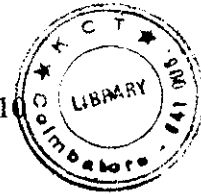


DATA HIDING IN AUDIO FILES

By
R.Dharani
Registration Number: 7120562101



Of
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE

A PROJECT REPORT
Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION
ENGINEERING

*In partial fulfillment of the requirements
For the award of the degree*

Of

MASTER OF COMPUTER APPLICATION

ANNA UNIVERSITY
CHENNAI 600 025

July 2008

BONAFIDE CERTIFICATE

Certified that this project report titled Data Hiding in Audio Files is the bonafide work of Ms. R.Dharani. (Registration Number: 71205621010) who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.




Supervisor

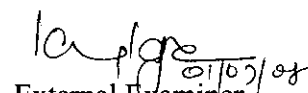


Head of the Department

Submitted to Project and Viva Examination held on 1.7.2008



Internal Examiner



External Examiner



BLUE CHIP TECHNOLOGIES
Joining the Poles apart

CORPORATE OFFICE:

NO. 78, 3RD FLOOR, USMAN ROAD, T.NAGAR,
CHENNAI - 600 017.
PH: 044-6536 0606
E-MAIL: ENQUIRY@BCPROJECTS.COM
WEB: WWW.BCPROJECTS.COM

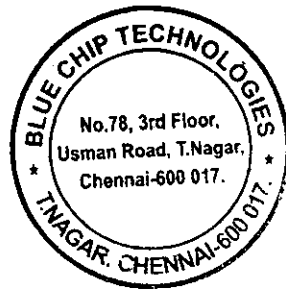
June 2nd, 2008

To Whomsoever It May Concern

This is to certify that Miss.DHARANI.R (71205621010), Student of Kumaraguru College of Technology, Coimbatore doing her Final Year, M.C.A (Computer Application) has successfully completed her project entitled as "DATA HIDING IN AUDIO FILES" under the guidance of Mr.Raghuram.J, Senior Software Engineer, from 04th Jan 2008 to 2nd June 2008.

During her Project duration her conduct and contribution has been excellent.

We wish her all the best for her future Endeavors.



For Blue Chip Technologies

Raghuram J.
(Project Co-ordinator)

ABSTRACT

Our project, **Data Hiding in Audio files** is the software developed for hiding information which uses the technology called as Steganography –derived from the Greek words meaning, “Covered writing”, is the art of hiding information in ways that prevent its detection.

It is a method akin to covert channels, and invisible links, which add another step in security. A message in cipher text may arouse suspicion while an invisible message is not.

Digital stenography uses a host data or message known as a “Container” or “Cover” to hide another data or message in it. The conventional way of protecting information was to use a standard symmetric or asymmetric key system in encryption. Steganography can also be used to place a hidden “trademark” in images, music, and software, a technique referred to as watermarking.

Steganography, if however used along with cryptography ,for example, if a message is encrypted using triple DES which requires a 112 bit key then the message has become quite secure as far as cryptanalytic attack are concerned. Now, if this cipher text is embedded in an image, video, voice, etc., it is even more secure. If an encrypted message is intercepted, the interceptor knows the text is an encrypted message. With Steganography, the interceptor may know the object contains a message.

In order to ensure the privacy of the communication between two parties, here we implement a technique for data hiding in audio images, known as Audio file Steganography. We will also be incorporating encryption of data to be hidden.

ACKNOWLEDGEMENT

I wish to express my sincere thanks to **Dr.JOSEPH V.THANIKAL Ph.D.**, Principal, Kumaraguru College of Technology,Coimbatore,for giving me the consent to undertake this project.

My deepest acknowledgement to **Dr.M.GURURAJAN Ph.D.**, Head of the Department, Computer Applications, Kumaraguru College of Technology, Coimbatore,for his timely help and guidance throughout this project.

I am greatly indebted to **Mr.A.MUTHUKUMAR MCA., MPhil.**, Project Coordinator, my Guide, Assistant Professor, Department of computer Applications and my sincere thanks to him for guiding and encouraging me at every stage of this project.

I express my sincere thanks to **MR.RAGHURAM.J**, Senior Project Leader, BLUE CHIP TECHNOLOGIES, Chennai for his support and assistance at various levels of my project work.

Finally, I owe my great deal of gratitude to my parents for helping me to overwhelm in all my proceedings. I bow my heart and head with heartfelt thanks to my department staffs and all those who taught me their warm service to succeed and achieve my work.

TABLE OF CONTENTS

| CHAPTER NO | TITLE | PAGE NO |
|-------------------|---------------------------------|----------------|
| | ABSTRACT | iv |
| | LIST OF FIGURES | ix |
| | LIST OF ABBREVIATIONS | x |
| | CHAPTERS | |
| 1 | INTRODUCTION | |
| | 1.1 Organization Profile | 1 |
| | 1.2 Abstract | 2 |
| | 1.3 Problem Definition | 3 |
| 2 | SYSTEM ANALYSIS | |
| | 2.1 Existing System | 4 |
| | 2.2 Proposed System | 4 |
| | 2.3 User Interface Requirements | 5 |
| 3 | DEVELOPMENT ENVIRONMENT | |
| | 3.1 H/W Environment | 6 |
| | 3.2 S/W Environment | 6 |
| | 3.2.1 Java Language | 6 |

| | | |
|----------|--|-----------|
| | 3.2.2 Java Platform | 8 |
| | 3.2.3 Java Swing | 9 |
| | 3.2.3.1 Javax.swing Package | 9 |
| | 3.2.4 ApplicationProgrammingInterfaces | 9 |
| | 3.2.5 Basic Foundation Classes | 13 |
| 4 | SYSTEM DESIGN | 14 |
| | 4.1 Context Diagram | 14 |
| | 4.2 Data Flow Diagram | 15 |
| 5 | SYSTEM IMPLEMENTATION | 16 |
| | 5.1 General Steganography | 16 |
| | 5.2 Steganography in Audio | 17 |
| | 5.3 Methods Of Encoding Message in Audio | 17 |
| | 5.3.1 Low bit Encoding | 17 |
| | 5.3.2 Phase Encoding | 19 |
| | 5.4 Audio File Format | 20 |
| | 5.5 Encryption Method Used | 22 |
| | 5.6 MD5 Algorithm | 30 |
| | 5.7 DES Algorithm | 31 |
| 6 | TESTING | 33 |
| | 6.1 Testing Objective | 33 |
| | 6.2 Types of Testing | 33 |

| | | |
|----------|------------------------------------|-----------|
| | 6.3 Different Levels of Testing | 34 |
| | 6.3.1 Unit Testing | 34 |
| | 6.3.2 Integration Testing | 34 |
| | 6.3.3 System Testing | 34 |
| | 6.3.4 Acceptance Testing | 34 |
| 7 | PERFORMANCE AND LIMITATIONS | 35 |
| | 7.1 Merits of the system | 35 |
| | 7.2 Limitations of the System | 34 |
| | 7.3 Future Enhancements | 36 |
| 8 | APPENDICES | 37 |
| | 8.1 Sample Screens | 37 |
| 9 | BIBLIOGRAPHY | 53 |

LIST OF FIGURES

| FIGURE NO | NAME | PAGE NO |
|------------------|---|--------------------|
| 3.2.1.1 | Execution of java program | 7 |
| 3.2.1.2 | Java Platform | 8 |
| 3.2.2.1 | Insulation of the Java program from hardware dependencies | 9 |
| 4.1.1 | Context Diagram | 14 |
| 4.2.1 | Data Flow Diagram | 15 |
| 5.3.1.1 | Encoding char 'A' in a stream of Audio Samples | 18 |
| 5.5.1 | Key Derivation Function | 27 |
| 5.5.2 | Encoding and decoding modules | 27 |
| 5.6 | MD5 Algorithm | 30 |
| 5.7 | Data Encryption Standard Algorithm | 31 |

