I would like to express my special thanks of gratitude to my teacher (Name of the teacher) as well as our principal (Name of the principal)who gave me the golden opportunity to do this wonderful project on the topic (Write the topic name), which also helped me in doing a lot of Research and i came to know about so many new things I am really thankful to them.

Secondly i would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

INTRODUCTION

Cryptography is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation. Modern cryptography intersects the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include ATM cards, computer passwords, and electronic commerce. [1]

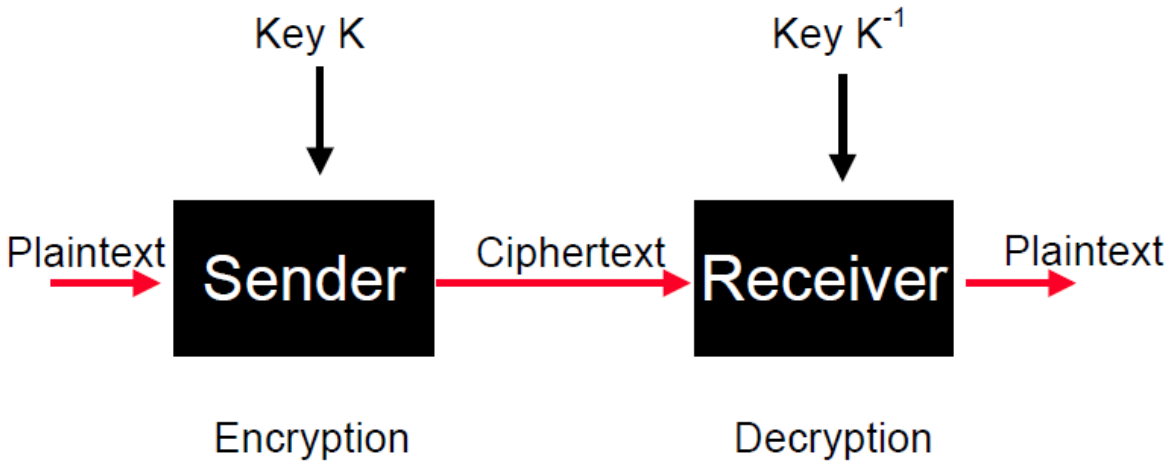

Figure 1: Block diagram to depict the cryptographic process

Data encryption technique converts data into an unreadable format so as to protect the information from external intrusion. It is thus useful in ensuring the privacy and security of the information transferred or shared between the systems. Text compression algorithms can be used to compact the text stored in the file and reduce the size of the file. It helps to reduce the consumption of resources, such as hard disk space or transmission bandwidth. Here, we propose a color coding scheme that can be used for data encryption which represents text in the form of colored blocks by grouping together binary bits and assigning them colors. [1]

At this point we need to describe the term ‘Steganography’. Steganography, literally means, "Covered Writing" which is derived from the Greek language. *Steganography is the art and science of communicating in a way which hides the existence of the communication.* In contrast to Cryptography, where the enemy is allowed to detect, intercept and modify

messages without being able to violate certain security premises guaranteed by a cryptosystem, the goal of Steganography is to hide messages inside other harmless messages in a way that does not allow any enemy to even detect that there is a second message present". [1]

In a digital world, Steganography and Cryptography are both intended to protect information from unwanted parties. Both Steganography and Cryptography are excellent means by which to accomplish this but neither technology alone is perfect and both can be broken. It is for this reason that most experts would suggest using both to add multiple layers of security. Steganographic technologies are a very important part of the future of Internet security and privacy on open systems such as the Internet. Steganographic research is primarily driven by the lack of strength in the cryptographic systems and the desire to have complete secrecy in an open-systems environment.

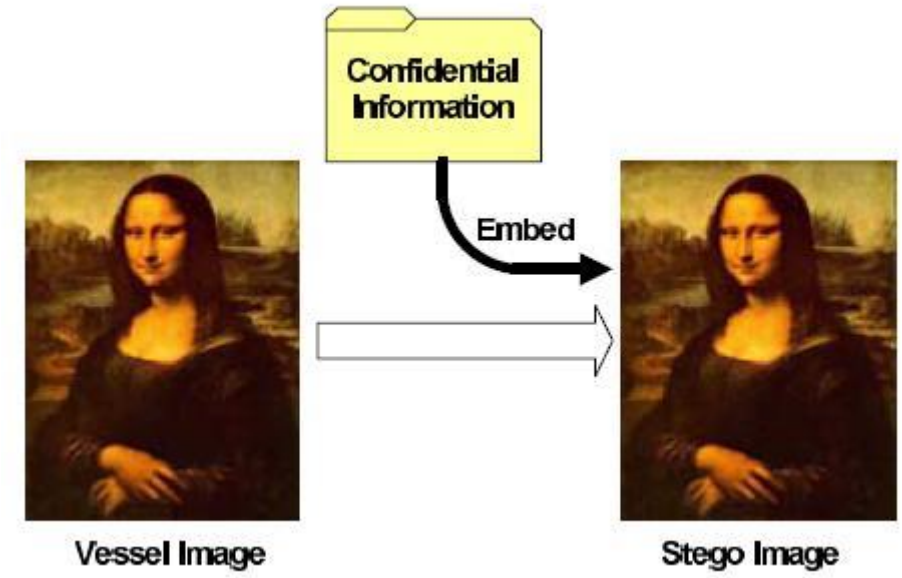

Figure 2: Information Hiding Using Steganography

PROBLEM STATEMENT

The aim of the project is to design a cryptosystem for text documents. The input to the system on the encoder side is the text entered by the user. The final output will be an image consisting of color patches. This image will be the input to the system on the decoder side. The output on the decoder side will be the text document.

The encoder will be required to enter three security parameters: key for encryption of the text entered, block size for determining the patch dimensions and color channel (R, G, or B). The decoder will also be required to enter these three parameters on the receiving side. The system is not expected to show any alert message on the wrong entry of any of these parameters. It will carry out all the necessary computations. But, any mismatch of the security parameters will lead to the decoder having to view a wrong text message.

The image generated after the encoding process can be stored in external storage devices such as hard drives, CD ROMs, and pen drives, or can be sent through email.

The basic working of the system is as follows. The text entered by the user is stored in a character array. A picture box is created and divided in to patches. The no of patches created depends on the block size entered by the encoder. Each character is extracted from the character array and its ASCII value is computed. Depending on the color channel selected, the color of each patch is created. The ASCII value of the character becomes the weight of the color whose channel has been selected. The default value of the other two color channels is 0. However, they can be adjusted. And thus, a color coded image is generated. This color coded image is opened on the decoding side and a somewhat reverse process takes place on that side.

SCOPE OF THE PROJECT

This project will have scope in a wide field of applications. Day by day there is an increasing need to keep data and communications secret in government, military, and business organizations. Employing encryption based on cryptographic algorithms to secure consumer data is of paramount importance today, especially in the area of ecommerce on the Internet. While there are many cryptosystems already existing, it is our attempt to develop yet another system which would try to cover up the holes in its former counterparts.

The reasons as to why our project may be a useful tool can be stated as follows:-

- Importance of intellectual property versus “brick and mortar” assets
- Threat of industrial espionage by competitors and even foreign governments
- Need for secure access to bank accounts and electronic transfers of funds
- Requirement for secure E-commerce
- Desire to avoid legal liability

Our project offers the means for protecting the data all the while preserving the privacy of critical personal financial, medical, and ecommerce data that might end up in the hands of those who shouldn’t have access to it.

LITERATURE SURVEY

4.1. EARLY VS. MODERN CRYPTOGRAPHY

Today's cryptography is vastly more complex than its predecessor. Unlike the original use of cryptography in its classical roots where it was implemented to conceal both diplomatic and military secrets from the enemy, the cryptography of today, even though it still has far-reaching military implications, has expanded its domain, and has been designed to provide a cost-effective means of securing and thus protecting large amounts of electronic data that is stored and communicated across corporate networks worldwide. Cryptography offers the means for protecting this data all the while preserving the privacy of critical personal financial, medical, and ecommerce data that might end up in the hands of those who shouldn't have access to it. [3]

There have been many advances in the area of modern cryptography that have emerged beginning in the 1970s as the development of strong encryption-based protocols and newly developed cryptographic applications began to appear on the scene. On January, 1977, the National Bureau of Standards (NBS) adopted a data encryption standard called the Data Encryption Standard (DES), which was a milestone in launching cryptography research and development into the modern age of computing technology. Moreover, cryptography found its way into the commercial arena when, on December, 1980, the same algorithm, DES, was adopted by the American National Standards Institute (ANSI). Following this milestone was yet another when a new concept was proposed to develop Public Key Cryptography (PKC), which is still undergoing research development today (Levy, 2001). [3]

When we speak of modern cryptography, we are generally referring to cryptosystems because the cryptography of today involves the study and practice of hiding information through the use of keys, which are associated with Web-based applications, ATMs, Ecommerce, computer passwords, and the like. [3]

Cryptography is considered not only a part of the branch of mathematics, but also a branch of computer science. There are two forms of cryptosystems: symmetric and asymmetric. Symmetric cryptosystems involve the use of a single key known as the secret key to encrypt and decrypt data or messages. Asymmetric cryptosystems, on the other hand, use one key (the public key) to encrypt messages or data, and a second key (the secret key) to decipher or decrypt those messages or data. For this reason, asymmetric cryptosystems are

also known as public key cryptosystems. The problem that symmetric cryptosystems have always faced is the lack of a secure means for the sharing of the secret key by the individuals who wish to secure their data or communications. Public key cryptosystems solve this problem through the use of cryptographic algorithms used to create the public key and the secret key, such as DES, which has already been mentioned, and a much stronger algorithm, RSA. The RSA algorithm is the most popular form of public key cryptosystem, which was developed by Ron Rivest, Adi Shamir, and Leonard Adleman at the Massachusetts Institute of Technology in 1977 (Robinson, 2008). The RSA algorithm involves the process of generating the public key by multiplying two very large (100 digits or more) randomly chosen prime numbers, and then, by randomly choosing another very large number, called the encryption key. The public key would then consist of both the encryption key and the product of those two primes. Ron Rivest then developed a simple formula by which someone who wanted to scramble a message could use that public key to do so. The plaintext would then be converted to cipher text, which was transformed by an equation that included that large product. Lastly, using an algorithm developed through the work of the great mathematician, Euclid, Ron Rivest provided for a decryption key—one that could only be calculated by the use of the original two prime numbers. Using this encryption key would unravel the cipher text and transform it back into its original plain text. What makes the RSA algorithm strong is the mathematics that is involved. Ascertaining the original randomly chosen prime numbers and the large randomly chosen number (encryption key) that was used to form the product that encrypted the data in the first place is nearly impossible (Levy, 2001). [3]

A very popular public key cryptosystem is known as Pretty Good Privacy (PGP), developed by Phil Zimmerman beginning in early 1991 (Levy, 2001). The strength of the keys that are created to encrypt and decrypt data or communications is a function of the length of those keys. Typically the longer the key, the stronger that key is. For example, a 56-bit key (consisting of 56 bits of data) would not be as strong as a 128-bit key. And, consequently, a 128-bit key would not be as strong as a 256- or 1024-bit key. [3]

REQUIREMENT ANALYSIS

5.1. PROJECT SPECIFICATION

Project Title

Color Coded Cryptography

Aim of the Project

The aim of the project is to design and build a cryptosystem for text documents. The text document will be encoded in an image form based on ASCII value computations.

User Requirements

This system encounters two types of users: the Sender and the Receiver or in other words, the Encoder and the Decoder.

The Sender/Encoder is required to enter only text document for encryption, otherwise the system will not be able to perform the encoding of the image. Also, he/she must select a key for encrypting the plain text message into cipher text. Two more criteria which the encoder needs to select are the Block Size and the Color Channel prior to the encoding process. All these parameters, viz. the key, the block size and the color channel need to be sent by the Sender/Encoder to the receiver or the decoder in some secure fashion.

The Receiver/Decoder is required to first open the image file sent to him/her by the Sender/Encoder. He/she then needs to enter the key, block size and the color channel shared between him/her and the sender. Only if these three parameters are the same on both the sides, the message will be properly decrypted.

