CHAPTER 1: STUDY ON AUDIO STEGNOGRAPHY

1.1 A BRIEF HISTORY OF STEGANOGRAPHY

The earliest recordings of Steganography were by the Greek historian Herodotus in his chronicles known as "Histories" and date back to around 440 BC. Herodotus recorded two stories of Steganographic techniques during this time in Greece. The first stated that King Darius of Susa shaved the head of one of his prisoners and wrote a secret message on his scalp. When the prisoner's hair grew back, he was sent to the Kings son in law. Aristogoras in Miletus undetected.

The second story also came from Herodotus, which claims that a soldier named Demeratus needed to send a message to Sparta that Xerxes intended to invade Greece. Back then, the writing medium was text written on wax- covered tablets. Demeratus removed the wax from the tablet, wrote the secret message on the underlying wood, recovered the tablet with wax to make it appear as a blank tablet and finally sent the document without being detected.

Romans used invisible inks, which were based on natural substances such as fruit juices and milk. This was accomplished by heating the hidden text, thus revealing its contents. Invisible inks have become much more advanced and are still in limited use today. During the 15th and 16th centuries, many writers including Johannes Trithemius (author of Steganographia) and Gaspari Schotti (author or Steganographica) wrote on Steganagraphic techniques such as coding techniques for text, invisible inks, and incorporating hidden messages in music.

Between 1883 and 1907, further development can be attributed to the publications of Auguste Kerckhoff (author of Cryptographic Militaire) and Charles Briquet (author of Les Filigranes). These books were mostly about Cryptography, but both can be attributed to the foundation of some steganographic systems and more significantly to watermarking techniques.

During the times of WWI and WWII, significant advances in Steganography took place. Concepts such as null ciphers (taking the 3rd letter from each word in a harmless message to create a hidden message, etc), image substitution and microdot (taking data such as pictures and reducing it to the size of a large period on a piece of paper) were introduced and embraced as great steganographic techniques. In the recent digital world of today, namely 1992 to present, Steganography is

being used all over the world on computer systems. Many tools and technologies have been created that take advantage of old steganographic techniques such as null ciphers, coding in images, audio, video and microdot. With the research this topic is now getting we will see a lot of great applications for Steganography in the near future.

1.2 PRINCIPLES OF STEGANOGRAPHY:

Steganography involves hiding data in an overt message and doing it in such a way that it is difficult for an adversary to detect and difficult for an adversary to remove. Based on this goal, three core principles can be used to measure the effectiveness of a given steganography technique: amount of data, difficulty of detection, and difficulty of removal.

Amount of data suggests that the more data we can hide, the better the technique.

Difficulty of detection relates to how easy it is for somebody to detect that a message has been hidden. There is usually a direct relationship between how much data can be hidden and how easy it is for someone to detect it. As we increase the amount of information that is hidden in a file, we increase the chance that someone will be able to detect that there is information hidden in the file.

Difficulty of removal involves the principle that someone intercepting the file should not be able to remove the hidden data easily.

1.3 METHOD OF HIDING:

There are essentially three ways to hide data: injection, substitution, and generation.

- **Injection** finds areas in a file that will be ignored and puts the covert message in those areas. For example, most files contain an EOF or end-of-file marker. When playing an audio file, the application that is playing the file will stop playing when it reaches the EOF because it thinks it is the end of the file. We can inject data after the EOF marker that does not have an effect on the sound of the file.
- **Substitution** finds insignificant information in the host file and replaces it with the covert data. For example, with sound files each unit of sound we hear is composed of several bytes. If we modify the least significant bit it will slightly modify the sound, but so slightly that the human ear cannot tell the difference.

- **Encryption (optional):** Encryption of the message file or encryption of the master audio file and allowing user to secure their file and message.
- **Generation** creates a new overt file based on the information that is contained in the covert message. For example, one generation technique will take the covert file and produce a picture that resembles a modern painting.

