rawLength, 0)];
            if (s.Read(buffer, 0, buffer.Length) != buffer.Length)
            {
                throw new SystemException("Did not read byte array
properly");
            }

            return buffer;
        }
    }
}
```

Texttoimage Helper Code

```
using System;
using System.Drawing;

namespace MYproject
{
    class TexttoimageHelper
    {
```

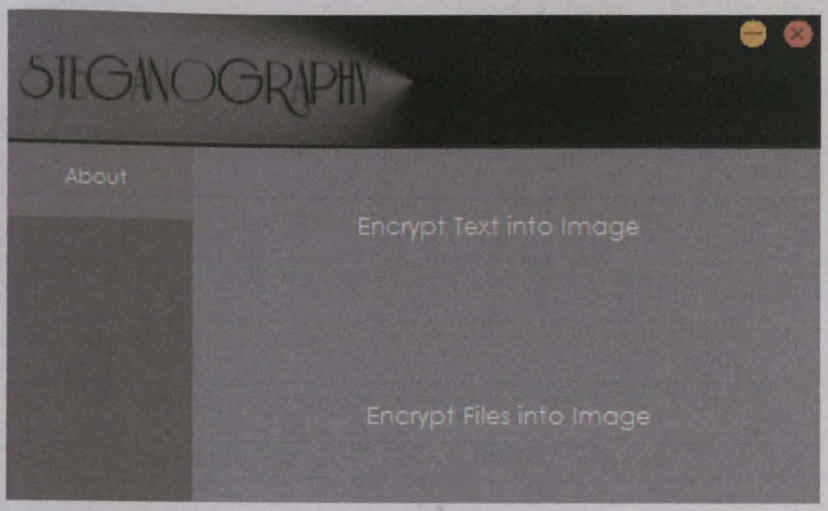
```
        public enum State
        {
            Hiding,
            Filling_With_Zeros
        };

        public static Bitmap embedText(string text, Bitmap bmp)
        {
            // initially, we'll be hiding characters in the image
            State state = State.Hiding;

            // holds the index of the character that is being hidden
            int charIndex = 0;

            // holds the value of the character converted to integer
            int charValue = 0;

            // holds the index of the color element (R or G or B) that is
currently being processed
            long pixelElementIndex = 0;

            // holds the number of trailing zeros that have been added when
finishing the process
            int zeros = 0;

            // hold pixel elements
            int R = 0, G = 0, B = 0;

            // pass through the rows
            for (int i = 0; i < bmp.Height; i++)
            {
                // pass through each row
                for (int j = 0; j < bmp.Width; j++)
                {
                    // holds the pixel that is currently being processed
                    Color pixel = bmp.GetPixel(j, i);

                    // now, clear the least significant bit (LSB) from each
pixel element
                    R = pixel.R - pixel.R % 2;
                    G = pixel.G - pixel.G % 2;
                    B = pixel.B - pixel.B % 2;

                    // for each pixel, pass through its elements (RGB)
                    for (int n = 0; n < 3; n++)
                    {
                        // check if new 8 bits has been processed
                        if (pixelElementIndex % 8 == 0)
                        {
                            // check if the whole process has finished
                            // we can say that it's finished when 8 zeros
are added
                            if (state == State.Filling_With_Zeros &&
zeros == 8)
                            {
                                // apply the last pixel on the image
                                // even if only a part of its elements have
been affected
                                if ((pixelElementIndex - 1) % 3 < 2)
                                {
```

```
                    bmp.SetPixel(j, i, Color.FromArgb(R, G,
B));
                }

                // return the bitmap with the text hidden
in
                return bmp;
            }

            // check if all characters has been hidden
            if (charIndex >= text.Length)
            {
                // start adding zeros to mark the end of
the text
                state = State.Filling_With_Zeros;
            }
            else
            {
                // move to the next character and process
again
                charValue = text[charIndex++];
            }
        }

        // check which pixel element has the turn to hide a
bit in its LSB
        switch (pixelElementIndex % 3)
        {
            case 0:
                {
                    if (state == State.Hiding)
                    {
                        // the rightmost bit in the
character will be (charValue % 2)
                        // to put this value instead of the
LSB of the pixel element
                        // just add it to it
                        // recall that the LSB of the pixel
element had been cleared
                        // before this operation
                        R += charValue % 2;

                        // removes the added rightmost bit
of the character
                        // such that next time we can reach
the next one
                        charValue /= 2;
                    }
                } break;
            case 1:
                {
                    if (state == State.Hiding)
                    {
                        G += charValue % 2;

                        charValue /= 2;
                    }
                } break;
            case 2:
                {
                    if (state == State.Hiding)
```

```
                                    {
                                        B += charValue % 2;

                                        charValue /= 2;
                                    }

                                    bmp.SetPixel(j, i, Color.FromArgb(R, G,
B));
                                } break;
                            }

                            pixelElementIndex++;

                            if (state == State.Filling_With_Zeros)
                            {
                                // increment the value of zeros until it is 8
                                zeros++;
                            }
                        }
                    }
                }

                return bmp;
            }

            public static string extractText(Bitmap bmp)
            {
                int colorUnitIndex = 0;
                int charValue = 0;

                // holds the text that will be extracted from the image
                string extractedText = String.Empty;

                // pass through the rows
                for (int i = 0; i < bmp.Height; i++)
                {
                    // pass through each row
                    for (int j = 0; j < bmp.Width; j++)
                    {
                        Color pixel = bmp.GetPixel(j, i);

                        // for each pixel, pass through its elements (RGB)
                        for (int n = 0; n < 3; n++)
                        {
                            switch (colorUnitIndex % 3)
                            {
                                case 0:
                                    {
                                        // get the LSB from the pixel element
(will be pixel.R % 2)
                                        // then add one bit to the right of the
current character
                                        // this can be done by (charValue =
charValue * 2)
                                        // replace the added bit (which value
is by default 0) with
                                        // the LSB of the pixel element, simply
by addition
                                        charValue = charValue * 2 + pixel.R %
2;
                                    } break;
```

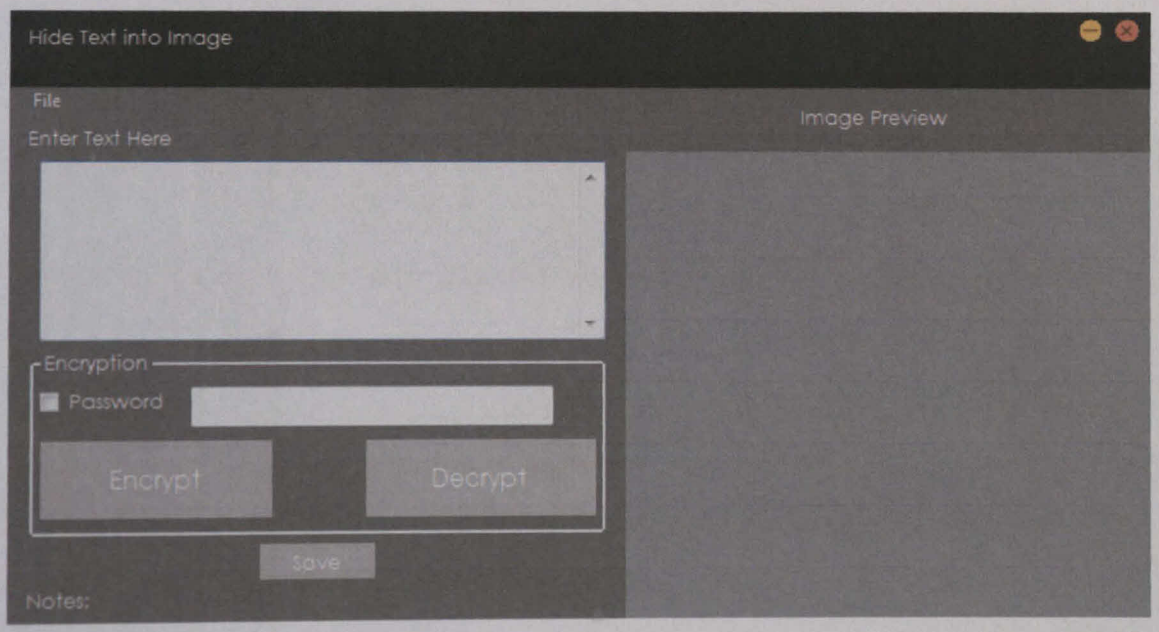
```
                            case 1:
                                {
                                    charValue = charValue * 2 + pixel.G %
2;
                                } break;
                            case 2:
                                {
                                    charValue = charValue * 2 + pixel.B %
2;
                                } break;
                        }