The interface on the encoding side must alert the Sender/Encoder if he/she has not entered a key or has not entered any block size. By default, the color channel selected is red. The Sender/Encoder must be able to save the color coded image.

The interface on the decoding side must also alert the Receiver/Decoder if he/she has not entered a key or block size and the color channel by default remains red. However, the interface is not designed to alert the receiver on entering any wrong value for the key, block size or the color channel.

5.2. USE CASE DIAGRAM

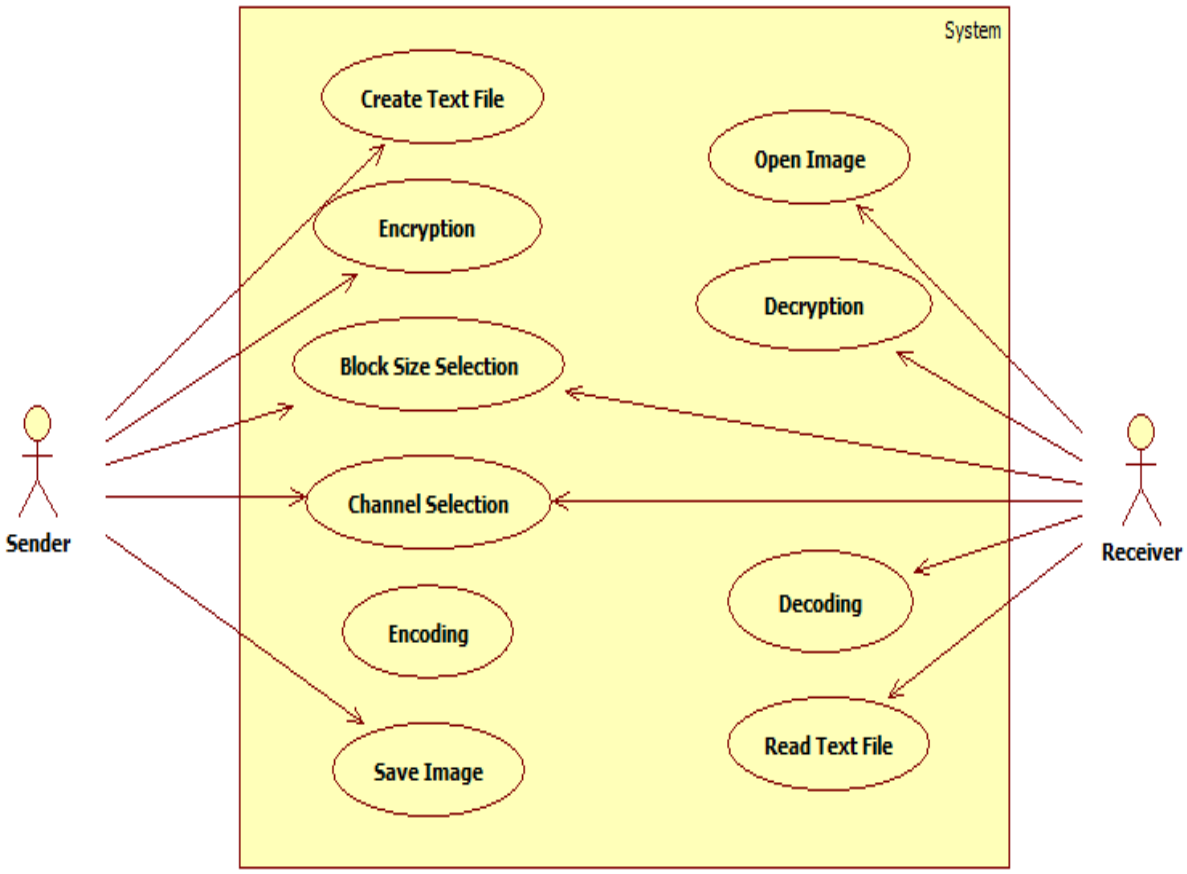

Figure 3: Use Case Diagram

5.3. ACTIVITY DIAGRAMS

5.3.1. Activity Diagram on Sender's Side

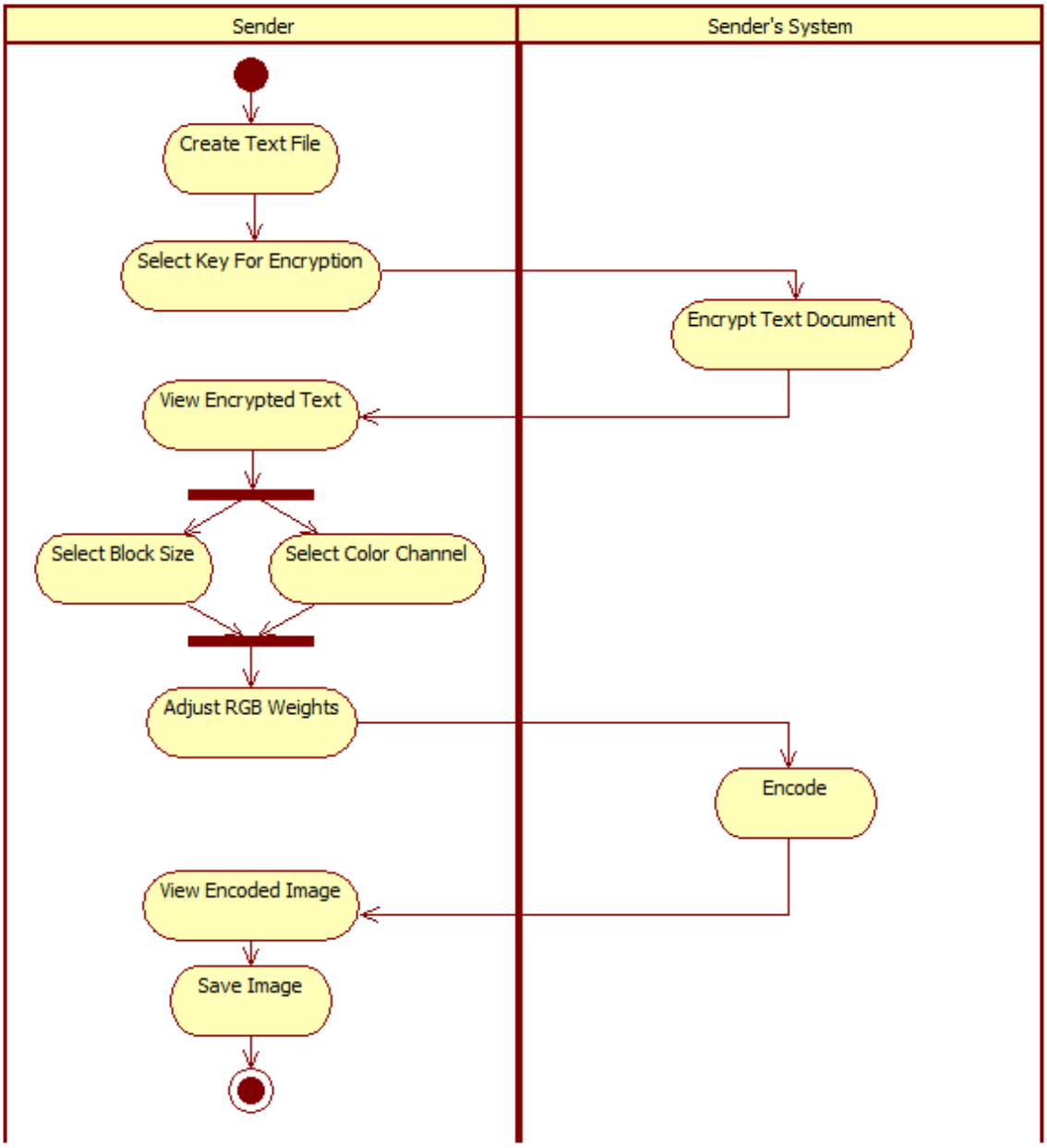

Figure 4: Activity Diagram on Sender's Side

5.3.2. Activity Diagram on Receiver's Side

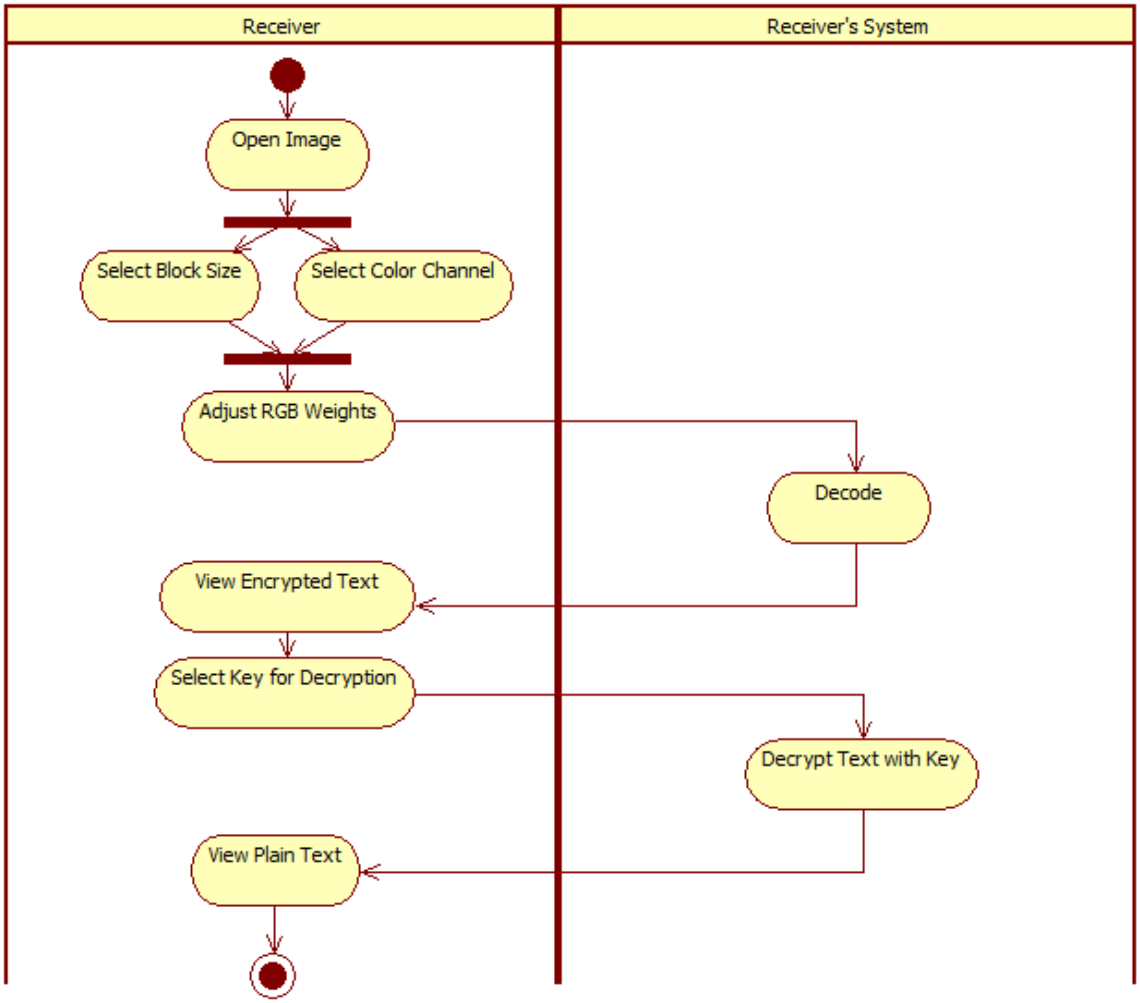

Figure 5: Activity diagram on Receiver's Side

5.4. PROJECT TIMELINE

5.4.1. Project Timeline for Semester 1

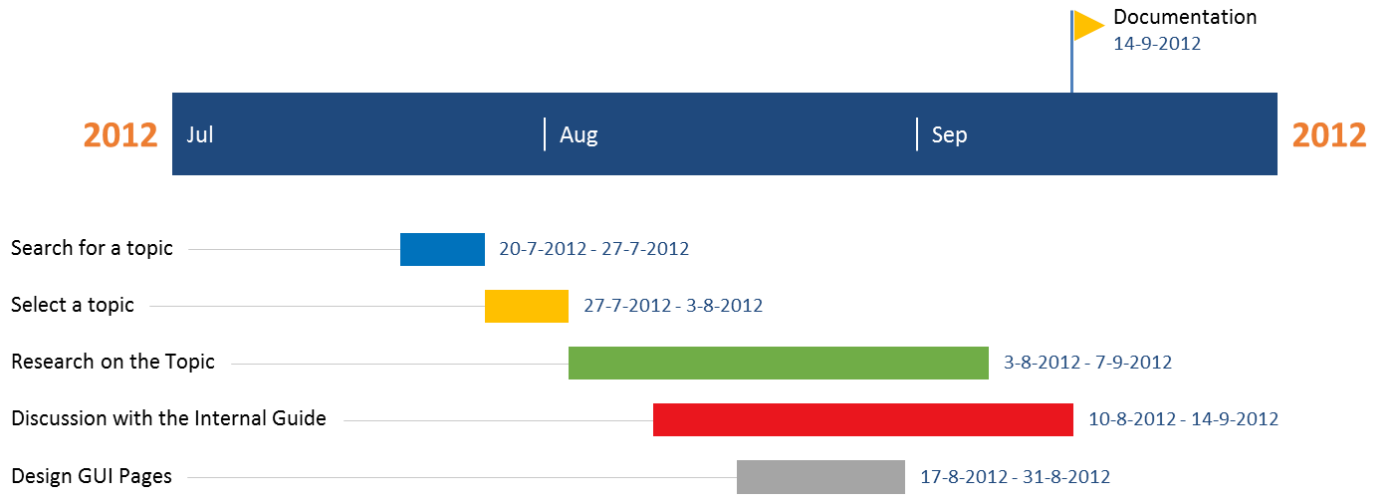

Figure 6: Timeline for Semester 1

5.4.2. Project Timeline for Semester 2

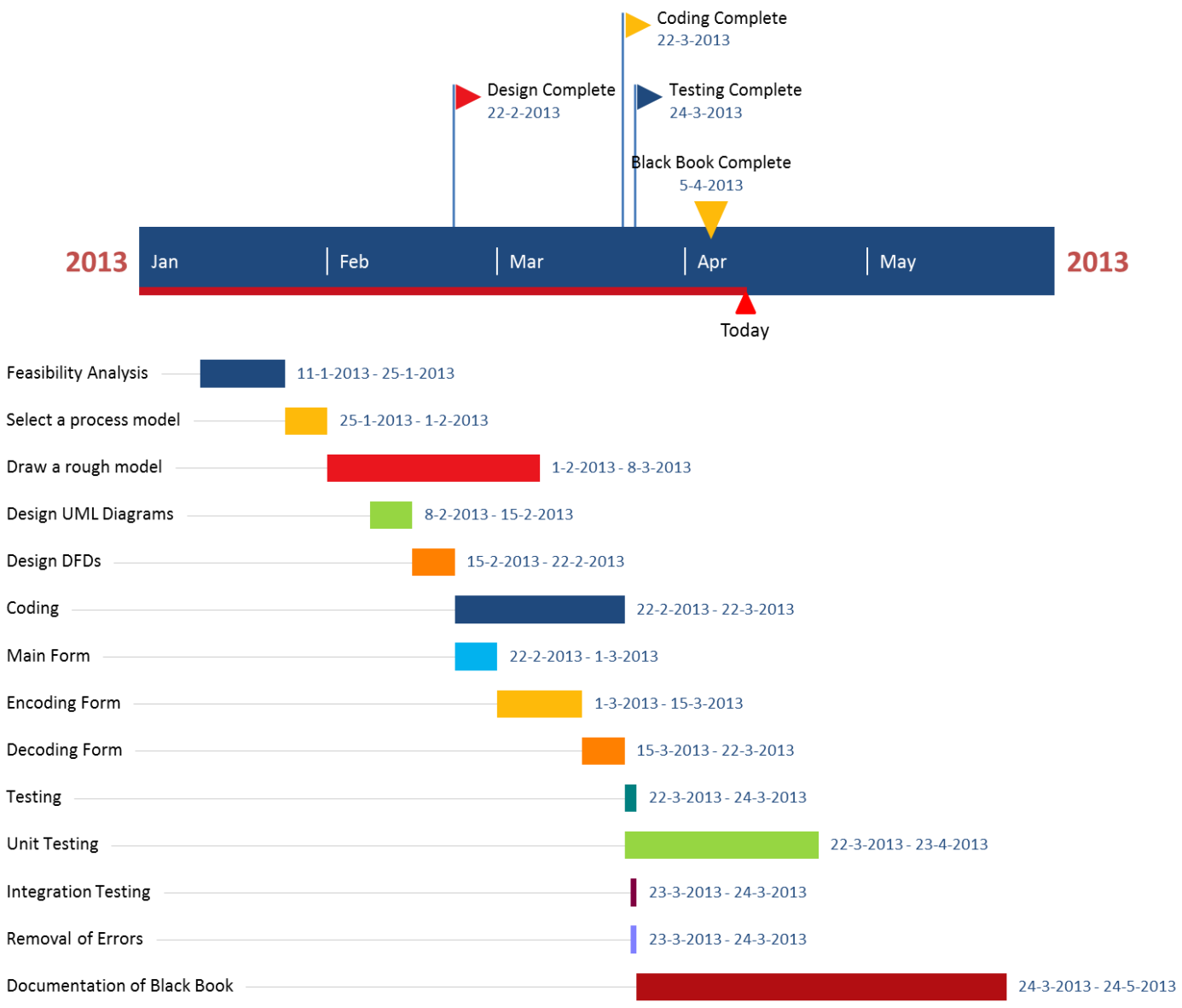

Figure 7: Timeline for Semester 2

5.5. PROJECT ARCHITECTURE

The Color Coded Cryptosystem mainly consists of the following components: Encryption of Text with Key, Encoding of Encrypted Text into Image, Decoding of Image into Encrypted Text, and Decryption of Text with Key.

The encryption of text is done by adding the value of the text message to that of the key and thereby creating the cipher text. The encoding of the encrypted text involves the following sub-components: Block Size Selection and Channel Selection. The decoding of the encrypted text also involves the two sub-components as those in encoding of the encrypted text. The decryption of text is done by subtracting the value of key from that of the text message and thereby retrieving the plain text.

