ABSTRACT

Steganography is a form of science that deals with cryptic information. It is the art of writing in cryptic text that is unrecognizable to a person who doesn't hold the key to decrypt it. Steganography is not a new form of science. In fact, Steganography is derived from the Greek word "steganos", which means hidden or secret and "graphy" means writing or drawing. Thus steganography means secret writing. In contemporary terms, steganography has evolved into a digital strategy of hiding a file in any form of multimedia such as an image, an audio file or even a video file.

This steganographic application "SteganoSense" works as a tool for hiding messages such as text, or image files in a wave file. The difference in SteganoSense tool and earlier tools is that the emphasis was more on security, lossless data compression and much more robustness. This work emphasizes various techniques by which steganography in audio can be implemented. SteganoSense offers more security with the use of Advanced Encryption Standard algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

1. BACKGROUND AND RATIONALE

1.1 Steganography

1.1.1 Terminology

The definition of steganography can be explained clearly by using adjectives like ‘cover’, ‘embedded’ and ‘stego’. Steganography simply grabs a piece of information and hides that information within another piece of information [Scribd 2007]. Many computer files such as audio files, text files, images contain few blocks of data that is either unused or not significant. Steganography takes advantage of these unused areas and hides the encrypted message.

1.1.2 Difference with Other Systems

The term “Information hiding” can be related to either steganography field or watermarking technology. Watermarking technology usually refers to various methods that conceal information in a data object so that the information is adjustable to future modifications [Wikipedia]. In essence, it should be impossible to remove the watermark without drastically modifying the quality of the object.

While on the other hand, steganography refers to hidden / concealed information that is fragile. Any small modification to the cover medium may destroy the concealed information. Also, the above mentioned two ways differ in one more way. In steganography, viewer or user must not know about the presence of the concealed information whereas in watermarking, this feature is optional.

1.1.3 History

The motive behind steganography is to communicate data secretly with somebody. An earlier account of steganography is found in a story by Herodotus, in

which a slave sent by his master, Histiaeus, to the Ionic city of Miletus with a secret message tattooed on his scalp [Cox, Miller, Bloom 2001]. Once when the slave's hair had grown back and had successfully hidden the message, the slave, Herodotus was sent to warn of the Persian's impending invasion on the Greece.

Another method to secretly deploy ciphered messages was to modify ancient writing tablets [Wikipedia]. In this method, the messages that need to be hidden were written on the layer of wax covering the surface of the tablets. The enhanced version of this method was developed by Demeratus [Wikipedia]. Demeratus was by birth a Greek but he was exiled into Persia. He devised a master sketch to embed and thus hide a message by removing the layer of wax and writing directly on the underlying wood. Demeratus implemented this enhanced method and was successful in sending a warning message to Sparta that the Persians were planning an invasion. After the message was written directly on the writing tablets, they were then covered again with wax and appeared unused to the examiners of the shipment.

Another classic application of ancient steganography is the method of wrapping a ribbon around a wooden staff from top to bottom [Brainos II 2003]. This method is the best example of a null cipher that was mentioned above. The key to this method is to write across all over the ribbon and unravel it. By doing so, the clear text will be all over the wooden staff and only someone with the same size diameter wooden staff similar to the original one could read the hidden ciphered message. In this method, the most important feature is the fact that the ciphered message even existed will be hidden from the outsiders.

Coming to not so ancient steganographic applications, few methods were introduced during the two World Wars, especially World War II. During this period, the German military created microdots which are a breakthrough technology at that time. They leveraged microfiche technology to create these microdots. Microdots consisted of pictures and text messages which were shrunk down to the size of a period and used in the text of an otherwise innocent letter or memorandum [Wikipedia]. The big breakthrough for steganography has happened in the early nineties when governments, industries, general citizens and even some of the extremist organizations began using software applications to embed messages and photos into various types of media like digital photos, digital videos, audio files and text files.

Most of the techniques proposed in the literature use texts or images as covers. For embedding a message in a text document, apparently invisible coding techniques are used. However, for embedding a message in an image, a different set of techniques such as least-significant bit insertion, masking and filtering, and subtle transformation of the image are used. These techniques or transformations do not cause any visible changes in the cover image when viewed.

1.1.4 Components of a Steganographic Message

Before going deep into the steganographic process, first and foremost, we need to understand the various components of a steganographic message. The below list covers all the possible components that will be present in the steganographic message.

- Secret message
- Cover data
- Stego message

The *secret message* refers to the part of the message which is intended to be hidden. This message will later be encrypted to make it even more difficult for anyone who tries to break the security to get hold of the hidden informatic message. This is the crucial component in a steganographic message. Next part is the *cover data* component. This component refers to the container in which the secret message is hidden. This cover data component can be anything like digital photos, digital videos, audio files and text files. The final component is the *stego message* which is as crucial as the *secret message*. The *stego message* component refers to the final product.

1.1.5 Steganographic Approaches

There are various types of cream layer steganographic approaches. They are:

- Top-down approach
- Bottom-up approach

These approaches are again subdivided into sub layers. From a top-down approach, there exist three types of steganographic approaches. They are:

- Pure steganography
- Private key steganography
- Public key steganography

These categories convey the level of security with which the stego message is embedded, transmitted and read.

1.1.5.1 Pure Steganography

Pure steganography is defined as a steganographic system that does not require the exchange of a cipher such as a stego-key. This method of steganography is the least secure means by which to communicate secretly because the sender and receiver can rely

only upon the presumption that no other parties are aware of this secret message [Brainos II 2003]. Using open systems such as the Internet, we know this is not the cause at all.

Pure steganography uses no keyed system to embed cleartext or null cipher text into the cover data in order to hide the existence of a secret message. Pure steganography is only secure in two aspects which are, the fact only the sending and receiving parties know of the secret message's existence and which steganographic algorithm was used to hide the message. In steganalysis, this type is the easiest to crack since once detected the message can only have been hidden in as many ways as the number of steganographic algorithms which exist.

The foremost difficult aspect is in the detection effort. The difficulty lies in the fact that the unlimited amount of screened data does not include pre-modification copies of themselves. For example, if any National Security Agency has to screen millions of Web pages for steganographic material, most of the authors of this material would not leave the original copies of the cover data in the Web site's directory or even on the computer which produced the stego message. In this way, the message will be virtually undetectable because of the fact that in the simplest form of the stego message, only the least significant bits of each byte representing a digital photo have been modified to carry the secret message. However, once detected, a pure stego message could be cracked very easily. This is the reason why the pure steganographic approach is the least secure method and thus least used.

1.1.5.2 Private Key Steganography

Private key steganography is also called as secret key steganography [Brainos II 2003]. This secret key steganography is defined as a steganographic system that requires

the exchange of a secret key prior to communication. Secret key steganography takes a cover message and embeds the secret message inside of it by using a secret key. This secret key is also called the stego key. Only the parties who know the secret key can reverse the process and read the secret message.

Unlike pure steganography where a perceived invisible communication channel is present, secret key steganography exchanges a stego key, which makes it more susceptible to interception [Anderson, Petitcolas 1998]. The benefit to secret key steganography is even if it is intercepted; only parties who know the secret key can extract the secret message.

This private key steganography method uses a mutual key for encrypting then hiding the secret message within the cover data. As in traditional encryption, the private key system is only as robust as the knowledge of the key. Since the private key system requires both parties to know the key, once it is compromised the entire stego message is non-secure.

1.1.5.3 Public Key Steganography

Public key steganography can be defined as a steganography system that uses a public key and a private key to secure the communication between the parties wanting to communicate secretly [Brainos II 2003]. The sender will use the public key during the encoding process and only the private key, which has a direct mathematical relationship with the public key, can decipher the secret message.

Public key steganography provides a more robust way of implementing a steganographic system because it can utilize a much more robust and researched technology in public key cryptography. It also has multiple levels of security in that

unwanted parties must first suspect the use of steganography and then they would have to find a way to crack the algorithm used by the public key system before they could intercept the secret message.

This public key encrypted steganography uses the key pair system to add a layer of robustness to the process. As in public key encryption, the public key of the recipient is used to encrypt the secret message and only that user's private key may decrypt it after extracting it from the cover data. This method is the most secure type of steganography. This approach is recommended since it combines the benefits of hiding the existence of a secret message with the security of encryption.

1.2 Previous Work

1.2.1 Binary Wave Encoding

There has been some work done in the field of audio steganography, particularly in binary encoding [Wikipedia]. One scheme to encode binary information in a wave file involves encoding each bit of the message in the lowest bit of several carrier "blocks" in the file. According to the scheme, one out of every few samples will carry one bit of the message in its lowest bit.

1.2.2 Binary MP3 Encoding

Another binary-encoding scheme is MP3Stego, which works by inserting a secret file, in text format, into the carrier MP3 file during the compression process [Hsieh, Li, Hung 2007]. MP3Stego is based on manipulating bits in the MP3 encoding process in order to store information in the final file. The program uses the psychoacoustic model to

determine an acceptable amount of noise in the cover file, and limits the capacity to encode data to that threshold. One facet of our project examines binary versus non-binary encoding for MP3 files, which compares the capacity of MP3Stego against an encoding scheme that we developed.

1.2.3 Non-Binary Wave Encoding

There is another description of a non-binary scheme, which uses amplitude modulation (AM) to hide a wave file within the data chunk of another wave file by modulating the message with a high frequency wave and adding it to the cover file [Hsieh, Li, Hung 2007]. Similarly to our scheme, this system uses a modulating frequency placed outside of normal human hearing to encode the additional data. A combination of low-pass filters seeks to minimize noise in the resulting output file. Section 4.2 describes our system, which works in much the same way.

1.3 Similarities and Differences with Cryptography

Often steganography is confused with cryptography [Anderson, Petitcolas 1998]. The confusion is not in terms with name but with its appearance and its usage. The easiest way to differentiate the two is to remember that steganography conceals not only the contents of the message but also the mere existence of a message. It is not the same with cryptography.

The initial steganographic applications used null ciphers. These null ciphers are also called cleartext in technical terms. A null cipher or cleartext ensures that the message has not been encrypted in any way. Either it is using basic character shifting or substitution or advanced modern day encryption algorithm. It implies that the message is

often used in plain view and it can neither be detected as been there always or cannot be seen once detected.

One more striking similarity between cryptography and steganography is that both these fields have their initial roots in military and government applications. Later, steganography was developed in a more advanced way in ingenuity and complexity. There are many approaches available for the process of steganography as well as steganalysis.

According to Mr. Brainos, "*Steganalysis is the method by which to detect the presence of a hidden message and attempt to reveal the true contents of this message*." [Scribd 2007]. Like steganography, steganalysis was also eventually evolved throughout its history. The main problem with steganalysis is that it often lags behind with new steganographic discovery as an attempt to discover the hidden messages and eventually decipher them.

1.4 Various Methods of Digital Steganography

There are various methods available to achieve digital steganography. Steganography is being used in the media of text, images, and audio files.

1.4.1 Encoding Secret Messages in Text

Encrypting secret messages in text can be a very tedious and often challenging task. This is because text files have a very small amount of redundant data to replace with a secret message [Fen, Zhi 2001]. If the text files have vast amount of redundant data, the secret message that needs to be encrypted can be easily be incorporated into the text file by simply replacing few bits in a particular order.