1.4 ADVANTAGE OF STEGANOGRAPHY:

The advantage of steganography over cryptography alone is that messages do not attract attention to themselves. Plainly visible encrypted messages - no matter how unbreakable - will arouse suspicion, and may in themselves be incriminating in countries where encryption is illegal. Therefore, whereas cryptography protects the contents of a message, steganography can be said to protect both messages and communicating parties.

1.5 STEGANOGRAPHY AND CRYPTOGRAPHY:

Cryptography does encrypt the message in a form so that it becomes tough for an intruder or an unauthenticated/unintended agency or person to understand the message by decrypting it. But then in that case the existence of a message is crystal clear to everyone, only thing they have to do is to decrypt and if they succeed then they get the message.

But in Steganography we try to avoid the external attacks or better to say: minimize the external attack on the message. We send an innocuous file to the recipient embedding our message in it in such a way that the file doesn't attract unwanted attention or doubt of any kind of message existence. In this technique we try to embed the message in such a way that the file looks ordinary.

1.6 ENCODING SECRET MESSAGES IN AUDIO:

Encoding secret messages in audio is the most challenging technique to use when dealing with Steganography. This is because the human auditory system (HAS) has such a dynamic range that it can listen over. To put this in perspective, the (HAS) 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. The only weakness in the (HAS) comes at trying to differentiate sounds (loud sounds drown out quiet sounds) and

this is what must be exploited to encode secret messages in audio without being detected.

There are two concepts to consider before choosing an encoding technique for audio. They are the digital format of the audio and the transmission medium of the audio. There are three main digital audio formats typically in use. They are Sample Quantization, Temporal Sampling Rate and Perceptual Sampling. Sample Quantization which is a 16-bit linear sampling architecture used by popular audio formats such as (.WAV and. AIFF). Temporal Sampling Rate uses selectable frequencies (in the KHz) to sample the audio. Generally, 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 ISO MPEG (MP3). Transmission medium (path the audio take's from sender to receiver) must also be considered when encoding secret messages in audio. W. Bender introduces four possible transmission mediums:

- Digital end to end - from machine to machine without modification.
- Increased/decreased re-sampling - the sample rate is modified but remains digital.
- Analog and re-sampled - signal is changed to analog and re-sampled at a different rate.
- Over the air - signal is transmitted into radio frequencies and re-sampled from a microphone.

More popular encoding methods for hiding data inside of audio: They are low-bit encoding, phase-coding and spread spectrum.

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.