LIST OF ABBREVIATIONS

| | | |
|---------------|---|--|
| DES | - | Data Encryption Standard |
| JVM | - | Java Virtual Machine |
| API | - | Application Programming Interface |
| AWT | - | Applet Windows Toolkit |
| HAS | - | Human Auditory System |
| DFT | - | Discrete Fourier Transform |
| PCM | - | Pulse Code Modulation |
| PBE | - | Password Based Encryption |
| MD5 | - | Message Digest Algorithm |
| KDF | - | Key Derivation Function |
| PBKDF1 | - | Password Based Key Derivation Function 1 |
| PBKDF2 | - | Password Based Key Derivation Function 2 |
| DK | - | Derived Key |
| EDE | - | Encryption Decryption Encryption |

CHAPTER-1

INTRODUCTION

1.1 ORGANIZATION PROFILE:

Blue chip technologies' is a leading monopoly technical organization in Chennai exclusively for project training, working with the sole mission of bringing the corporate and education worlds together. BCT is promoted by Iwiz Technologies (p) Ltd, which has been emerged as a reputed software consultant in last decade. So far Iwiz technologies provide software solutions for wide range of industries from retail to top-notch technologies. Moreover we are one of the major technology and management-training providers with more than 3,00,000 trained candidates in various streams through our technology-training arm, T-Jun. Furthermore, through our Iwiz Jobs Consultant, we resource lakhs of candidates to wide range of industries.

Blue chip facilitate real time working environment with cutting edge tools and eminent trainers. It offers Cost effective intelligent IT solutions to clients in the areas of Financial Services, Communication, Retail, Manufacturing, High-Technology, Travel and Transport Public sector industries. They offer IT services in the areas of Application Development, Application Management, Enterprise Business Solution, and Software Testing through a global delivery model that provides good quality.

The main Objective is to provide a global delivery model that ensures security, cost effectiveness and Quality for clients. To create real time working environment with cutting edge tools .To minimize the industry-institute gap. To make aware of different kinds of Software Life Cycle models like SDLC, Water fall, Spiral, User experience etc. Our research projects had participated in various National and International Conferences. Most of our projects were identified by the industries as suitable for their needs.

1.2 ABSTRACT

Our project, **Data Hiding in Audio files** is the software developed for hiding information which uses the technology called as Steganography –derived from the Greek words meaning, “Covered writing”, is the art of hiding information in ways that prevent its detection. It is a method akin to covert channels, and invisible links, which add another step in security. A message in cipher text may arouse suspicion while an invisible message is not.

Digital stenography uses a host data or message known as a “Container” or “Cover” to hide another data or message in it. The conventional way of protecting information was to use a standard symmetric or asymmetric key system in encryption. Steganography can also be used to place a hidden “trademark” in images, music, and software, a technique referred to as watermarking.

Steganography, if however used along with cryptography ,for example, if a message is encrypted using triple DES which requires a 112 bit key then the message has become quite secure as far as cryptanalytic attack are concerned. Now, if this cipher text is embedded in an image, video, voice, etc., it is even more secure. If an encrypted message is intercepted, the interceptor knows the text is an encrypted message. With Steganography, the interceptor may know the object contains a message. When performing data hiding on audio, one must exploit the weakness of the Human Auditory System (HAS), while at the same time being aware of the extreme sensitivity of the human auditory system.

In order to ensure the privacy of the communication between two parties, here we implement a technique for data hiding in audio images, known as *Audio file Steganography*. However, Steganography alone is not able to provide a sufficiently high enough level of security. In order to improve the security of our technique, we will also be incorporating encryption of data to be hidden.

1.3 PROBLEM DEFINITION

We are of the belief that the easiest way to keep something from prying eyes is to place it right in front of the person looking for it and make it look as innocuous as possible. Everyone has a taste for a certain kind of music. Hence, it is more than likely that the person will have that kind of music on the storage device of his computer. Also, it is quite common case where people share and transfer different music files to one another. If one were able to hide the message can be. Also, transfer of this message can be done quite conveniently without raising any eyebrows.

Our aim is to come up with a technique of hiding the message in the audio file in such a way, that there would be no perceivable changes in the audio file after the message insertion. At the same time, if the message that is to be hidden were encrypted, the level of security would be raised to quite a satisfactory level. Now, even if the hidden message were to be discovered the person trying to get the message would only be able to lay his hands on the encrypted message with no way of being able to decrypt it. When performing data hiding on audio, one must exploit the weakness of the Human Auditory System (HAS), while at the same time being aware of the extreme sensitivity of the human auditory system.

Communication holds the key to business, personal life, etc. As people tend to rely on these new means of communication, more and more important information is being conveyed along these new lines.

CHAPTER 2

SYSTEM ANALYSIS

2.1 Existing System

Nowadays, several methods are used for communicating secret messages for defense purposes or in order to ensure the privacy of communication between two parties. So we go for hiding information in ways that prevent its detection. Some of methods used for privacy communication are the use of

- invisible inks
- covert channels
- digital signatures

The above are some of existing systems that are used to convey the messages.

2.2 PROPOSED SYSTEM

The proposed system uses Audio file as a carrier medium which add another step in security. The objective of the newly proposed system is to create a system that makes it very difficult for an opponent to detect the existence of a secret message by encoding it in the carrier medium as a function of some secret key and that remains as the advantage of this system.

In order to ensure the privacy of the communication between two parties, various new methods are being developed. Cryptography being the mother to all those projects. However, cryptography is like a tool, it can do as well as it is programmed to do.

Also, there are various different techniques that can be implemented to attain a certain level of security. This System can be used to:

- Covert communication of sensitive data

- Facilitate 'theft of data'
- Embedding hidden 'trademarks' in images, music, etc.,

Common Elements that are used in this system are

- Carrier medium : Audio file
- Embedded message : Text file
- Key : Password

2.3 USER INTERFACE REQUIREMENTS

- 1 Module I (Media Reading & Analyzing) :
 - Java language
- 2 Module II (Message encryption and embedding into audio)
 - Java language
- 3 Module III (Extracting of Message from Audio and encryption)
 - Java language

CHAPTER 3

DEVELOPMENT ENVIRONMENT

3.1 HARDWARE ENVIRONMENT

| | |
|----------|---|
| Monitors | : 800x600 minimum resolutions at 256 colors minimum |
| Memory | : Approximately 64 MB of on board memory |
| I/O | : two or three button mouse and standard 101-key keyboard |
| MHZ | : At least 166 MHZ processor |

3.2 SOFTWARE ENVIRONMENT

3.2.1 THE JAVA LANGUAGE

What Is Java?

Java is two things: a programming language and a platform.

The Java Programming Language

Java is a high-level programming language that is all of the following:

- 1 Simple
- 2 Object-oriented
- 3 Distributed
- 4 Interpreted
- 5 Robust
- 6 Secure
- 7 Architecture-neutral
- 8 Portable
- 9 High-performance

10 Multithreaded

11 Dynamic

Java is also unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called Java byte codes—the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java byte code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how this works.

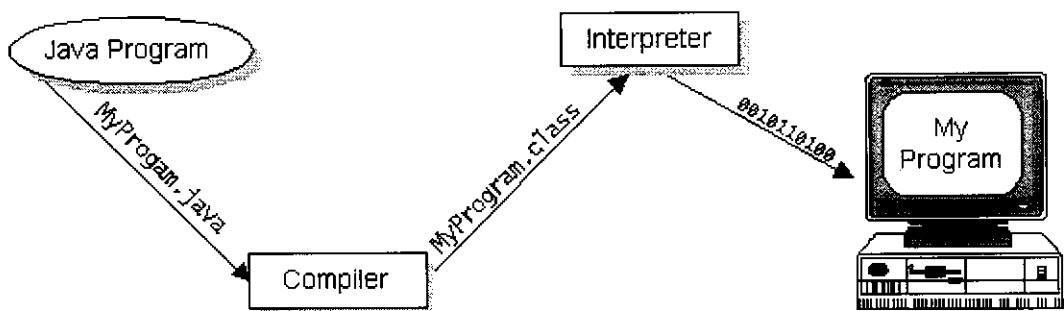


Fig 3.2.1.1

Java byte codes can be considered as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java byte codes help make "write once, run anywhere" possible. The Java program can be compiled into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh.