Unfortunately, this is not the case with most of the text files. They have little to no redundant data. Another drawback is the ease of which text based steganography can be altered by unwanted parties by just changing the text itself or reformatting the text to some other form or format. This format may vary from .txt to .pdf to .doc etc.

There are many available methods by which to achieve the required text based steganography [Artz 2001].

1.4.2 Encoding Secret Messages in Audio files

Encoding secret messages in audio is the most challenging technique to use when dealing with steganography. This is the most challenging because of the human ability to distinguish small variations in sounds. The range of the human auditory power is dynamic. Humans can distinguish small changes in the audio with crystal clarity.

Audio steganography is a useful means for transmitting covert military information via a cover audio signal which is virtually untraceable. “The two primary criteria for successful embedding of a covert message are that the stego signal resulting from embedding is perceptually indistinguishable from the host audio signal, and the embedded message is recovered correctly at the receiver” [Gopalan 2003].

Other requirements such as robustness of embedding, data recovery without the original cover signal may depend upon the type of applications. In addition to these, for covert communication of key information, the embedded information must withstand channel noise and intentional attacks or jamming on the signal.

It is proven that the human auditory system perceives over a range of power greater than one million to one and a range of frequencies greater than one thousand to

one making it extremely hard to add or remove data from the original data structure [Gopalan 2005]. The only problem with this system is with the ability to differentiate sounds and this is what must be exploited to encode secret message in audio without being detected.

There are three main digital audio formats that are in extensive use today. They are:

- Sample Quantization
- Temporal Sampling Rate
- Perceptual sampling

Sample quantization which is a 16-bit linear sampling architecture used by popular audio formats such as wave format and AIFF format. Coming to *temporal sampling rate*, it uses selectable frequencies, preferably in the KHz, to sample the audio. In general, the higher the sampling rate is, the higher the usable data space gets. The last audio format is *perceptual sampling*. This format changes the statistics of the audio drastically by encoding only the parts the listener perceives, thus maintaining the sound but changing the signal. This format is used by the most popular digital audio on the Internet today in MP3 [Bao 2004].

Transmission medium must also be considered when encoding secret messages in audio. In general, the transmission medium is defined as the path the audio takes from sender to receiver. There are four possible transmission mediums introduced by Bender et al. [Bender 1996]. They are:

- Digital end to end
- Increased / decreased resampling

- Analog and resampled
- Over the air

In these transmission mediums, digital end to end represents machine to machine transmission without modification. In increased or decreased resampling, the sample rate is modified but the sample rate always remains digital [Cox, Miller, Bloom 2001]. Where as in analog and resampled transmission medium, the signal is changed to analog and resampled at a different rate. The last transmission medium is the over the air medium. In this medium, the signal is transmitted into radio frequencies and resampled from a microphone [Scribd 2007].

There are many encoding methods for performing audio steganography. The three of the most popular encoding methods for hiding data inside of an audio file are:

- Low-bit coding
- Phase coding
- Spread spectrum

1.4.2.1 Low-bit coding

Low-bit coding embeds secret data into the Least Significant Bit (LSB) of the audio file. The channel capacity in this encoding method is 1 KB per second per Kilohertz. This method is easy to incorporate but is very susceptible to data loss due to channel noise and resampling [Pooyan, Delforouzi 2007]. This method of coding is the simplest way to embed information in a digital audio file. By substituting the least significant bit of each sampling point with a binary message, Low-bit coding allows for a

large amount of data to be encoded. Figure 1.1 illustrates the message ‘HEY’ encoded as a 16-bit quality sample using the LSB method.

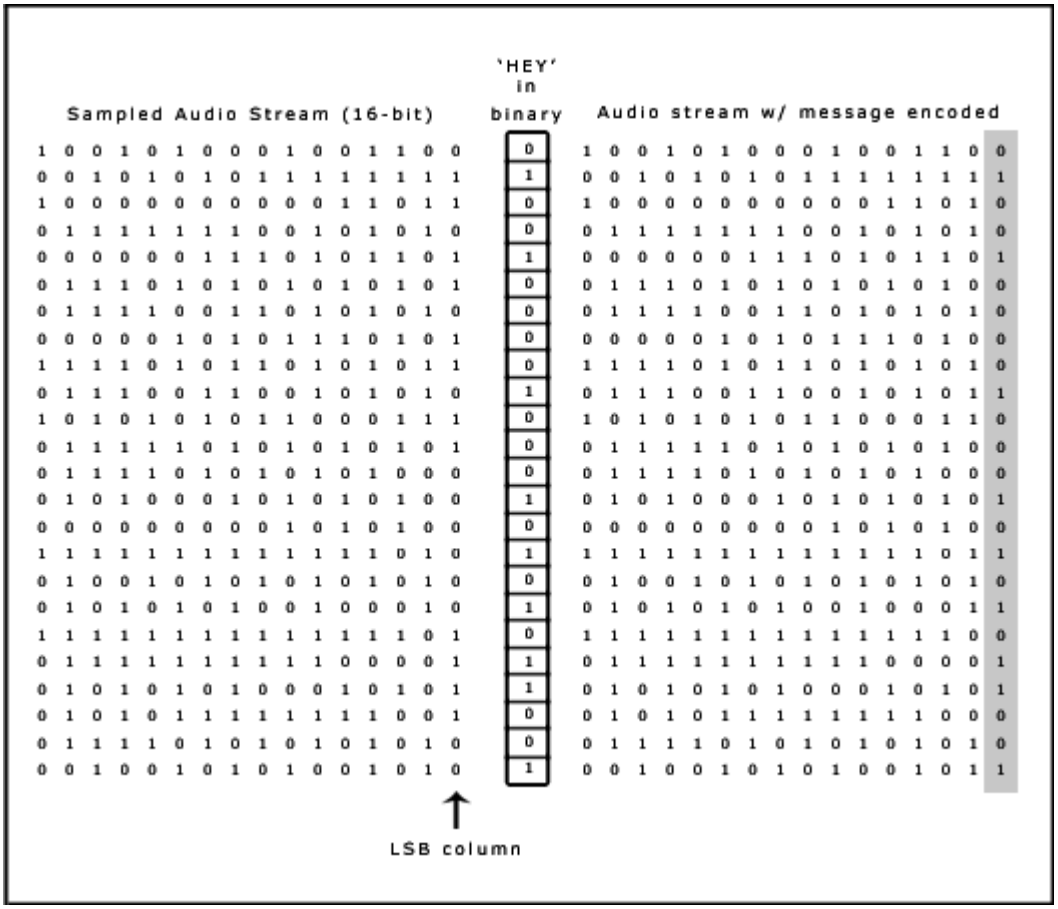

Figure 1.1 Low-bit coding example [Artz 2001].

To extract a secret message from an LSB encoded sound file, the receiver needs access to the sequence of sample indices used in the embedding process. Normally, the length of the secret message to be encoded is smaller than the total number of samples in a sound file. One must decide then on how to choose the subset of samples that will contain the secret message and communicate that decision to the receiver. One trivial technique is to start at the beginning of the sound file and perform LSB coding until the message has been completely embedded, leaving the remaining samples unchanged.

1.4.2.2 Phase coding

Phase coding substitutes the phase of an initial audio segment with a reference phase that represents the hidden data. This can be thought of, as sort of an encryption for the audio signal by using what is known as Discrete Fourier Transform (DFT), which is nothing more than a transformation algorithm for the audio signal. Phase coding addresses the disadvantages of the noise-inducing methods of audio steganography [Snortmonkey 2000].

Phase coding relies on the fact that the phase components of sound are not as perceptible to the human ear as noise is. Rather than introducing perturbations, the technique encodes the message bits as phase shifts in the phase spectrum of a digital signal, achieving an inaudible encoding in terms of signal-to-perceived noise ratio [Kavitha, Murugan 2007]. Figure 1.2 given below illustrates the difference between the original signal and the encoded signal after applying phase coding.

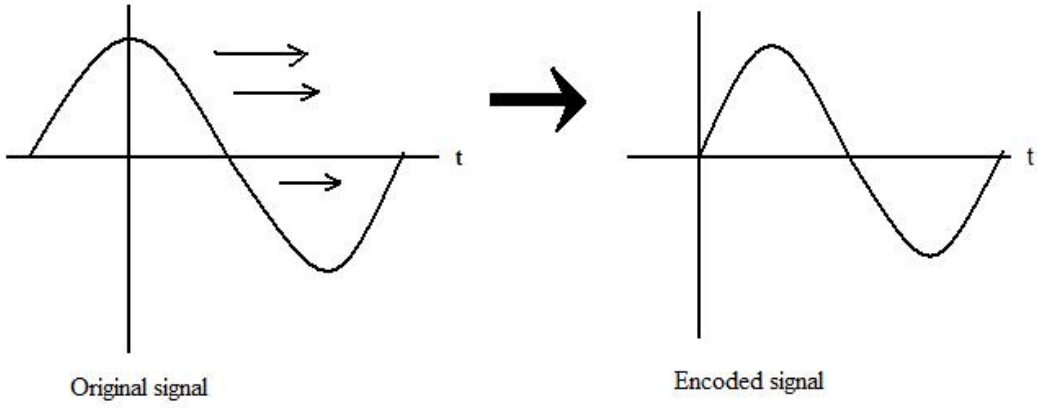

Figure 1.2 Phase coding example.

In phase coding, the original sound signal is broken up into smaller segments whose lengths equal the size of the message to be encoded. After this step, A Discrete

Fourier Transform is applied to each segment to create a matrix of the phases and Fourier transform magnitudes. Once the matrix is created, phase differences between adjacent segments are calculated.

After the phase differences were calculated, a new phase matrix is created using the new phase of the first segment and the original phase differences. The final step is to reconstruct the sound signal using the new phase matrix and original magnitude matrix by applying the inverse DFT and then concatenate the sound segments back together.

To extract the secret message from the sound file, the receiver must know the segment length. The receiver can then use the DFT to get the phases and extract the information. One disadvantage associated with phase coding is a low data transmission rate due to the fact that the secret message is encoded in the first signal segment only.

1.4.2.3 Spread spectrum

The spread spectrum method encodes the audio over almost the entire frequency spectrum. It then transmits the audio over different frequencies which will vary depending on what spread spectrum method is used. Spread spectrum encoding techniques are the most secure means by which to send hidden messages in audio, but it can introduce random noise to the audio thus creating the chance of data loss [Dunbar 2002].

Spread Spectrum is one of the audio steganography methods that analyze the frequency masking threshold with the help of a psycho acoustic model [Matsuoka 2006]. This model helps in embedding the spread signal in the audio just below the frequency masking threshold. The spread spectrum audio steganography reduces the error probability by increasing the spreading rate and coding gain.

2. STEGANOSENSE

The primary goal of SteganoSense is to provide end users the ability to apply steganography on wave audio files and secretly convey messages to the other end users. This project mainly concentrates on applying steganography to audio files. As discussed earlier, steganography can also be applied to video, image and text files. The main emphasis of this project is on developing a tool “SteganoSense” for use in audio steganography. The main purpose of SteganoSense is to transfer encrypted messages between two different parties. Also emphasis is based on preventing the intruders from detecting the encrypted or secret message.

