P-1820

VIDEO STEGANOGRAPHY – HIDING A TEXT FILE IN A VIDEO FILE

A PROJECT REPORT

Submitted by

ARUNSHANKARAN.K	71203104004
B.MOHANRAJ	71203104020
RAJESH KUMAR.S.R	71203104028

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2007

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**VIDEO STEGANOGRAPHY – HIDING A TEXT FILE IN A VIDEO FILE**" is the bona fide work of "**ARUNSHANKARAN.K, MOHANRAJ.B, RAJESH KUMAR.S.R** who carried out the project under my supervision.

[illegible]	[illegible] 23/04/2007
SIGNATURE	SIGNATURE
Prof. S. Thangasamy, Ph.D,	Mr.K.Sivan Arul Selvan, M.E,
HEAD OF THE DEPARTMENT,	SUPERVISOR,
	Senior Lecturer,
Department of Computer Science and Engineering,	Department of Computer Science and Engineering,
Kumaraguru College of Technology,	Kumaraguru College of Technology,
Coimbatore – 641 006.	Coimbatore – 641 006.

The candidates with University Register Nos.: **71203104004,71203104020,71203104028** were examined by us in the project viva-voce examination held on 25/04/2007

[illegible] 25·4

INTERNAL EXAMINER

[illegible] 25/4/07

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere thanks to our chairman **Arutselvar Dr. N. Mahalingam, B.Sc, F.I.E.** and correspondent Prof. **K. Arumugam, B.E., M.S., M.I.E.,** for all their support and ray of strengthening hope extended.

We are immensely grateful to our principal **Dr. Joseph V. Thanikal, M.E., Ph.D., PDF., CEPIT.,** for his invaluable support to the come outs of this project.

We are extremely thankful to **Dr. S. Thangasamy,Ph.D.,** Head of the Department, Department of Computer Science and Engineering for his valuable advice and suggestions throughout this project.

We are indent to express our heartiest thanks to **Ms. S. Rajini, B.E.,** project coordinators who have helped us to perform the project work extremely well.

We are indent to express our heartiest thanks to **Mr. K.Sivan Arul Selvan, M.E.,** project guide who rendered his valuable guidance and support to perform our project work extremely well.

We are also thankful to all the faculty members of the department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore for their valuable guidance, support and encouragement during the course of our project work.

We express our humble gratitude and thanks to our parents and family members who have supported and helped us to complete the project and our friends, for lending us valuable tips, support and co-operation throughout our project work.

Above all we would like to thank **God**, The Almighty for showering his blessings on us to successfully implement the project.

ABSTRACT

Steganography is the art of hiding data. It is a fast emerging field and is being used nowadays in large context. Recent technique involves hiding a text data, audio file, picture file in a picture file (usually JPEG). JPEG operates in transform space unlike GIF or BMP which operates in structural space. So visually no changes can be seen in JPEG but it can be seen in GIF. In JPEG technique the redundant bits are found and they are replaced by the message bits. This is encoded with a secret key so that no one else can see it. But what usually happens is that the file size changes sometimes so that existence of hidden data is found out. It can also be found out if any suspicious images are sent.

In our project we are implementing the idea of hiding text file in a video file using Least Significant Bit (LSB) Technique. First the Video file in which the text has to be hidden is taken (Overt File). Next the text file which is to be hidden is taken. The text file is encrypted using Vignere's algorithm and embedded in the overt file. The sender and the receiver have an application that is used to cover and uncover the text file. The sender enters the password in the interface and the overt file is sent. To remove the Covert file the password is given in the interface and the text file is obtained.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Steganography is the art of hidden or covered writing. This is in practice since ancient times, for secret communication between the allies during war times. Since the enemies should be kept unaware of such communications, the message is passed by some hidden means ensuring no suspicion.

Steganography and encryption are both used to ensure data confidentiality. However the main difference between them is that with encryption anybody can see that both parties are communicating in secret. Steganography hides the existence of a secret message and in the best case nobody can see that both parties are communicating in secret. This makes steganography suitable for some tasks in which encryption aren't, such as copyright marking. Adding encrypted copyright information to a file could be easy to remove but embedding it within the contents of the file itself can prevent it being easily identified and removed.

Steganography provides a means of secret communication which cannot be removed without significantly altering the data in which it is embedded. The embedded data will be confidential unless an attacker can find a way to detect it. Due to availability of the images in internet and many digital media, there is a serious threat to the images from the digital thieves, etc. As a consequence, the ownership of the image might be misinterpreted. In this context, research work is needed to resolve rightful ownership. The owner should be able to hide some information in the image and when needed he/she should be able to extract that information to prove his ownership.

1.1 EXISTING SYSTEM

Steganography has been implemented in two different ways, based on the cover medium used for hiding.

1.1.1 Images as cover medium:

The secret message, either as text or image, is hidden behind an image file. The image file used as cover may be BMP, GIF, or JPEG file. In case of hiding an image under another image, the size should be proportionate. i.e., the carrier file should be eight times the size of the file to be hidden. The principle used for hiding is "LSB Insertion Technique". Here, the Least Significant bit of the cover medium is altered to contain the secret message bit. Altering any other bit to contain the message would affect the quality of the cover image.

1.1.2. Audio file as cover medium:

The existing software can be used to hide the text messages or images (.gif, .bmp, .jpeg, etc.) in audio stream. As human ear is not very sensitive to detect small changes in the audio, the noise of the audio stream is used to hide the secret message bits.

1.2 PROPOSED SYSTEM AND ITS ADVANTAGES:

The proposed software hides a text file (TXT) in a video file (Audio Video Interleaved). The message is hidden such that it is not visible to the human eye. Only the authenticated person can extract the hidden information.

The following are the advantages of the proposed system.

- Secret Communication – The message in the form of text file can be communicated secretly to the receiver without the knowledge of the third person.
- Enhanced Security – The text file which is being hidden in the video file is encrypted using Vignere's Algorithm. This gives added security apart from hiding in the video file.
- Untraceable video file – The source video file is shot personally to make steganalysis difficult and eliminates comparisons on size and quality which is possible in the case of popular videos available on the web.

2. PROPOSED DEVELOPMENT METHODOLOGY

The proposed system can hide secret text messages in video files (AVI). The bitmap files are then embedded with the text file bits using Least Significant Bit (LSB) technique and converted into AVI frames. Even if the video is intercepted and again split into frames by a third person, the secret text message cannot be retrieved as it is encrypted using secret key and password. Hence additional security is provided. The password and the key file are provided for extraction, and then decryption is done and the secret message is retrieved.

3. PROPOSED APPROACH

The software is used to hide the text file in the video file. AVI files can contain one or more streams of four different types (Video, Audio, Midi and Text). The implementation involves two main processes – Hiding and Extracting the secret text from the carrier file (the video file in which the secret text has been embedded).

During Hiding, the video stream of the AVI file is obtained and frames are decompressed using AVI library functions. The AVI frames are converted into BMP files. The application can use the extracted bitmaps just like any other image file. It extracts the first non-header frame to a temporary file, opens it and hides a part of the message. Then it writes the resulting bitmap to a new video stream, and continues with the next frame. After the last frame the application closes both video files and deletes the temporary bitmap files.

During Extraction, the application opens the AVI carrier file, the individual frames are obtained and the hidden secret message is extracted. Since compression destroys the hidden message by changing colors and sound and would make the files even larger, uncompressed AVI files are used.

4. PROGRAMMING ENVIRONMENT

4.1 HARDWARE REQUIREMENTS:

Processor: Intel Pentium 4

Hard disk: 80 GB

System RAM: 256 MB

Processor Speed: 3.02 GHz

4.2 SOFTWARE REQUIREMENTS:

Operating System: Windows XP

Language: C#, .NET Framework

5. SYSTEM REQUIREMENTS

5.1 PURPOSE:

The software is used for the communication of secret text without the knowledge of the interceptor.

5.2 PROBLEM STATEMENT:

Hiding and extraction of a text file in a video file.

5.3 SCOPE:

This software embeds the text file (TXT) in a video file (AVI). The text is hidden in such a way that it will not be visible to the normal users and leaves not even intimation to the hackers and get transmitted in a normal way. The authorized person can decode the file and extract the text.

5.4 SPECIFIC REQUIREMENTS:

5.4.1 Functional Requirements:

5.4.1.1 Embedding:

To hide the text file contents into a video file to provide secret communication.

Input: Text file, original video file.

Output: Covert video file.

5.4.1.2 Retrieving:

To extract the text file contents from the covert video file.

Input: Covert video file.

Output: Text file contents, original video file.

5.4.2 Performance Requirements:

5.4.2.1 Invisibility:

The text file contents are hidden in such a way that data quality is not degraded and attackers are prevented from finding and deleting it.

5.4.2.2 Security:

The text file contents which is being hidden in the video is previously encrypted using Vignere's algorithm which provides an added security.

5.4.2.3 Robustness:

The use of sounds, images and video signals in digital form commonly involves many types of distortions, such as lossy compression, or in this case, changes in the contents of the hidden text. Steganography tools must grant that the embedded information is not removed by interception or any other attack.

5.4.2.4 Invertibility:

It is said to be invertible if authorized users can extract the contents of the text file from the Covert video file.