The following figure shows the overall system model for color coded cryptosystem:

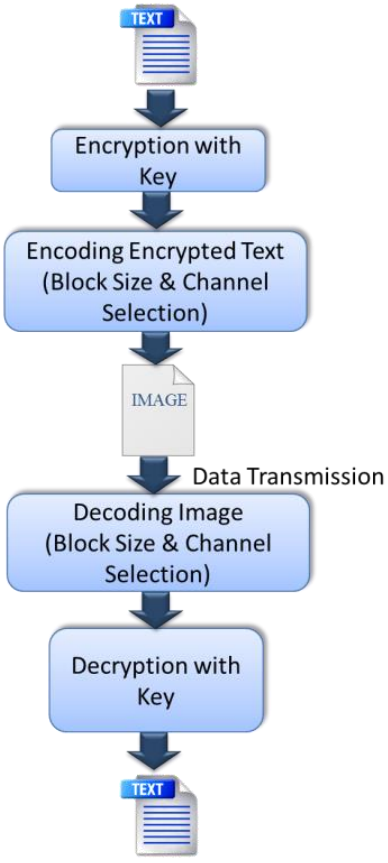

Figure 8: System Model

5.5.1. Encryption With Key

The text document entered by the sender is first encrypted using a symmetric key algorithm. The encryption is done by adding the ASCII value of the key with the ASCII value of the text message and then converting the resultant to character, thereby obtaining the cipher text. Since, a symmetric key algorithm is being used the same key is to be used by the receiver or the decoder.

5.5.2. Encoding of Encrypted Text Into Image

The cipher text thus obtained in the previous component is now subjected to encoding process. This entire process consists of two major steps: Block Size Selection and Channel Selection. These two criteria help in optimizing the level of security furthermore. In other words, in addition to the private or symmetric key, these two parameters also act as keys. The block size and the channel selected by the sender must be known by the receiver to properly decode the image. This encoded image will now be saved by the sender to be later sent to the receiver.

5.5.3. Decoding of Image Into Encrypted Text

After the intended receiver has received the color coded image he/she will have to carry out the decoding process which primarily involves entering the correct block size and channel. If these two parameters do not match those entered by the sender while he/she was encoding the message, then it will lead to false decoding of the image.

5.5.4. Decryption With Key

After decoding the image the receiver obtains a cipher text document which he/she has to decrypt using the symmetric key shared between him/her and the sender. After decryption with the right key, the receiver obtains the correct plain text message.