SteganoSense is developed in such a way that it takes a wave file and the message to be encrypted as inputs and create a new stego wave file. The encrypt method that will be used in SteganoSense encrypts the file to be hidden first and later embeds into the wave file. For encrypting the file, the system uses AES as block cipher.

2.1 Process of Encryption and Decryption in Steganography

In general, steganographic process include various steps including encrypting the message, providing the key, hiding the message, decrypting the message and unhiding the message. The entire process of creating a steganographic wave file from a normal wave file is shown in Figure 2.1.

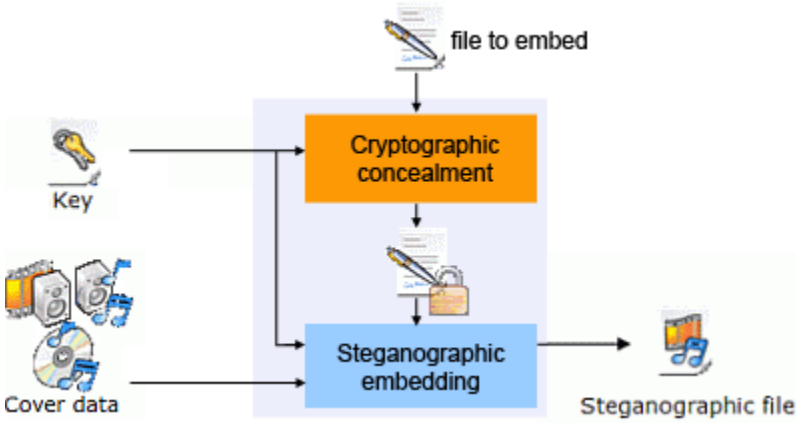

Figure 2.1 Steganographic file creation process.

The first and foremost step in the steganography process is to choose the file to be embedded or hidden and is known as the data file. In the present scenario, the file to be embedded would be an audio file with wave extension.

After selecting the data file, the next step in the steganography process is selecting the file that will serve as the hiding place. This file is termed as cover file or carrier file. In selecting the carrier file, there are several precautions that need to be taken care of. The most important one is the size of the carrier file. The size of the carrier file must be at least eight times as large as the data file in order for the encryption process to work.

The public key encryption is vital in the entire steganographic process because it plays crucial role in protecting the secrecy of the secrecy of the data file by using the key pair system to add a layer of robustness to the entire process. In this public key encryption, the public key of the receiver is used to encrypt the secret message and only the receiver's private key may decrypt it after extracting the data file from the cover data.

Encrypting the file can be done once the data file and the carrier file were selected. The output file is called the result file of encrypted steganographic file. The overview of the steganographic process is shown in Figure 2.1.

2.2 Hiding Covert Message

SteganoSense accepts a container wave file and a source data file as inputs. Wave file can be selected by clicking the “Browse” button near Source text box. Data file or covert file can be selected by clicking the “Browse” button near Container text box. If the user chose to encrypt the data, it is possible by checking “Enable encryption” check box. Checking “Enable Encryption” check box requires entering password twice to provide additional layer of security. The resulting wave file can be saved in the desired location by clicking the “Browse” button near Destination text box. This is shown in Figure 2.2.

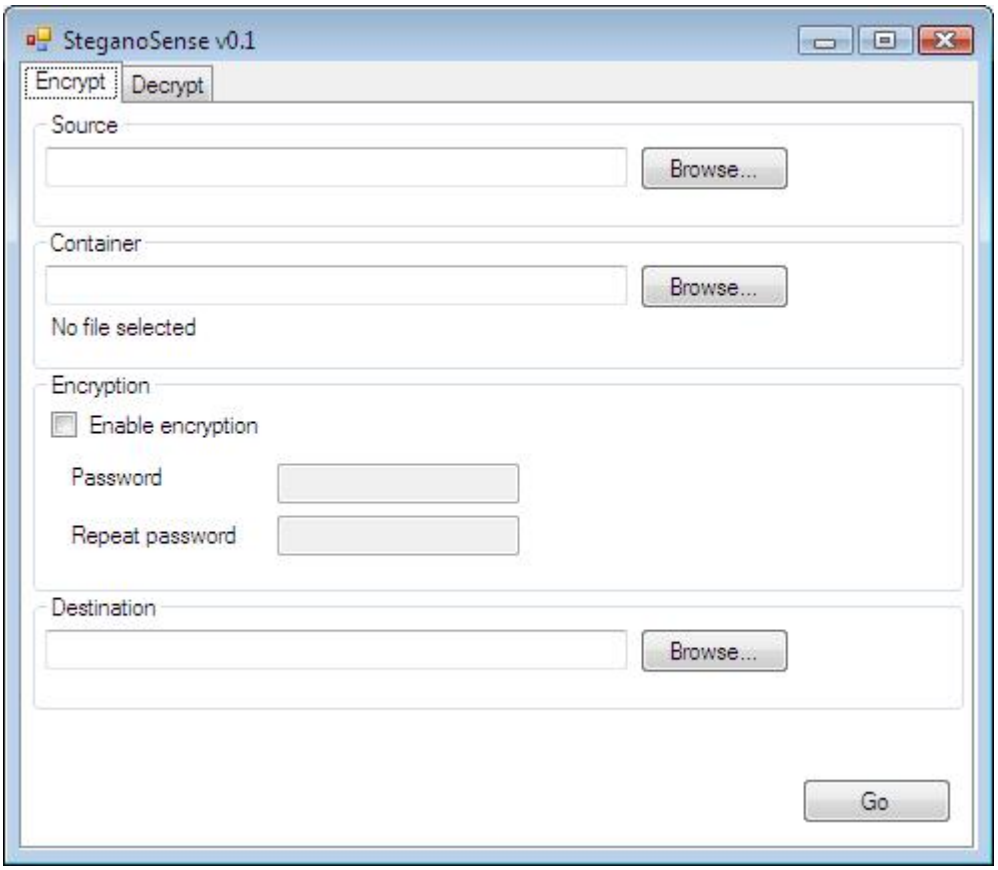

Figure 2.2 GUI for both encryption and decryption.

2.2.1 Selecting the Source File

Here, in this example, the source file is the text file. The only restriction that the SteganoSense imposes on selecting the source file is that the size of the source file must be less than one eighth of the size of the container file, in this case the wave file selected. This is shown in Figure 2.3.

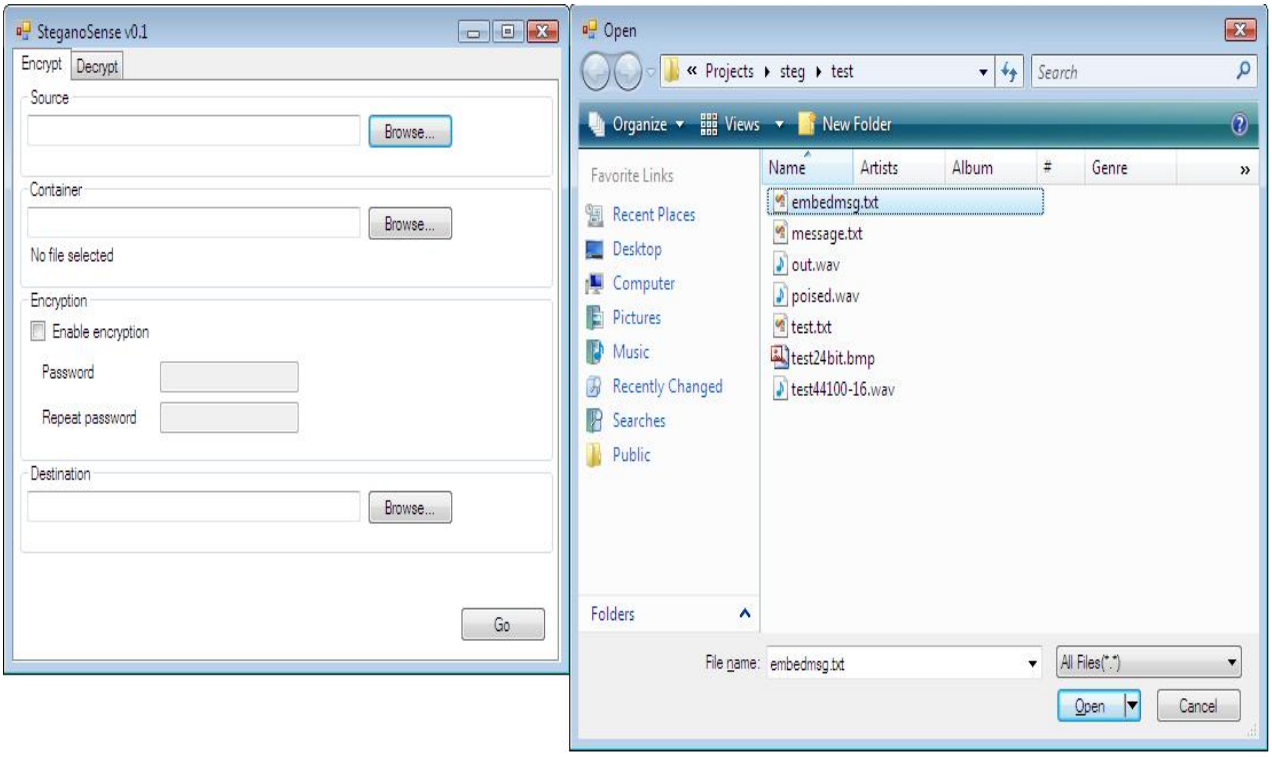

Figure 2.3 Selecting the source file.

2.2.2 Selecting the Container File

Here, the user selects the container file by using the browse button. In this scenario, the container file is the wave file. SteganoSense accepts any wave file of PCM audio format as input. A user hits the “Browse” button and selects a text file and hits “Open”. This is shown in Figure 2.4.