6. SYSTEM DESIGN

6.1 READING

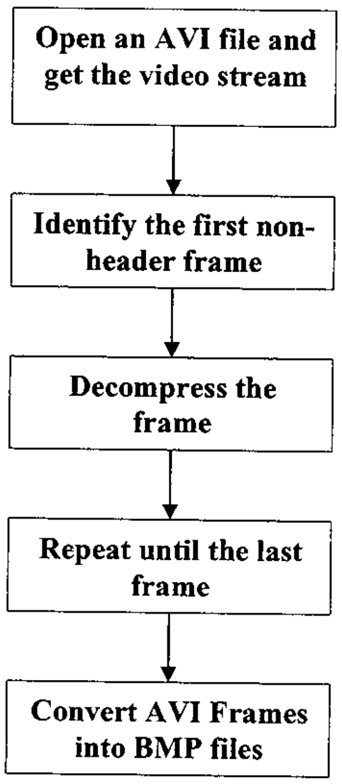

Open an AVI file and get the video stream
Identify the first non-header frame
Decompress the frame
Repeat until the last frame
Convert AVI Frames into BMP files

Fig. 6.1 Flowchart for Reading

6.2 HIDING

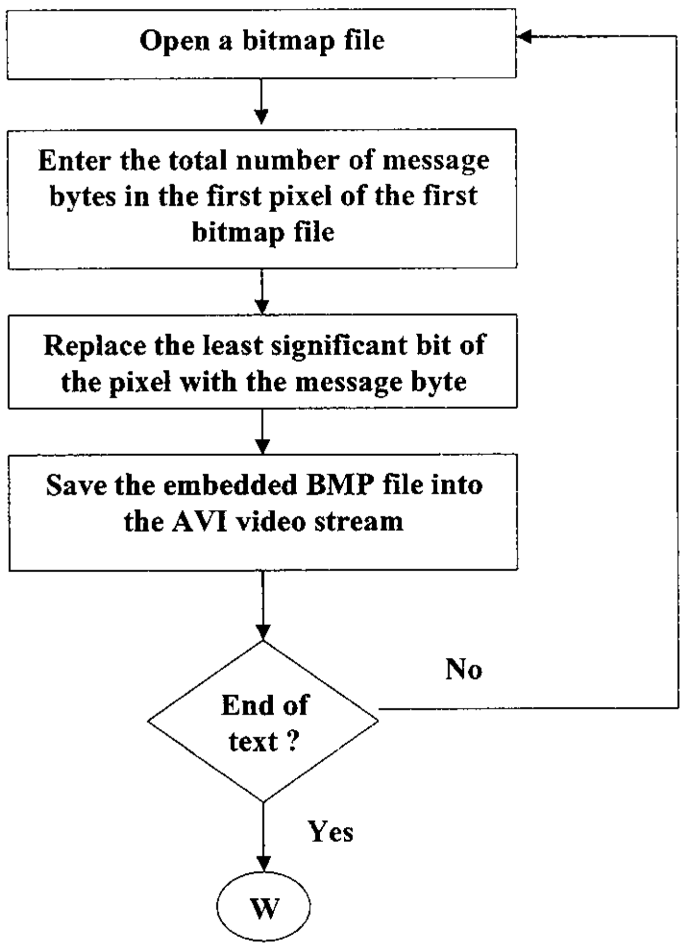

Open a bitmap file
Enter the total number of message bytes in the first pixel of the first bitmap file
Replace the least significant bit of the pixel with the message byte
Save the embedded BMP file into the AVI video stream
End of text ?
No
Yes
W

Fig. 6.2 Flowchart for Hiding

6.3 WRITING

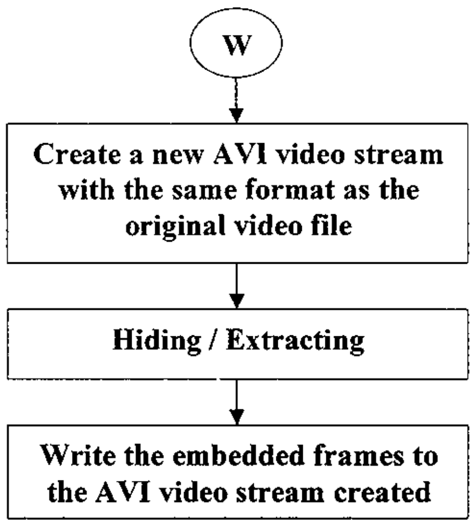

W
Create a new AVI video stream with the same format as the original video file
Hiding / Extracting
Write the embedded frames to the AVI video stream created

Fig. 6.3 Flowchart for Writing

6.4 EXTRACTING

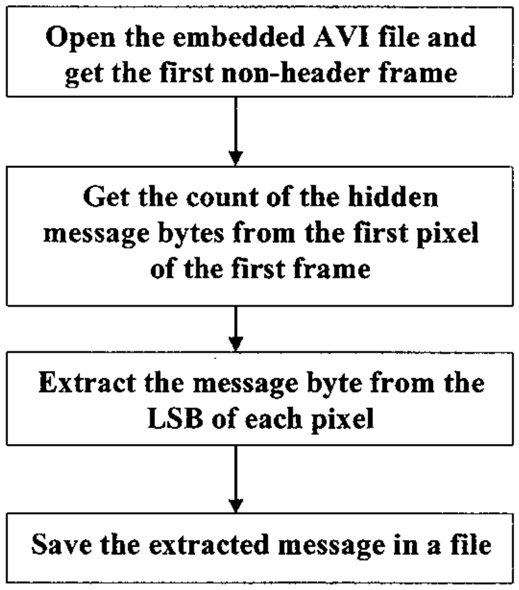

Open the embedded AVI file and get the first non-header frame
Get the count of the hidden message bytes from the first pixel of the first frame
Extract the message byte from the LSB of each pixel
Save the extracted message in a file

Fig. 6.2 Flowchart for Extracting

7. DETAILED DESIGN

Modules:

1. Reading and Splitting
2. Encryption
3. Hiding
4. Writing
5. Extracting

7.1 READING AND SPLITTING:

In this module, the carrier AVI file specified by the user is opened and the video stream is extracted. Then, the video file is split into frames and converted into BMP files. The steps involved are:

- Initialize the AVI library
- Open an AVI file
- Get a stream from an open AVI file
- Identify the beginning of the first non-header frame
- Decompress the frames
- Construct a BMP file header for AVI – BMP conversion
- Copy the image data of the frame into the temporary BMP file.
- Release an open AVI stream
- Release an open AVI file
- Close the AVI library

7.2 ENCRYPTION:

VIGNERE'S ENCRYPTION ALGORITHM:

- The encipherer chooses a piece of plaintext:

  e.g.: VIGENERE

- The encipherer chooses a keyword

  e.g.: CIPHCIPH

- To encipher letter L1 of the plaintext, the encipherer creates a new alphabet wherein A is shifted to letter L1 of the ciphertext, B is shifted to the next letter, etc.:

  ABCDEFGHIJKLMNOPQRSTUVWXYZ -->
  CDEFGHIJKLMNOPQRSTUVWXYZAB

- The encipherer finds the letter that corresponds to L1 in the new alphabet.

  This is now L1 of the plaintext:

  V --> X

- This is repeated for each letter in the plaintext and its corresponding letter in the key:

  VIGENERE + CIPHCIPH --> XQVLPMFL

7.3 HIDING:

The hiding involves opening the BMP files and embedding the secret text in them. The steps involved are:

- Open the BMP file
- Calculate the position of the pixel to hide the message byte.
- Replace the least significant bit of the pixel with the message byte.

- After all the frames have been processed, write the embedded BMPs to the newly created AVI file (Carrier Video).

Least Significant Bit (LSB) Insertion Method:

Pixels of the Cover medium:

```
00010010 10001000 00101100
01011100 01010101 11001100
00001111 10101010 00101111
```

Let character to be hidden be 'A'

ASCII of A: 0100 0001

Result:

Hiding is done in the last bit. For hiding 1 byte, 8 bytes are required.

0001001**0** 1000100**1** 0010110**0**

0101110**0** 0101010**0** 1100110**0**

0000111**0** 1010101**1** 00101111

7.4 WRITING:

When the application opens an AVI carrier file, it creates another AVI file for the resulting bitmaps. The new video stream must have the same size and frame rate as the original stream. And, the BMPs with embedded text are written to the newly created video stream.

Steps involved:

- Create a new AVI file
- Create a new stream in the new AVI file
- Write the BMPs with embedded text, one by one as is processed, into the stream.
- Delete the temporary BMP.

7.5 EXTRACTING:

The extracting process involves opening the carrier AVI file and retrieving the secret text hidden. The steps involved are:

- Get the frames with embedded text, one by one, using the reading module.
- Decrypt using the same key file.
- Calculate the position of the pixel to extract the next message byte.
- Get the message byte from the calculated position and store it in a separate file.

8. FUTURE ENHANCEMENTS

The video steganography software is used for secret communication in the form of text which is embedded in the video file. The software works on LSB technique for hiding the text file in the video file which replaces the least significant bit of each pixel. In the future, it can be extended to hide secret messages in other formats such as images, audio and video files in a video file. Currently the software application works on AVI files alone as the overt file. It can be extended to other video formats such as MPEG and DAT as the video file.

9. CONCLUSION

Video Steganography has enabled embedding of secret text in video files. This enables the communication of secret text without the knowledge of someone intercepting it. Even if the interceptor detects the embedding of the text, the text cannot be comprehended because it has been encrypted with a key and password. So it provides a higher degree of secrecy and integrity. Only the authorized users can decrypt it using the key and the password, which is unique.