5.6. CLASS DIAGRAMS

5.6.1. Class Diagram On Sender's Side

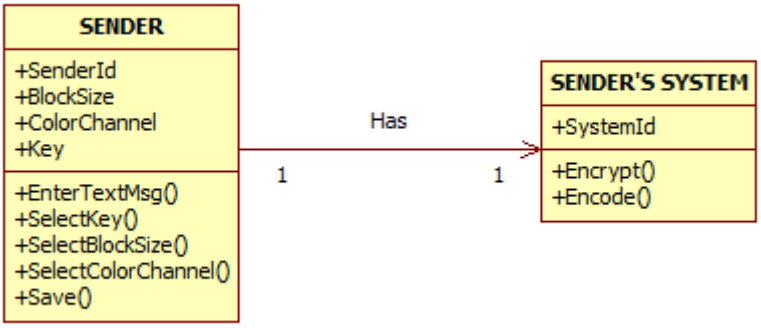

Figure 9: Class Diagram on Sender's Side

5.6.2. Class Diagram On Receiver's Side

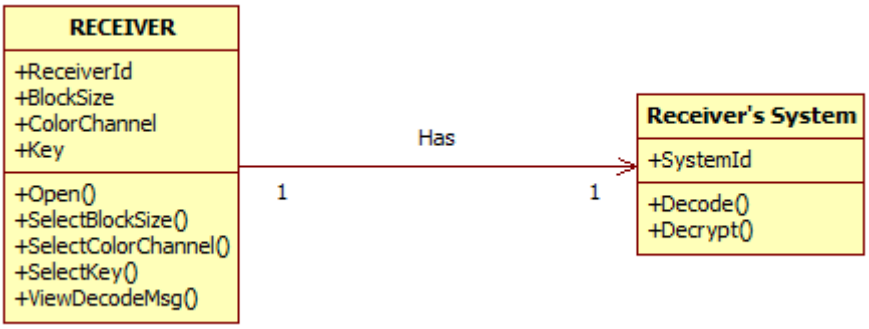

Figure 10: Class Diagram on Receiver's Side

5.7. STATE CHART DIAGRAM

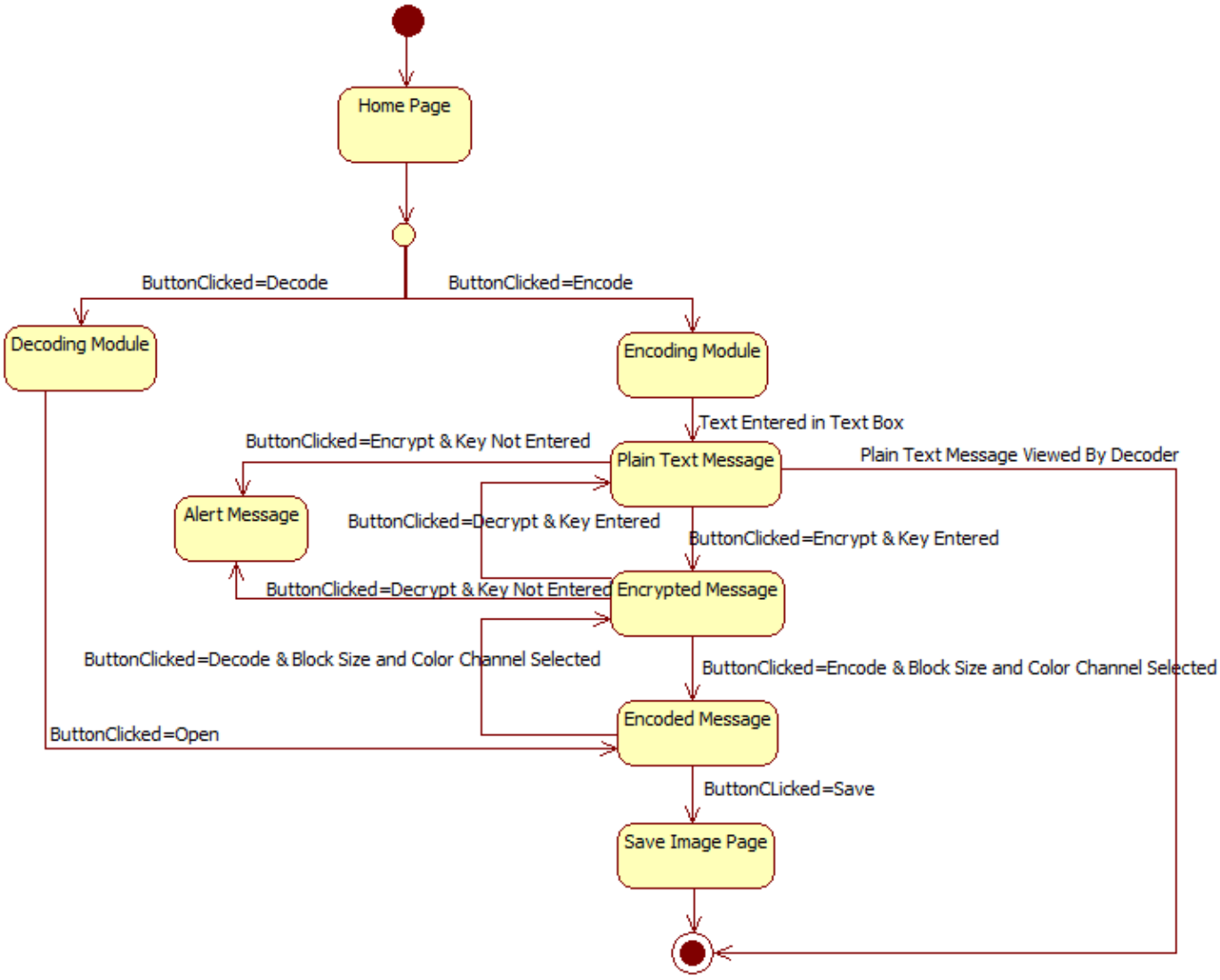

Figure 11: State Chart Diagram

5.8. PRESENTATION MODELING

5.8.1. Sequence Diagrams

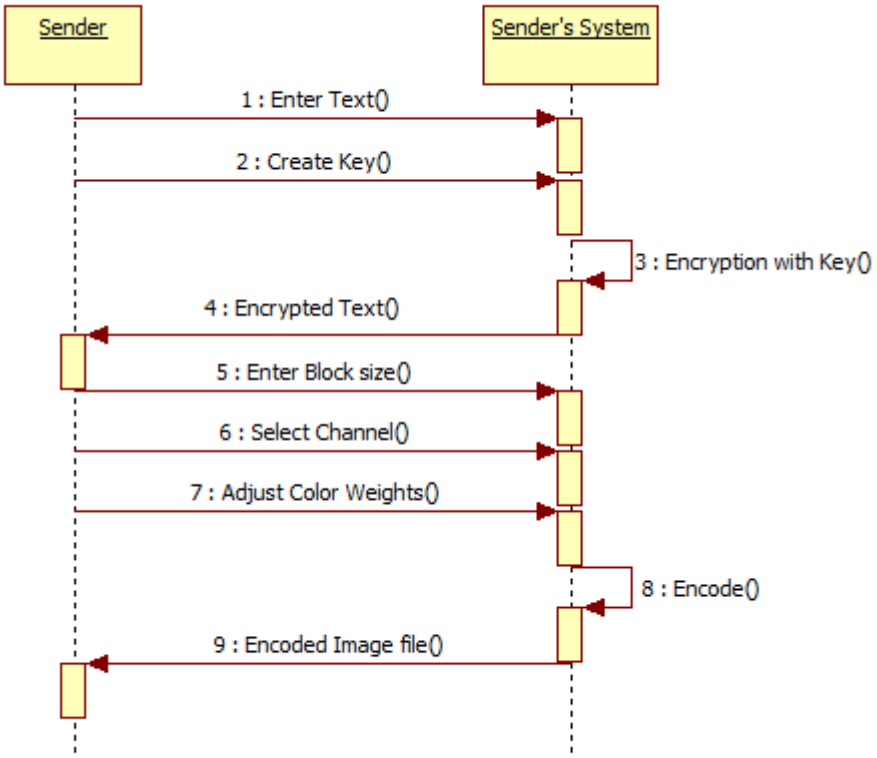

Figure 12: Sequence Diagram on Sender's Side

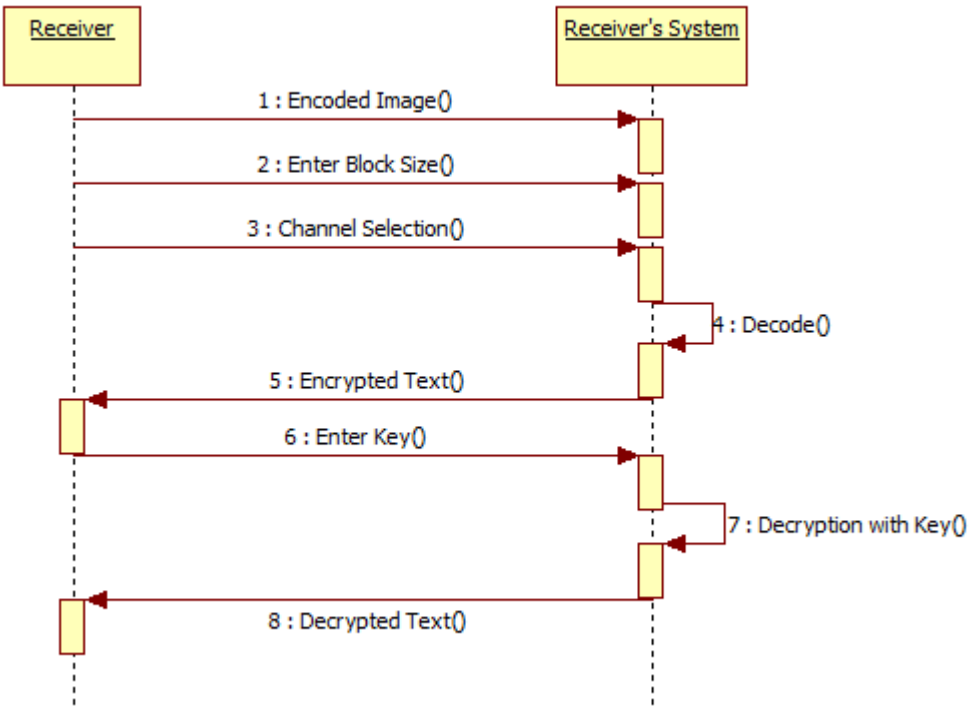

Figure 13: Sequence Diagram on Receiver's Side

5.8.2. Collaboration Diagrams

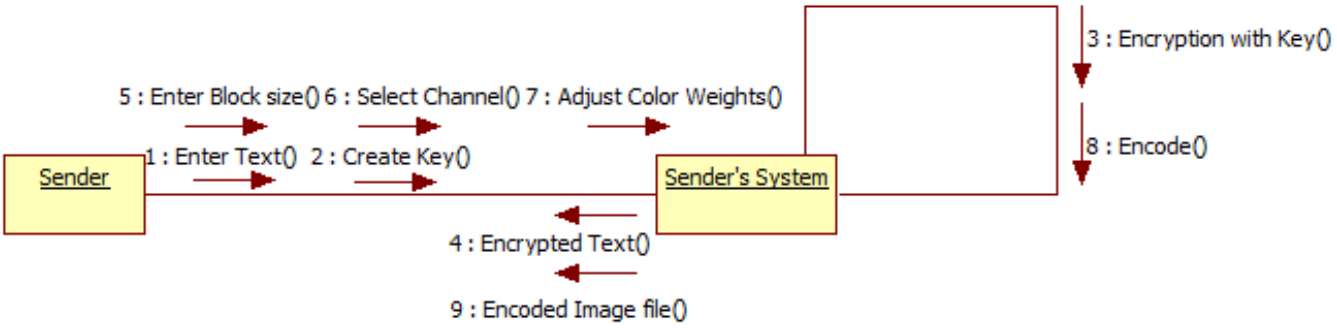

Figure 14: Collaboration Diagram on Sender's Side

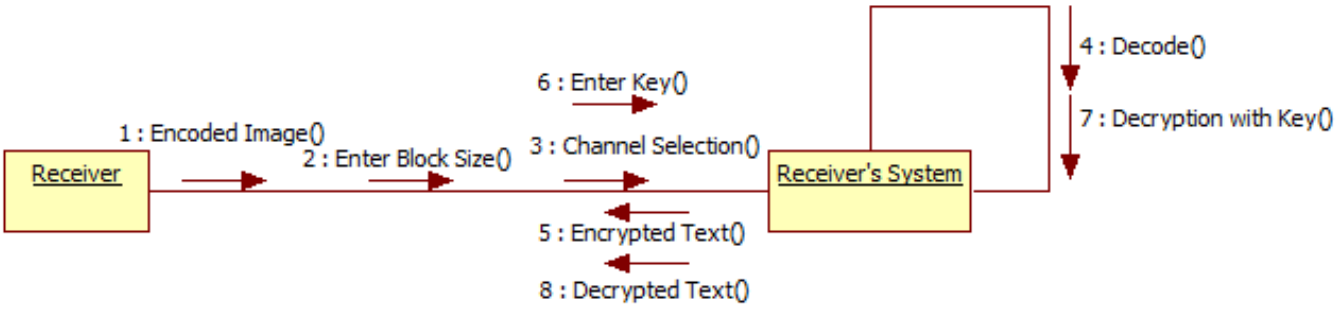

Figure 15: Collaboration Diagram on Receiver's Side

5.9. SOFTWARE REQUIREMENTS SPECIFICATIONS

This is an SRS document for the Color Coded Cryptographic System.

5.9.1. Introduction

5.9.1.1. Purpose

The purpose of this document is to describe the requirements for the Color Coded Cryptographic System. The intended audience includes all stakeholders in the potential system. These include the system users and the developers.

5.9.1.2. Scope

The proposed software product is the Color Coded Cryptographic System. The system will be used to encode text messages in a unique color coded image. Currently, no such system exists. The intention of the system is to increase information security which can be used in areas where data integrity and confidentiality is of high importance.

5.9.1.3. Definitions, Acronyms, and Abbreviations

Encoder – The user who writes the text message and converts it into color coded image.

Decoder – The user who opens the color coded image and converts it into text message.

Key – Secret key shared between encoder and decoder for encryption and decryption.

Block Size – Decides the dimensions of the color patches.

Color Channel – Any one of R, G, or B channel can be selected which decides the color of the patches.

R – Red

G – Green

B – Blue

5.9.1.4. References

[1] www.wikipedia.org

[2] "Color Coded Cryptography", Aditya Gaitonde, International Journal of Scientific & Engineering Research, Volume 3, Issue 7, July-2012

[3] "Literature Review of Cryptography and its Role in Network Security Principles and Practice", Daniel Lloyd Calloway, Capella University, OM8302, § 4, 8 September 2008

5.9.1.5. Overview

This Software Requirements Specification (SRS) is the requirements work product that formally specifies the Color Coded Cryptographic System. It includes the results of both business analysis and systems analysis efforts. Various techniques were used to elicit the requirements and we have identified your needs, analyzed and refined them. The objective of this document therefore is to formally describe the system's high level requirements including functional requirements, non-functional requirements and business rules and constraints. The detail structure of this document is organized as follows:

Section 2 of this document provides an overview of the business domain that the proposed Color Coded Cryptographic System will support. These include a general description of the product, user characteristics, general constraints, and any assumptions for this system. This model demonstrates the development team's understanding of the business domain and serves to maximize the team's ability to build a system that truly does support the business.