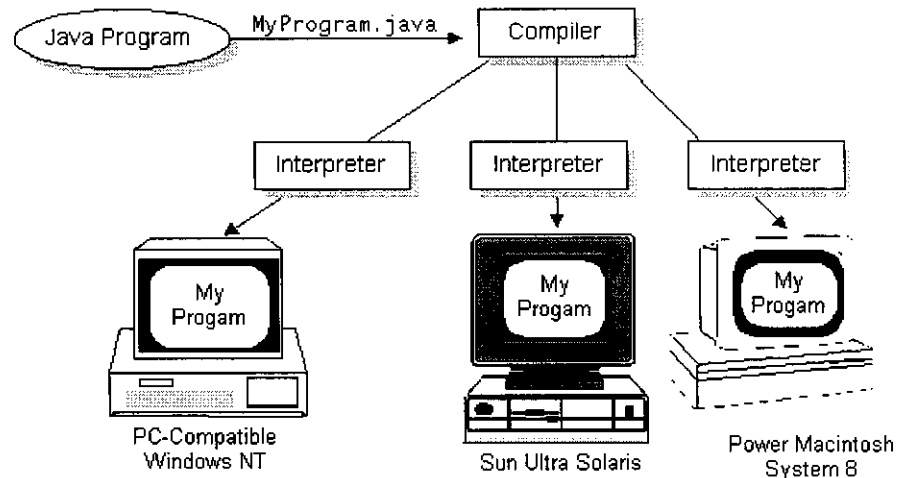


Fig 3.2.1.2

3.2.2 THE JAVA PLATFORM

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components:

- The Java Virtual Machine (Java VM)
- The Java Application Programming Interface (Java API)

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (*packages*) of related components.

The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

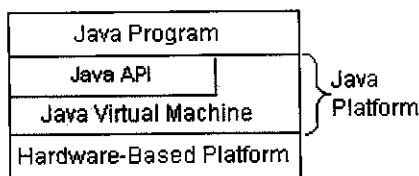


Fig 3.2.2.1

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring Java's performance close to that of native code without threatening portability.

3.2.3 JAVA SWING

The swing is a set of classes that provides more powerful and flexible components than are possible with the AWT. Unlike AWT components Swing components are not implemented by platform specific code. They are written entirely in java and, therefore, are platform-independent. The term lightweight is used to describe such elements. The number of classes and interfaces in the swing packages is substantial. Swing is area that you will want to explore further on your own

3.2.3.1 JAVAX.SWING PACKAGE

The Swing API has been included in the Java 2 platform. JFC is short for Java Foundation Classes, which encompass a group of features for building graphical user interfaces GUI. Use Swing components in building a user interface. First we examine the simplest Swing application you can write. Then we present several progressively complicated examples of creating user interfaces using components in the javax.swing package. We cover several Swing components, such as buttons, labels, and text areas.

| | |
|---------------------------------|--|
| javax.swing | Provides a set of "lightweight" (all-Java language) components that, to the maximum degree possible, work the same on all platforms. |
| javax.swing.border | Provides classes and interface for drawing specialized borders around a Swing component. |
| javax.swing.colorchooser | Contains classes and interfaces used by the JColorChooser component. |
| javax.swing.event | Provides for events fired by Swing components. |
| javax.swing.filechooser | Contains classes and interfaces used by the JFileChooser component. |

3.2.4 APPLICATION PROGRAMMING INTERFACES

Listeners are created by implementing one or more of the interfaces defined by the **java.awt.event** package. When an event occurs, the event source invokes the appropriate method defined by the listener.

Action Listener interface

This interface defines the `actionPerformed ()` method that is Invoked when an action event occurs. Its general form is shown

```
Void actionPerformed (ActionEvent ae)
```

The mouset listener interface

This interface defines five methods. If the mouse is pressed and released at the same point, `mouseClicked()` is invoked. When the mouse enters a component, the `mouseEntered()` method is called. When it leaves, `mouseExited()` is called. The `mousePressed` and `mouseReleased()` methods are invoked when the mouse is pressed and released, respectively.

The general forms of these methods are shown here:

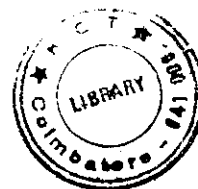
Void mouseClicked(MouseEvent me)

Void mouseEntered(MouseEvent me)

Void mouseExited(MouseEvent me)

Void mousePressed(MouseEvent me)

Void mouseReleased(MouseEvent me)



9-2259

The MouseMotionListener interface

This interface defines two methods. The mouseDragged() method is called multiple times as the mouse is dragged. The mouseMoved() method multiple times as the mouse is moved. Their general forms are shown here:

Void mouseDragged(MouseEvent me)

Void mouseMoved(MouseEvent me)

The TextListener interface

This interface defines the textChanged() method that is invoked when a change occurs in a text area or text field. Its general form is shown here:

Void textChanged(TextEvent te)

The WindowListener interface

This interface defines seven methods. The windowActivated() and windowDeactivated() methods are invoked when a window is activated or deactivated, respectively. If a window is iconified, the windowIconified() method is called. When a window is deiconified, the windowDeiconified() method is called. When a window is opened

or closed, the `windowOpened()` or `windowClosed()` methods are called, respectively. The `windowClosing()` method is called when a window is being closed. The general forms of these methods are

```
Void windowActivated(WindowEvent we)
Void windowClosed(WindowEvent we)
Void windowClosing(WindowEvent we)
Void windowDeactivated(WindowEvent we)
Void windowDeiconified(WindowEvent we)
Void windowIconified(WindowEvent we)
Void windowOpened(WindowEvent we)
```

Using the delegation event model

Now that you have learned the theory behind the delegation event model and have had an over view of its various components, it is time to see it in practice. applet programming using the delegation event model is actually quite easy. Just follow these two steps:

1. Implement the appropriate interface in the listener so that it will receive the type of event desired.
2. Implement code to register and unregister (if necessary) the listener as a recipient for the event notifications.

Remember that a source may generate several types of events. each event must be registered separately. also, an object may register to receive several types of events, but it must implement all of the interfaces that are required to receive these events.

To see how the delegation model works in practice, we will look at examples that handle the two most commonly used event generators: the mouse and keyboard.

3.2.5 BASIC FOUNDATION CLASSES

APPLET

Applet provides all necessary support for execution, such as starting and stopping. It also provides methods that load and display images and methods that load and play audio clips. Applet extends the AWT class panel. In turn panel extends container, which extends component. These classes provide support for java's window-based, graphical interface. Thus applet provides all of the necessary support for window-based activities.

IMAGE

This class provides support for imaging. Images are objects of the Image class, which is a part of the java.awt package. There are a large number of imaging classes and interfaces defined by java.awt.image and it is not possible to examine them all.

EVENT

The classes that represent events are at the core of java's event handling mechanisms. They provide a consistent, easy-to-use means of encapsulating events. At the root of the java event class hierarchy is EventObject, which is in java.util. It is the super class for all events. Its one constructor is shown

```
EventObject(Object src)
```

CHAPTER 4

SYSTEM DESIGN

4.1 CONTEXT DIAGRAM

A System Context Diagram (SCD) is the highest level view of a system, similar to Block Diagram, showing a (normally software-based) system as a whole and its inputs and outputs from/to external factors. Context Diagrams show the interactions between a system and other actors with which the system is designed to face. SCD is very helpful in understanding the context in which the system.