APPENDICES

APPENDIX 1

SAMPLE CODE:

Form1.cs

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

namespace PictureKey
{
	public class frmMain : System.Windows.Forms.Form
	{
		private System.Windows.Forms.GroupBox grpPicture;
		private System.Windows.Forms.GroupBox grpKey;
		private System.Windows.Forms.Button btnHide;
		private System.Windows.Forms.Button btnExtract;
		private System.Windows.Forms.TabPage tabPage1;
		private System.Windows.Forms.TabPage tabPage2;
		private System.Windows.Forms.GroupBox grpMessage;
		private System.Windows.Forms.RadioButton rdoMessageText;
		private System.Windows.Forms.TextBox txtMessageFile;
		private System.Windows.Forms.TextBox txtMessageText;
		private System.Windows.Forms.Button btnMessage;
		private System.Windows.Forms.RadioButton rdoMessageFile;
		private System.Windows.Forms.GroupBox groupBox3;
		private System.Windows.Forms.Label label1;
		private System.Windows.Forms.Label label3;
		private System.Windows.Forms.Button btnKeyFile;
		private System.Windows.Forms.TextBox txtExtractedMsgFile;
		private System.Windows.Forms.Button btnExtractedMsgFile;
		private System.Windows.Forms.TextBox txtExtractedMsgText;
		private System.Windows.Forms.Label lblKeyFiles;
		private System.ComponentModel.Container components = null;
		private System.Windows.Forms.Button btnImageFile;
		private System.Windows.Forms.Label lblImageFiles;
		private System.Windows.Forms.GroupBox grpPictureExtract;
		private System.Windows.Forms.Button btntImageFileExtract;
		private System.Windows.Forms.TabControl tabAction;
private System.Windows.Forms.Label lblTextFilesExtract;
private System.Windows.Forms.GroupBox grpSplitBytes;
private System.Windows.Forms.CheckBox chkSplitBytes;

private FilePasswordPair[] keys = new FilePasswordPair[0];
private CarrierImage[] textHide = new CarrierImage[0];
private CarrierImage[] textExtract = new CarrierImage[0];
public frmMain(){
        InitializeComponent();
}


protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if (components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
}


private void InitializeComponent()
{
        this.grpPicture = new System.Windows.Forms.GroupBox();
        this.lblImageFiles = new System.Windows.Forms.Label();
        this.btnImageFile = new System.Windows.Forms.Button();
        this.grpKey = new System.Windows.Forms.GroupBox();
        this.lblKeyFiles = new System.Windows.Forms.Label();
        this.btnKeyFile = new System.Windows.Forms.Button();
        this.btnHide = new System.Windows.Forms.Button();
        this.btnExtract = new System.Windows.Forms.Button();
        this.tabAction = new System.Windows.Forms.TabControl();
        this.tabPage1 = new System.Windows.Forms.TabPage();
        this.grpMessage = new System.Windows.Forms.GroupBox();
        this.rdoMessageText=new System.Windows.Forms.RadioButton();
        this.txtMessageFile = new System.Windows.Forms.TextBox();
        this.txtMessageText = new System.Windows.Forms.TextBox();
        this.btnMessage = new System.Windows.Forms.Button();
        this.rdoMessageFile =new System.Windows.Forms.RadioButton();
        this.tabPage2 = new System.Windows.Forms.TabPage();
        this.grpPictureExtract = new System.Windows.Forms.GroupBox();
        this.lbltextFilesExtract = new System.Windows.Forms.Label();
```

```
                    this.btntImtextFileExtract = new
System.Windows.Forms.Button();
                    this.groupBox3 = new System.Windows.Forms.GroupBox();
                    this.label1 = new System.Windows.Forms.Label();
                    this.txtExtractedMsgFile=new System.Windows.Forms.TextBox();
                    this.txtExtractedMsgText=new System.Windows.Forms.TextBox();
                    this.btnExtractedMsgFile = new System.Windows.Forms.Button();
                    this.label3 = new System.Windows.Forms.Label();
                    this.grpSplitBytes = new System.Windows.Forms.GroupBox();
                    this.chkSplitBytes = new System.Windows.Forms.CheckBox();
                    this.grpPicture.SuspendLayout();
                    this.grpKey.SuspendLayout();
                    this.tabAction.SuspendLayout();
                    this.tabPage1.SuspendLayout();
                    this.grpMessage.SuspendLayout();
                    this.tabPage2.SuspendLayout();
                    this.grpPictureExtract.SuspendLayout();
                    this.groupBox3.SuspendLayout();
                    this.grpSplitBytes.SuspendLayout();
                    this.SuspendLayout();

                    // grpPicture

                    this.grpPicture.Controls.AddRange(new
System.Windows.Forms.Control[]
                    {
                    this.lbltextFiles,
                    this.btnltextFile});
                    this.grpPicture.Location = new System.Drawing.Point(456, 16);
                    this.grpPicture.Name = "grpPicture";
                    this.grpPicture.Size = new System.Drawing.Size(312, 184);
                    this.grpPicture.TabIndex = 0;
                    this.grpPicture.TabStop = false;
                    this.grpPicture.Text = "Carrier Bitmaps";

                    this.lblTextFiles.Location = new System.Drawing.Point(16, 32);
                    this.lblTextFiles.Name = "lblTextFiles";
                    this.lblTextFiles.Size = new System.Drawing.Size(168, 23);
                    this.lblTextFiles.TabIndex = 3;
                    this.lblTextFiles.Text = "No text files specified";
                    //
                    // btnTextFile
                    //
                    this.btnTextFile.Location = new System.Drawing.Point(184, 32);
                    this.btnTextFile.Name = "btnTextFile";
                    this.btnTextFile.Size = new System.Drawing.Size(112, 23);
```

```
                        this.btntextFile.TabIndex = 2;
                        this.btntextFile.Text = "Add/Remove...";
                        this.btntextFile.Click += new
System.EventHandler(this.btntextFile_Click);


                        this.grpKey.Controls.AddRange(new
System.Windows.Forms.Control[] {

                         this.lblKeyFiles,
                         this.btnKeyFile});
                        this.grpKey.Location = new System.Drawing.Point(8, 16);
                        this.grpKey.Name = "grpKey";
                        this.grpKey.Size = new System.Drawing.Size(792, 72);
                        this.grpKey.TabIndex = 1;
                        this.grpKey.TabStop = false;
                        this.grpKey.Text = "Keys";
                        this.lblKeyFiles.Location = new System.Drawing.Point(16, 32);
                        this.lblKeyFiles.Name = "lblKeyFiles";
                        this.lblKeyFiles.Size = new System.Drawing.Size(152, 23);
                        this.lblKeyFiles.TabIndex = 3;
                        this.lblKeyFiles.Text = "No key files specified";
                        this.btnKeyFile.Location = new System.Drawing.Point(168, 32);
                        this.btnKeyFile.Name = "btnKeyFile";
                        this.btnKeyFile.Size = new System.Drawing.Size(112, 23);
                        this.btnKeyFile.TabIndex = 2;
                        this.btnKeyFile.Text = "Add/Remove...";
                        this.btnKeyFile.Click += new
System.EventHandler(this.btnKeyFile_Click);
                        this.btnHide.Enabled = false;
                        this.btnHide.Location = new System.Drawing.Point(608, 216);
                        this.btnHide.Name = "btnHide";
                        this.btnHide.Size = new System.Drawing.Size(160, 23);
                        this.btnHide.TabIndex = 2;
                        this.btnHide.Text = "Hide Message";
                        this.btnHide.Click += new
System.EventHandler(this.btnHide_Click);

                        this.btnExtract.Enabled = false;
                        this.btnExtract.Location = new System.Drawing.Point(608, 216);
                        this.btnExtract.Name = "btnExtract";
                        this.btnExtract.Size = new System.Drawing.Size(160, 23);
                        this.btnExtract.TabIndex = 2;
                        this.btnExtract.Text = "Extract Hidden Text";
```

```
			this.btnExtract.Click += new
System.EventHandler(this.btnExtract_Click);

			this.tabAction.Controls.AddRange(new
System.Windows.Forms.Control[] {
			this.tabPage1,
			this.tabPage2});
			this.tabAction.Location = new System.Drawing.Point(8, 184);
			this.tabAction.Name = "tabAction";
			this.tabAction.SelectedIndex = 0;
			this.tabAction.Size = new System.Drawing.Size(792, 280);
			this.tabAction.TabIndex = 2;

			this.tabPage1.Controls.AddRange(new
System.Windows.Forms.Control[] {

			this.grpMessage,
			 this.btnHide,
			this.grpPicture});
			this.tabPage1.Location = new System.Drawing.Point(4, 25);
			this.tabPage1.Name = "tabPage1";
			this.tabPage1.Size = new System.Drawing.Size(784, 251);
			this.tabPage1.TabIndex = 0;
			this.tabPage1.Text = "Hide";
			this.grpMessage.Controls.AddRange(new
System.Windows.Forms.Control[] {

			this.rdoMessageText,
			 this.txtMessageFile,
			 this.txtMessageText,
			 this.btnMessage,
			this.rdoMessageFile});
			this.grpMessage.Location = new System.Drawing.Point(16, 16);
			this.grpMessage.Name = "grpMessage";
			this.grpMessage.Size = new System.Drawing.Size(424, 184);
			this.grpMessage.TabIndex = 0;
			this.grpMessage.TabStop = false;
			this.grpMessage.Text = "Message";

			// rdoMessageText

			this.rdoMessageText.Checked = true;
			this.rdoMessageText.Location = new
System.Drawing.Point(16,48);
			this.rdoMessageText.Name = "rdoMessageText";
			this.rdoMessageText.Size = new System.Drawing.Size(72, 24);
```

```
                        this.rdoMessageText.TabIndex = 3;
                        this.rdoMessageText.TabStop = true;
                        this.rdoMessageText.Text = "Text";
                        this.rdoMessageText.Click += new
System.EventHandler(this.rdoMessage_Click);