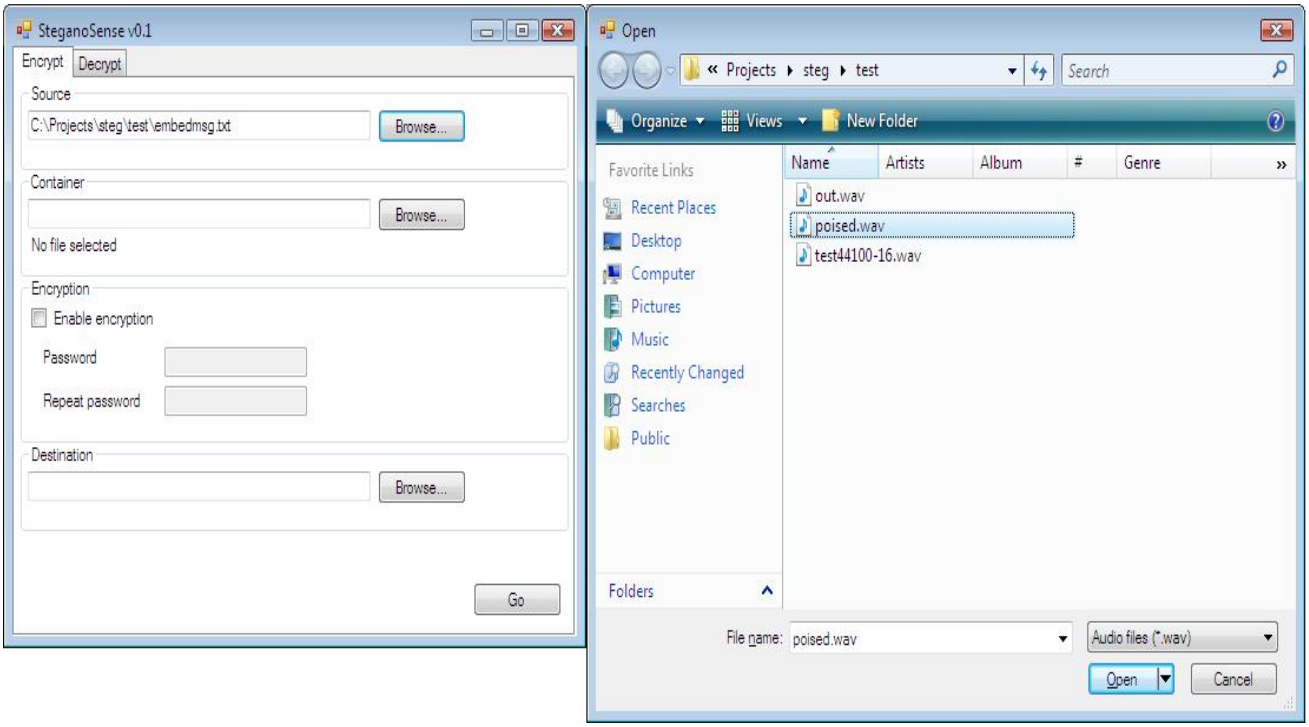

Figure 2.4 Selecting the covert file.

The user selects the wave file by using “Browse” button. Once any of the wave file was selected, user hits “Open” and the path loads in the subsequent text box.

2.2.3 Functionality of Password Match / Mismatch

The password fields are used for dual purposes. Both the sender and the receiver know the exact password to either hide or extract the hidden covert / secret file. While hiding the bits of a secret file into Least Significant Bits of a wave file, the chunk to start replacing LSBS can begin from anywhere. In general, the chunk to start replacing LSBs begins from the very first byte of the wave file. But here, the starting chunk to begin anywhere in the wave file and if it reaches the end, it comes back to the starting byte and completes replacing LSBs.

The starting chunk to begin replacing LSBs is determined by using the above password provided. The password fields are shown in the Figure 2.5.

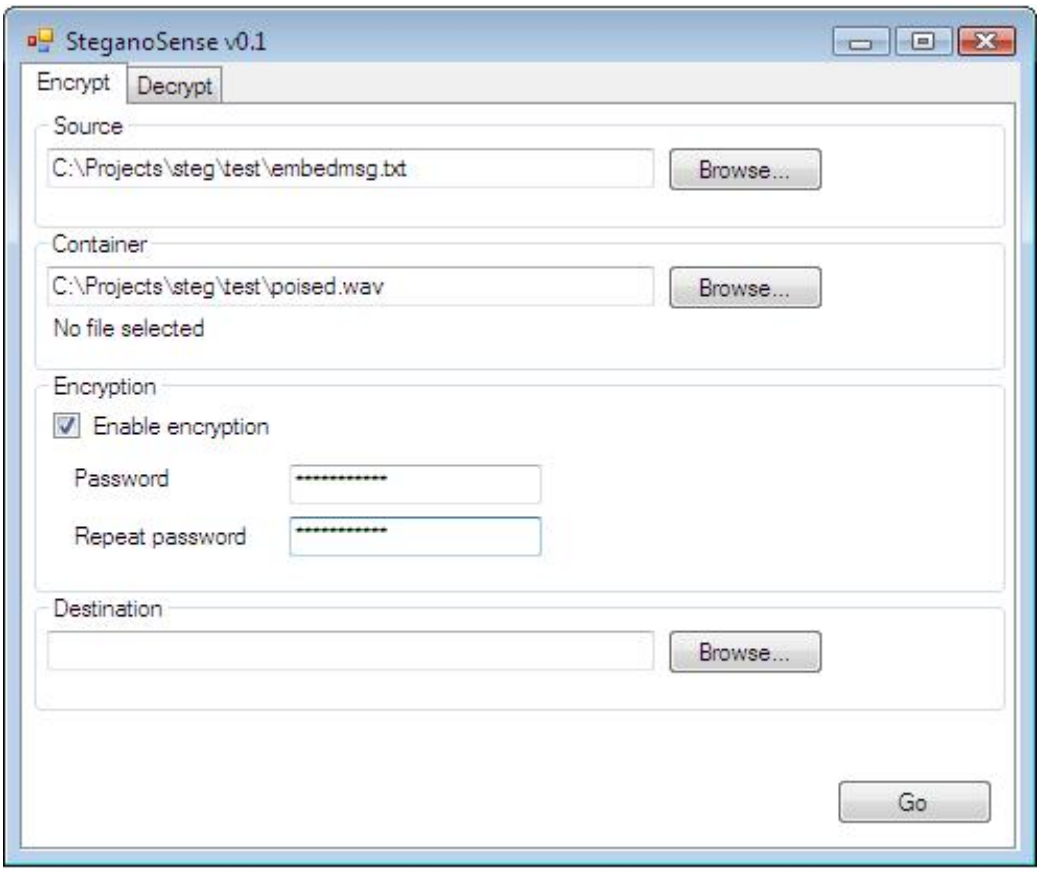

Figure 2.5 Password field functionality.

2.2.4 Naming the Output File

After all the fields are filled, the user selects the Encrypt button. Upon hitting the encrypt button, SteganoSense asks the user to enter the name of the output file and destination folder to save it. This output file is important in retrieving the secret message in the decryption process. This is shown in Figure 2.6.

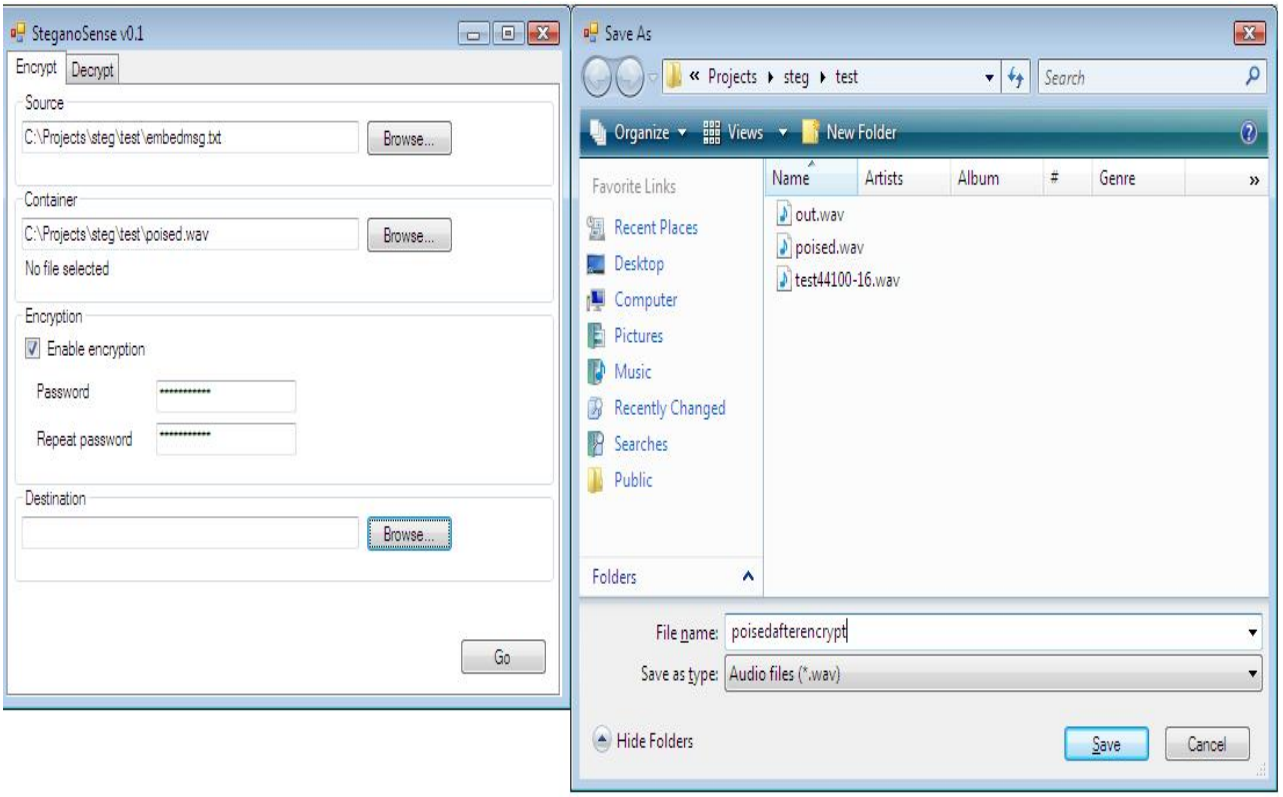

Figure 2.6 Creating by naming the output wave file.

2.2.5 Successful Creation of Output Wave File

When the user correctly selects the wave file, data file, and key file and enters the password in their respective fields and when the encryption is successful, SteganoSense creates the output wave file. This is shown in Figure 2.7.

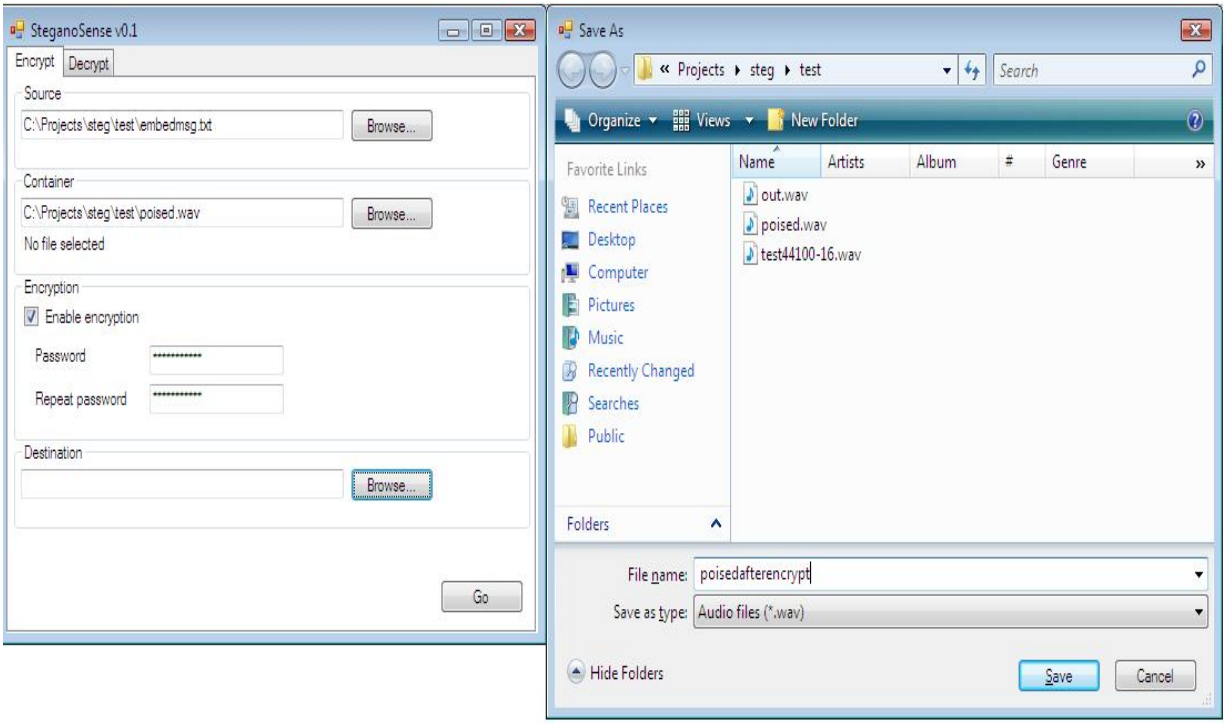

Figure 2.7 Successful creation of output wave file.