Section 3 presents the detail requirements, which comprise the domain model.

5.9.2. General Description

5.9.2.1. Product Perspective

This system is a self-contained system that encodes text messages into color coded image files and conversely decodes them also. Various stakeholders are involved in this system.

5.9.2.2. Product Functions

The system functions can be described as follows:

Encryption: The text message entered by user is first encrypted into cipher text.

Encoding: The cipher text is encoded into color coded image.

Decoding: The color coded image is decoded into cipher text message.

Decryption: The cipher text message is decrypted into plain text message.

5.9.2.3. User Characteristics

The system can have two kinds of users: Encoder and Decoder. The system is designed to be user friendly and uses an interactive GUI.

Encoder: The encoder is responsible for entering the text message, selecting a key for encryption, selecting block size, and choosing the color channel so that the process of encoding can be performed.

Decoder: The decoder is responsible for decoding the color coded image by entering the correct key, block size, and color channel as provided to him/her by the encoder.

5.9.2.4. General Constraints

- Only text documents or messages can be encoded.
- The image cannot be stored as a hard copy.
- The system must be user friendly.
- Encoding and decoding should be performed perfectly without any loss of information.

5.9.2.5. Assumptions

- The user on either side has basic computer knowledge.
- The user's system will have the necessary hardware and software requirements.

5.9.3. Specific Requirements

5.9.3.1. Functional Requirements

SRS001 Encrypt

The system shall perform proper encryption of the plain text message, provided a valid key is entered.

SRS002 Encode

The system shall perform proper encoding of the encrypted message, provided a valid block size and color channel is selected.

SRS003 Save

The encoder should be able to save the color coded image.

SRS004 Open

The decoder should be able to successfully open the color coded image saved on its system.

SRS005 Decode

The system shall perform proper decoding of the color coded image, provided a valid block size and color channel is selected.

SRS006 Decrypt

The system shall perform proper decryption of the cipher text message, provided a valid key is entered.

5.9.3.2. Design Constraints

SRS007 Operating System

The Operating system should one of the following: Windows XP, Windows 2000, and Windows 7.

SRS008 Software

The users' systems must have Microsoft Visual Studio installed in them.

5.9.3.3. Non Functional Requirements

5.9.3.3.1. Security Parameters

SRS009 Key

The encoder must provide a valid key for encryption of text message and the decoder must provide the same for decryption

SRS010 Block Size and Color Channel

The encoder must provide the block size and color channel for encoding process and the decoder must provide the same for decoding.

5.9.3.3.2. Performance Requirements

SRS011 Response Time

The system shall give responses in 1 second after each process.

SRS012 User Interface

The user interface must be interactive and respond quickly.

5.9.3.3.3. Reliability

SRS013 Availability

The system must be available all the time to the users.

5.9.4. Conclusion

This SRS document is used to give details regarding Color Coded Cryptographic System. In this all the functional and non-functional requirements are specified in order to get a clear cut idea to develop a project.

PROJECT DESIGN

6.1. ALGORITHMS

6.1.1. Algorithm for btnEncode_click()

The plaintext message is encoded into image form after calling this function.

1. Check whether the key field is empty.
   If yes, show alert message “Please Encrypt the Message”.
   Else, continue.
2. Call CCCInit() function for preprocessing.
3. Convert text data into buffer array.
4. Check for the buffer length (must be less than the number of patches).
5. Call Encode() function.
6. Update the picture box.

6.1.2. Algorithm for CCCInit() (Preprocessing) on the Encoding Side

1. Create Image of the size of the picture box.
2. Initialize the block size.
3. Calculate the number of rows and columns and also the number of patches.
   Number of Patches = Number of Rows * Number of Columns
4. PatchXY Array is created to hold the location of a single patch.
5. Fill the patch with the color.

6.1.3. Algorithm for Encode()

1. Read character from buffer array.
2. Compute its ASCII value.
3. Depending on the color channel selected, assign the value to the respective color.
4. If end of buffer, go to step 5. Else, go back to step 1.
5. Fill the remaining patches with the color white.

6.1.4. Algorithm for btnDecode_click()

1. Check whether picture box is empty.
   If yes, show alert message “Please Select Color Coded Image File”.
   Else, continue.
2. Call CCCInit() function for preprocessing.
3. Call Decode() function.
4. Update the text box.

6.1.5. Algorithm for CCCInit() on the Decoding Side

1. Initialize the block size.
2. Calculate the number of rows and columns and also the number of patches.
   Number of Patches = Number of Rows * Number of Columns
3. PatchXY Array is created to hold the location of a single patch.
4. The color of the patch is extracted.

6.1.6. Algorithm for Decode()

1. Create a buffer array of size same as the number of patches.
2. Read patch from the buffer array.
3. Calculate the x and y coordinates of the patch.
4. Depending on the color channel selected, compute the ASCII value of the patch’s color.
5. Compute the character value.
6. If color of the patch is white, then stop. Else, go back to step 2 for the next patch.

6.1.7. Algorithm for btnEncrypt_click()

1. Add the ASCII value of key to that of the message.
2. Convert the resultant ASCII value to character string.
3. Return the processed string.

6.1.8. Algorithm for btnDecrypt_click()

1. Subtract the ASCII value of key from that of the message.
2. Convert the resultant ASCII value to character string.
3. Return the processed string.

IMPLEMENTATION DETAILS

7.1. CODING

7.1.1. Main Form (frmMain.cs)

```
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;

namespace CCC
{
   public partial class frmMain : Form
   {

      public frmMain()
      {
         InitializeComponent();
      }

      private void btnEncode_Click(object sender, EventArgs e)
      {
         frmEncode frm = new frmEncode();
         frm.MdiParent = this;
         frm.WindowState = FormWindowState.Maximized;
         frm.Show();
      }

      private void btnDecode_Click(object sender, EventArgs e)
      {
         frmDecode frm = new frmDecode();
         frm.MdiParent = this;
         frm.WindowState = FormWindowState.Maximized;
         frm.Show();
      }


   }
}
```

7.1.2. Form for Encoding (frmEncode.cs)

```
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;

namespace CCC
{
   public partial class frmEncode : Form
   {
      //System Variables-----------------------------------------------------------------------

      Bitmap Img;

      int BlockSize;

      int Rows;
      int Cols;
      int Patches;
      int[,] PatchXY;

      string StrEncypted;

      char[] Buff;


      //-------------------------------------------------------------------------------------
      public frmEncode()
      {
         InitializeComponent();
      }

      /// <summary>
      /// ////////////////////////////////////////////////////////////////////////////////
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void btnEncode_Click(object sender, EventArgs e)
      {
         if (StrEncypted == null)
         {
            MessageBox.Show("Please Encrypt the Message !");
            return;
```

```
        }

        CCCInit();

        Buff = StrEncypted.ToCharArray();

        if (Patches < Buff.Length)
        {
            MessageBox.Show("Please Reduce the Length of DATA !");
            return;
        }

        Encode();

        pic.Image = Img;
    }
    //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    /// <summary>
    /// //////////////////////////////////////////////////////////////////////////////
    /// </summary>
    private void CCCInit()
    {
        Img = new Bitmap(pic.Width, pic.Height);

        BlockSize = int.Parse(txtBlockSize.Text);

        Rows = Img.Height / BlockSize;
        Cols = Img.Width / BlockSize;
        Patches = Rows * Cols;

        PatchXY = new int[Patches, 2];

        for (int r = 0; r < Rows; r++)
        {
            for (int c = 0; c < Cols; c++)
            {
                PatchXY[(r * Cols) + c, 0] = r * BlockSize;
                PatchXY[(r * Cols) + c, 1] = c * BlockSize;
            }
        }

    }
    //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    /// <summary>
    /// //////////////////////////////////////////////////////////////////////////////
    /// </summary>
    private void Encode()
    {
        //encoding the Buff----------------------------------------------------------------------
        for (int idx = 0; idx < Buff.Length; idx++)
```

```
            {
               for (int x = PatchXY[idx, 0]; x < PatchXY[idx, 0] + BlockSize; x++)
               {
                  for (int y = PatchXY[idx, 1]; y < PatchXY[idx, 1] + BlockSize; y++)
                  {

                     if (optRed.Checked)
                     {
                        Img.SetPixel(y, x, Color.FromArgb(Buff[idx],
int.Parse(txtGreen.Text), int.Parse(txtBlue.Text)));
                     }
                     if (optGreen.Checked)
                     {
                        Img.SetPixel(y, x, Color.FromArgb(int.Parse(txtRed.Text), Buff[idx],
int.Parse(txtBlue.Text)));
                     }
                     if (optBlue.Checked)
                     {
                        Img.SetPixel(y, x, Color.FromArgb(int.Parse(txtRed.Text),
int.Parse(txtGreen.Text), Buff[idx]));
                     }
                  }
               }
            }
            //coloring the remaining image--------------------------------------------------------
            for (int idx = Buff.Length; idx < Patches; idx++)
            {
               for (int x = PatchXY[idx, 0]; x < PatchXY[idx, 0] + BlockSize; x++)
               {
                  for (int y = PatchXY[idx, 1]; y < PatchXY[idx, 1] + BlockSize; y++)
                  {
                       Img.SetPixel(y, x, Color.FromArgb(255, 255, 255));
                  }
               }
            }
         }
        //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
        /// <summary>
        /// ////////////////////////////////////////////////////////////////////////////////////////////
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnSave_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.InitialDirectory =
Convert.ToString(Environment.SpecialFolder.MyDocuments);
            saveFileDialog1.Filter = "JPEG (*.JPG)|*.jpg|All Files (*.*)|*.*";
            saveFileDialog1.FilterIndex = 1;
```

```
            if (pic.Image != null)
            {
                if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    pic.Image.Save(saveFileDialog1.FileName);
                    MessageBox.Show("Image Saved Scuccessfully.");
                }
            }
        }

        private void btnEncrypt_Click(object sender, EventArgs e)
        {

            int messageCount, keyCount, value = 0;

            StringBuilder strRetVal = new StringBuilder();
            if (txtMsg.Text.Length > 0 && txtKey.Text.Length > 0)
            {
                try
                {
                    for (messageCount = 0, keyCount = 0; messageCount <
txtMsg.Text.Length; messageCount++, keyCount++)
                    {
                        if (keyCount >= txtKey.Text.Length) keyCount = 0; // Rotates the Key
string counter to Zero when character value is traversed till end
                        value = (txtMsg.Text[messageCount]) + (txtKey.Text[keyCount]);
//When Encryption is required we add ascii value of key with Message
                        strRetVal.Append((char)(value)); // Here we convert the ascii value to
character and Append that to Stringbuilder
                    }
                }
                catch (Exception ex)
                {
                    // Clears the StringBuilder and Append Error Message in it.
                    strRetVal.Remove(0, strRetVal.Length);
                    strRetVal.Append("Error in Ecrypting:-" + ex.Message);
                }
            }
            else
            {
                // Append Return message with proper valid string required.
                strRetVal.Append("Enter valid Message and Key");
            }

            StrEncypted = strRetVal.ToString(); //Returns the Processed String.
            txtEncrypted.Text = StrEncypted;
        }
        //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    }
}
```