                        this.txtMessageFile.Location = new
System.Drawing.Point(104,24);
                        this.txtMessageFile.Name = "txtMessageFile";
                        this.txtMessageFile.Size = new System.Drawing.Size(232, 22);
                        this.txtMessageFile.TabIndex = 1;
                        this.txtMessageFile.Text = "";
                        this.txtMessageFile.Enter += new
System.EventHandler(this.txtMessageFile_Enter);

                        this.txtMessageText.Location = new
System.Drawing.Point(16,72);
                        this.txtMessageText.Multiline = true;
                        this.txtMessageText.Name = "txtMessageText";
                        this.txtMessageText.Size = new System.Drawing.Size(400, 96);
                        this.txtMessageText.TabIndex = 4;
                        this.txtMessageText.Text = "";
                        this.txtMessageText.Enter += new
System.EventHandler(this.txtMessageText_Enter);

                        this.btnMessage.Location = new System.Drawing.Point(336, 24);
                        this.btnMessage.Name = "btnMessage";
                        this.btnMessage.Size = new System.Drawing.Size(80, 23);
                        this.btnMessage.TabIndex = 2;
                        this.btnMessage.Text = "Browse...";
                        this.btnMessage.Click += new
System.EventHandler(this.btnMessage_Click);

                        // rdoMessageFile

                        this.rdoMessageFile.Location = new
System.Drawing.Point(16,24);
                        this.rdoMessageFile.Name = "rdoMessageFile";
                        this.rdoMessageFile.Size = new System.Drawing.Size(88, 24);
                        this.rdoMessageFile.TabIndex = 0;
                        this.rdoMessageFile.Text = "Filename";
                        this.rdoMessageFile.Click += new
System.EventHandler(this.rdoMessage_Click);

                        this.tabPage2.Controls.AddRange(new
System.Windows.Forms.Control[] {
```

```
                        this.grpPictureExtract,
                        this.groupBox3,
                        this.btnExtract});
                        this.tabPage2.Location = new System.Drawing.Point(4, 25);
                        this.tabPage2.Name = "tabPage2";
                        this.tabPage2.Size = new System.Drawing.Size(784, 251);
                        this.tabPage2.TabIndex = 1;
                        this.tabPage2.Text = "Extract";

                        this.grpPictureExtract.Controls.AddRange(new
System.Windows.Forms.Control[] {
                        this.lblTextFilesExtract,
                        this.btntTextFileExtract});
                        this.grpPictureExtract.Location = new
System.Drawing.Point(456,16);
                        this.grpPictureExtract.Name = "grpPictureExtract";
                        this.grpPictureExtract.Size = new System.Drawing.Size(312, 184);
                        this.grpPictureExtract.TabIndex = 3;
                        this.grpPictureExtract.TabStop = false;
                        this.grpPictureExtract.Text = "Carrier Bitmaps";


                        this.lblTextFilesExtract.Location = new
System.Drawing.Point(16,32);
                        this.lblTextFilesExtract.Name = "lblTextFilesExtract";
                        this.lblTextFilesExtract.Size = new System.Drawing.Size(168, );
                        this.lblTextFilesExtract.TabIndex = 3;
                        this.lblTextFilesExtract.Text = "No text files specified";

                        this.btntTextFileExtract.Location = new
System.Drawing.Point(184, 32);
                        this.btntTextFileExtract.Name = "btntTextFileExtract";
                        this.btntTextFileExtract.Size = new System.Drawing.Size(112,23);
                        this.btntTextFileExtract.TabIndex = 2;
                        this.btntTextFileExtract.Text = "Add/Remove...";
                        this.btntTextFileExtract.Click += new
System.EventHandler(this.btnTextFile_Click);

                        this.groupBox3.Controls.AddRange(new
System.Windows.Forms.Control[]{this.label1,this.txtExtractedMsgFile,this.txtExtracted
MsgText,        this.btnExtractedMsgFile,this.label3});

                        this.groupBox3.Location = new System.Drawing.Point(16, 16);
                        this.groupBox3.Name = "groupBox3";
                        this.groupBox3.Size = new System.Drawing.Size(424, 184);
                        this.groupBox3.TabIndex = 0;
                        this.groupBox3.TabStop = false;
                        this.groupBox3.Text = "Message";

                        this.label1.Location = new System.Drawing.Point(16, 32);
                        this.label1.Name = "label1";
                        this.label1.Size = new System.Drawing.Size(376, 16);
                        this.label1.TabIndex = 10;
                        this.label1.Text = "Save Extracted Message to File";

                        // txtExtractedMsgFile

                        this.txtExtractedMsgFile.Location = new
System.Drawing.Point(16, 48);
                        this.txtExtractedMsgFile.Name = "txtExtractedMsgFile";
                        this.txtExtractedMsgFile.Size = new
System.Drawing.Size(312,22);
                        this.txtExtractedMsgFile.TabIndex = 0;
                        this.txtExtractedMsgFile.Text = "";

                        // txtExtractedMsgText

                        this.txtExtractedMsgText.Location = new
System.Drawing.Point(16, 96);
                        this.txtExtractedMsgText.Multiline = true;
                        this.txtExtractedMsgText.Name = "txtExtractedMsgText";

                        this.txtExtractedMsgText.ReadOnly = true;
                        this.txtExtractedMsgText.Size = new
System.Drawing.Size(392,72);
                        this.txtExtractedMsgText.TabIndex = 5;
                        this.txtExtractedMsgText.Text = "";

                        // btnExtractedMsgFile

                        this.btnExtractedMsgFile.Location = new
System.Drawing.Point(328, 48);
                        this.btnExtractedMsgFile.Name = "btnExtractedMsgFile";
                        this.btnExtractedMsgFile.Size = new System.Drawing.Size(80,23);
                        this.btnExtractedMsgFile.TabIndex = 1;
                        this.btnExtractedMsgFile.Text = "Browse...";
                        this.btnExtractedMsgFile.Click += new
System.EventHandler(this.btnExtractedMsgFile_Click);
```

```
			this.label3.Location = new System.Drawing.Point(16, 80);
			this.label3.Name = "label3";
			this.label3.Size = new System.Drawing.Size(376, 16);
			this.label3.TabIndex = 10;
			this.label3.Text = "Extracted UnicodeText";

			this.grpSplitBytes.Controls.AddRange(new
System.Windows.Forms.Control[] {this.chkSplitBytes});
			this.grpSplitBytes.Location = new System.Drawing.Point(8, 96);
			this.grpSplitBytes.Name = "grpSplitBytes";
			this.grpSplitBytes.Size = new System.Drawing.Size(792, 72);
			this.grpSplitBytes.TabIndex = 1;
			this.grpSplitBytes.TabStop = false;
			this.grpSplitBytes.Text = "Split Bytes";

			this.chkSplitBytes.Location = new System.Drawing.Point(16, 32);
			this.chkSplitBytes.Name = "chkSplitBytes";
			this.chkSplitBytes.Size = new System.Drawing.Size(736, 24);
			this.chkSplitBytes.TabIndex = 4;

			this.AutoScaleBaseSize = new System.Drawing.Size(6, 15);
			this.ClientSize = new System.Drawing.Size(810, 473);
			this.Controls.AddRange(new System.Windows.Forms.Control[]
{this.tabAction, this.grpKey, this.grpSplitBytes});
			this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedSingle;
			this.Name = "frmMain";

			this.grpPicture.ResumeLayout(false);
			this.grpKey.ResumeLayout(false);
			this.tabAction.ResumeLayout(false);
			this.tabPage1.ResumeLayout(false);
			this.grpMessage.ResumeLayout(false);
			this.tabPage2.ResumeLayout(false);
			this.grpPictureExtract.ResumeLayout(false);
			this.groupBox3.ResumeLayout(false);
			this.grpSplitBytes.ResumeLayout(false);
			this.ResumeLayout(false);

		}


		static void Main()
		{
			Application.Run(new frmMain());
		}
		private void btnHide Click(object sender, System.EventArgs e)
		{
			//get a stream for the message to hide
			Stream messageStream = GetMessageStream();
			if(messageStream.Length == 0)
			{
				MessageBox.Show("Please select a file.");
				txtMessageText.Focus();
			}
			else
			{
				this.Cursor = Cursors.WaitCursor;
				//try{
					//hide the message
					CryptUtility.HideMessageInBitmap(messageStream,
imagesHide, keys, chkSplitBytes.Checked);
					//}catch(Exception ex)
					{


				this.Cursor = Cursors.Default;
			}
			messageStream.Close();
		}

		private void btnExtract_Click(object sender, System.EventArgs e) {
			//empty stream for the extracted message
			Stream messageStream = new MemoryStream();

			this.Cursor = Cursors.WaitCursor;

			try{
				//extract the hidden message from the bitmap
				CryptUtility.ExtractMessageFromBitmap(imagesExtract,
keys, ref messageStream, chkSplitBytes.Checked);

				//save the message, if a filename is available
				if(txtExtractedMsgFile.Text.Length > 0){
					messageStream.Seek(0, SeekOrigin.Begin);
					FileStream fs = new
FileStream(txtExtractedMsgFile.Text, FileMode.Create);
					byte[] streamContent = new
Byte[messageStream.Length];
					messageStream.Read(streamContent, 0,
streamContent.Length);
```

```
                                        fs.Write(streamContent, 0, streamContent.Length);
                                }

                                                        messageStream.Seek(0,
SeekOrigin.Begin);
                                StreamReader reader = new StreamReader(messageStream,
UnicodeEncoding.Unicode);
                                String readerContent = reader.ReadToEnd();
                                if(readerContent.Length >
txtExtractedMsgText.MaxLength){
                                        readerContent = readerContent.Substring(0,
txtExtractedMsgText.MaxLength);
                                }
                                txtExtractedMsgText.Text = readerContent;