                        colorUnitIndex++;

                        // if 8 bits has been added, then add the current
character to the result text
                        if (colorUnitIndex % 8 == 0)
                        {
                            // reverse? of course, since each time the
process happens on the right (for simplicity)
                            charValue = reverseBits(charValue);

                            // can only be 0 if it is the stop character
(the 8 zeros)
                            if (charValue == 0)
                            {
                                return extractedText;
                            }

                            // convert the character value from int to char
                            char c = (char)charValue;

                            // add the current character to the result text
                            extractedText += c.ToString();
                        }
                    }
                }
            }

            return extractedText;
        }

        public static int reverseBits(int n)
        {
            int result = 0;

            for (int i = 0; i < 8; i++)
            {
                result = result * 2 + n % 2;

                n /= 2;
            }

            return result;
        }
    }
}
```

User Manual



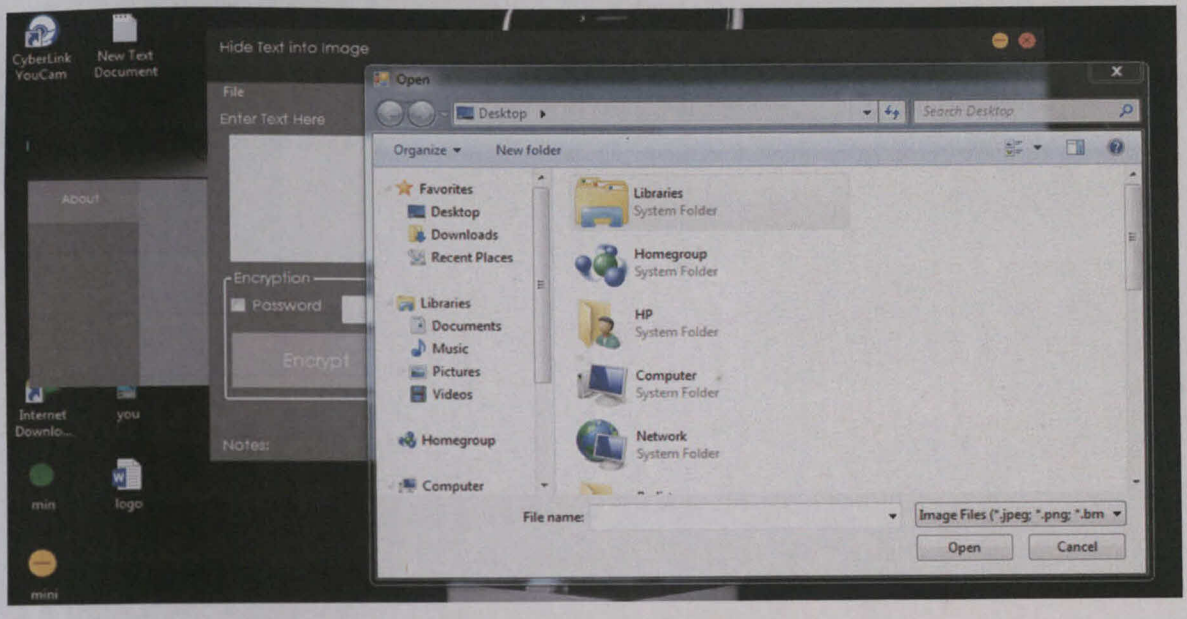
This is the first screen which has two panel option — one is the Encrypt Text into image for Message encryption and another is the Encryption Files into image for files Encryption

Encrypt Text into image System

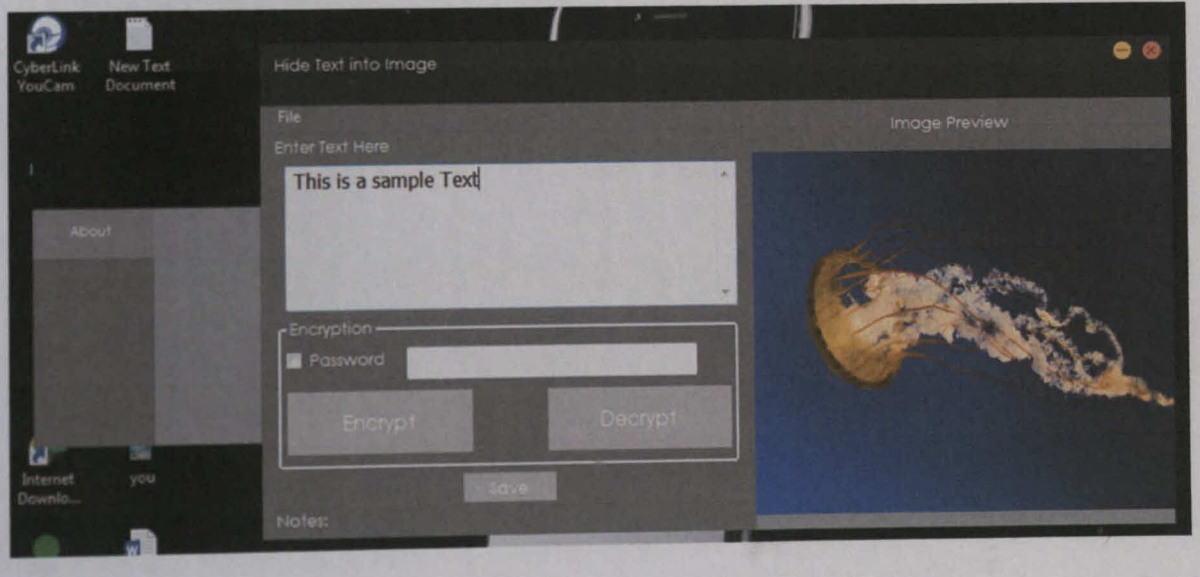


Encryption

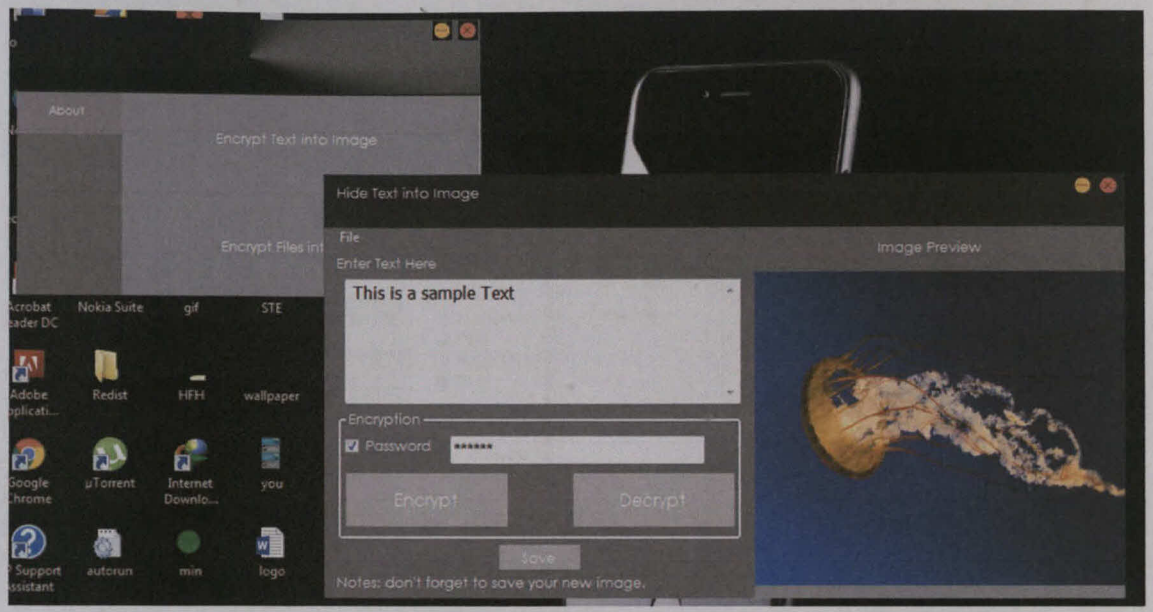
1. Load the image by clicking on File item to open image. The file open dialog box will displays as follows, select the Image file, which you want to use hide information and click on Open button.

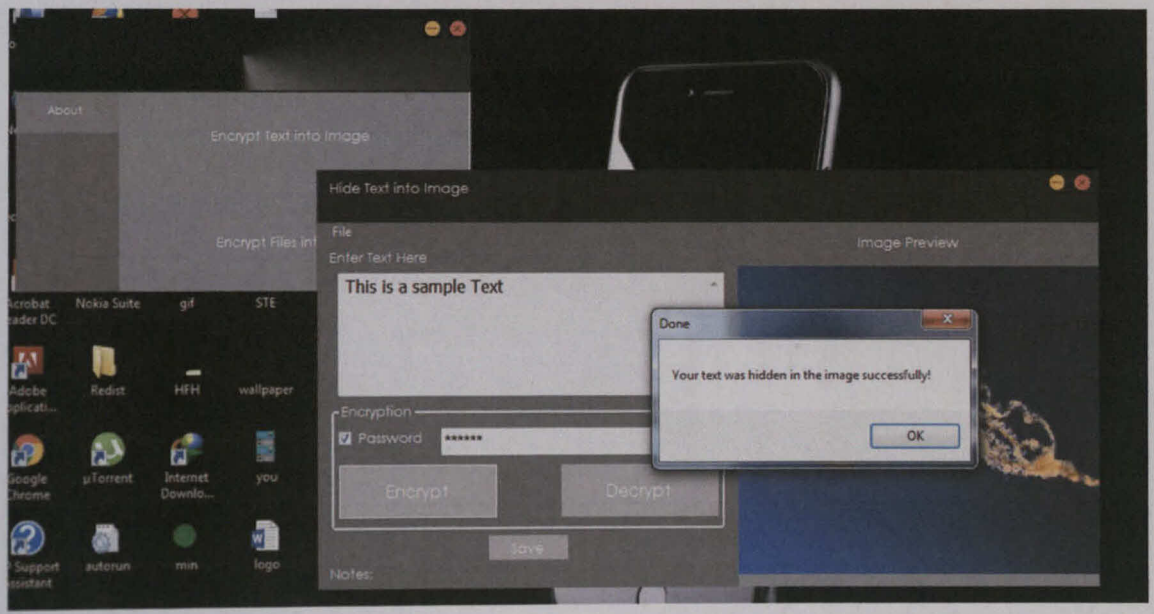


2. The image will open and display as follows. Next Enter the message that you want to hide on the Text area as follows

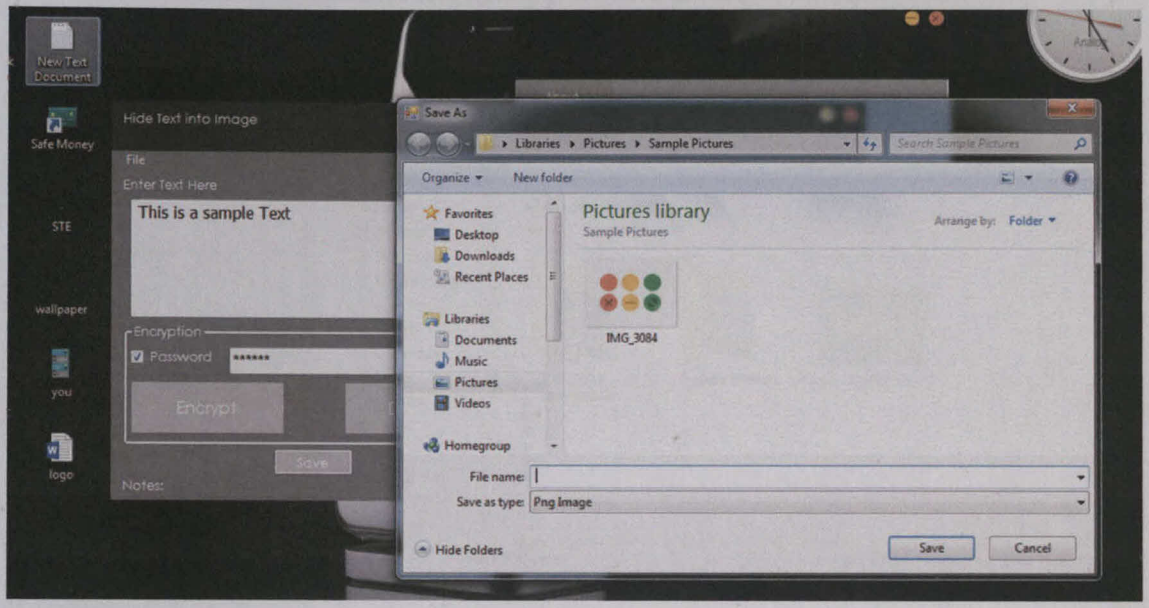