Here the password is entered and validation is done. The message is then encrypted, and the encrypted data is embedded using the techniques in steganography.

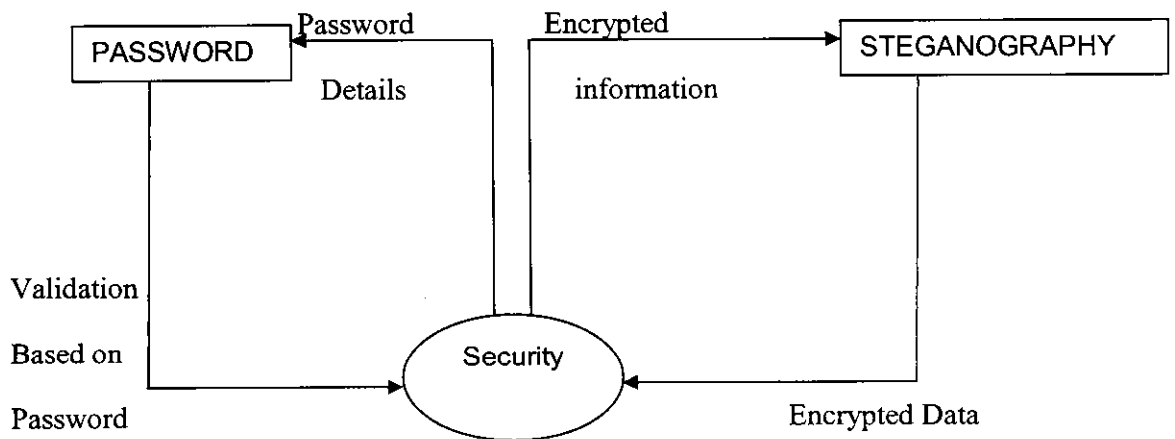


Fig 4.1.1

4.2 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. A data flow diagram can also be used for the visualization of data processing. There are 4 key elements in a Data Flow diagram, Processes, Data Flows, Data Stores & External entities.

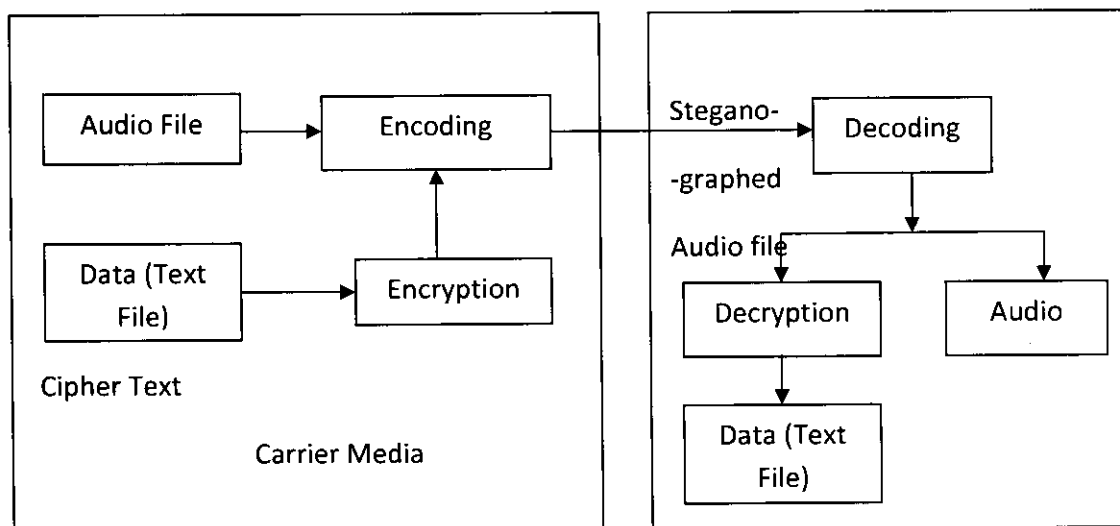


Fig 4.2.1

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 GENERAL STEGANOGRAPHY

There are several data hiding techniques available today. In each technique, the host data type is fixed, but the embedded data type can be varied as per requirement. Data hiding technique should be capable of embedding data in a host signal with the following restrictions and features:

- 1 The host signal should be nonobjectionally degraded and the embedded data should be minimally perceptible. What that means is that the observer should not be able to notice the presence of the data even if it were perceptible.
- 2 The embedded data should be directly encoded into the media rather than into a header or a wrapper so that the data remain intact across varying data file formats.
- 3 The embedded data should be immune to modifications ranging from intentional and intelligent attempts at removal to anticipated manipulations e.g. channel noise, re-sampling, cropping, etc.
- 4 Asymmetrical coding of the embedded data is desirable since the purpose of data hiding is to keep the data in the host signal but not necessarily to make the data difficult to access.
- 5 The embedded data should be self clocking or arbitrarily re-entrant. This ensures that the embedded data can be recovered even when only fragments of information are available.

5.2 STEGANOGRAPHY IN AUDIO

Data hiding in audio signals is especially challenging, because the Human Auditory System (HAS) operates over a wide dynamic range. The HAS perceives over a range of power greater than one billion to one and a range of frequencies greater than thousand to one. Sensitivity to additive random noise is also acute.

The perturbations in a sound file can be detected as low as one part in ten million which is 80dB below ambient level. However there are some 'holes' available. While the HAS has a large dynamic range, it has a fairly small differential range. As a result, loud sounds tend to mask out the quieter sounds.

Additionally, the HAS is unable to perceive absolute phase, only relative phase. Finally there are some environmental distortions so common as to be ignored by the listener in most cases.

We have tried to exploit these traits to our advantage in the methods discussed further while being careful to bear in mind the extreme sensitivities of the HAS.

5.3 METHODS OF ENCODING A MESSAGE IN AUDIO

5.3.1 LOW-BIT ENCODING

Low-bit encoding is the one of the simplest way to embed data into other data structures. By replacing the least significant bit of each sampling point by a coded binary string, we can encode a large amount of data in an audio signal.

Ideally, the channel capacity is 1 kb per second (kbps) per 1 kilohertz(kHz), e.g., in a noiseless channel, the bit rate will be 8 kbps in an 8 kHz sampled sequence and 44 kbps in a

44kHz sampled sequence. In return for this large channel capacity, audible noise is introduced. The impact of this noise is a direct function of the content of the host signal, e.g., crowd noise during a live sports event would mask low-bit encoding noise that would be audible in a string quartet performance.

Adaptive data attenuation has been used to compensate this variation. The major advantage of this method is its poor immunity to manipulation. Encoded information can be destroyed by channel noise, re-sampling, etc., unless it is encoded using redundancy techniques.

In order to be robust, these techniques reduce the data rate which could result in the requirement of a host of higher magnitude, often by one to two orders of magnitude. In practice, this method is useful only in closed, digital-to-digital environments.