                        }

                        this.Cursor = Cursors.Default;

                        //close the stream
                        messageStream.Close();
                }


                private Stream GetMessageStream(){
                        Stream messageStream;
                        if(rdoMessageText.Checked){
                                byte[] messageBytes =
UnicodeEncoding.Unicode.GetBytes(txtMessageText.Text);
                                messageStream = new MemoryStream(messageBytes);
                        }else{
                                messageStream = new FileStream(txtMessageFile.Text,
FileMode.Open, FileAccess.Read);
                        }
                        return messageStream;
                }


                private String GetFileName(String filter){
                        OpenFileDialog dlg = new OpenFileDialog();
                        dlg.Multiselect = false;
                        if(filter.Length > 0){ dlg.Filter = filter; }

                        if( dlg.ShowDialog(this) != DialogResult.Cancel){
                                return dlg.FileName;
                        }else{
```

```
                        return null;
                }
        }

        private void rdoMessage_Click(object sender, System.EventArgs e) {
                txtMessageFile.Enabled = rdoMessageFile.Checked;
                txtMessageText.Enabled = rdoMessageText.Checked;
        }

        private void txtMessageFile_Enter(object sender, System.EventArgs e) {
                rdoMessageFile.Checked = true;
        }

        private void txtMessageText_Enter(object sender, System.EventArgs e) {
                rdoMessageText.Checked = true;
        }

        private void btnMessage_Click(object sender, System.EventArgs e) {
                String fileName = GetFileName(String.Empty);
                if(fileName != null){
                        txtMessageFile.Text = fileName;
                        rdoMessageFile.Checked = true;
                }
        }

        private void btnExtractedMsgFile_Click(object sender,
System.EventArgs) {

                SaveFileDialog dlg = new SaveFileDialog();
                if( dlg.ShowDialog() == DialogResult.OK ){
                        txtExtractedMsgFile.Text = dlg.FileName;
                }
        }


                                        lblKeyFiles.Text = keys.Length.ToString()
                            + " key file specified";
                                        btnHide.Enabled = (textHide.Length >0);
                                        btnExtract.Enabled = (textExtract.Length
                                    > 0);
                                        break; }
                        }
                }
        }

        private void btntexteFile_Click(object sender, System.EventArgs e) {
```

```
                        Label lblFeedback;
                        Button btnAction;
                        CarrierImage[] images;
                        ImageFilesDialog dlg;
                        if(sender == btntextFile){

                                }

                                switch(i.Length){
                                        case 0:{
                                                lblFeedback.Text = "No carrier files
specified";
                                                btnAction.Enabled = false;
                                                break; }
                                        case 1:{
                                                lblFeedback.Text = "1 carrier file specified";
                                                btnAction.Enabled = (keys.Length > 0);
                                                break; }

                                        default:{
                                                lblFeedback.Text =
images.Length.ToString() + " carrier file specified";
                                                btnAction.Enabled = (keys.Length > 0);
                                                break; }
                                }
                        }
                }

        }
}
```

11.1.4. TypeDefs.cs:

```
using System;
using System.Drawing;
namespace PictureKey
{
        public struct FilePasswordPair{
                public String fileName;
                public String password;
                public FilePasswordPair(String fileName, String password){
                        this.fileName = fileName;
                        this.password = password;
                }
}
public struct CarrierImage{
        //file name of the clean text
        public String sourceFileName;

        //file name to save the new text
        public String resultFileName;
        //width * height
        public long countPixels;
        //count of frames in the video stream, or 0
        public int aviCountFrames;
        public bool useGrayscale;
        //how many bytes will be hidden in this text - this field is set by
                CryptUtility.HideOrExtract()
        public long messageBytesToHide;
        public long[] aviMessageBytesToHide;

        public void SetCountBytesToHide(long messageBytesToHide){
                this.messageBytesToHide = messageBytesToHide;
                aviMessageBytesToHide = new long[aviCountFrames];
                //calculate count of message-bytes to hide in (or extract from) each
                      frame
                long sumBytes = 0;
                for(int n=0; n<aviCountFrames; n++){
                        aviMessageBytesToHide[n] = (long)Math.Ceiling(
                            (float)messageBytesToHide / (float)aviCountFrames );
                        sumBytes += aviMessageBytesToHide[n];
                }
                if(sumBytes > messageBytesToHide){
                 //correct Math.Ceiling effects
                aviMessageBytesToHide[aviCountFrames-1] -= (sumBytes -
                      messageBytesToHide);
                }
        }

        public CarrierImage(String sourceFileName, String resultFileName, long
              countPixels, int aviCountFrames, bool useGrayscale){
                this.sourceFileName = sourceFileName;
                this.resultFileName = resultFileName;
                this.countPixels = countPixels;
                this.aviCountFrames = aviCountFrames;
                this.useGrayscale = useGrayscale;
                this.messageBytesToHide = 0;
                this.aviMessageBytesToHide = null;
        }
}
```

```
        public struct BitmapInfo{
                //uncompressed image
                public Bitmap bitmap;
                //position of the frame in the AVI stream
                public int aviPosition;
                //count of frames in the AVI stream
                public int aviCountFrames;
                //path and name of the bitmap file
                public String sourceFileName;
                //how many bytes will be hidden in this text
                public long messageBytesToHide;
                public void LoadBitmap(String fileName){
                        bitmap = new Bitmap(fileName);
                        sourceFileName = fileName;
                }
        }}
```

AviReader.cs:

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

namespace PictureKey {

        /// Extract bitmaps from AVI files
        public class AviReader{

                private int firstFrame = 0, countFrames = 0;
                private int aviFile = 0;
                private int getFrameObject;
                private IntPtr aviStream;
                private Avi.AVISTREAMINFO streamInfo;

                public int CountFrames{
                        get{ return countFrames; }
                }
        public UInt32 FrameRate{
                get{ return streamInfo.dwRate / streamInfo.dwScale; }
        }

        public Size BitmapSize{
                        get{ return new Size( (int)streamInfo.rcFrame.right,
                        (int)streamInfo.rcFrame.bottom); }
        }

/// Opens an AVI file and creates a GetFrame object
        public void Open(string fileName) {
                //Intitialize AVI library
                Avi.AVIFileInit();

                //Open the file
                int result = Avi.AVIFileOpen(
                        ref aviFile, fileName,
                        Avi.OF_SHARE_DENY_WRITE, 0);

                if(result != 0){ throw new Exception("Exception in AVIFileOpen:
                    "+result.ToString()); }

                //Get the video stream
                result = Avi.AVIFileGetStream(
                        aviFile,
                        out aviStream,
                        Avi.StreamtypeVIDEO, 0);

                if(result != 0){ throw new Exception("Exception in
                    AVIFileGetStream: "+result.ToString()); }

                //get start position and count of frames
                firstFrame = Avi.AVIStreamStart(aviStream.ToInt32());
                countFrames = Avi.AVIStreamLength(aviStream.ToInt32());

                //get header information
                streamInfo = new Avi.AVISTREAMINFO();
                result = Avi.AVIStreamInfo(aviStream.ToInt32(), ref streamInfo,
                          Marshal.SizeOf(streamInfo));

                if(result != 0){ throw new Exception("Exception in
                        AVIStreamInfo: "+result.ToString()); }
        //Open frames

        Avi.BITMAPINFOHEADER bih = new
                Avi.BITMAPINFOHEADER();

        //construct the expected bitmap header
        bih.biBitCount = 24;
        bih.biClrImportant = 0;
        bih.biClrUsed = 0;
        bih.biCompression = 0; //BI_RGB;
        bih.biHeight = (Int32)streamInfo.rcFrame.bottom;
        bih.biWidth = (Int32)streamInfo.rcFrame.right;
        bih.biPlanes = 1;
        bih.biSize =(UInt32)Marshal.SizeOf(bih);
        bih.biXPelsPerMeter = 0;
        bih.biYPelsPerMeter = 0;

        //prepare to decompress DIBs (device independent bitmaps)
        getFrameObject = Avi.AVIStreamGetFrameOpen(aviStream, ref
            bih);
        if(getFrameObject == 0){ throw new Exception("Exception in
            AVIStreamGetFrameOpen!"); }
}

/// Closes all streams, files and libraries
public void Close() {
        if(getFrameObject != 0){
                Avi.AVIStreamGetFrameClose(getFrameObject);
                getFrameObject = 0;
        }
        if(aviStream != IntPtr.Zero){
                Avi.AVIStreamRelease(aviStream);
                aviStream = IntPtr.Zero;
        }
        if(aviFile != 0){
                Avi.AVIFileRelease(aviFile);
                aviFile = 0;
        }
        Avi.AVIFileExit();
}

        /// Exports a frame into a bitmap file bitmap
public void ExportBitmap(int position, String dstFileName){
        if(position > countFrames){
                throw new Exception("Invalid frame position");
}

//Decompress the frame and return a pointer to the DIB
int pDib = Avi.AVIStreamGetFrame(getFrameObject, firstFrame +
    position);

//Copy the bitmap header into a managed struct
Avi.BITMAPINFOHEADER bih = new
      Avi.BITMAPINFOHEADER();
 bih = (Avi.BITMAPINFOHEADER)Marshal.PtrToStructure(new
      IntPtr(pDib), bih.GetType());

if(bih.biSizeImage < 1){
throw new Exception("Exception in AVIStreamGetFrame:Bitmap
            not decompressed.");
}

//Copy the image
byte[] bitmapData = new byte[bih.biSizeImage];
int address = pDib + Marshal.SizeOf(bih);
for(int offset=0; offset<bitmapData.Length; offset++){
        bitmapData[offset] = Marshal.ReadByte(new
        IntPtr(address));
     address++;
}

//Copy bitmap info
byte[] bitmapInfo = new byte[Marshal.SizeOf(bih)];
IntPtr ptr;
ptr = Marshal.AllocHGlobal(bitmapInfo.Length);
Marshal.StructureToPtr(bih, ptr, false);
address = ptr.ToInt32();
for(int offset=0; offset<bitmapInfo.Length; offset++){
     bitmapInfo[offset] = Marshal.ReadByte(new
     IntPtr(address));
     address++;
}

//Create file header
Avi.BITMAPFILEHEADER bfh = new
        Avi.BITMAPFILEHEADER();
bfh.bfType = Avi.BMP;
bfh.bfSize = (Int32)(55 + bih.biSizeImage); //size of file as written
  to disk
bfh.bfReserved1 = 0;
bfh.bfReserved2 = 0;
```

```
                        bfh.bfOffBits = Marshal.SizeOf(bih) + Marshal.SizeOf(bfh);

                        FileStream fs = new FileStream(dstFileName,
                                System.IO.FileMode.Create);
                        BinaryWriter bw = new BinaryWriter(fs);

                        //Write header
                        bw.Write(bfh.bfType);
                        bw.Write(bfh.bfSize);
                        bw.Write(bfh.bfReserved1);
                        bw.Write(bfh.bfReserved2);
                        bw.Write(bfh.bfOffBits);
                        //Write bitmap info
                        bw.Write(bitmapInfo);
                        //Write bitmap data
                        bw.Write(bitmapData);
                        bw.Close();
                        fs.Close();
                }
        }
}
```