3. Next tick the password checkbox if you wish to enter the password to protect the message for better security or you can untick and encrypt the message without the password. Then click Encrypt to encrypt the message into an image, after clicking encrypt button then a message will appear that your text is successfully hidden within the image as follows



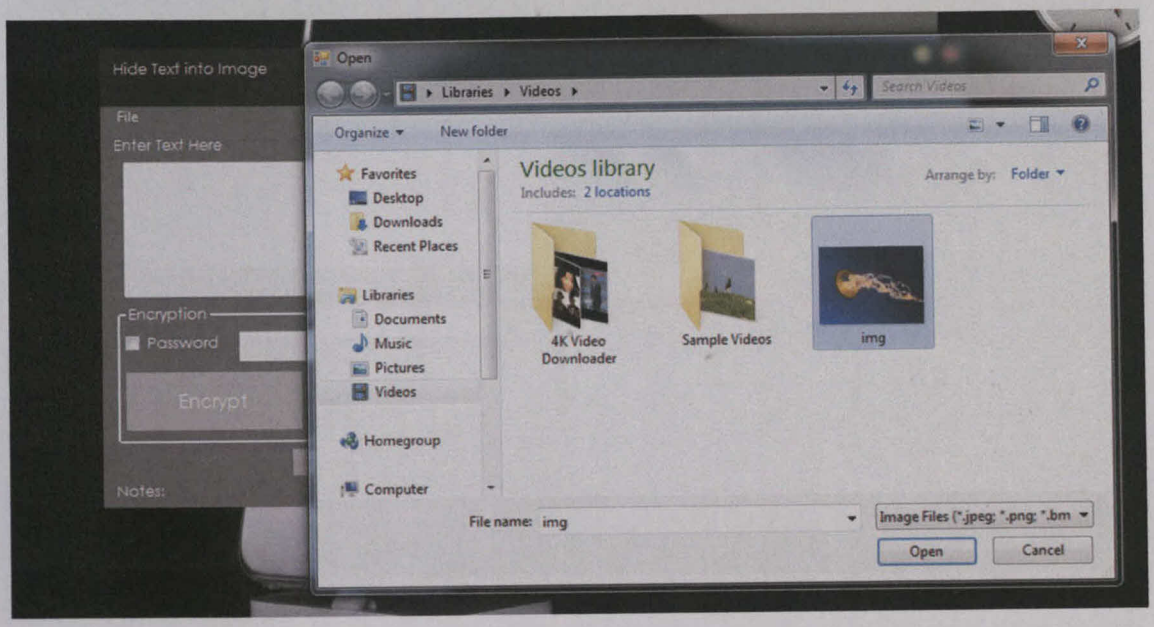


4. To save the file, click on the save button, Again the file open dialog box will appear, select the folder and give the name for the file to save and click on save button.

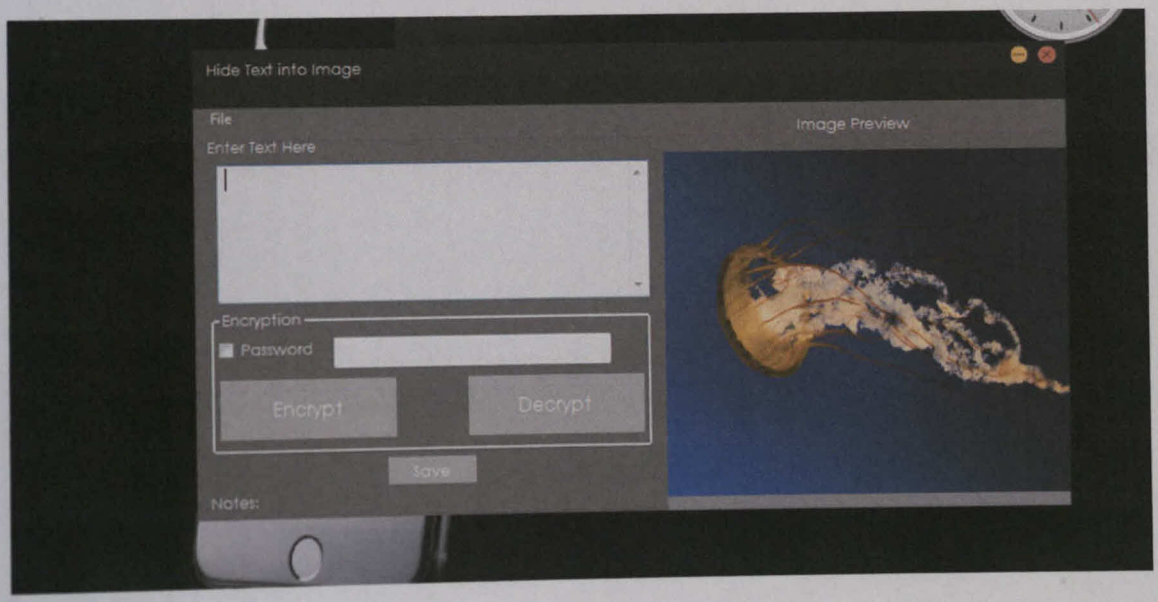


Decryption

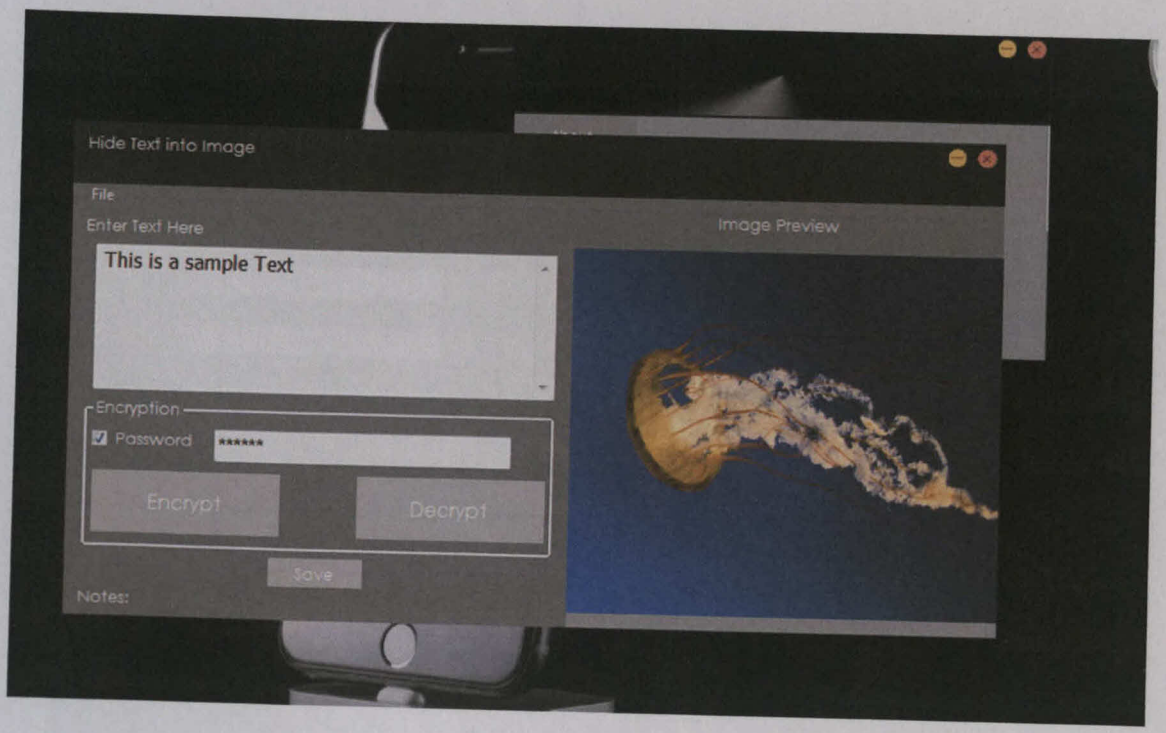
1. **Click on the "File" button, which open the Open file dialog box, here you have to select the image which is Encrypted and has hidden Message. Select the image file and click on Open button.**



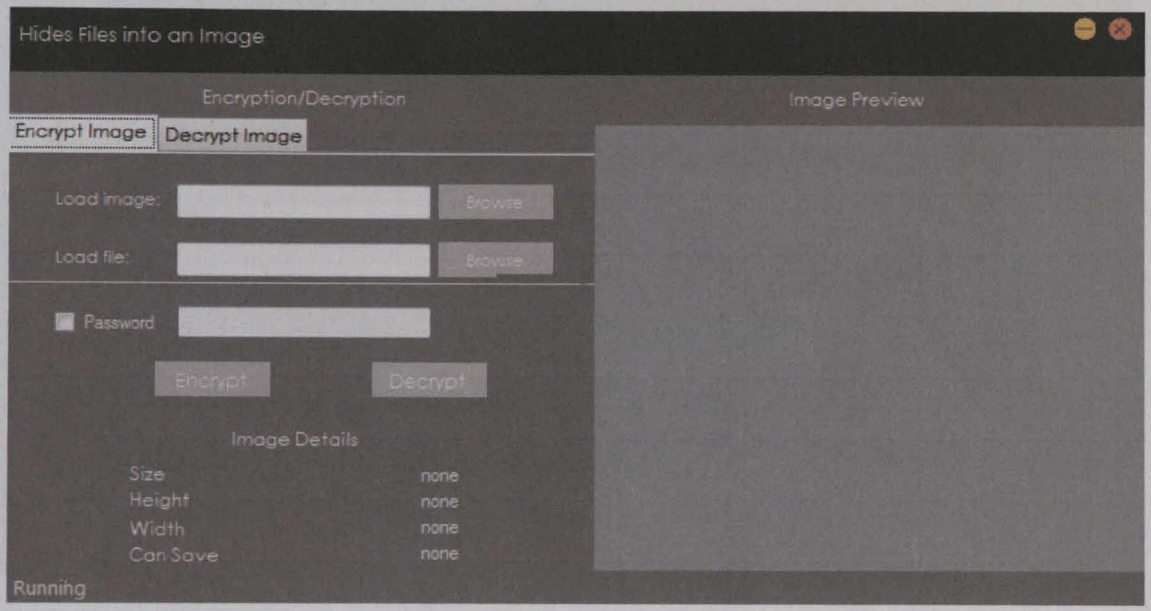
2. **The image file will be display as follow**



3. Enter the password which was used for the encrypted file and click on decrypt button then the message will be display as follows