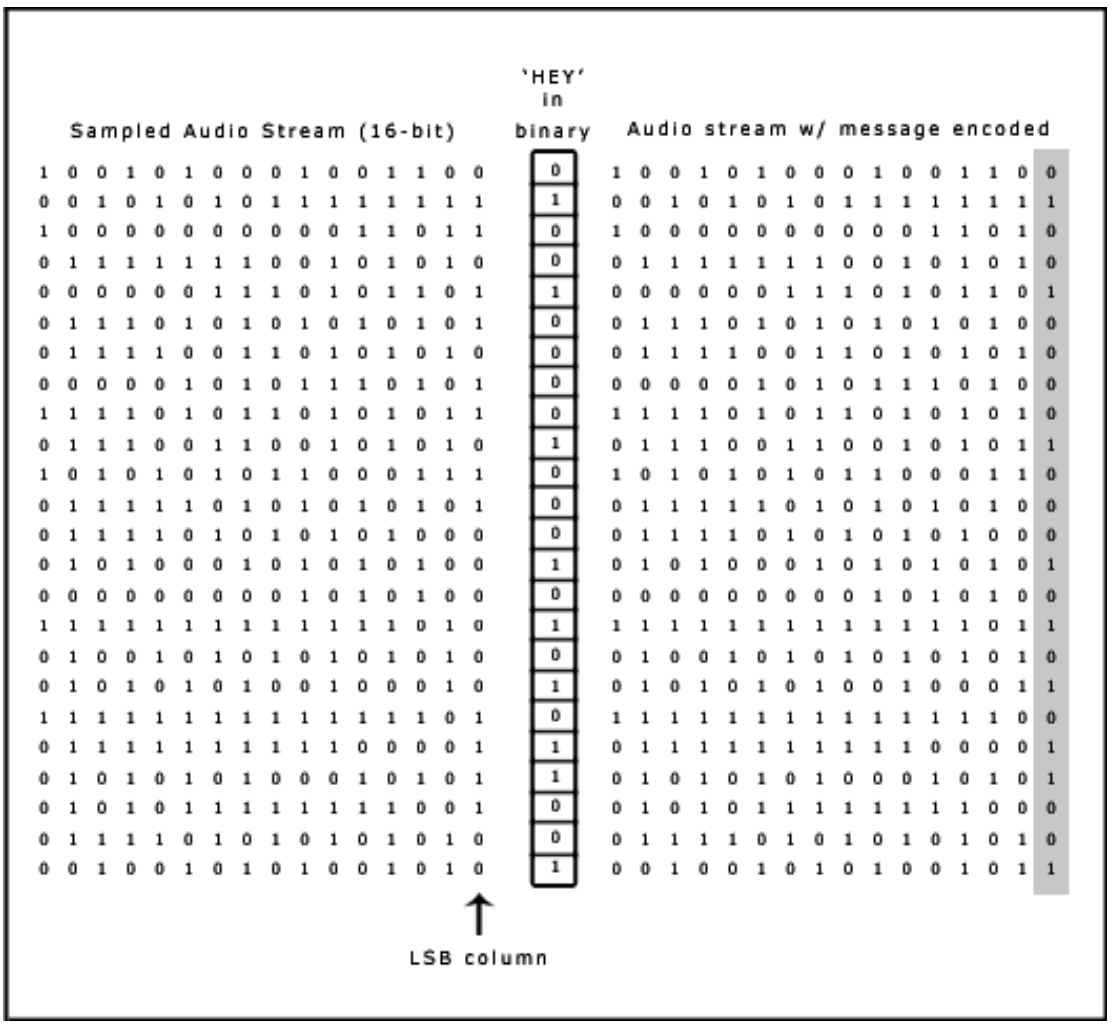
Spread spectrum encodes the audio over almost the entire frequency spectrum. It then transmits the audio over different frequencies which will vary depending on what spread spectrum method is used. Direct Sequence Spread Spectrum (DSSS) is one such method that spreads the signal by multiplying the source signal by some pseudo random sequence known as a (CHIP). The sampling rate is then used as the chip rate for the audio signal communication.

1.7 ADVANTAGES OF LEAST SIGNIFICANT BIT ALGORITHM OVER OTHER ALGORITHMS:

There are many coding techniques for audio Steganography like Parity Coding, Phase Coding and Echo hiding. But Least Significant Bit Algorithm has got certain valuable advantages over them like:

- There are two main disadvantages associated with the use of methods like **parity coding**. The human ear is very sensitive and can often detect even the slightest bit of noise introduced into a sound file, although the parity coding method does come much closer to making the introduced noise inaudible. Another problem is robustness.
- 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. Phase coding method is used when only a small amount of data needs to be considered.

1.8 LSB ALGORITHM DIAGRAMTIC REPRESENTATION:



1.9 POSSIBLE WAY OF ATTACKS:

As Fabien A.P. Petitcolas points out, there are six general protocols used to attack the use of Steganography.

- Stego only attack - only the stego object is available for analysis.
- Known cover attack - the original cover object and the stego object are available for analysis.
- Known message attack - the hidden message is available to compare with the stego- object.
- Chosen stego attack - the stego tool (algorithm) and stego-object are available for analysis.
- Chosen message attack - takes a chosen message and generates a stego object for future analysis.
- Known stego attack - the stego tool (algorithm), the cover message and the stego-objects are available for analysis.