AviWriter.cs:

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace PictureKey {

        /// Create AVI files from bitmaps
        public class AviWriter {

                private int aviFile = 0;
                private IntPtr aviStream = IntPtr.Zero;
                private UInt32 frameRate = 0;
                private int countFrames = 0;
                private int width = 0;
                private int height = 0;
                private UInt32 stride = 0;
                private UInt32 fccType = Avi.StreamtypeVIDEO; // vids
                private UInt32 fccHandler = 1668707181;

                /// Creates a new AVI file
                public void Open(string fileName, UInt32 frameRate) {
                        this.frameRate = frameRate;
```

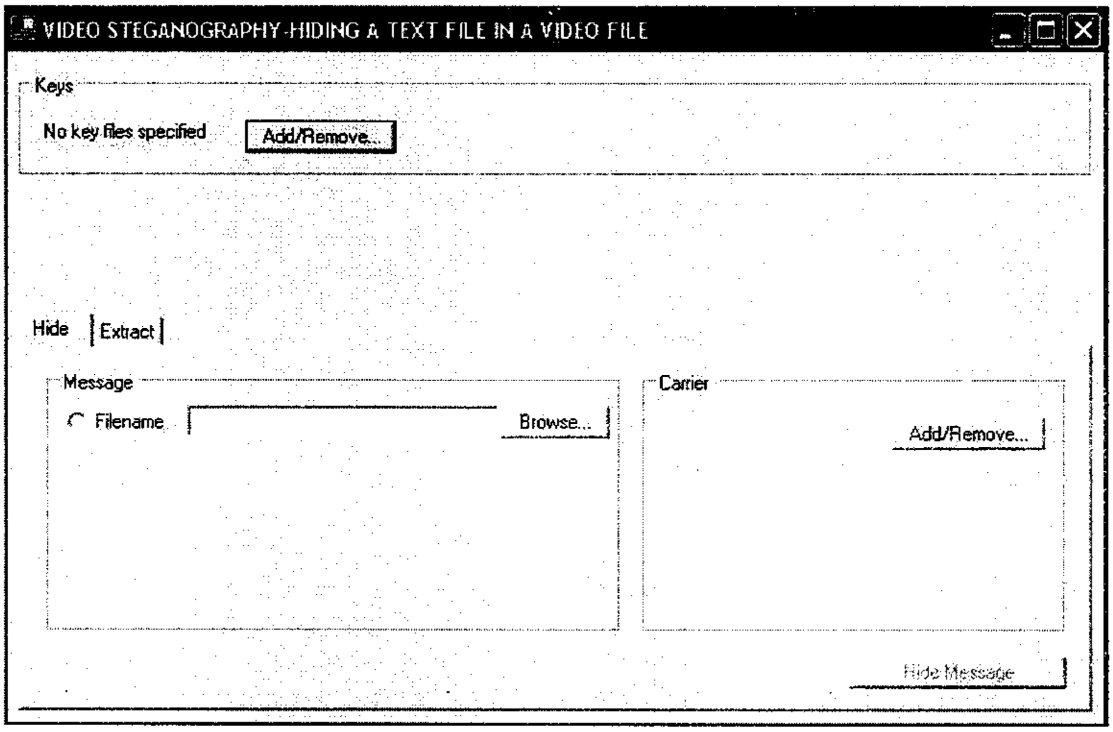
```
Avi.AVIFileInit();

int hr = Avi.AVIFileOpen(
                ref aviFile, fileName,
                4097 /* OF_WRITE | OF_CREATE */, 0);
if (hr != 0) {
                throw new Exception("Error in AVIFileOpen:
                "+hr.ToString());
        }
}

/// Adds a new frame to the AVI stream
public void AddFrame(Bitmap bmp) {

bmp.RotateFlip(RotateFlipType.RotateNoneFlipY);

BitmapData bmpDat = bmp.LockBits(
        new Rectangle(0, 0, bmp.Width, bmp.Height),
        ImageLockMode.ReadOnly,PixelFormat.Format24bppRgb);

if (countFrames == 0) {
        //this is the first frame - get size and create a new stream
        this.stride = (UInt32)bmpDat.Stride;
        this.width = bmp.Width;
        this.height = bmp.Height;
        CreateStream();
}

int result = Avi.AVIStreamWrite(aviStream,
        countFrames, 1, bmpDat.Scan0, (Int32) (stride  * height), 0, 0, 0);
if (result != 0) {
      throw new Exception("Error in AVIStreamWrite:
                "+result.ToString());
}

bmp.UnlockBits(bmpDat);
countFrames ++;
}

/// Closes stream, file and AVI library
public void Close() {
if(aviStream != IntPtr.Zero){
        Avi.AVIStreamRelease(aviStream);
        aviStream = IntPtr.Zero;
}
```

```
                if(aviFile != 0){
                        Avi.AVIFileRelease(aviFile);
                        aviFile = 0;
                }
                Avi.AVIFileExit();
                }

                /// Creates a new video stream in the AVI file
                private void CreateStream() {
                Avi.AVISTREAMINFO strhdr = new Avi.AVISTREAMINFO();
                strhdr.fccType = fccType;
                strhdr.fccHandler = fccHandler;
                strhdr.dwScale = 1;
                strhdr.dwRate = frameRate;
                strhdr.dwSuggestedBufferSize = (UInt32)(height * stride);
                strhdr.dwQuality= 10000;
                strhdr.rcFrame.bottom = (UInt32)height;
                strhdr.rcFrame.right = (UInt32)width;
                strhdr.szName  = new UInt16[64];

                int result = Avi.AVIFileCreateStream(aviFile, out aviStream, ref strhdr);
                if(result != 0){ throw new Exception("Error in AVIFileCreateStream:
                        "+result.ToString()); }