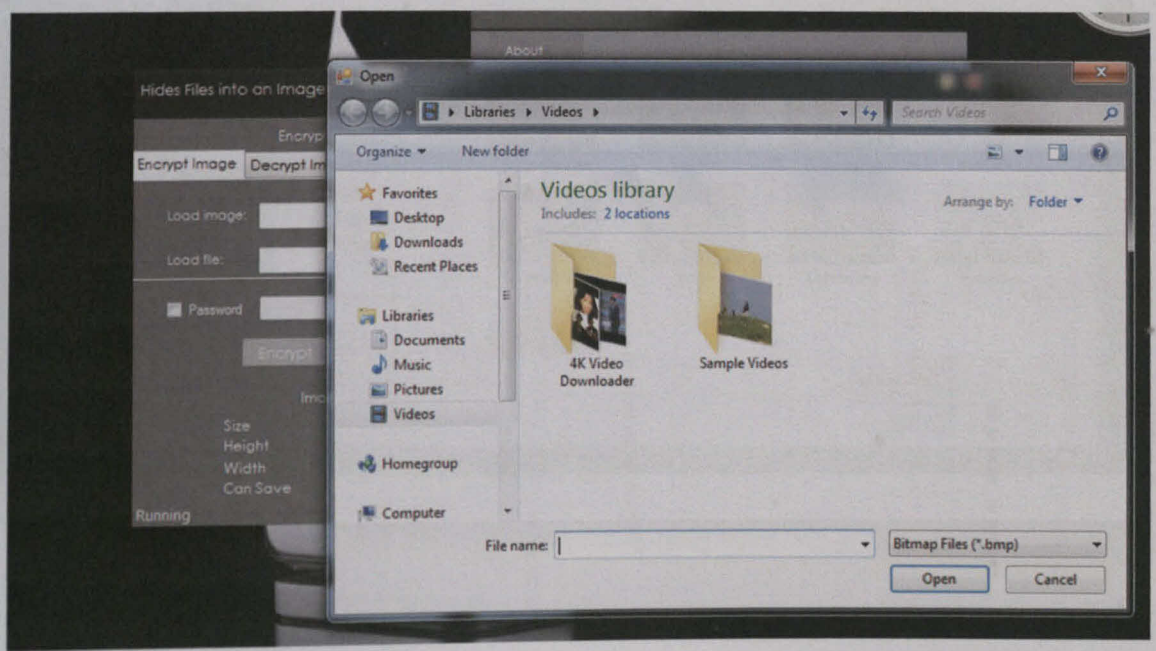


Encrypt Files into image System

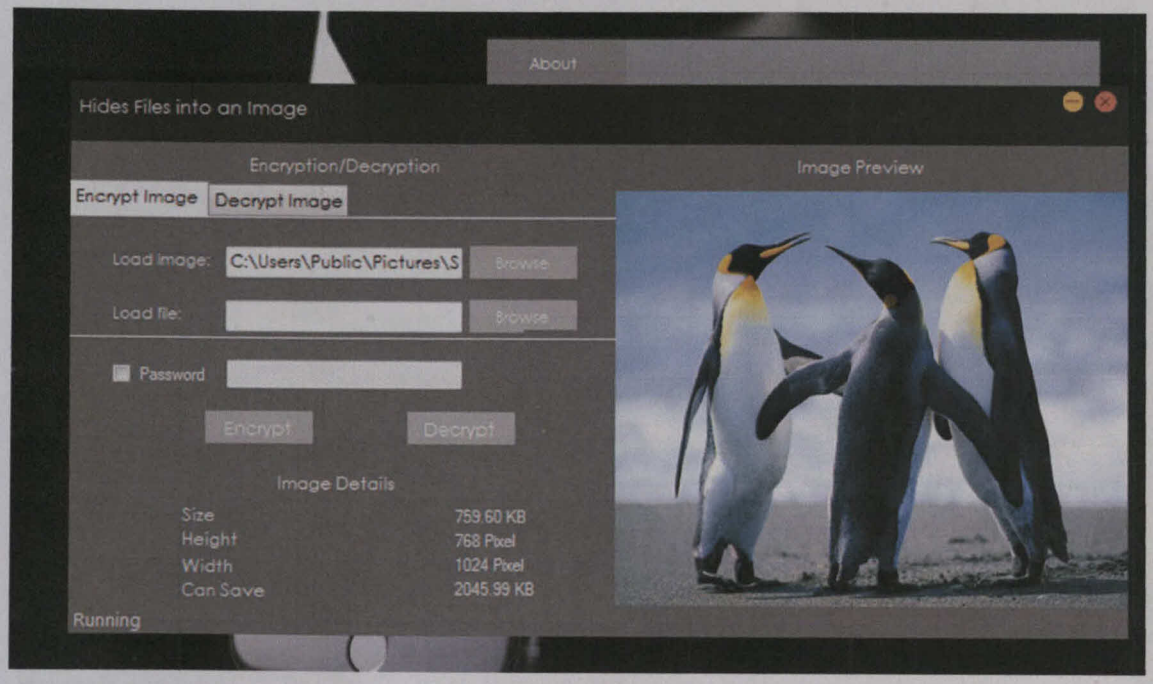


Encryption

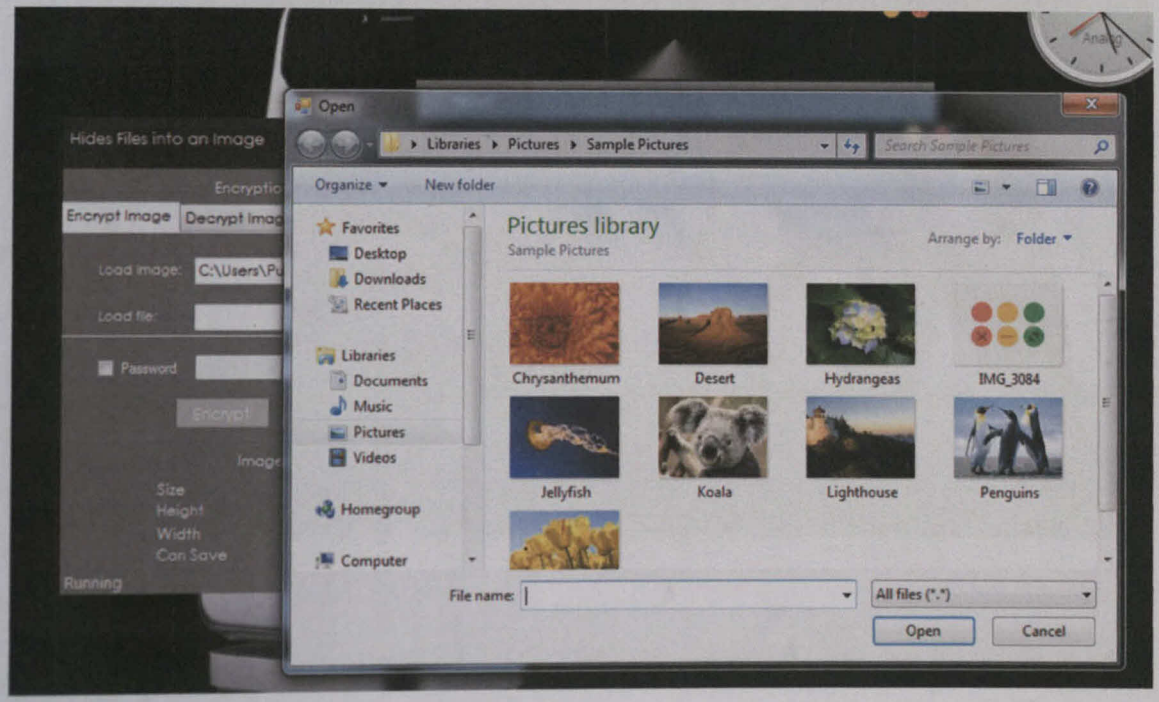
1. For load image click on button "Browse" that is next to the Load Image Textbox. The file open dialog box will displays as follows, select the Image file, which you want to use hide information and click on Open button.



2. The image file will opened and is displays as follows. Next, click on "Browse" button that is next to the Load File textbox.

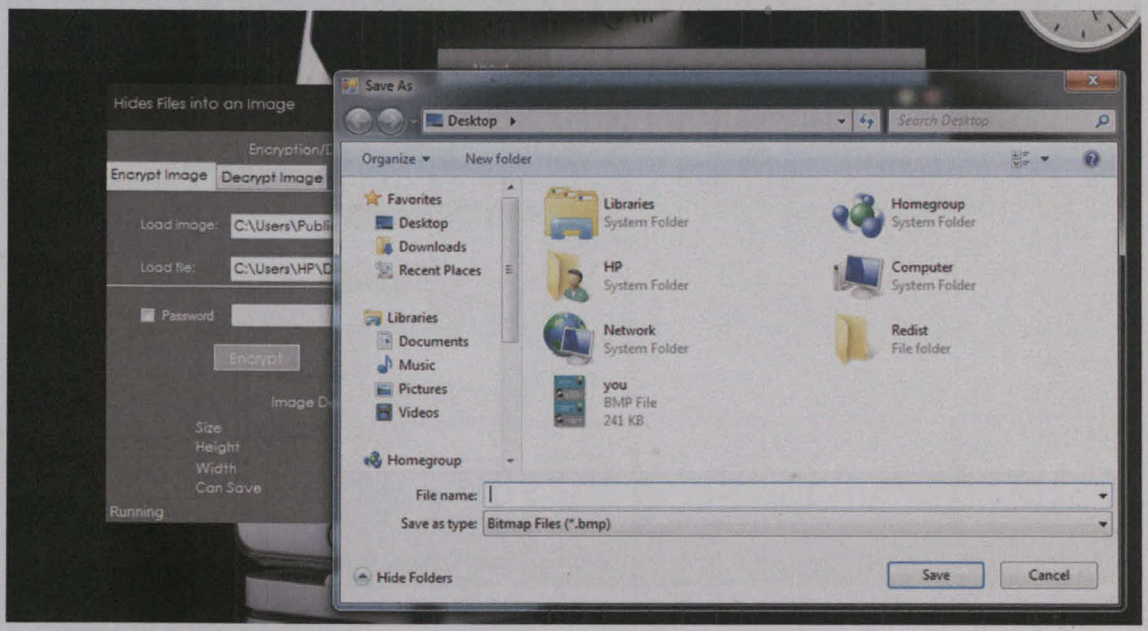


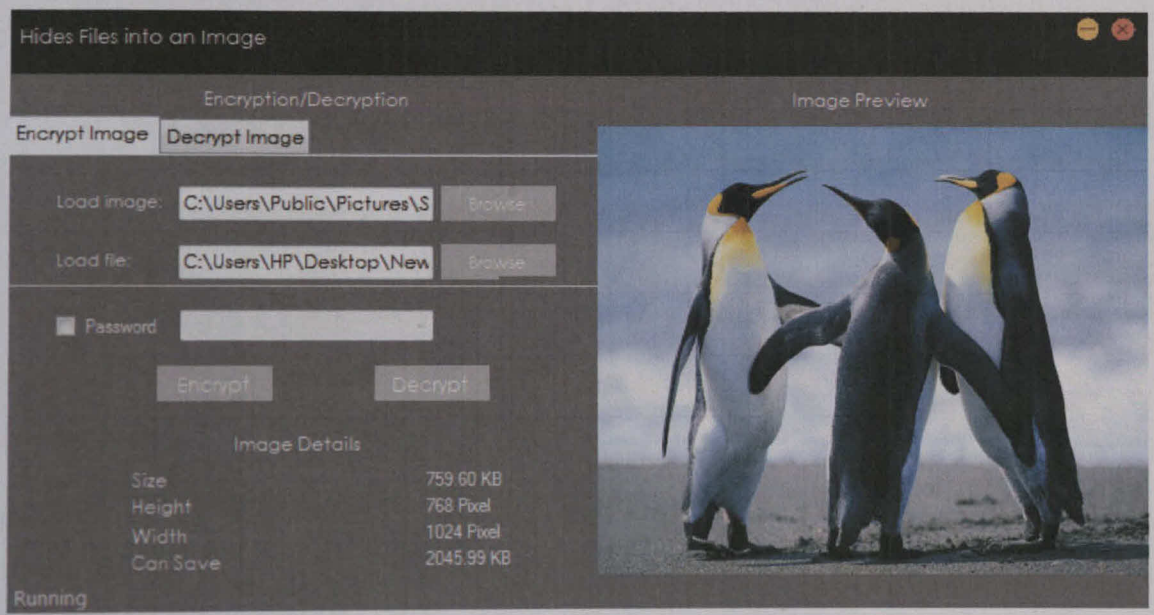
3. Again the file open dialog box will appear, select any type of file whatever you want to hide with the image and click on ok button.



Note: Lets skip the password for this tutorial but if you wish to put password you can do so.

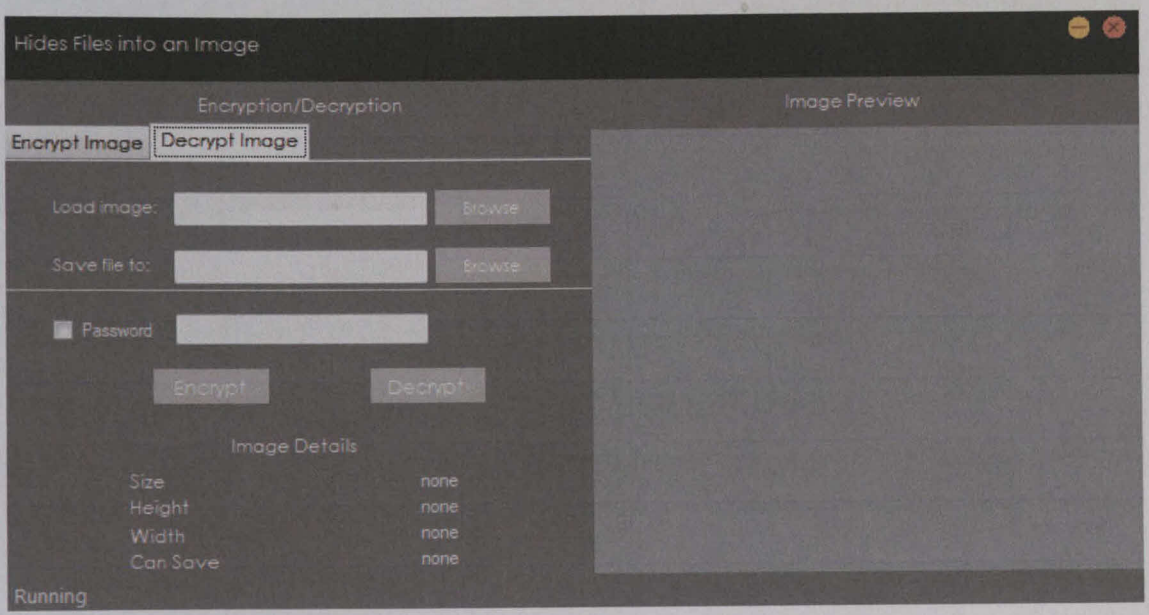
4. The next step is to encrypt the file. Now click on "Encrypt" button, it will open the save dialog box which ask you to select the path to save the New image file and the Image file name. The default format of image file is BMP.