2.3 Decryption of the Message

Decryption of the message process includes three major steps. They are: selecting the encrypted file, selecting the key file and successful extraction of the message from the cover file.

2.3.1 Selecting the Encrypted File

Here, the user selects the encrypted file by using the browse button. In this scenario, the encrypted file is the newly created wave file. SteganoSense accepts that wave file as input. This is shown in Figure 2.8.

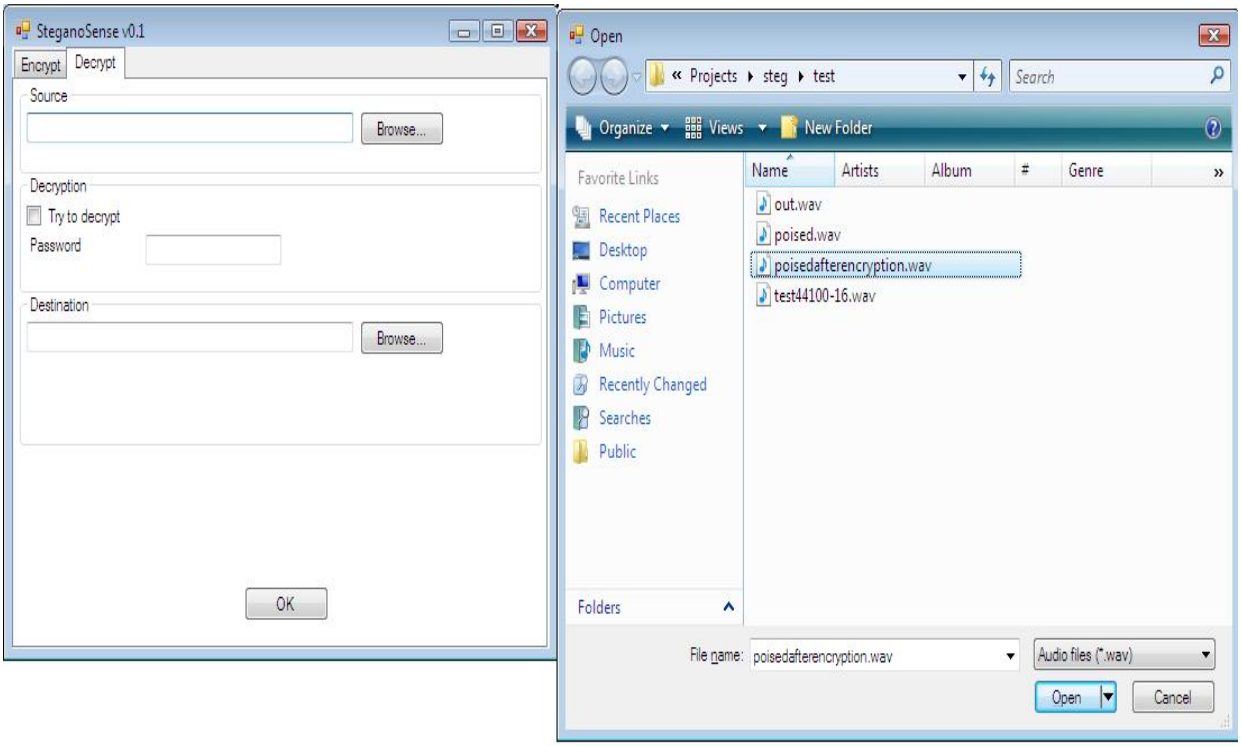

Figure 2.8 Selecting the encrypted file.

2.3.2 Successful Extraction of the Message

When the user correctly selects the encrypted wave file and the key file and enters the password in their respective fields and when the decryption is successful, SteganoSense successfully extracts the hidden message. This is shown in Figure 2.9.

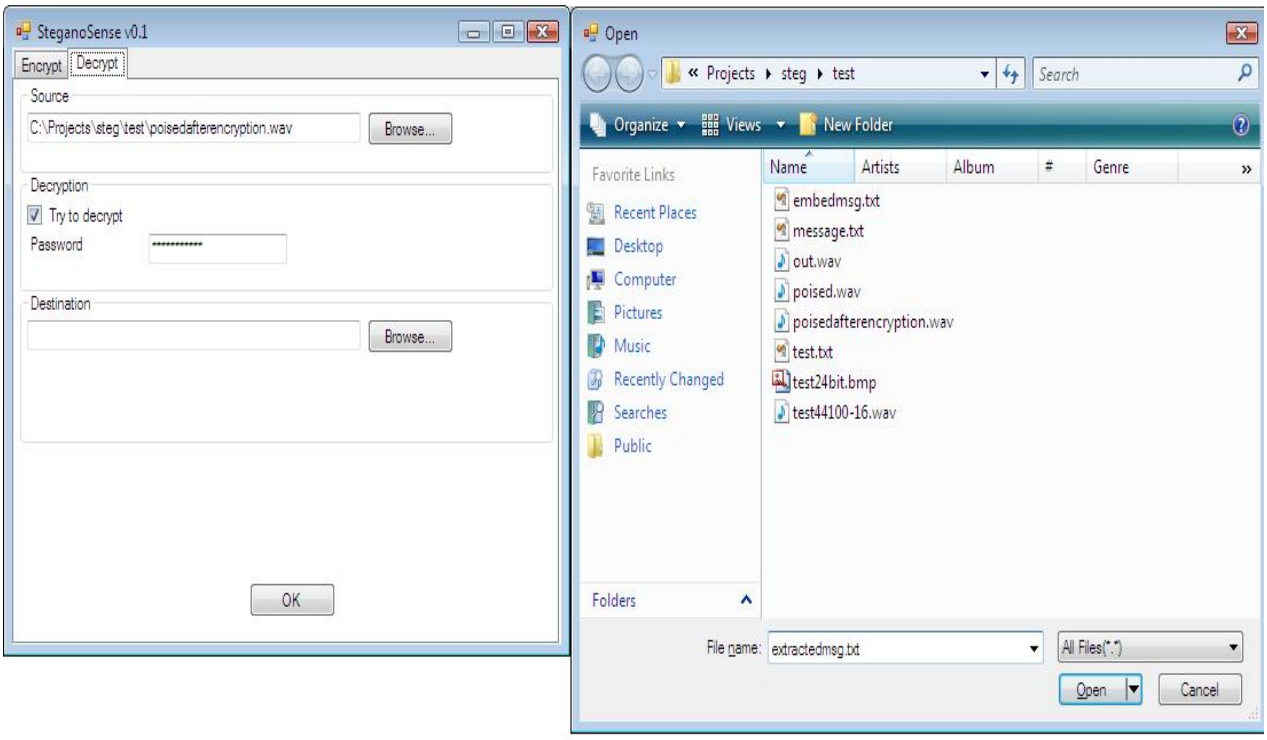

Figure 2.9 Successful extraction of the message.

3. SYSTEM DESIGN

SteganoSense can hide any file into an audio file. Here any file can be used as a secret file, and .wav file as a carrier file. The system first takes an audio file as carrier file, and accepts any secret file to hide into it. It uses Advanced Encryption Standard (AES) algorithm to encrypt secret file before hiding into audio.

The Advanced Encryption Standard (AES) is a computer security standard that became effective in 2002 to replace the standard DES [Wikipedia]. The cryptography scheme is a symmetric block cipher that encrypts and decrypts 128-bit blocks of data. Lengths of 128, 192 and 256 bits are standard key lengths used by AES Algorithm. The algorithm consists of four stages that make up a round which is iterated 10 times for a 128-bit length key, 12 times for a 192-bit key and 14 times for a 256-bit key [Jamil 2004].

The first stage “SubBytes” transformation is a non-linear byte substitution for each byte of the block. The second stage “ShiftRows” transformation cyclically shifts the bytes within the block. The third stage “MixColumns” transformation groups 4-bytes together forming 4-term polynomials and multiplies the polynomials with a fixed polynomial. In the final stage, “AddRoundKey” transformation adds the round key with the block of data. In AES algorithm, Plaintext refers to the data to be encrypted. Ciphertext refers to the data after going through the cipher as well as the data that will be going into the decipher.

3.1 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is a cryptographic algorithm that was approved by Federal Information Processing Standards Publications, also known as FIPS, to protect and secure any form of electronic data. AES algorithm can perform both encryption and decryption to obtain enciphering and deciphering of the information. It can do so because AES was classified as a symmetric block cipher.

The main difference between a block cipher and a stream cipher lies in the fact that a block cipher breaks up an unambiguous text into fixed length blocks and then performs the necessary encryption to the broken down blocks to obtain fixed-length blocks whereas, stream ciphers will work on continuous chunks of data that arrives in real time. In essence, stream ciphers operate on information bit by bit rather than on chunks by chunks or block-by-block.

The AES algorithm is capable of using cryptographic keys of 128, 192 and 256 bits to encrypt data and also can decrypt data in blocks of 128 bits.

3.1.1 AES Algorithm Specification

For the AES algorithm, the length of the input block, the output block and the state is 128 bits. The length of the Cipher Key, *K*, is 128, 192 or 256 bits. In this method, the number of rounds to be performed during the execution of the algorithm is dependent on the key size.

In AES algorithm, Key length is represented by N_k, Block size is represented by N_b and the number of rounds is represented by N_r. The only possible Key, block, round combinations are as illustrated in Figure 3.1.

	Key Length (N_k)	Block Size (N_b)	Total Rounds (N_r)
AES (128 bits)	4	4	10
AES (192 bits)	6	4	12
AES (256 bits)	8	4	14

Figure 3.1 Possible key combination in AES algorithm.

For both encryption and decryption process, AES algorithm uses a round function that is composed of four different byte-oriented transformations. They are:

- Byte substitution using a substitution table
- Shifting rows of the state array by different offsets
- Mixing the data within each column of the state array
- Adding a round key to the state.

3.1.2 Rijndael Algorithm

Rijndael (pronounced rain-dahl) is the algorithm that has been selected by the U.S. National Institute of Standards and Technology (NIST) as the candidate for the Advanced Encryption Standard (AES). It was selected from a list of five finalists that were themselves selected from an original list of more than 15 submissions. Rijndael will begin to supplant the Data Encryption Standard (DES) - and later Triple DES - over the next few years in many cryptography applications. The algorithm was designed by two Belgian cryptologists, Vincent Rijmen and Joan Daemen, whose surnames are reflected

in the cipher's name. Rijndael has its origins in Square, an earlier collaboration between the two cryptologists.