1.10 AIM OF THIS PROJECT:

- Allow users to choose any audio file (WAV format) of any size.
- Allow users to choose the desired message file to be stored or sent.
- Allow users to choose multiple message files if the required message to be sent is in multiple files or there are multiple messages to be sent.
- Allow users to embed any kind of message into the carrier audio file. So the message can be anything i.e., an image, text, video, etc.
- Allow users to choose the option for applying password lock when hiding the messages.
- Allow users to secure the message file with the help of encryption.
- Ensure least possible distortion in quality of audio file after hiding the message file.
- Provide a user-friendly graphical user interface for the software.
- Handle errors, exceptions and inappropriate user actions effectively and display the messages with dialogue boxes to users.
- Allow users to rename the output audio file after embedding message into it.
- Allow user to choose the directory or folder in which the extracted messages (during de-staganography) have to put into.

CHAPTER 2: ALGORITHM

The basic algorithm used in this project is the 'Least Significant Bit' algorithm. To provide the necessary features and security, LSB algorithm is used in conjugation with a custom header and encryption tools. The software's, programming languages and libraries used in this project are as follows:

1. **Programming Languages: C, Python.**
   The C programming language has been used to make two CLI based application – 'hide.out' and 'unhide.out'. These applications deal with the audio files and do the necessary bit changes to hide and unhide message files in an audio file. All the inputs have to be provided to these applications via command line arguments.
   Python has been used to provide the Graphical User Interface and error handling.
2. **Python Libraries Used: PyQt4, os, sys, python-crypto.**
   PyQt is the python library which provides Qt support in python.
   Os and sys provide the library for system programming.
   Python-crypto provides the necessary library which has been used in this project for encryption.
3. **GUI Interface Designer Used: Qt-Designer.**
   Qt-Designer is a GUI based application which helps in designing the API in no time. It creates an xml file with extension .ui which can be used by various programming languages to create GUI based application.
4. **Utility Used: pyuic4.**
   This utility converts the .ui file generated by Qt-Designer to python class.

2.1 HEADER STRUCTURE:

1. Main Header (size 48 bits + variable):

Version (4 bits)	Header Length (24 bits)	Bit gap (8 bits)
Encrypt All (4 bits)	No. of message files (8 bits for 255 files)	
Password size (16 bits)	Password (password size bytes, variable)	

2. Optional Header(size 44 + variable)

File Size (32 bits)	Filename Size (8 bits)	Encryption (4 bits)
File Name (max 255 bytes , variable)		

Possible header size = 48 bits to 141845 bits

2.2 HEADER DESCRIPTION:

The main header stores the basic information needed to hide message files effectively in an audio file. The details of various variables used in the main Header are described as follows:

1. Version: It stores the version of header being used. This can be useful for future changes. The current version of the protocol is 1.
2. Header Length: It stores the total size of the header. The header length varies according to the no of message files, their file names and password.
3. Bit Gap: To provide least distortion in the audio file after hiding the message file, the 'Bit Gap' variable is used. It helps to skip few LSB's when storing consecutive bits of message file. The minimum value is 1 and maximum value is 255. This variable is only applicable for message files and not the header itself. The bit gap for storing the header itself is constant.
4. Encrypt All: Possible value – 0 to 15. When 0, the message file stored is not encrypted and when it is non zero, the file is encrypted. The possible 15 non-zero values can also help to define various encryption technologies being used to store the message file. This variable also determines whether password is being used or not.
5. No. of message files: This variable determines the no. of message files stored in the audio file. The current size of this field is 8 bits which gives us the flexibility to store at max 255 files in an audio file.

6. Password size: It stores the length of the password. However this variable is available for use only if the 'Encrypt All' variable is non-zero.
7. Password: It stores the password used in hiding the message files. This variable is also available for use only if the 'Encrypt All' variable is non-zero.