7.1.3. Form for Decoding (frmDecode.cs)

```
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;

namespace CCC
{
   public partial class frmDecode : Form
   {
      //System Variables-----------------------------------------------------------------------

      Bitmap Img;

      int BlockSize;

      int Rows;
      int Cols;
      int Patches;
      int[,] PatchXY;

      //string StrEncypted;

      char[] Buff;

      //-------------------------------------------------------------------------------------

      public frmDecode()
      {
         InitializeComponent();
      }
      /// <summary>
      /// ////////////////////////////////////////////////////////////////////////////////
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      private void btnOpen_Click(object sender, EventArgs e)
      {
         OpenFileDialog dlg = new OpenFileDialog();

         dlg.Title = "Open Image";
         dlg.Filter = "JPEG files (*.JPG)|*.JPG";
```

```
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        Img = new Bitmap(dlg.FileName);

        pic.Image = Img;
    }

    dlg.Dispose();
}
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
/// <summary>
/// ////////////////////////////////////////////////////////////////////////////////////////
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnDecode_Click(object sender, EventArgs e)
{
    if (pic.Image == null)
    {
        MessageBox.Show("Please Select Color Coded Image File !");
        return;
    }

    CCCInit();

    Decode();

    txtDecrypted.Text = new string(Buff);

}
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
/// <summary>
/// ////////////////////////////////////////////////////////////////////////////////////////
/// </summary>
private void CCCInit()
{

    BlockSize = int.Parse(txtBlockSize.Text);

    Rows = Img.Height / BlockSize;
    Cols = Img.Width / BlockSize;
    Patches = Rows * Cols;

    PatchXY = new int[Patches, 2];

    for (int r = 0; r < Rows; r++)
    {
        for (int c = 0; c < Cols; c++)
        {
            PatchXY[(r * Cols) + c, 0] = r * BlockSize;
```

```
            PatchXY[(r * Cols) + c, 1] = c * BlockSize;
        }
    }
}
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
/// <summary>
/// ////////////////////////////////////////////////////////////////////////////////
/// </summary>
private void Decode()
{
    Buff = new char[Patches];

    int idx;
    for (idx = 0; idx < Patches; idx++)
    {
        int x = PatchXY[idx, 0] + BlockSize / 2;
        int y = PatchXY[idx, 1] + BlockSize / 2;

        if (optRed.Checked)
        {
            Buff[idx] = (char)Img.GetPixel(y, x).R;
        }
        if (optGreen.Checked)
        {
            Buff[idx] = (char)Img.GetPixel(y, x).G;
        }
        if (optBlue.Checked)
        {
            Buff[idx] = (char)Img.GetPixel(y, x).B;
        }

        if (Buff[idx] == (char)(255)) break;
    }

    Buff[idx] = '\0';

}

private void btnDecrypt_Click(object sender, EventArgs e)
{
    int messageCount, keyCount, value = 0;

    if (txtDecrypted.Text == null)
    {
        MessageBox.Show("Please Decode the Message First !");
        return;
    }

    StringBuilder strRetVal = new StringBuilder();
    if (txtDecrypted.Text.Length > 0 && txtKey.Text.Length > 0)
```

```
            {
                try
                {
                    for (messageCount = 0, keyCount = 0; messageCount <
txtDecrypted.Text.Length; messageCount++, keyCount++)
                    {
                        if (keyCount >= txtKey.Text.Length) keyCount = 0; // Rotates the Key
string counter to Zero when character value is traversed till end
                        value = (txtDecrypted.Text[messageCount]) - (txtKey.Text[keyCount]);
//When Encryption is required we add ascii value of key with Message
                        strRetVal.Append((char)(value)); // Here we convert the ascii value to
character and Append that to Stringbuilder
                    }
                }
                catch (Exception ex)
                {
                    // Clears the StringBuilder and Append Error Message in it.
                    strRetVal.Remove(0, strRetVal.Length);
                    strRetVal.Append("Error in Ecrypting:-" + ex.Message);
                }
            }
            else
            {
                // Append Return message with proper valid string required.
                strRetVal.Append("Enter valid Message and Key");
            }

            txtMsg.Text = strRetVal.ToString(); //Returns the Processed String.

        }
        //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    }
}
```