The Rijndael algorithm is a new generation symmetric block cipher that supports key sizes of 128, 192 and 256 bits, with data handled in 128-bit blocks - however, in excess of AES design criteria, the block sizes can mirror those of the keys. Rijndael uses a variable number of rounds, depending on key/block sizes, as follows:

- 9 rounds if the key/block size is 128 bits.
- 11 rounds if the key/block size is 192 bits.
- 13 rounds if the key/block size is 256 bits.

3.1.2.1 Description of the cipher

Strictly speaking, AES is not precisely Rijndael (although in practice they are used interchangeably) as Rijndael supports a larger range of block and key sizes; AES has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits, whereas Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits.

Due to the fixed block size of 128 bits, AES operates on a 4×4 array of bytes, termed the STATE (versions of Rijndael with a larger block size have additional columns in the state). Most of AES calculations are done in a special finite field.

High-level cipher algorithm:

- KeyExpansion using Rijndael's key schedule
- Initial Round
- AddRoundKey

3.1.2.2 Rounds

SubBytes — a non-linear substitution step where each byte is replaced with another according to a lookup table.

ShiftRows — a transposition step where each row of the state is shifted cyclically a certain number of steps.

MixColumns — a mixing operation which operates on the columns of the state, combining the four bytes in each column.

AddRoundKey — each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule.

Final Round (no MixColumns).

3.1.2.2.1 **The Sub-Bytes Step:**

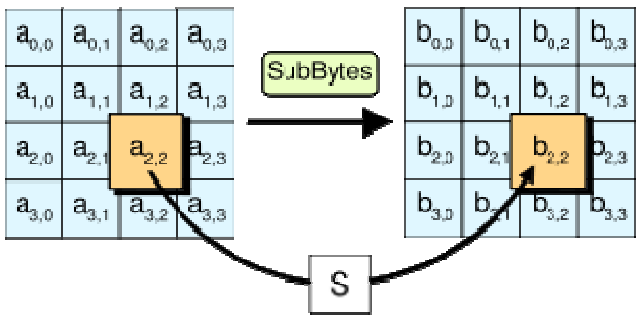

Figure 3.2 Diagram illustrating SubBytes step.

In the SubBytes step as shown in Figure 3.2, each byte in the state is replaced with its entry in a fixed 8-bit lookup table S as $b_{ij} = S(a_{ij})$.

In the SubBytes step, each byte in the array is updated using an 8-bit S-box. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF (28), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), and also any opposite fixed points.

3.1.3 Optimization of the Cipher

On systems with 32-bit or larger words, it is possible to speed up execution of this cipher by combining SubBytes and ShiftRows with MixColumns, and transforming them into a sequence of table lookups. This requires four 256-entry 32-bit tables, which utilizes a total of four kilobytes (4096 bytes) of memory--a kilobyte for each table. A round can now be done with 16 table lookups and 12 32-bit exclusive-or operations, followed by four 32-bit exclusive-or operations in the AddRoundKey step.

If the resulting four kilobyte table size is too large for a given target platform, the table lookup operation can be performed with a single 256-entry 32-bit table by the use of circular rotates. Using a byte-oriented approach it is possible to combine the SubBytes, ShiftRows, and MixColumns steps into a single round operation.

AES algorithm encryption technique generates a key based on the password, and thus the password is external to the system and not saved either in the code or the stego file generated. This can be given to recipient through a different medium of communication to ensure security. Thus the recipient needs the password, the original source code and the file extension of the hidden file to retrieve the secret message.

3.2 Wave File Format and Specification

The wave file was originally designed as a subset of Microsoft's RIFF specification. It can include lots of different kinds of data and information. Like any other files, wave files contain two standard parts namely the header part and the data part. The data part contains large chunks of bytes that represent the audio. In this, the chunk header specifies the type and size of the chunk data bytes. The header part follows the standard RIFF file format structure.

3.2.1 RIFF File Format

RIFF is a file format that can be used for storing any kind of multimedia data. This RIFF is based on chunks and sub-chunks. The entire RIFF file is a big chunk that contains all the other chunks. The form type of any file format will be stored in the contents of the RIFF chunk. This form type describes the overall type of the file's contents. Figure 3.3 below represents all RIFF chunks and therefore, wave chunks in general.

Offset	Size	Description
0x00	4	Chunk ID
0x04	4	Chunk Data Size
0x08	Chunk Data Bytes	

Figure 3.3 RIFF chunk format

The first four bytes in wave file represents chunk id. These first four bytes represents RIFF in a typical wave file. The next four bytes represents file size. The maximum file size that can be represented in wave file is stored in here. The next four bytes represents the file format. As this is a wave file, these four bytes should have WAVE in it. The subsequent sixteen bytes represents sub chunk id, sub chunk size, audio format, number of channels and sample rate. The next subsequent sixteen bytes represents byte rate, block alignment, bits per sample, sub chunk2 id and data size. The total wave file format can be best described in the Figure 3.4.

Bytes Position	Field Name	Field Size (in bytes)	Description of the format
0	Chunk ID	4	This should just contain "RIFF"
4	File Size	4	The size of the rest of the file after this field. Entire File Size – 8
8	File Format	4	This should just contain "WAVE"
12	SubChunk1 ID	4	This should just contain "fmt "
16	SubChunk1 Size	4	For PCM files, this will be 16.
20	Audio Format	2	Should be 1 for uncompressed audio. If this is something other than 1, you may have a compressed file.
22	Number of Channels	2	Either 1 or 2, or ???
24	Sample Rate	4	CD quality would be 44100.
28	Byte Rate	4	SampleRate * NumChannels * BitsPerSample / 8
32	Block Alignment	2	Channels * BitsPerSample / 8
34	Bits Per Sample	2	8, 16, 24 ...
36	SubChunk2 ID	4	This should just contain "data"
40	Data Size	4	The number of bytes following the header. The size of the data.

Figure 3.4 Wave file format specification.

3.3 Base Tool and the Previous Work

A tool developed by Mr. Odeti [Odeti 2007] is the base tool for this application. This application can hide any type of file into a wave file. Hiding secret file into a wave file can be accomplished in the following way.

3.3.1 Design of the Current Tool

SteganoSense tool developed here first extracts the size of the wave file from its header. This size information of the wave file is useful in determining the maximum size of the covert file. SteganoSense then calculates the size of the covert file (such a text or image file) in bytes and stores the calculated size in the first thirty two bytes of data in the wave audio file.

SteganoSense uses least significant bit insertion as the technique to replace wave audio LSBs with covert file bits. The least significant bit affects the smallest change of the eight bits. Suppose the size of the encrypted covert file is 2560 bits. This number can be represented in binary as “101000000000”. But this results in just twelve bits. The remaining 20 bits must be filled to fill in the first thirty two bits of the wave audio file. The best way to do this is to pad the first twenty bits with all zeroes. Filling any number of zeroes on left doesn’t change the total value of the binary representation. Adding twenty zeroes to the above number finally gives us “00000000000000000000101000000000”. This representation is then stored in the first 32 least significant bits of the wave audio file. Even though some data space has been taken up to add this information, it is very important to have this information in the wave audio file.

It is important to note that the covert file size can be no greater than one eighth of the size of the wave audio file. SteganoSense does not allow the covert file size to be greater than one eighth of the size of the wave file. As SteganoSense does not even touch the header part of the wave audio file, the properties remain intact no matter all the remaining least significant bits of the wave audio file are replaced. This is important as

SteganoSense creates new wave audio file that cannot be distinguished from the original wave audio file.

The two important distinguished features in SteganoSense tool are encrypt and hide process and decrypt and retrieve process.

3.4 Encrypt and Hide Process

Encryption and hiding of the covert file into a wave file is achieved in SteganoSense in following steps. All the following steps have been implemented correctly to achieve perfect wave audio steganography. The entire process can be broken down into few important steps:

- Convert the wave file and the covert file into binary representation.
- Encrypt the covert file using AES algorithm.
- Create a random unique number from password given.
- Use LSB substitution to replace wave file bits with encrypted data bits.
- After substitution is done, create a new stego wave file.

3.4.1 Convert Wave file and the Source File into Binary

To convert a wave file and the covert file into their binary form of representation, the files were opened using ***fstream*** in the function call that converts both files into binary representation. To know the beginning chunk of the input file (either the wave file or covert file), the function uses ***seekg*** member function. A second overload of this ***seekg*** member function takes a second parameter that allows specifying starting of the file, ending of the file or the current position in the file.

After this the function simply reads in bytes as ***chars*** from the wave file input stream and writes them out formatted appropriately as text to the data file stream. The sample wave file is shown in Figure 3.5. The example binary bit representation is shown in Figure 3.6.

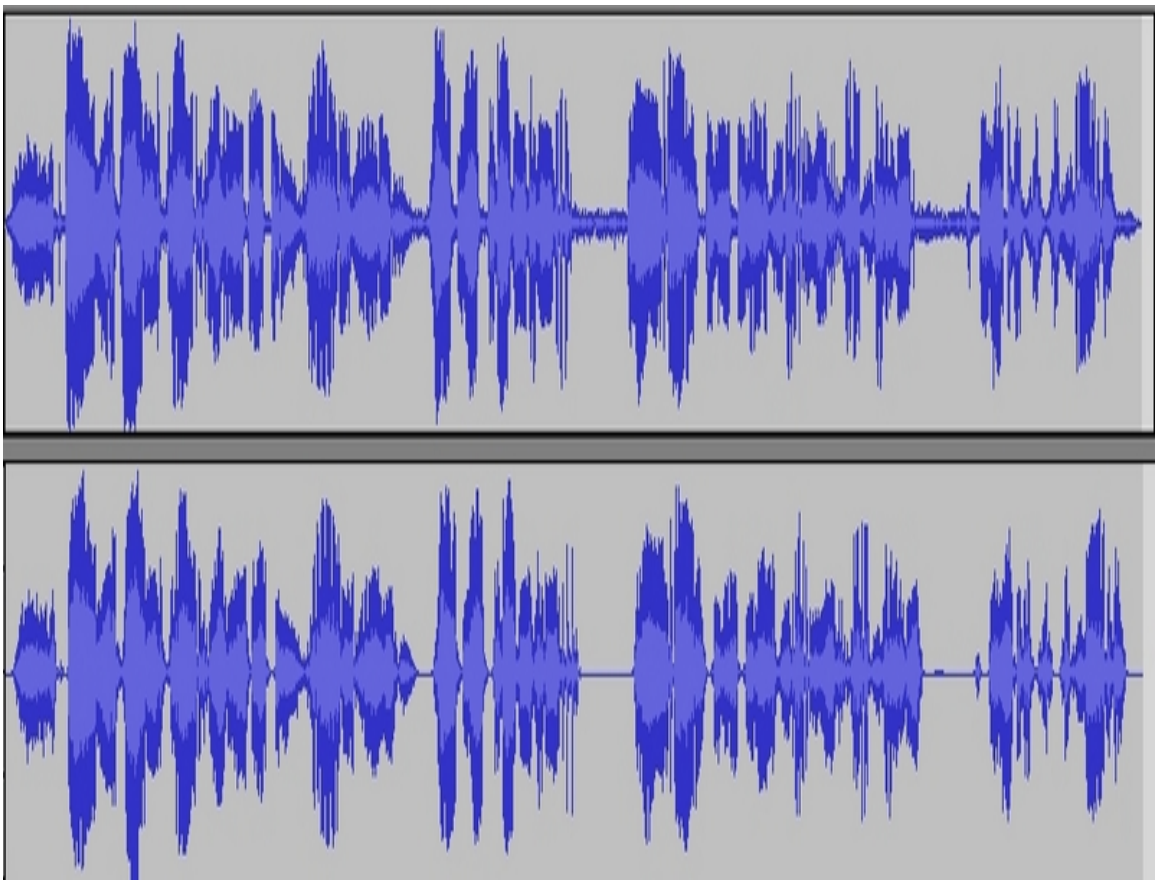

Figure 3.5 Sample Wave File in Analog representation.