                //define the image format
                Avi.BITMAPINFOHEADER bi = new Avi.BITMAPINFOHEADER();
                bi.biSize      = (UInt32)Marshal.SizeOf(bi);
                bi.biWidth     = (Int32) width;
                bi.biHeight    = (Int32) height;
                bi.biPlanes    = 1;
                bi.biBitCount  = 24;
                bi.biSizeImage = (UInt32)(stride*height);
                result = Avi.AVIStreamSetFormat(aviStream, 0, ref bi,
                        Marshal.SizeOf(bi));
                if(result != 0){ throw new Exception("Error in AVIStreamSetFormat:
                        "+result.ToString()); }
            }
        }
    }
```

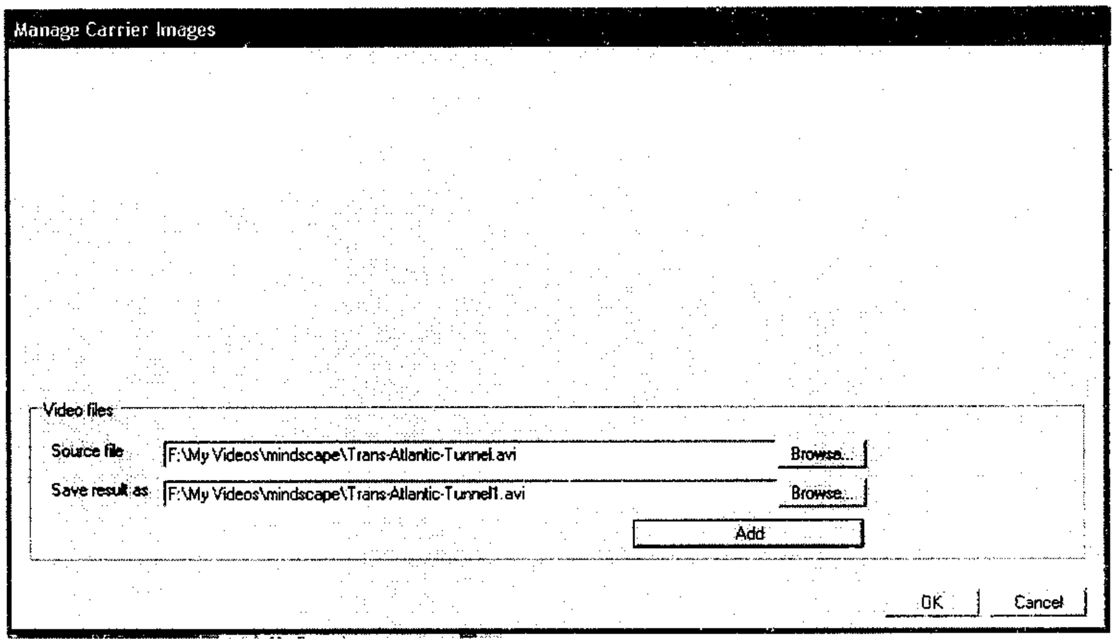
APPENDIX 2

SAMPLE OUTPUT:

USER INTERFACE FORM



COVERT AND OVERT FILE SELECTION

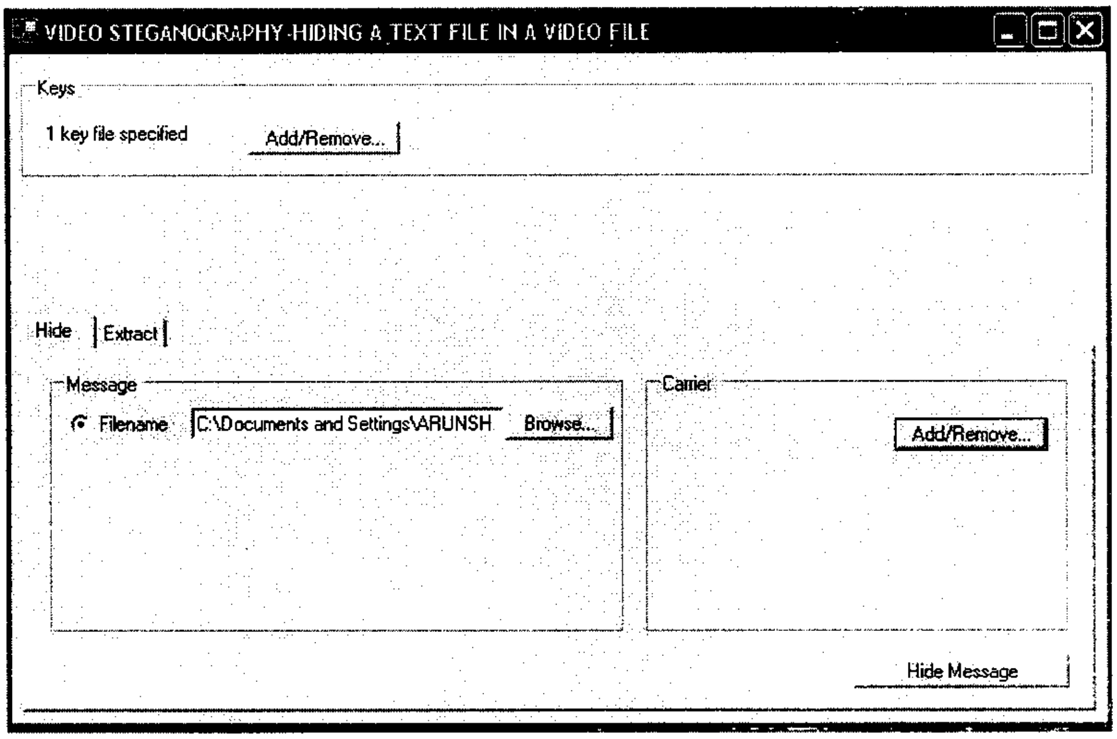

Manage Carrier Images

Video files

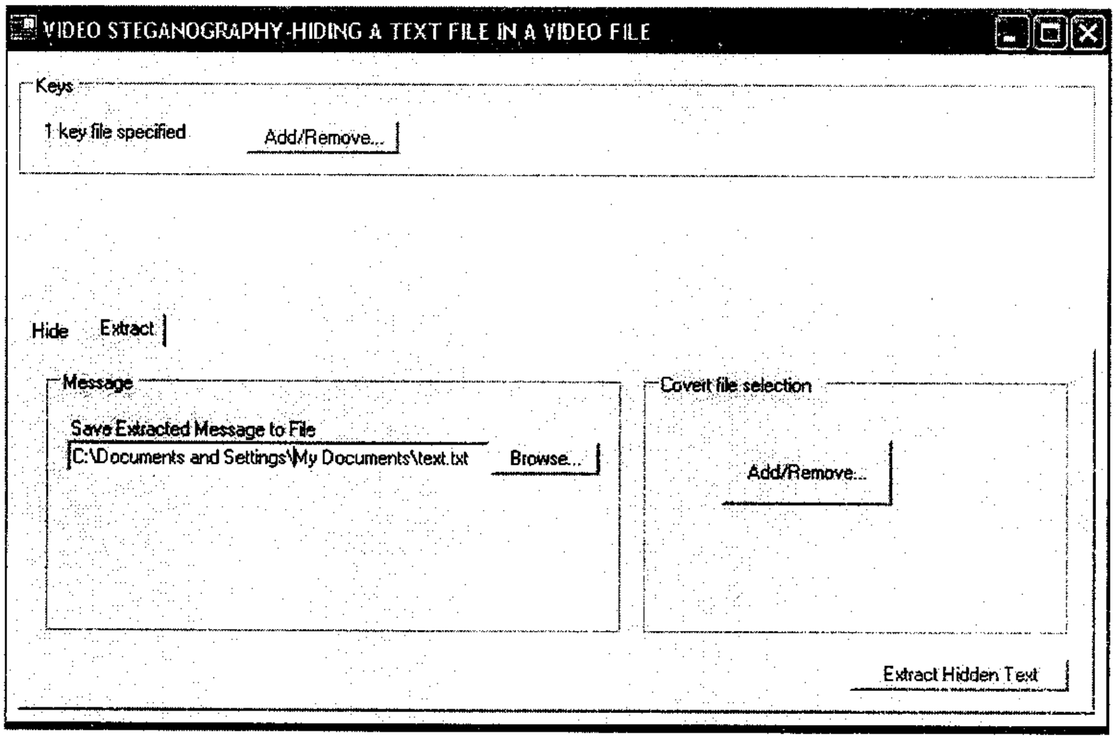
Source file: F:\My Videos\mindscape\Trans-Atlantic-Tunnel.avi

Save result as: F:\My Videos\mindscape\Trans-Atlantic-Tunnel1.avi

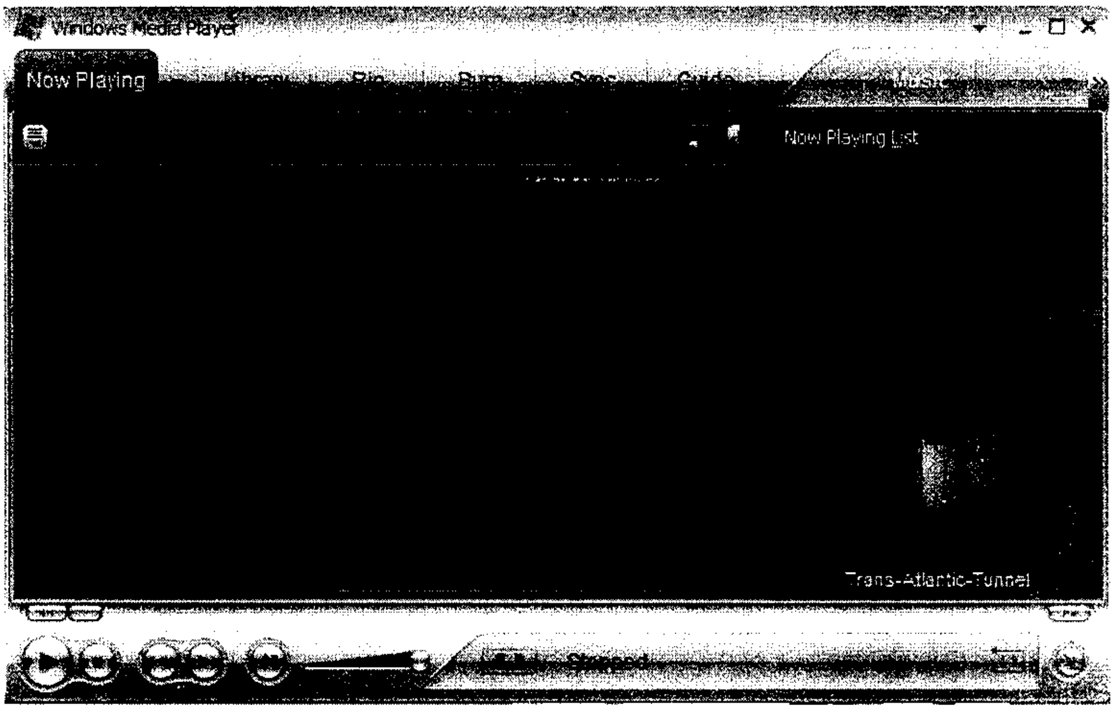
HIDING INTERFACE



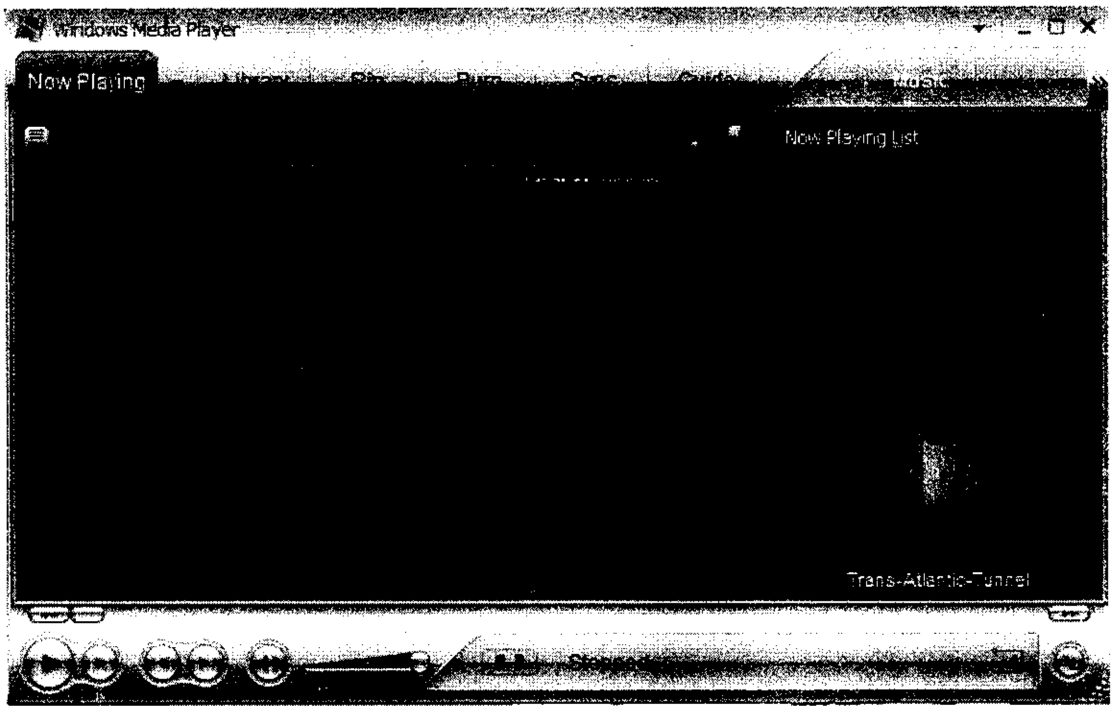
EXTRACTION INTERFACE


VIDEO STEGANOGRAPHY-HIDING A TEXT FILE IN A VIDEO FILE
Keys
1 key file specified
Add/Remove...
Hide
Extract
Message
Save Extracted Message to File
C:\Documents and Settings\My Documents\text.txt
Browse...
Covert file selection
Add/Remove...
Extract Hidden Text

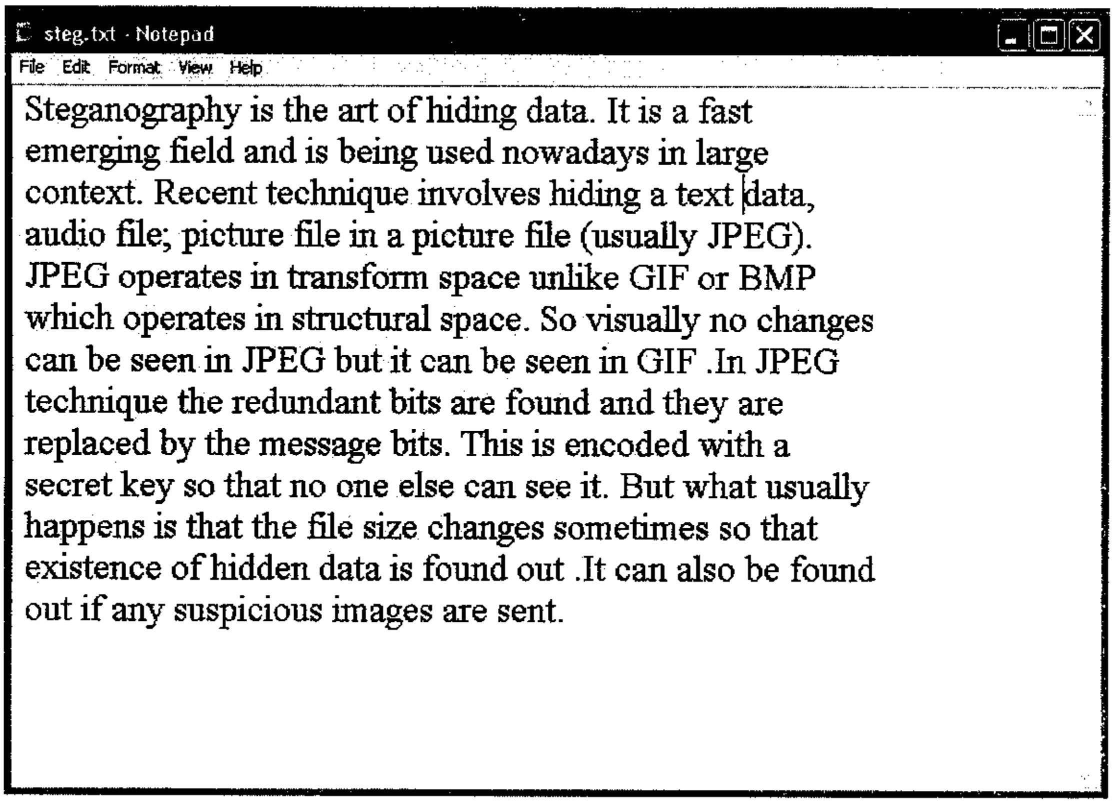
VIDEO FILE BEFORE EMBEDDING TEXT CONTENTS



VIDEO FILE AFTER EXTRACTION OF THE HIDDEN TEXT

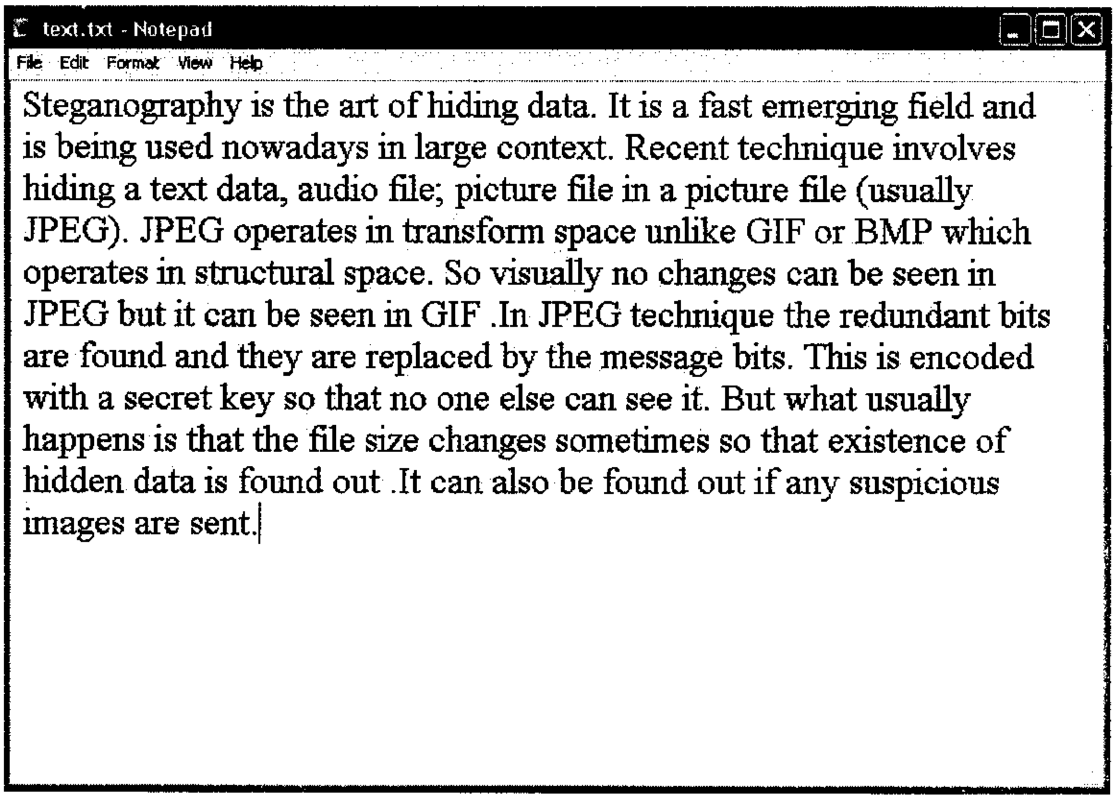


TEXT FILE CONTENTS BEFORE HIDING