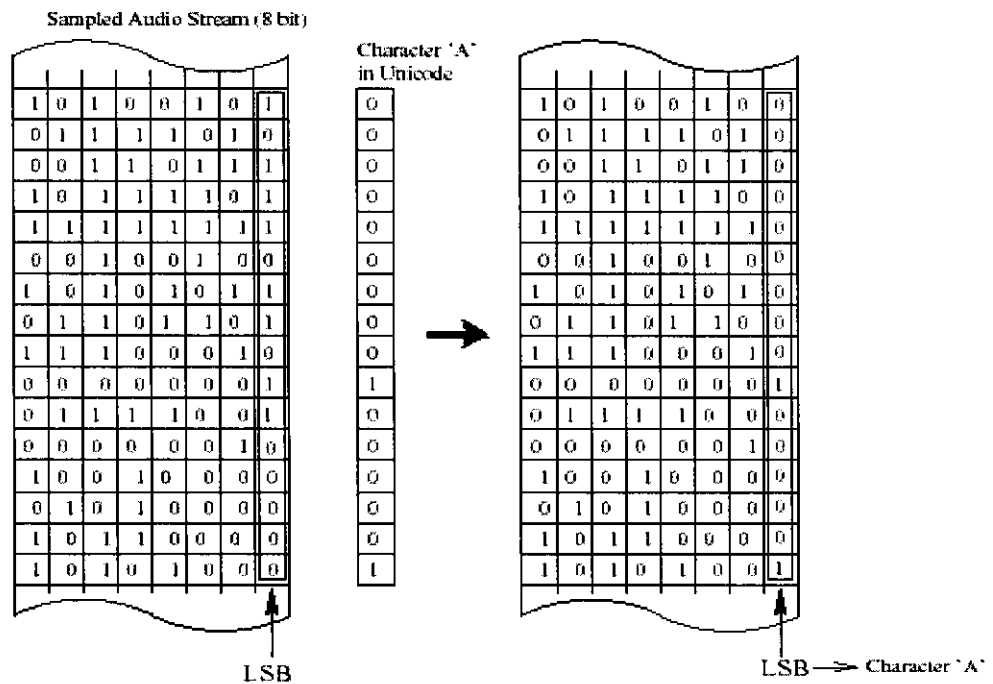


Fig 5.3.1.1

5.3.2 PHASE ENCODING

The phase coding method works by substituting the phase of an initial audio segment with a reference phase that represents the data. The phase of subsequent segments is adjusted in order to preserve the relative phase between segments. Phase coding, is one of the most effective coding methods in terms of the signal to perceived noise ratio. When the phase relation between each frequency component is dramatically changed, a noticeable phase dispersion will occur. However, as long as the modification of the phase is sufficiently small an inaudible coding can be achieved. The phase coding method works by substituting the phase of an initial audio segment with a reference phase that represents the data. The procedure for phase coding is as follows:

- 1 The original sound sequence is broken into a series of N short segments.
- 2 A Discrete Fourier Transform is applied to each segment, to break create a matrix of the phase magnitude.
- 3 The phase difference between each adjacent segment is calculated.
- 4 For segment S_0 , the first segment, an artificial absolute phase p_0 is created.
- 5 For all other segments, new phase frames are created.
- 6 The new phase and original magnitude are combined to get a new segment S_n .
- 7 Finally, the new segments are concatenated to create the encoded output.

For the decoding process, the synchronization of the sequence is done before the decoding. The length of the segment, the DFT points, and the data interval must be known at the receiver. The value of the underlying phase of the first segment is detected as 0 or 1, which represents the coded binary string.

This is interesting from our point of view for several reasons. First, a watermark that is embedded in the phase of the DFT would be quite robust to tampering. The core information

contained in watermarks is almost always encoded with a high degree of redundancy. Therefore background clutter and phase distortions deliberately introduced by an ‘enemy’ to impede transmission of the watermark would have to be noticeably large in order to be successful. This would cause unacceptable damage to the quality of the image. Second, from communications theory, it is well known that angle modulation possesses superior noise immunity when compared to amplitude modulation. We also find that phase is relatively robust to changes in image contrast.

5.4 AUDIO FILE FORMAT

In order to demonstrate the use of Steganography techniques combined with encryption, the AU format used on Sun and NeXT machines was chosen as the host audio file. Sun AU format is well documented elsewhere and Java Sound API provides convenient ways to handle formatted audio data.

Formatted audio data refers to sound in any of a number of standard formats. The Java Sound API distinguishes between data formats and file formats. A data format tells you how to interpret a series of bytes of “raw” sampled audio data, such as samples that have been captured from the microphone input. You might need to know, for example how many bits constitute one sample (the representation of the shortest instant of sound), and similarly you might need to know the sound’s sample rate (how fast the samples are supposed to follow one another). When setting up for playback or capture, you specify the data format of the sound you are capturing or playing.

In the Java Sound API, a data format is represented by an `AudioFormat` object, which includes the following attributes:

- 1 Encoding technique, usually pulse code modulation (PCM)
- 2 Number of channels (1 for mono, 2 for stereo, etc.)
- 3 Sample rate (number of samples per second, per channel)

- 4 Number of bits per sample (per channel)
- 5 Frame rate
- 6 Frame size in bytes
- 7 Byte order (big-endian or little-endian)

PCM is one kind of encoding of the sound waveform. Compact disks, for example, use linear PCM-encoded sound. Mu-law encoding and a-law encoding are common nonlinear encodings that provide a more compressed version of the audio data.

A frame contains the data for all channels at a particular time. For PCM-encoded data, the frame is simply the set of simultaneous samples in all channels, for a given instant in time, without any additional information. In this case, the frame rate is equal to the sample rate, and the frame size in bytes is the number of channels multiplied by the sample size in bits, divided by the no of bits in a byte.

A file format specifies the structure of a sound file, including not only the format of the raw audio data in the file, but also other information that can be stored in the file. Sound files come in various standard varieties, such as WAVE (also known as WAV, and often associated with PCs), AIFF (often associated with Macintosh OS), and AU (often associated with UNIX systems). The different types of sound files have different structures. For example, they might have a different arrangement of data in the file's 'header'. A header contains descriptive information that typically precedes the file's actual audio samples, although some file formats allow successive "chunks" of descriptive and audio data. The header includes a specification of the data format that was used for storing the audio in the second file. Any of these types of sound file can contain various data formats (although usually there is only one data format within a given file), and the same data format can be used in files that have different file formats.

In the Java Sound API, a file format is represented by an `AudioFileFormat` object, which contains:

- 1 The file type (WAVE, AIFF, AU, etc.)
- 2 The files length in bytes
- 3 The length, in frames, of the audio data contained in the file
- 4 An `AudioFormat` object that specifies the data format of the audio data contained in the file. In our implementation, we treat the AU file as 8-bit Mu-law encoded.

5.5 ENCRYPTION METHOD USED

We chose a password Based Encryption scheme with MD5 and DES algorithms. The recommendations are intended for general application within computer and communications systems, and as such include a fair amount of flexibility. They are particularly intended for the protection of sensitive information such as private keys. It is expected that application standards based on these specifications may include additional constraints.

Other cryptographic techniques based on passwords, such as password-based key entity authentication and key establishment protocols are outside the scope of this document. Guidelines for the selection of passwords are also outside the scope.

Salt and iteration count

In as much as salt and iteration count are central to the techniques defined in this document, some further discussion is warranted.

Salt

A salt in password-based cryptography has traditionally served the purpose of producing a large set of keys corresponding to a given password, among which one is selected at random

according to the salt. An individual key in the set is selected by applying a key derivation function KDF , as

$$DK = KDF(P, S)$$

where DK is the derived key, P is the password, and S is the salt. This has two benefits:

- 1) It is difficult for an opponent to precompute all the keys corresponding to a dictionary of passwords, or even the most likely keys. If the salt is 64 bits long, for instance, there will be as many as 2^{64} keys for each password. An opponent is thus limited to searching for passwords after a password-based operation has been performed and the salt is known.
- 2) It is unlikely that the same key will be selected twice. Again, if the salt is 64 bits long, the chance of “collision” between keys does not become significant until about 2^{32} keys have been produced, according to the Birthday Paradox. This addresses concerns about interactions between multiple uses of the same key, which may apply for some encryption and authentication techniques.

In password-based encryption, the party encrypting a message can gain assurance that these benefits are realized simply by selecting a large and sufficiently random salt when deriving an encryption key from a password. A party generating a message authentication code can gain such assurance in a similar fashion.