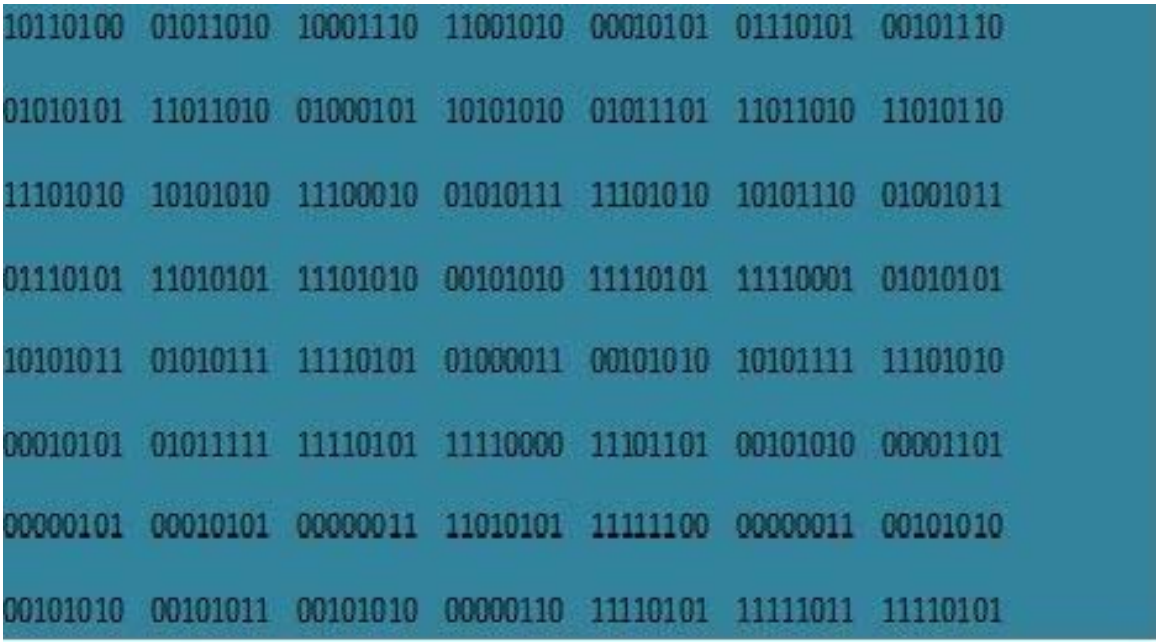

Figure 3.6 Example binary bits representation.

3.4.2 Encrypt Covert File using AES Algorithm

Encryption of the covert file was achieved in this tool, SteganoSense, by successfully implementing Advanced Encryption Standard (AES) Algorithm. AES, also known as Rijndael, is a substitution linear transformation cipher, not requiring a Feistel network. It uses triple discreet invertible uniform transformations (layers). Specifically, these are: Linear Mix Transform; Non-linear Transform and Key Addition Transform [Jamil 2004]. Even before the first round, a simple key addition layer is performed, which adds to security. Thereafter, there are *Nr*-1 rounds and then the final round. The transformations form a State when started but before completion of the entire process.

The State can be thought of as an array, structured with 4 rows and the column number being the block length divided by bit length (for example, divided by 32). The cipher key similarly is an array with 4 rows, but the key length divided by 32 to give the

number of columns. The blocks can be interpreted as uni-dimensional arrays of 4-byte vectors.

The exact transformations occur as follows: the byte subtransformation is nonlinear and operates on each of the State bytes independently - the invertible S-box (substitution table) is made up of 2 transformations. The **shiftrow** transformation sees the State shifted over variable offsets. The shift offset values are dependent on the block length of the State. The **mixcolumn** transformation sees the State columns take on polynomial characteristics over a Galois Field values (28), multiplied $x^4 + 1$ (modulo) with a fixed polynomial. Finally, the **roundkey** transform is XORed to the State. The key schedule helps the cipher key determine the round keys through key expansion and round selection.

3.4.3 Generating Random Unique Number from Password

Generating a random unique number from the password entered during the "encryption and hiding" process in SteganoSense is very important as this number is used to decide the chunk number to start replacing least significant bits of the wave audio file. To generate this random unique number, the password was hashed first and using that hash the random number was generated.

The hashing of the password was done using MD5 hash function. The MD5 functions were used to generate a condensed representation of the message. The MD5 functions are considered to be more secure than the other hash functions with which they share a similar interface.

MD5 algorithm used in this project first changes a variable-length message into a fixed length output of 128 bits. The input message is broken down into chunks of 512 bit blocks. The 128 bits are divided again into four blocks of 32-bit words.

The hash function take a sequence of integers k=k1,..,kn and produce a small integer bucket value. In general, in c++, a character is a char variable which is an 8-bit integer. But ASCII uses only seven of these eight bits. Of these seven bits, the common characters use only the low-order six bits and the first of these six bits primarily indicates the case of characters which is relatively insignificant. So, the hash function used concentrated on preserving as much information as possible from the last five bits of each number and make less use of the first three bits.

The hash function used was pre computed and was stored in an array that returns random numbers. The process done was to add each character to the output value, do a one bit left circular shift of the output value and then perform exclusive OR function. The pseudo code for the random generating hash function was as follows:

Extract high order bit from the output value

Shift output value left by one bit

Move them to the low order end and

Perform XOR into output value

XOR output value and the random value for Ki.

This random number was used to identify the starting chunk to start replacing least significant bits. While replacing the LSBs of the wave audio file, if the chunk reaches the end of the file and still has some LSBs to replace, the process will start from the beginning chunk of the data part of the wave file. This can be viewed as the circular LSB replacement. No LSB of the header part of the wave file will be replaced as the header part was the key to hold the original properties of the wave file intact.

3.4.4 Use LSB Substitution to Replace Wave File Bits with Encrypted Data Bits

Least significant bit insertion works on the basic fact that the accuracy level in the audio and image formats are much greater than that perceivable by human listening capacity and image vision. Any alterations made to the original cover file will be impossible to tell apart going by the standard human levels.

A 24-bit bitmap has a bitmap header in the following notation.

10010101 00001101 11001001 10010110 00001111 11001011 10011111 00010000

For example, if the character “S” is to be inserted into the eight bytes of the cover file, it is done in the following way. An “S” is represented in the American Standard Code for Information Interchange (ASCII) as the binary string 01000111. These eight bits can be inserted into least significant bit of the above header in the following way.

1001010*0* 0000110*1* 1100100*0* 1001011*0* 0000111*0* 1100101*1* 1001111*1* 0001000*1*

The italicized bits are the LSB inserted bits.

In the Least Significant Bit substitution method, the first thing to do is to extract the wave file size from its header. The Least Significant Bit substitution method is used in this project to replace wave file bits with encrypted covert file data bits. This process was achieved in SteganoSense in following steps:

- Extract the wave file size from its header.
- Obtain the message or secret file size after encryption.
- Store this information in the first thirty two bytes in the data part.
- Calculate displacement. This will be used in pin-pointing exact replacement bits.
- Convert the secret file to bit representation.
- Create a random unique number using the password given to identify the chunk to start replacing LSBs.
- Start replacing each LSB beginning from the starting chunk.
- Move forward by the displacement.

3.4.5 Creating New Stego Wave File

All the above steps are used in creating a new stego wave file. As explained above, the header part of the new stego wave audio file will be exactly same as the original wave audio file. The remaining data part after the replacement of least significant bits is written into the data chunk part of the wave file.

The crucial process in wave steganography is to hide any kind of data or chunk of information in wave files. The purpose of this project is to hide any kind of file into an audio file, and transmit the audio file over the network from a sender to a receiver and

block other potential intruders from knowing the fact that an encrypted message was being transmitted. Thus, any intruder or hacker on the network may understand that an audio file is being transmitted, but the actual message hidden in audio file in invisible.

3.5 Decrypt and Retrieve Process

Decryption and retrieval of the covert file from a stego wave file is achieved by using the following steps. The entire process can be broken down into few important steps.

- Retrieve the replaced bits in the wave audio file to get back the covert file.
- Decrypt the encrypted bits of the covert file.
- Generate the plaintext covert file.

3.5.1 Retrieving the Replaced Bits in Wave Audio File

The process of retrieving the replaced bits in the wave audio file to get back the original covert file in its form in entirety involves few important steps. They are:

- SteganoSense reads the header file of the newly created encrypted wave file to obtain information about the size of the cover file. The file size information is and will always be stored in the header section of that particular file.
- After obtaining the header file, the *SteganoSense* tool reads the first thirty two bits of the header file to obtain information about the size of the covert file.

- After obtaining the covert file size, the *SteganoSense* tool calculates the displacement value. The displacement value can be obtained by dividing the value of the wave file size with the value of the covert file size. This displacement value points out the position from which the least significant bits of the wave file were replaced by least significant bits of the covert file.
- After obtaining the displacement position value, the *SteganoSense* tool starts extracting the replaced bits to form the original covert file.
- After the first position, SteganoSense moves forwards by the value of the displacement and extracts all the remaining replaced bits and this process is continued until reaching the end of the wave file.

3.5.2 Decrypt the Encrypted Bits

In order to decrypt the extracted encrypted bits, the end user must know the password that was used to encrypt the file. If the correct password is entered, the SteganoSense begins the process of decrypting the encrypted bits. This decrypting process only works in case of the correct password. If the password mismatch occurs, SteganoSense does not go any further and warns the user about the mismatch.

3.5.3 Generate New Source File

After the process of decryption was done, SteganoSense uses the extracted original bits to construct the covert file.

4. TESTING, EVALUATION, AND RESULTS

Correctness is the minimum requirement of software, the essential purpose of testing. To ensure that the designed system will work without any problems, few testing approaches like black-box testing, white-box testing can be applied on the system [Howden 1987]. The important factor in this testing is that both the black-box and white-box testing ideas are not limited to correctness testing, but can be applied in various other testing techniques.

4.1 Black-box Testing

The black-box approach is a testing method in which test data are derived from the specified functional requirements without regard to the final program structure. Black-box testing can also be called as data-driver testing, input or output driver testing or requirements-based testing. It can also be referred to as functional testing because it is a testing method emphasized on executing the functions and examination of their input and output data [Howden 1987].

In this testing method, the system that has to be tested must be treated like a black box; only the inputs, outputs and specification are visible, and the functionality is determined by observing the outputs to corresponding inputs. When testing the system, various inputs are exercised and the outputs are compared against specification to validate the correctness.