Steganography is the art of hiding data. It is a fast emerging field and is being used nowadays in large context. Recent technique involves hiding a text data, audio file; picture file in a picture file (usually JPEG). JPEG operates in transform space unlike GIF or BMP which operates in structural space. So visually no changes can be seen in JPEG but it can be seen in GIF .In JPEG technique the redundant bits are found and they are replaced by the message bits. This is encoded with a secret key so that no one else can see it. But what usually happens is that the file size changes sometimes so that existence of hidden data is found out .It can also be found out if any suspicious images are sent.

TEXT FILE CONTENTS AFTER EXTRACTION



Steganography is the art of hiding data. It is a fast emerging field and is being used nowadays in large context. Recent technique involves hiding a text data, audio file; picture file in a picture file (usually JPEG). JPEG operates in transform space unlike GIF or BMP which operates in structural space. So visually no changes can be seen in JPEG but it can be seen in GIF .In JPEG technique the redundant bits are found and they are replaced by the message bits. This is encoded with a secret key so that no one else can see it. But what usually happens is that the file size changes sometimes so that existence of hidden data is found out .It can also be found out if any suspicious images are sent.

REFERENCES

1. Simon Robinson, Christian Nagel, Karli Watson, Jay Glynn, Morgan Skinner, Bill Evjen, "Professional C#", 3rd edition, Wiley Publishing, Inc.

Web sites:

1.www.steganography.com
2.www.computerworld.com
3.www.jjtc.com/Steganography.html