The party decrypting a message or verifying a message authentication code, however, cannot be sure that a salt supplied by another part has actually been generated at random. It is possible, for instance, that an opponent may have copied the salt from another password-based operation, in an attempt to exploit interactions between multiple uses of the same key. For instance, the opponent may take the salt for an encryption operation with a 80-bit key and provide it to a party as though it were for a 40-bit key. If the party performs a decryption with

the resulting key, the opponent may be able to determine the 40-bit key from the result of the decryption operation, and thereby solve for half of the 80-bit. Similar attacks are possible in the case of message authentication.

To defend against such attacks, either the interactions between multiple uses of the same key should be carefully analyzed, or the salt should contain data that explicitly distinguishes between different operations. For instance, the salt might have an additional, non-random octet that specifies whether the derived key is for encryption, for message authentication, or for some other operation. Based on this, the following is recommended for salt selection:

1. If there is no concern about interactions between multiple uses of the same key with the password-based encryption and authentication techniques supported for a given password, then the salt may be generated at random. It should be at least eight octets (64 bits) long.
2. Otherwise, the salt should contain data that explicitly distinguishes between different operations, in addition to a random part that is at least eight octets long. For instance, the salt could have an additional non-random octet that specifies the purpose of the derived key. Alternatively, it could be the encoding of a structure that specifies detailed information about the derived key, such as the encryption or authentication technique and a sequence number among the different keys derived from the password. The particular format of the additional data is left to the application.

Note. If a random number generator or pseudorandom generator is not available, a deterministic alternative for generating the salt (or the random part of it) is to apply a password-based key derivation function to the password and the message M to be processed. For instance, the salt could be computed with a key derivation function as $S = KDF(P, M)$. This approach is not recommended if the message M is known to belong to a small message

space (e.g., “Yes” or “No”), however, since then there will only be a small number of possible salts.

Iteration count

An iteration count has traditionally served the purpose of increasing the cost of producing keys from a password, thereby also increasing the difficulty of attack. For the methods in this document, a minimum of 1000 iterations is recommended. This will increase the cost of exhaustive search for passwords significantly, without a noticeable impact in the cost of deriving individual keys.

Key derivation functions

A key derivation function produces a derived key from a base key and other parameters. In a password-based key derivation function, the base key is a password and the other parameters are a salt value and an iteration count.

The primary application of the password-based key derivation functions defined here is in the encryption schemes in Section 6 and the message authentication scheme in Section 7. Other applications are certainly possible, hence the independent definition of these functions.

Two functions are specified: PBKDF1 and PBKDF2. PBKDF2 is recommended for new applications; PBKDF1 is included only for compatibility with existing applications, and is not recommended for new applications.

A typical application of the key derivation functions defined here might include the following steps:

1. Select a salt S and an iteration count c .
2. Select a length in octets for the derived key, $dkLen$.

3. Apply the key derivation function to the password, the salt, the iteration count and the key length to produce a derived key.
4. Output the derived key.

Since a password is not directly applicable as a key to any conventional cryptosystem, however, some processing of the password is required to perform cryptographic operations with it. Moreover, as passwords are often chosen from a relatively small space, special care is required in that processing to defend against search attacks. A general approach to password based cryptography, for the protection of password tables, is to combine a password with a salt to produce a key. The salt can be viewed as an index into a large set of keys derived from the password, and need not be kept secret. Although it may be possible for an opponent to construct a table of possible passwords (a so-called “dictionary attack”), constructing a table of possible keys will be difficult, since there will be many possible keys for each password. An opponent will thus be limited to searching through passwords separately for each salt.

Another approach to password-based cryptography is to construct key derivation techniques that are relatively expensive, thereby increase the cost of exhaustive search. One way to do this is to include an iteration count in the key derivation technique, indicating how many times to iterate some underlying function by which keys are derived. A modest number of iterations, say 1000, is not likely to be a burden for legitimate parties when computing a key, but will be a significant burden of opponents.

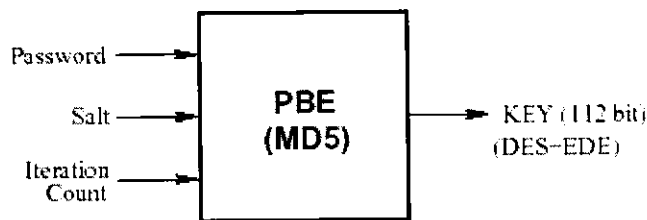


Fig 5.5.1

Salt and iteration count formed the basis for password-based encryption in PKCS . The PBE schemes here are based on an underlying, conventional encryption schemes (for an example, in this implementation, triple DES-EDE with two keys in CBC mode), where the key for the conventional scheme is derived from the password.

A salt in password-based cryptography has traditionally served the purpose of producing a large set of keys corresponding to a given password, among which one is selected at random according to the salt. An individual key in the set is selected by applying a key derivation function KDF, as

$$DK = KDF (P, S)$$

where DK is the derived key, P is the password and S is the salt.

With this scheme, it is difficult for an opponent to precompute all the keys corresponding to a dictionary of passwords, or even the most likely keys. If the salt is 64 bits long, for instance, there will be as many as 2^{64} keys for each password. It is unlikely that the same key will be selected twice. Again, if the salt is 64 bits long, the chance of “collision” between keys does not become significant until about 2^{32} keys have been produced.

An iteration count has traditionally served the purpose of increasing the cost of producing keys from a password, thereby also increasing the difficulty of attack. Of the two functions defined, we chose PBKDF1, which employs a hash function, in this case, MD5.

PBKDF1 Algorithm

PBKDF1 (P, S, c, dkLen)

| | | |
|----------|-------|--|
| Options: | Hash | underlying hash function |
| Input: | P | password, an octet string |
| | S | salt, an eight-octet string |
| | C | iteration count, a positive integer |
| | dkLen | intended length in octets of derived key, a positive integer, at most 16 for MD2 or MD5 and 20 for SHA-1 |
| Output: | DK | derived key, a dkLen-octet string |

Steps:

1. If $dkLen > 16$ for MD2 and MD5, or $dkLen > 20$ for SHA-1, output “derived key too long” and stop.
2. Apply the underlying hash function for c iterations to the concatenation of the password P and the salt S , then extract the first $dkLen$ octets to produce a derived key DK :

$$T_1 = \text{Hash}(P\|S),$$

$$T_2 = \text{Hash}(T_1), \dots$$

$$T_c = \text{Hash}(T_{c-1}),$$

$$DK = T_c \langle 0..dkLen-1 \rangle.$$

3. Output the derived key DK.

The Java™ Cryptography Extension (JCE) [4] provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The software also supports secure streams and sealed objects. JCE was previously an optional package (extension) to the Java™ 2 SDK, Standard Edition (J2SDK), versions 1.2.x and 1.3.x. JCE has now been integrated into the J2SDK, v 1.4. JCE provides an implementation of the MD5 with DES-CBC password-based encryption (PBE) algorithm defined in PKCS #5 together with “Secret-key factories” for bi-directional conversions between opaque DES, Triple DES and PBE key objects and transparent representations of their underlying key material. Thus the implementation is shown in 5.5.2