The functionality of SteganoSense is tested using black-box testing method using two wave files and two text files. The first wave test file are of size of 21 MB and the text file (covert file) is of size 1.2 MB. After hiding the text file in audio file, the size of the wave audio file is exactly 21 MB. The second wave test file of the size of 1.2 MB and the

text file were of 124 KB. After performing the steganography process on these files, the size of the wave file remained same at 1.2 MB.

Also, after decryption and retrieving the text file, the size of the text files remained same as the original covert files. Also, there are no additional lines or any other special characters in the text files. Hence, black-box testing was successful.

4.2 White-Box Testing

Contrary to black-box testing, the system that need to be tested is viewed as a white-box, or glass-box in white-box testing, as the structure and flow of the software under test are visible to the tester. In white-box testing, testing plans are made according to the details of the software implementation, such as programming language, logic, and styles. Test cases are derived from the program structure. White-box testing is also called glass-box testing or logic-driven testing or design-based testing [Hetzel, 1988].

The code written to develop SteganoSense is successfully tested and there have been no errors or warnings when building and debugging the solution. Also, all the modules written are individually tested resulting in no errors.

4. 3 Performance Testing

Not all software systems have specifications on performance explicitly. But every system will have implicit performance requirements. The software should not take infinite time or infinite resource to execute. "Performance bugs" sometimes are used to refer to those design problems in software that cause the system performance to degrade.

Performance has always been a great concern and a driving force of computer evolution. Performance evaluation of a software system usually includes: resource usage,

throughput, and stimulus-response time and queue lengths detailing the average or maximum number of tasks waiting to be serviced by selected resources. Typical resources that need to be considered include network bandwidth requirements, CPU cycles, disk space, disk access operations, and memory usage [Smith 1990].

All the above mentioned tests are performed on SteganoSense and the results are satisfactory.

4. 4 Usability Testing

Usability testing is usually conducted to obtain data and statistics about a project or a product that is yet to be released. These statistics are important to assess the stability and user readiness of the software product. Various advantages of the usability testing include calculating the time it takes for a user to understand the software product, the speed with which the product was run, its smoothness and the rate of human error, and the user satisfaction in using the product.

Conducting usability tests has its advantages. Usability tests alone can cover many other testing types like functional testing, system integration testing, smoke testing et cetera. This helps in covering all possible directions in the field of testing. Usability testing is very inexpensive if planning is good and yet it can be highly successful and advantageous. The main advantage is that this form of testing helps in revealing many potential bugs that the developers are unaware of.

Usability testing measures few vital points like how much time the user and system took to finish the basic flow while using the product and how much time users

took to understand the system and whether they made any mistakes while understanding the flow of the product.

As part of conducting usability testing, the project executable "*SteganoSense*" was given to four users to test the product. The main features that are considered in this testing procedure are

- Measure the level of understanding of the users while testing *SteganoSense*. With the easier and simple nature of graphical user interface, all the testers had no problems in the flow of the product.
- Asked the users if they observed any difference in the file size or the quality of the wave audio and quality of BMP while testing *SteganoSense* for at least five times and with five different files of varying size specifications. None of the users observed any changes in the file size or the quality.

Figures 4.1 to 4.4 depict the first set of the various files with their description details. Figure 4.1 shows the initial size and description of the wave file that was used as cover file. Figure 4.2 shows the initial size and description of the text file that was used as the covert file. Figure 4.3 show the size and description of the wave file that was created by *SteganoSense* after encryption. Figure 4.4 shows the size and description of the text file that was extracted by *SteganoSense* after decryption.

Set 1:

Initial Wave file size

Figure 4.1 Initial wave file description.

Initial text file size

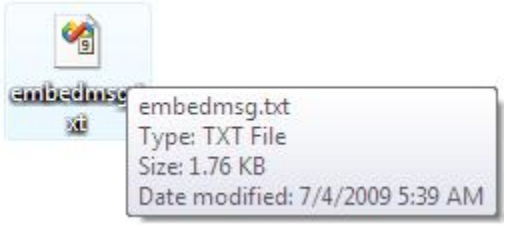

Figure 4.2 Initial text file description.

Wave file size after applying Steganography

Figure 4.3 Encrypted wave file description.

Text file size after decryption

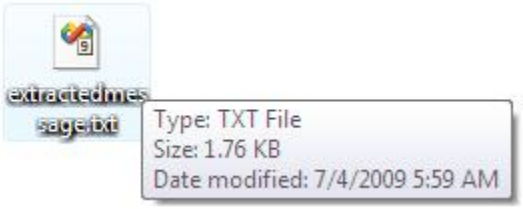

Figure 4.4 Decrypted text file description.

Figures 4.5 to 4.10 depict the second set of the various files with their description details. Figure 4.5 shows the initial size and description of the wave file that was used as cover file. Figure 4.6 shows the initial size and description of the image file that was used as the covert file. Figure 4.7 displays the original image file that was used before encryption. Figure 4.8 shows the size and description of the wave file that was created by *SteganoSense* after encryption. Figure 4.9 shows the size and description of the image file that was extracted by *SteganoSense* after decryption. Figure 4.10 displays the extracted image file that was obtained by SteganoSense after decryption.

Set 2:

Initial wave file size

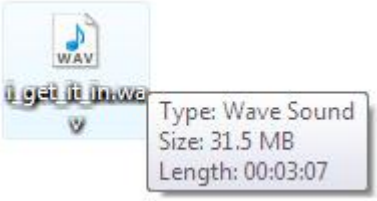

Figure 4.5 Initial wave file size description.

Initial image file size

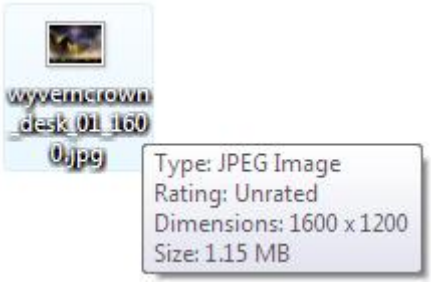

Figure 4.6 Initial image file size description.

Original Image used as covert file:

Figure 4.7 Original image used before encryption.

Wave file size after applying Steganography

Figure 4.8 Wave file size after applying steganography.

Image file size details after decryption

Figure 4.9 Extracted Image file size description.

Extracted image after decryption:

Figure 4.10 Extracted image after decryption.

5. FUTURE WORK

Communicating secretly without giving away any kind of crucial information is very important now a days in many fields. The application developed in this project, SteganoSense, achieves the above mentioned goal. In this application, a secret message file that needs to be transmitted must be small in size to achieve successful transmission. SteganoSense primarily concentrates on this issue.

To hide the secret file covertly in an audio file, the Least Significant Bits of the audio file are replaced with each bit of the secret file. SteganoSense can be used not only for Wav audio files but also few of the image formats like BMP to apply steganography onto them. SteganoSense uses Advanced Encryption Standard algorithm for the encryption process. SteganoSense provides more robustness and security compared to the waveStego tool.

SteganoSense tool can be extended to make it work on Video file format and other formats like 3GP, AVI etc. At present, Wave audio files that are of PCM audio format works with steganoSense tool. Other audio formats require some form of compression. This can be solved in future work. Also, Instead of LSB, spread specturm method or phase coding method can be used to insert secret message bits.

ACKNOWLEDGEMENTS

The preparation of this report and completion of the project was successful because of the never ending support of Dr. Dulal Kar, Associate Professor of the Department of Computing Sciences, Texas A & M University – Corpus Christi. Dr. Dulal Kar's suggestions, comments, and guidance throughout the project had tremendously helped to ensure to the success of the project.

I would like to express my sincere thanks to Dr. Longzhuang Li, Associate Professor of the Department of Computing Sciences, Texas A&M University – Corpus Christi for his unending support and for providing guidance which helped me towards completion of my project.

I would like to express my sincere thanks to Dr. Ajay Katangur, Assistant Professor of Computing Sciences, Texas A&M University – Corpus Christi, for accepting to serve as my committee member without which the completion of the process would have been much difficult.

I would like to express my sincere thanks to Dr. David Thomas, Associate Professor of Computing Sciences, Texas A&M University – Corpus Christi, for his unending support and warm wishes that helped me to concentrate on completing my project.

My sincere heartfelt thanks to all the faculty, and staff of the Department of Computing Sciences for their outstanding support.

Last but not least, I would like to thank my parents, family and few of my friends who provided the much needed moral support and boosted me in reaching the successful completion of the project.

BIBLOGRAPHY AND REFERENCES

[Anderson, Petitcolas 1998] *On the Limits of Steganography.* IEEE Journal on Selected Areas in Communications, Vol. 16, No. 4, May 1998. IEEE.

[Artz 2001] Artz, D. *Digital Steganography : Hiding Data within Data.* IEEE Internet Computing, May 2001. IEEE.

[Bender 1996]. *Techniques for Data Hiding, IBM system Jorunal, Pgs 313-336.*.

[Brainos II 2003] Brainos II, A. C. *A Study of Steganography And The Art of Hiding Information,* East Carolina University, November 13, 2003.

[Cox, Miller, Bloom 2001] Cox, J. Miller, L. & Bloom, A. *Digital Watermarking.* Boston: Morgan Kaufmann, 2001.

[Dunbar 2002] Dunbar, B. *A detailed look at Steganographic Techniques and their use in an Open-Systems Environment*. SANS Institute 2002.

[Fen, Zhi 2001] Fen, S. A. Zhi, L. *Detection of Random LSB Image Steganography*. IEEE 2004.

[Gopalan 2005] Gopalan, K. *Audio Steganography by Cepstrum Modification*. ICASSP 2005, IEEE.

[Gopalan 2003] Gopalan, K. *Audio Steganography Using Bit Modification*. ICASSP 2003, IEEE.

[Howden 1987] William E. Howden. *Functional program Testing and Analysis.* McGraw-Hill, 1987.

[Hsieh, Li, Hung 2007] Hsieh, C. Li, J. Hung, C. *A Robust Audio Fingerprinting Scheme for MP3 Copyright*. IIHMSP 2007, IEEE.

[Jamil 2004] Jamil, T. *The Rijndael Algorithm*. IEEE Potentials 2004. IEEE.

[Kavitha, Murugan 2007] Kavitha, R. Murugan, A. *Lossless Steganography on AVI File using Swapping Algorithm.* ICCIMA 2007. IEEE.

[Matsuoka 2006] Matsuoka, H. *Spread Spectrum Audio Steganography using Sub-band Phase Shifting*. Proceedings of the 2006 International Conference on IIH-MSP'06. IEEE.

[Pooyan, Delforouzi 2007] Pooyan, M. Delforouzi, A. *LSB-based Audio Steganography Method Based on Lifting Wavelet Transform.* IISSPIT 2007. IEEE.

[Smith 1990] Smith, C. U. *Performance Engineering of Software Systems.* Addison-Wesley, 1990.

[Odeti 2007] Odeti, S. R. *Wave Steganography.* Master's Project. 2007.

[Scribd 2007] http://www.scribd.com/doc/20529/Seminar-on-Steganography (visited April, 2008)

[Snortmonkey 2000] http://www.snotmonkey.com/work/school/405/methods.html (visited April, 2008)

[Wikipedia] Available from http://en.wikipedia.org/wiki/ (visited April, 2008).