The optional header stores the information about various messages being stored in the audio file. This header is appended at the end of main header according to the number of message files being used. All the variables in this header are added per message file. The details of various variables used in the optional Header are described as follows:

1. File Size: This variable stores the size of message file in bytes. It plays significant role in distinguishing between message files.
2. Filename Size: This variable stores the length of message filename. It helps to store the filename of message file efficiently.
3. Encryption: Possible value – 0 to 15. When 0, the message file stored is not encrypted and when it is non zero, the file is encrypted. The possible 15 non-zero values can also help to define various encryption technologies being used to store the message file.
4. File name: This variable stores the filename of the message being stored. It helps to ensure that when the message is extracted from the audio file, it has the same filename.

2.3 ALGORITHM TO WRITE 'n' bit INTEGER IN LSB's OF AUDIO FILE WITH PROVIDED BIT GAP:

```
write_integer( int_variable, master_file, output_file,
               no_of_bits, bit_gap, cursor_position,
               cursor_loop_max)

1.   ms_mask ← 2 ( integer_size * 8 - 1 )
2.   cursor_loop_max ← cursor_loop_max  + number_of_bits *
     bit_gap
3.   for cursor_position ← cursor_position to cursor_loop_max
4.     do tmp_char ← get_character( master_file )
5.        ms_bit ← int_variable & ms_mask
6.        if ms_bit = 0
7.          then ms_bit = 0
8.        else
9.          ms_bit ← 1
10.       int_variable ← int_variable << 1
11.       tmp_char ← (tmp_char & (-2)) | ms_bit
12.       write_character( tmp_char, output_file)
     # Provide Bit gap if bit_gap > 1
```

```
13.       for i ← 1 to (bit_gap-1)
14.         do tmp_char ← get_character(master_file)
15.            write_character(tmp_char, output_file)
16.            cursor_position ← cursor_position + 1
17.  return 0
```

2.4 ALGORITHM TO WRITE A CHARACTER IN LSB's OF AUDIO FILE WITH PROVIDED BIT GAP:

```
write_character( char_variable, master_file, output_file,
                 bit_gap, cursor_position, cursor_loop_max)

1.   ms_mask ← 2 ( character_size * 8 - 1 )
2.   cursor_loop_max ← cursor_loop_max  + 8 * bit_gap
3.   for cursor_position ← cursor_position to cursor_loop_max
4.     do tmp_char ← get_character( master_file )
5.        ms_bit ← char_variable & ms_mask
6.        if ms_bit = 0
7.          then ms_bit = 0
8.        else
9.          ms_bit ← 1
10.       char_variable ← char_variable << 1
11.       tmp_char ← (tmp_char & (-2)) | ms_bit
12.       write_character( tmp_char, output_file)
     # Provide Bit gap if bit_gap > 1
13.       for i ← 1 to (bit_gap-1)
14.         do tmp_char ← get_character(master_file)
15.            write_character(tmp_char, output_file)
16.            cursor_position ← cursor_position + 1
17.  return 0
```

2.5 ALGORITHM TO EXTRACT 'n' bit INTEGER FROM AUDIO FILE WITH PROVIDED BIT GAP:

```
extract_integer( carrier_file, no_of_bits, bit_gap,
                 cursor_position, cursor_loop_max)

1.   cursor_loop_max ← cursor_loop_max  + number_of_bits *
     bit_gap
2.   integer_stored ← 0
3.   for cursor_position ← cursor_position to cursor_loop_max
4.     do tmp_char ← get_character( carrier_file )
5.        ls_bit ← tmp_char & 1
6.         integer_stored ← (integer_stored << 1) | ls_bit
      # skip the some space if bit_gap > 1
7.        for i ← 1 to (bit_gap-1)
8.          do get_character( carrier_file )
9.             cursor_position = cursor_position + 1
10.  return integer_stored
```

2.6 ALGORITHM TO EXTRACT A CHARACTER FROM AUDIO FILE WITH PROVIDED BIT GAP:

```
extract_character( carrier_file, bit_gap, cursor_position,
                   cursor_loop_max)

1.   cursor_loop_max ← cursor_loop_max  + 8 * bit_gap
2.   character_stored ← 0
3.   for cursor_position ← cursor_position to cursor_loop_max
4.     do tmp_char ← get_character( carrier_file )
5.        ls_bit ← tmp_char & 1
6.         character_stored ← (character_stored << 1) | ls_bit
      # skip the some space if bit_gap > 1
7.        for i ← 1 to (bit_gap-1)
8.          do get_character( carrier_file )
9.             cursor_position = cursor_position + 1
10.  return character_stored
```