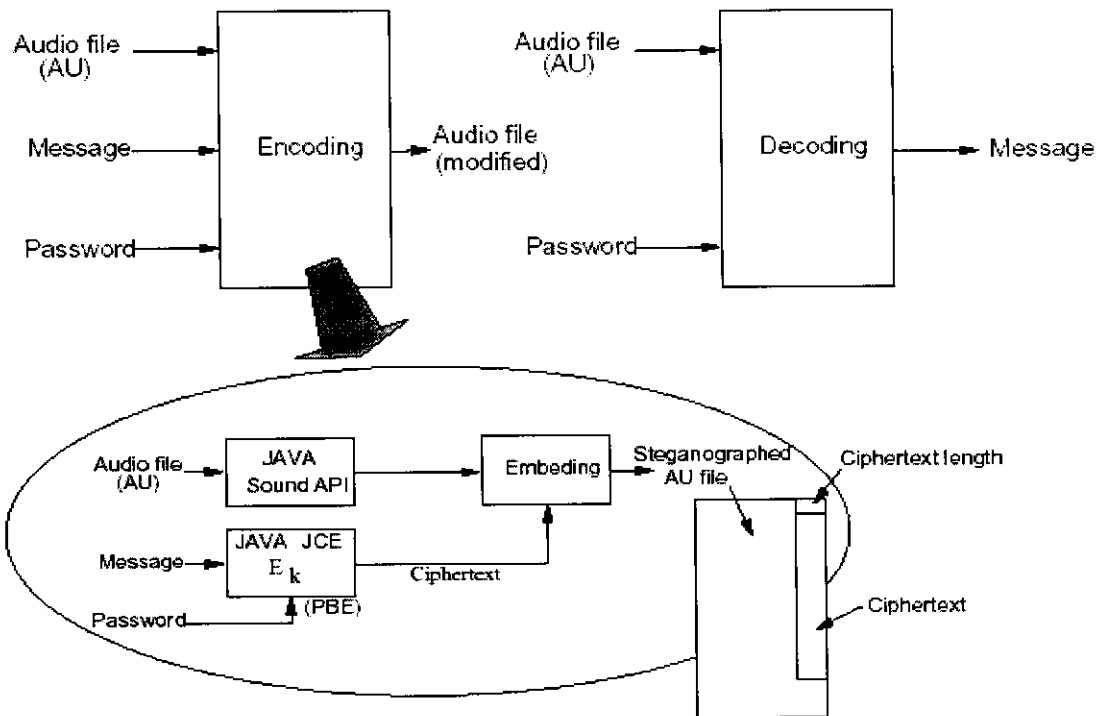


Fig 5.5.2

5.6 MD5 ALGORITHM

MD5 digests have been widely used in the software world to provide some assurance that a transferred file has arrived intact.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A , B , C and D . These are initialized to certain fixed constants. The main algorithm then operates on each 512-bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed *rounds*; each round is composed of 16 similar operations based on a non-linear function F , modular addition, and left rotation.

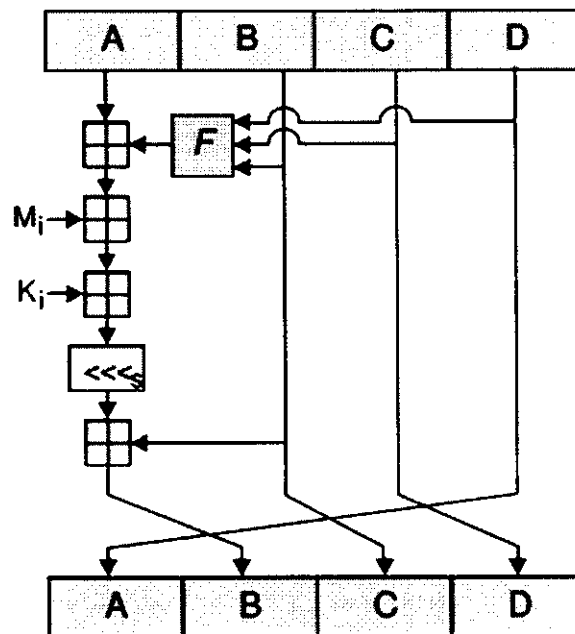


Fig.5.6

Here One MD5 operation-MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. F is a nonlinear function; one function is used in each round. M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation.

\lll_s denotes a left bit rotation by s places; \oplus denotes addition modulo 2^{32} .

5.7 DES ALGORITHM

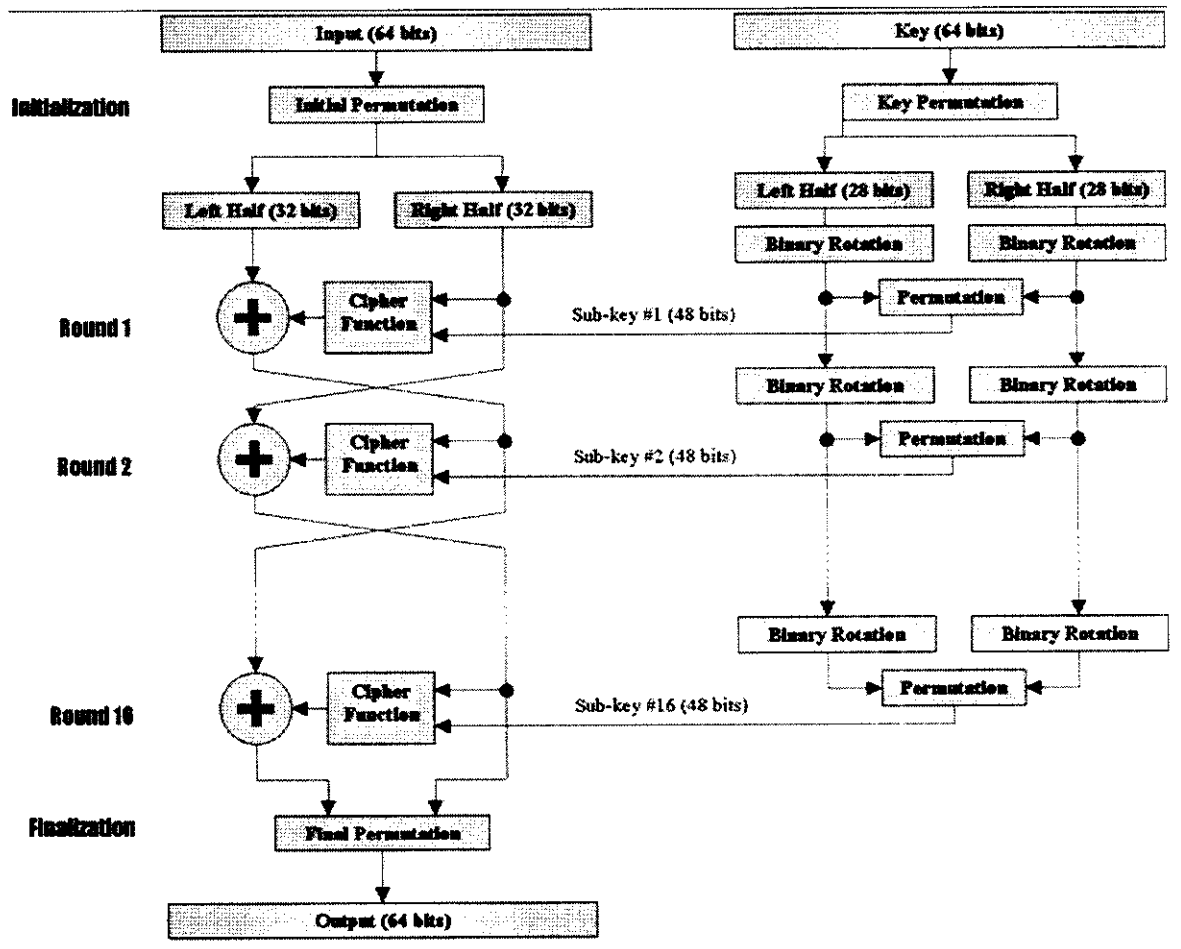


Fig 5.7

The **Data Encryption Standard (DES)** is a cipher (a method for encrypting information). DES is the archetypal block cipher — an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another cipher text bit string of the same length. In the case of DES, the block size is 64 bits.

DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits, and it is usually quoted as such.