7.2. SNAPSHOTS

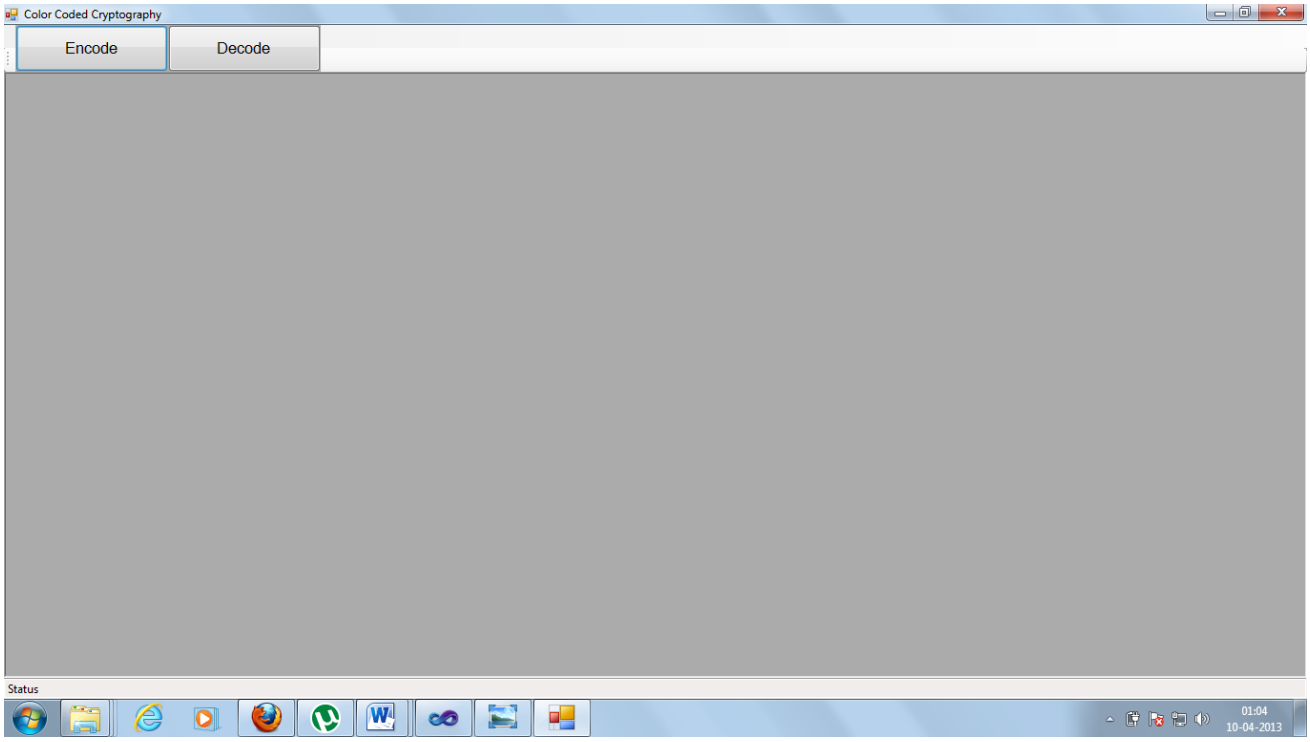

Figure 16: Main Form

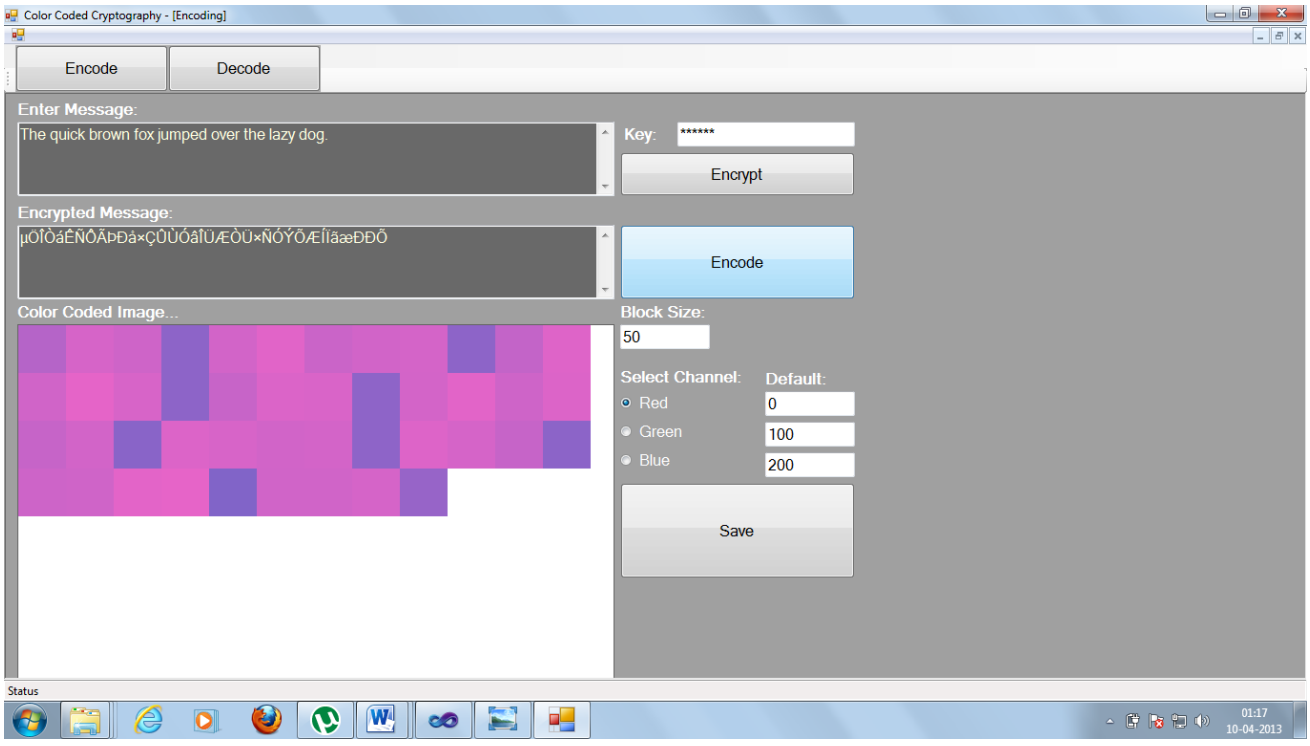

Figure 17: Encoding Form

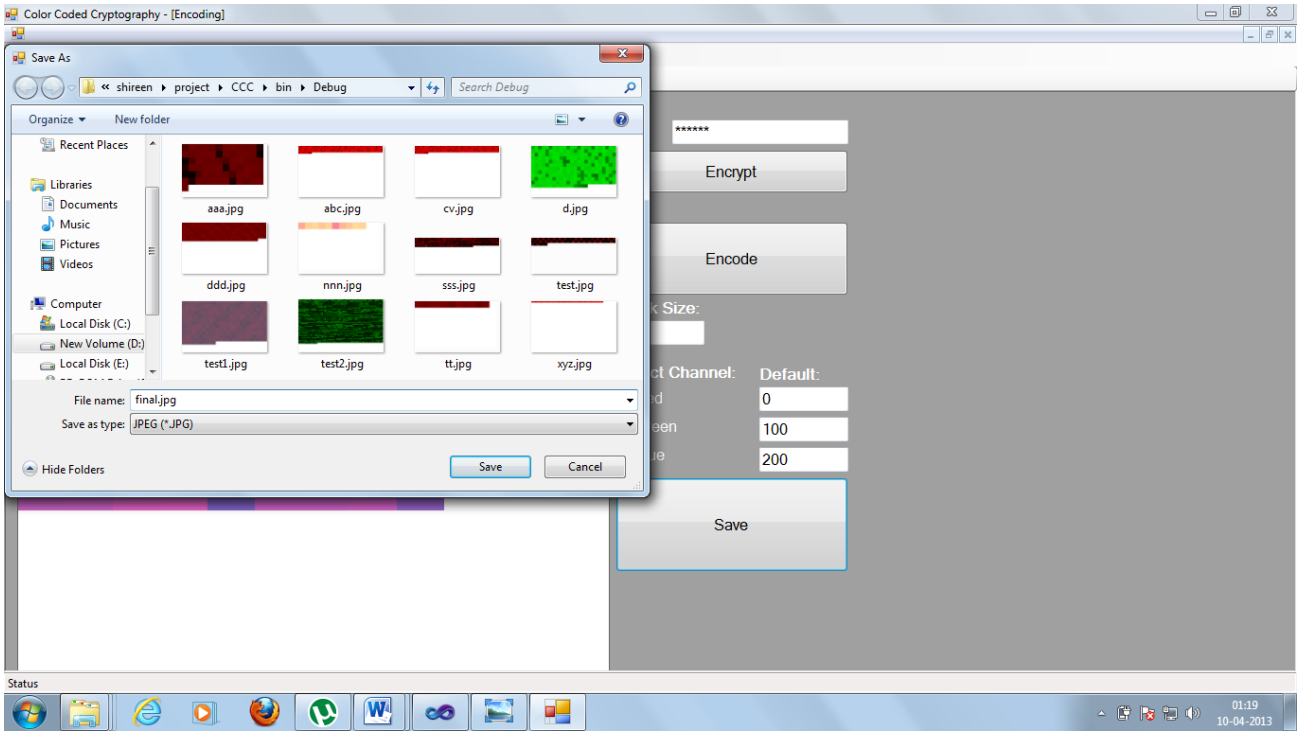

Figure 18: Save Dialog Box

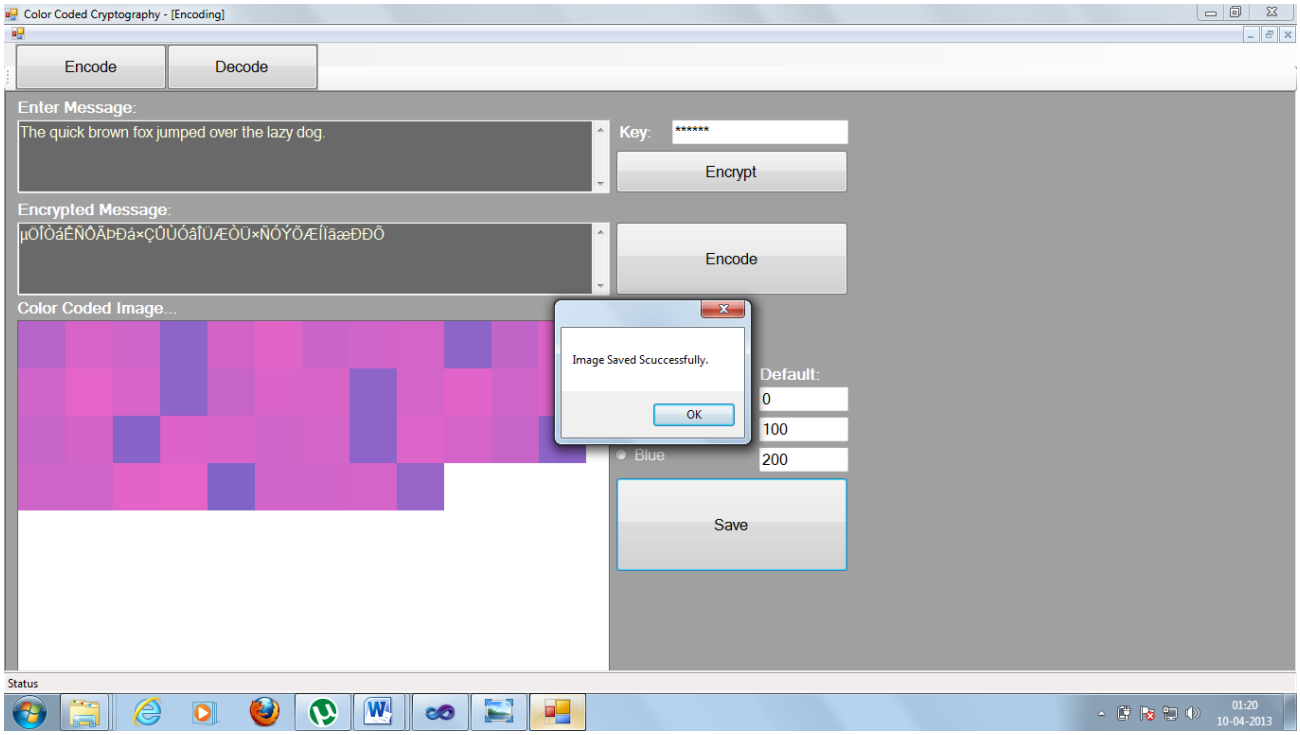

Figure 19: Image saved successfully

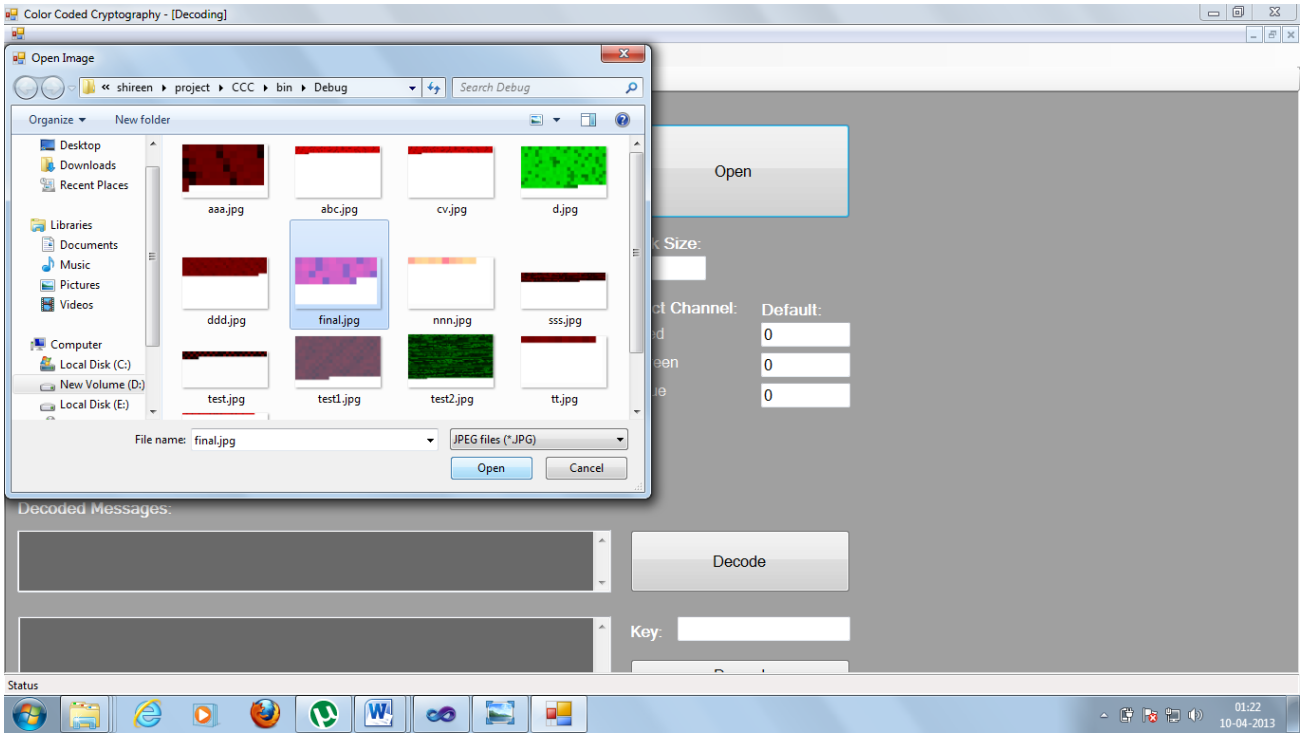

Figure 20: Open Dialog Box

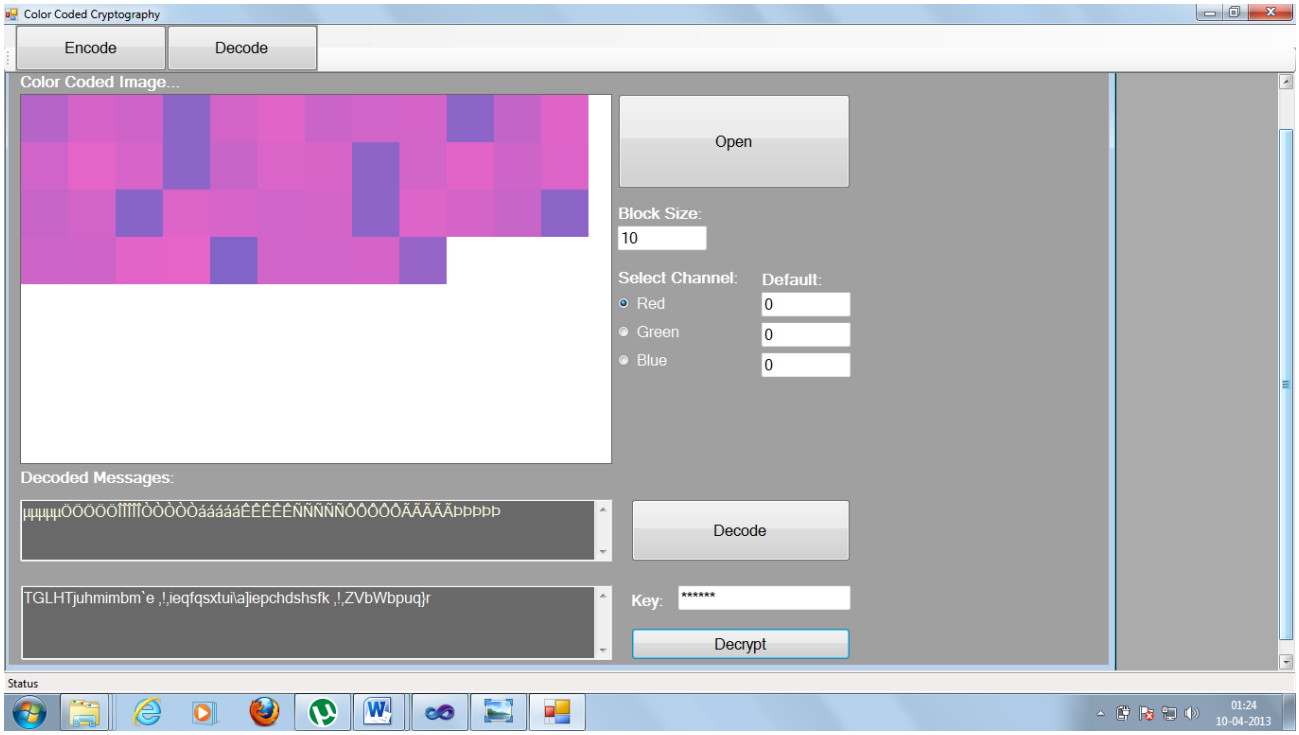

Figure 21: Output when block size entered is not same as used by encoder

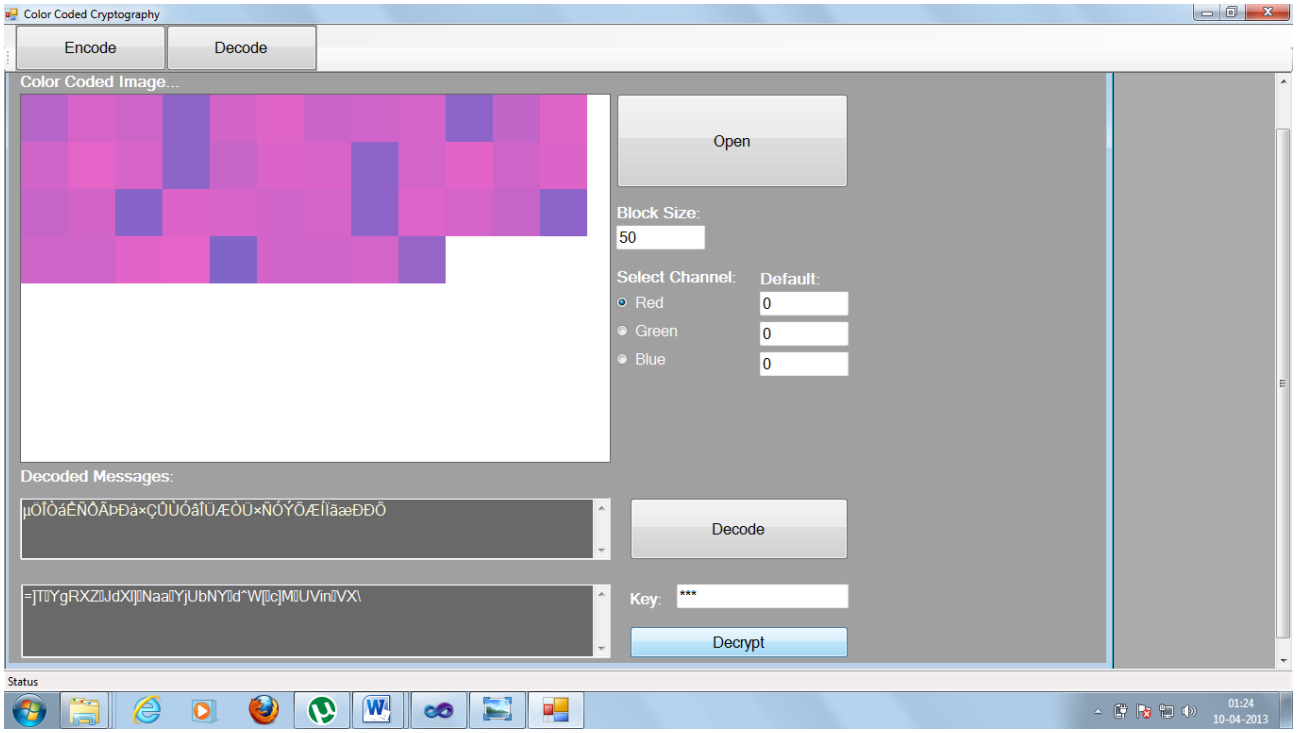

Figure 22: Output when key entered is not same as used by encoder

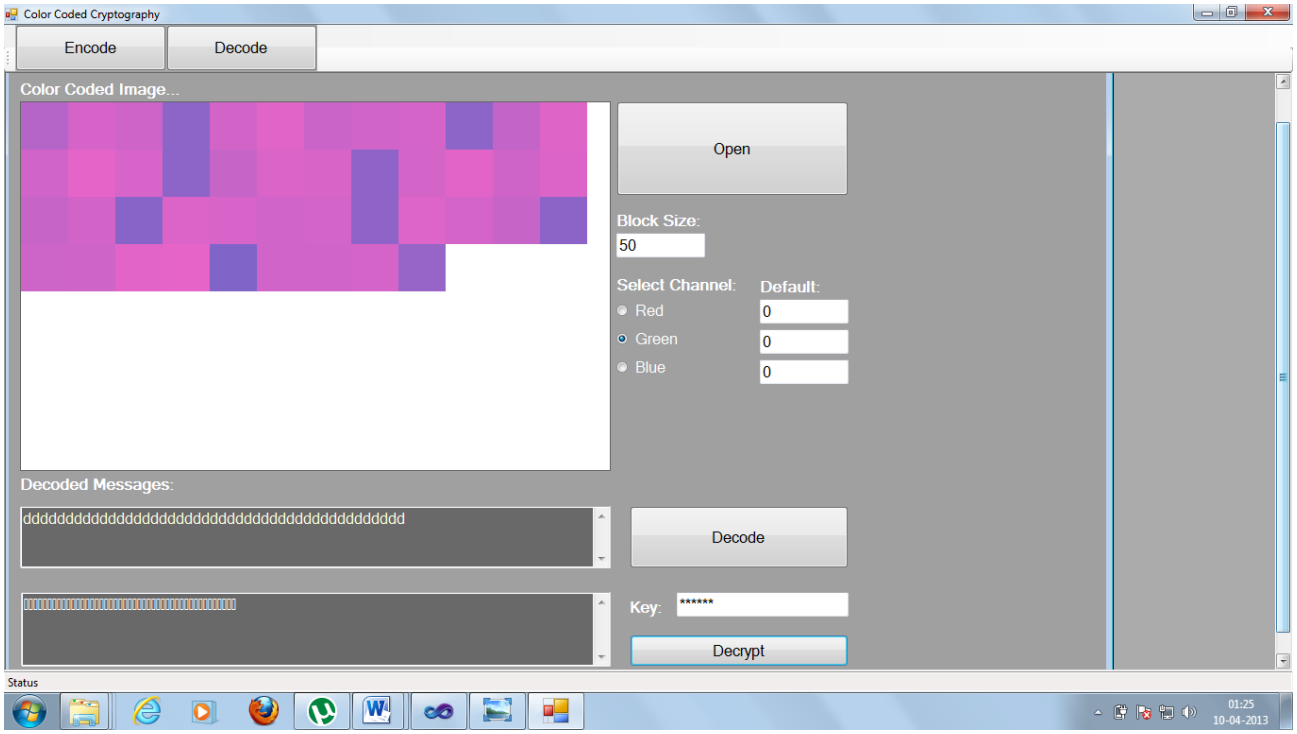

Figure 23: Output when color channel selected is not same as selected by encoder

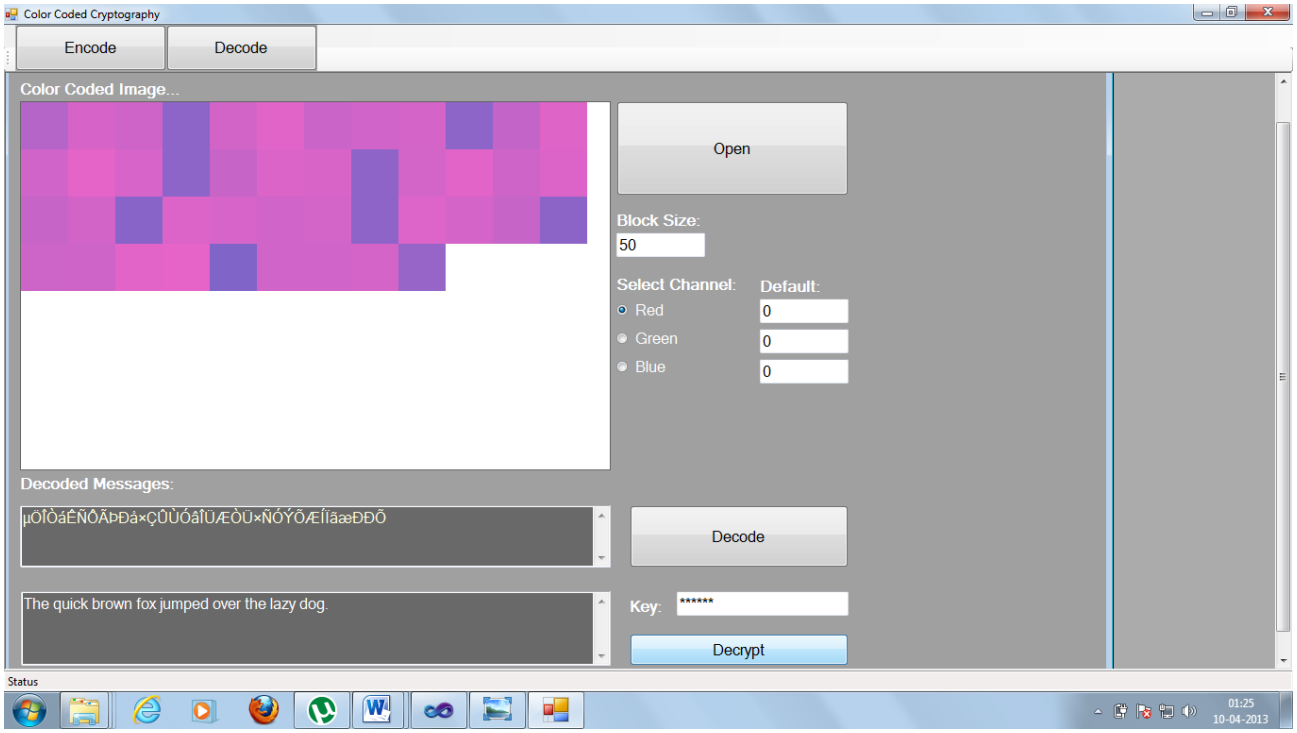

Figure 24: Output when all the three security parameters are entered correctly

TECHNOLOGIES USED

8.1. MICROSOFT VISUAL STUDIO 2010

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop console and graphical user interface applications along with Windows Forms or WPF applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

Visual Studio provides a suite of component-based development tools and other technologies that you can use to build powerful, high-performance applications. In addition, Visual Studio is optimized for team-based design, development, and deployment of enterprise solutions.

Visual Studio supports different programming languages by means of language services, which allow the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010). Support for other languages such as M, Python, and Ruby among others is available via language services installed separately. It also supports XML/XSLT, HTML/XHTML, JavaScript and CSS. Individual language-specific versions of Visual Studio also exist which provide more limited language services to the user: Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

8.2. PROGRAMMING LANGUAGE USED: C#

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO

(ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

C# is intended to be a simple, modern, general-purpose, object-oriented programming language. Its development team is led by Anders Hejlsberg. The most recent version is C# 5.0, which was released on August 15, 2012.

Microsoft Visual C# is Microsoft's implementation of the C# specification, included in the Microsoft Visual Studio suite of products. While multiple implementations of the specification exist, Visual C# is by far the one most commonly used.

TESTING

9.1. TEST PLAN

The system has been tested manually according to the following test plan:

Table 1: Test Plan

Test module	Features to be tested	Responsible member	Testing tools
Main form	1. Connectivity to succeeding forms.	Supriya Adep	Microsoft Visual Studio 2010
Encoding Form	1. Proper Encryption of message. 2. Alert messages on not entering key or message. 3. Proper encoding of the message. 4. Does the output come any different if any of the three parameters are changed?	Shireen Munawar	Microsoft Visual Studio 2010