The above four algorithms have been used extensively in our application to store and extract information in audio files. The program which hides the message in audio file inserts the message into the audio file in the following steps:

1. Skip first 50 bytes because this space contains the header information of wav audio format. So the first 50 bytes is copied as it is in the master file to output file.
2. Writes the custom header information described above using the write_integer and write_character functions in audio file. The bit gap provided here is 1.
3. Writes the message files one by one serially with maximum bit gap possible by using the same functions.
4. Copies the remaining bytes from master file to output file if message writing is complete.

Similarly the program which extracts the messages from audio file does it in the following steps:

1. Skip first 50 bytes because this space contains the header information of wav audio format.
2. Extracts the custom header information described above using the extract_integer and extract_character functions from audio file. The bit gap provided here is 1.
3. Extracts the message files one by one serially with the help of information from the header extracted. The same functions help to do this task as well.

CHAPTER 3: EXECUTION RESULTS AND DISCUSSION

3.1 SNAPSHOTS:

A. MESSAGE HIDING INTERFACE:

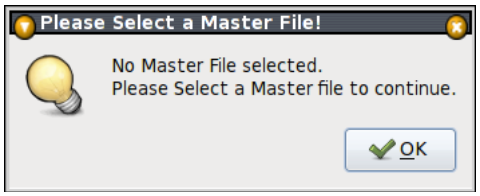
- User is allowed to browse the desired audio file in which the message is to be hidden with the help of "browse" button.
- The table view provided just below gives the user idea about the File size and location of the added file.
- User can add multiple message files with the help of "Add" button.
- User can remove the any file with the help of "Remove" button.
- User can enter password and reconfirm it and secure the message file.

B. MESSAGE UNHIDING INTERFACE:

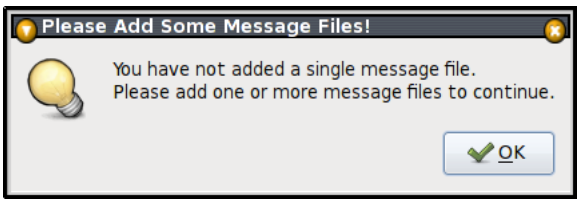
- User can input the desired audio file in which the message is hidden.
- And if the file is locked by a password then password field should be populated accordingly to unlock the message file.
- After clicking the UNHIDE button the embedded messages will be extracted and will be stored in the directory specified by the user.

C. ERROR HANDLING MESSAGES FOR HIDE FORM:

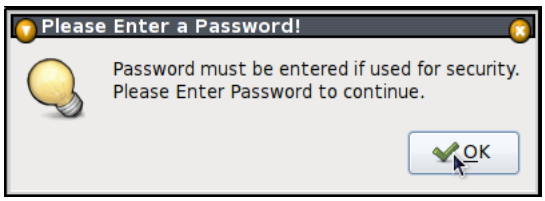
a. User clicks on HIDE button without selecting a master file:



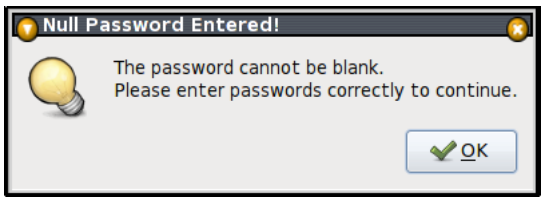
b. User clicks on HIDE button without adding any message files:



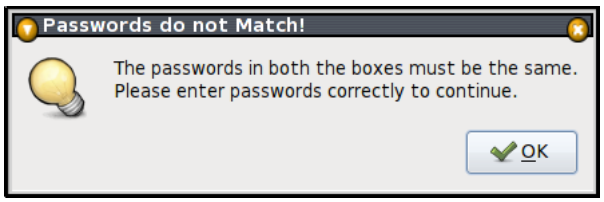
c. User forgets to type a password after checking the use password checkbox:



d. User enters null password:



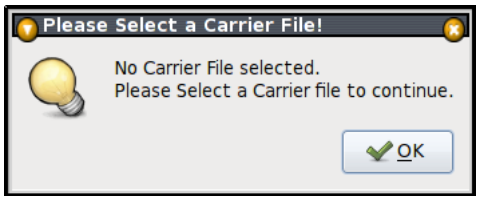
e. User confirm password does not match with the given password:



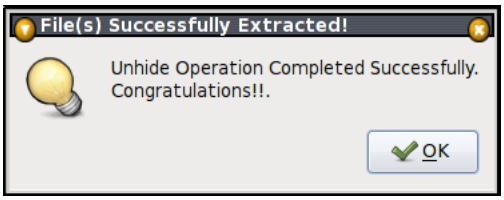
f. Message box if all goes well:

D. ERROR HANDLING MESSAGES FOR UNHIDE FORM:

a. User clicks on UNHIDE button without selecting a carrier file:



b. Message box if all goes well:



3.2 APPLICATIONS OF STEGANOGRAPHY IN OPEN SYSTEMS ENVIRONMENT:

Covert channels in TCP/IP involve masking identification information in the TCP/IP headers to hide the true identity of one or more systems. This can be very useful for any secure communications needs over open systems such as the Internet when absolute secrecy is needed for an entire communication process and not just one document as mentioned next. Using containers (cover messages) to embed secret messages into is by far the most popular use of Steganography today. This method of steganography is very useful when a party must send a top secret, private or highly sensitive document over an open systems environment such as the Internet. By embedding the hidden data into the cover message and sending it, we can gain a sense of security by the fact that no one knows we have sent more than a harmless message other than the intended recipients. Although not a pure steganographic technique, digital watermarking is very common in today's world and does use Steganographic techniques to embed information into documents.

Digital watermarking is usually used for copy write reasons by companies or entities that wish to protect their property by either embedding their trademark into their property or by concealing serial numbers/license information in software, etc. Digital watermarking is very important in the detection and prosecution of software pirates/digital thieves.

CHAPTER 4: LIMITATIONS AND FUTURE SCOPE

FUTURE DEVELOPMENTS IN THIS PROJECT CAN BE DONE IN THE FOLLOWING AREAS:

1. Adding support for Windows and MAC OS.
2. Adding support for other popular audio formats.
3. Adding a web based interface.
4. Adding themes in GUI.
5. Adding multiple encryption technology support.

The application is made using standard libraries which are available in all popular operating systems. So, the application can be easily ported in any OS with little effort. The above software is designed to work on raw audio formats. Work can be done to add support for compressed audio formats which are more popular like mp3 format, RAW, VOX format, etc.

The interface is made by using a very dynamic and high level programming language – python. Work can be done to add a web based interface and design a website for the same. The theme support for the software is already there, because the widgets in python are also object variables and can be changed when and where needed. So various themes can be designed and option to switch between them can be easily added in the software. The header format used in the software uses 4 bit encryption variable. So, an application can store 15 non-zero values in it which can be useful to map each value with a different encryption technique. Since a stronger encryption technique significantly increases the file size of message file, this will be useful to have a tradeoff between size of message being stored and security.

REFERENCES

BOOKS:

- Mastering C by K R Venugopal and S R Prasad
- The C Programming Language by Brian W. Kernigan & Dennis M. Ritchie

WEBSITES:

- www.docs.python.org
- www.doc.qt.nokia.com
- www.riverbankcomputing.co.uk/static/Docs/PyQt4/html/
- www.google.com
- www.wikipedia.org

IRC:

- Internet Relay Chatting (irc.freenode.net)