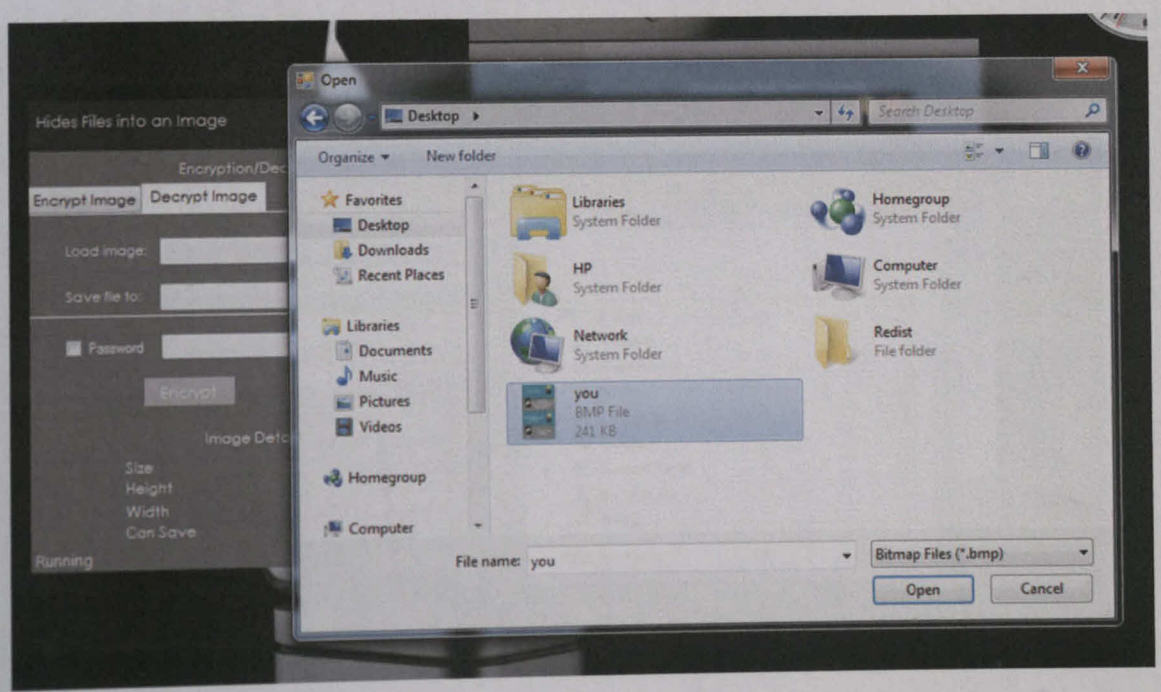


Decryption

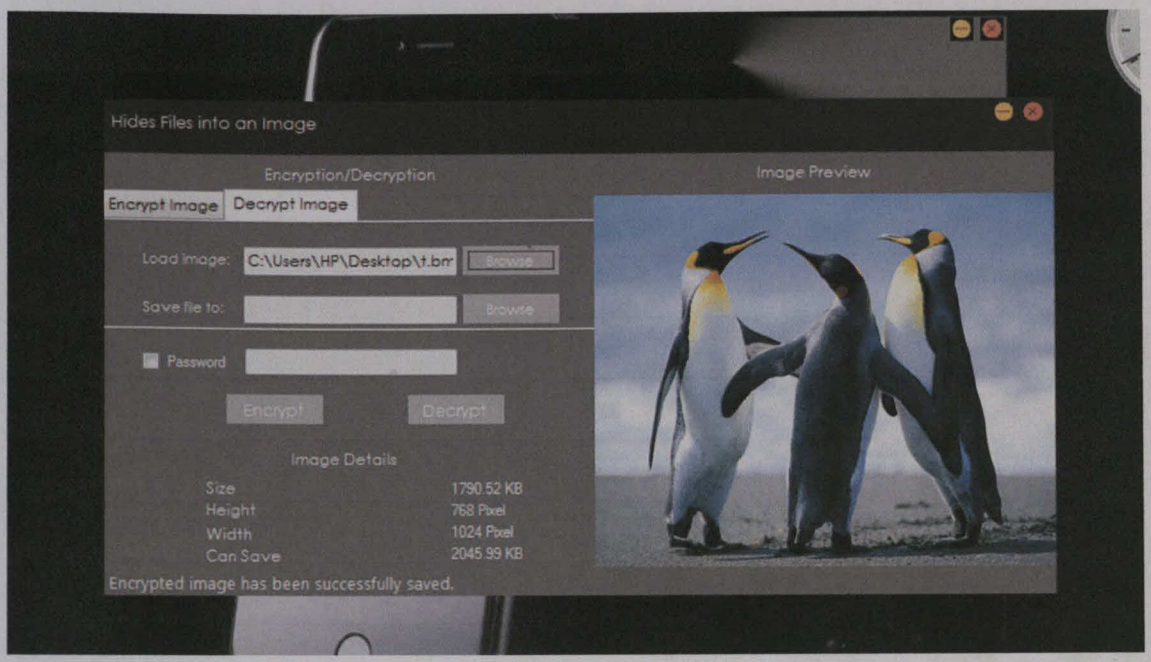
1. Select the Decryption Image tab option.



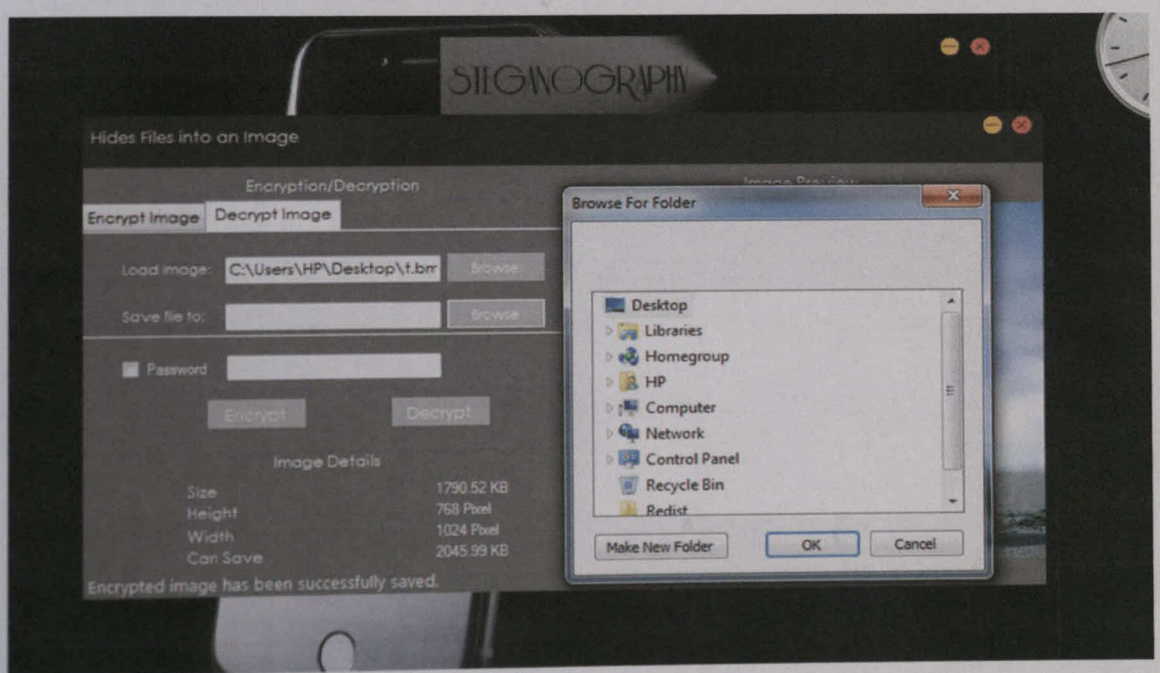
2. Next click on the "Browse" button, which open the Open file dialog box, here you have to select the image which is encrypted and has hidden information file. Select the image file and click on Open button.



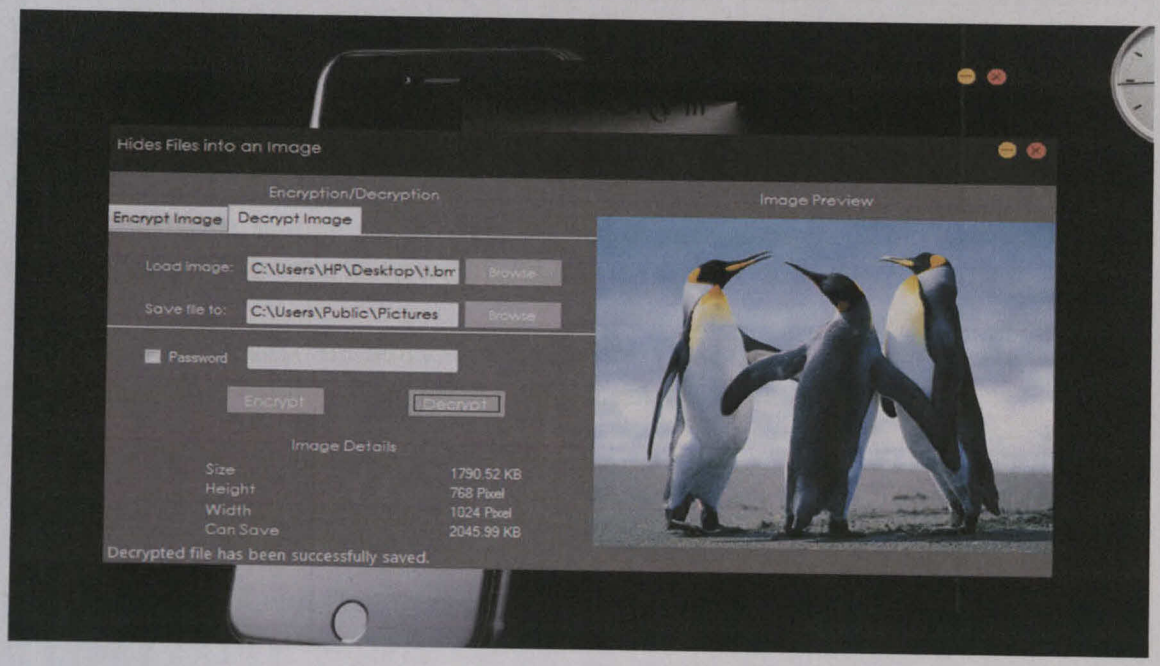
3. The image file displayed as follows:



4. Now click on "Browse" button which is next to "Save file to" textbox. It will open a dialog box that is "Browse for folder". It ask you to select the path or folder, where you want to extract the hidden file. Select the folder and click on Ok button.



5. Now click on Decrypt button, it will decrypt the image, the hidden file and image file is saved into selected folder. The message for successful decryption is displayed on the status bar which is places at bottom of the screen.



Acknowledgement