Decoding Form	1. Proper Decryption of message. 2. Alert message on not opening any color coded image. 3. Proper decoding of the message. 4. Does the system produce wrong output if any of the three parameters do not match those entered by the selector?	Pooja Nair	Microsoft Visual Studio 2010

9.2. TEST CASES

Test Case No. 1

Step Description: Click Encode in main form

Expected Result: Encoding form should appear

Test Case No. 2

Step Description: Click Decode in main form

Expected Result: Decoding form should appear

Test Case No. 3

Step Description: Keep key field empty

Expected Result: Alert message “Please Enter Valid Key”

Test Case No. 4

Step Description: Keep message field empty

Expected Result: Alert message “Please Enter Valid Message”

Test Case No. 5

Step Description: Click Encrypt in encoding form

Expected Result: Message should get encrypted

Test Case No. 6

Step Description: Click Encode in encoding form

Expected Result: Color coded image should appear

Test Case No. 7

Step Description: Increase/ decrease block size

Expected Result: Size of patch should change accordingly

Test Case No. 8

Step Description: Change color channel and change color weights

Expected Result: Color of patches should change accordingly

Test Case No. 9

Step Description: Click Save in encoding form

Expected Result: The image should get saved

Test Case No. 10

Step Description: Keep picture box empty

Expected Result: Alert message “Please open color coded image”

Test Case No. 11

Step Description: Click Open in decoding form

Expected Result: Color coded image should be opened

Test Case No. 12

Step Description: Enter wrong key, block size, or color channel in decoding form

Expected Result: Wrong message should be seen

CONCLUSION AND FUTURE SCOPE

Thus, we have been successful in building an innovative cryptosystem for text documents. The system correctly encrypts and encodes the text message into color coded image on the encoding side. Conversely, it also successfully decrypts and decodes the color coded image into plain text message. Also, we have tested that if the decoder does not provide the correct key, block size, or color channel, he would not be privileged to see the correct message.

Depending on the availability of time and resources, we hope to work on the future enhancements for this system. With the assistance of a good printer and high calibrated scanner, this system can be extended to incorporate a hard copy version of the encrypted information which can be scanned at the receiver's end using a high-end scanner. Another extension can be the use of the system on other formats of files such as audio and video files, images etc.

REFERENCES

[1] www.wikipedia.org

[2] "Color Coded Cryptography", Aditya Gaitonde, International Journal of Scientific & Engineering Research, Volume 3, Issue 7, July-2012

[3] "Literature Review of Cryptography and its Role in Network Security Principles and Practice", Daniel Lloyd Calloway, Capella University, OM8302, § 4, 8 September 2008

CHAPTER 12

APPENDIX