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to my Guide Mr Donald Thabah for his guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & member of the Department of Computer Science and Application for their kind co-operation and encouragement which help me in completion of this project.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

Lastly, I thank almighty, brother, sisters and friends for their constant encouragement without which this assignment would not be possible.

Bibliography

Websites

Following websites are referring to create this project reports.

- http://www.youtube.com
- http://www.wikihow.com
- http://www.null-byte.wonderhowto.com
- http://www.aesencryption.net
- http://www.tutorialpoints.com
- http://www.google.com
- http://www.microsoft.com
- http://www.itworld.com
- http://www.lifehacker.com
- http://www.wikipedia.org

Conclusion

Steganography is a really interesting subject and outside of the mainstream cryptography and system administration that most of us deal with day after day.

Steganography can be used for hidden communication. We have explored the limits of steganography theory and practice. We printed out the enhancement of the image steganography system using LSB approach to provide a means of secure communication. A stego-key has been applied to the system during embedment of the message into the cover image.

This steganography application software provided for the purpose to how to use any type of image formats to hiding any type of files inside their. The master work of this application is in supporting any type of pictures without need to convert to bitmap, and lower limitation on file size to hide, because of using maximum memory space in pictures to hide the file.

Since ancient times, man has found a desire in the ability to communicate covertly. The recent explosion of research in watermarking to protect intellectual property is evidence that steganography is not just limited to military or espionage applications. Steganography, like cryptography, will play an increasing role in the future of secure communication in the "digital world".