There are 16 identical stages of processing, termed rounds. There is also an initial and final permutation, termed IP and FP, which are inverses (IP "undoes" the action of FP, and vice versa). IP and FP have almost no cryptographic significance. Before the main rounds, the block is divided into two 32-bit halves and processed alternately; this criss-crossing is known as the Feistel scheme. The Feistel structure ensures that decryption and encryption are very similar processes — the only difference is that the subkeys are applied in the reverse order when decrypting. The rest of the algorithm is identical. This greatly simplifies implementation, particularly in hardware, as there is no need for separate encryption and decryption algorithms.

CHAPTER 6

TESTING

Testing is the process of executing the program with the intent of finding errors. During testing, the program to be tested is executed with a set of test cases and the output of the program for the test cases is evaluated to determine the program is performing as it is expected. Error is the testing fundamental and is defined as the difference between the actual output of a software and a correct output i.e., difference between the actual and ideal testing is usually relied upon to detect these faults in the coding phase for this, different levels of testing are used which performs different tasks and aim to the test different aspects of the system.

6.1 TESTING OBJECTIVES

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has high probability of finding an as yet undiscovered error.

6.2 TYPES OF TESTING

1. Functional Testing

Here the structure of the program is not considered. Only the test cases are decided solely on the basis of the requirements or specification of the program or module and the internal details of the module or the program is not considered for the selection of test cases. This is also called "*Black Box Testing*".

2. Structural Testing

It is considered with testing the implementation of the program. The intention of the structural testing is not to exercise all the different input and output conditions but to exercise the different programming and data files used in the program. This testing is also called “*White Box Testing*”.

6.3 DIFFERENT LEVEL OF TESTING

6.3.1 UNIT TESTING

In it different modules are tested against the specifications produced during design for the modules. It is essential for verification of the code produced during the code phase and the goal is to test the internal logic of the module.

6.3.2 INTEGRATION TESTING

The goal here is to see if the modules can be integrated properly, the emphasis being on testing interfaces between modules. After structural testing and functional testing we get error free modules these modules are to be integrated to get the required results of the system.

6.3.3 SYSTEM TESTING

Here the entire software is tested. The reference document for this process is the requirement document and the goal is to see whether the software needs its requirements. The system was tested for various test cases with various inputs.

6.3.4 ACCEPTANCE TESTING

It is some times performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here focus on the external behavior of the system, the internal logic of the program is not emphasized. In acceptance test the system is tested for various inputs. Thus various tests have been performed.

CHAPTER 7

PERFORMANCE AND LIMITATIONS

7.1 MERITS OF THE SYSTEM:

The aim of providing a secure communication is achieved by hiding the data in a audio file and transferring it across the system. It has merits such as

- It ensure the privacy of communication
- Hiding information such that even its presence can't be detected
- There would be no perceivable changes in audio file after message insertion

The encryption adds another level of security to the conventional stegosystem. It is necessary to maintain the same salt and iteration count to generate the same key in both encryption and decryption process. This reduces the key space in a brute force attack. But the salt which is compiled with the source code itself makes it harder to guess. Also the Triple DES (EDE), used here, is regarded as a better alternative to conventional DES.

7.2 LIMITATIONS OF THE SYSTEM

The major disadvantage of low-bit encoding is, its poor immunity to manipulation. Encoded information can be destroyed by channel noise, re-sampling etc., unless it is encoded with redundancy techniques. In order to be robust, these techniques reduce the data rate, often by one to two orders of magnitude. In practice this method is useful in closed, digital environments. Other data hiding techniques such as Phase encoding, Spread spectrum and Echo hiding have better immunity to manipulation.

Steganalysis focuses on two aspects: detection of embedded message, and destruction of embedded message. In our implementation we noticed that the detection process is extremely difficult since HAS has a fairly small differential range.

Examples of Steganography attacks are: Stego-only attack. Host-stego attack (known host signal attack), and chosen message attack. Trivially, our implementation is strong against those types of attacks since the attacker is unaware of the encryption algorithm and also the parameters of the algorithm. There is no secure stegosystem if the attacker knows both host medium, and stego medium. The difference reveals that Steganography was used. But the encryption adds another level of security to the conventional stegosystem.

It is necessary to maintain the same salt and iteration count to generate the same key in both encryption and decryption process. This reduces the key space in a brute force attack. But the salt which is compiled with the source code itself, makes it harder to guess. Also the Triple DES (EDE), used here, is regarded as a better alternative to conventional DES.

7.3 FUTURE ENHANCEMENTS

- 1 Public key cryptography system suggests a very secure and convenient way of communicating. The audio containing the embedded text can be sent through the network, and that can be extracted at the other end.
- 2 Support for other audio formats can easily be added since Java Sound API can handle WAVE, AIFF, etc.. However support for MP3 (MPEG layer3) format which uses different encoding scheme, needs additional sound encoding/decoding method to be implemented. This can be simplified by the use of a freeware GPL (rather, LGPL) encoder such as BladeEnc.
- 3 Choice of various encryption algorithms such as IDEA, AES, etc., can be implemented with Java Cryptography Extension or using another provider.
- 4 Robustness (i.e. resistance to data manipulation) can be increased by using other techniques of data hiding.

CHAPTER-8

APPENDICES

8.1 SAMPLE SCREEN

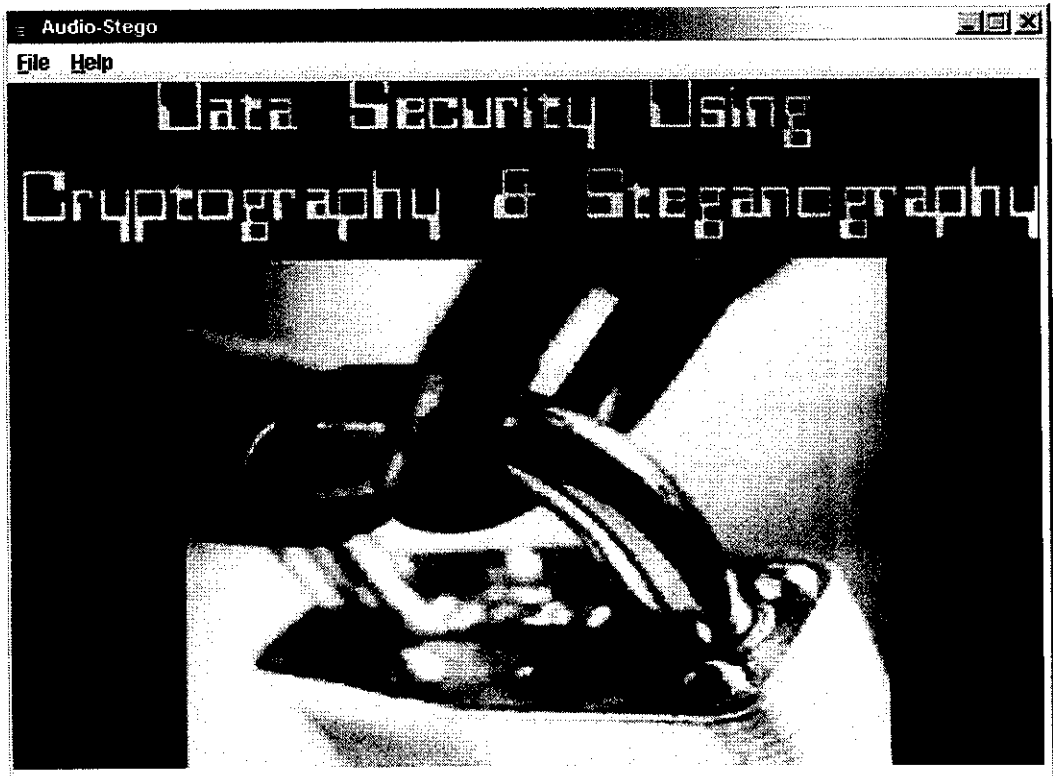


Fig A.1 Front Screen

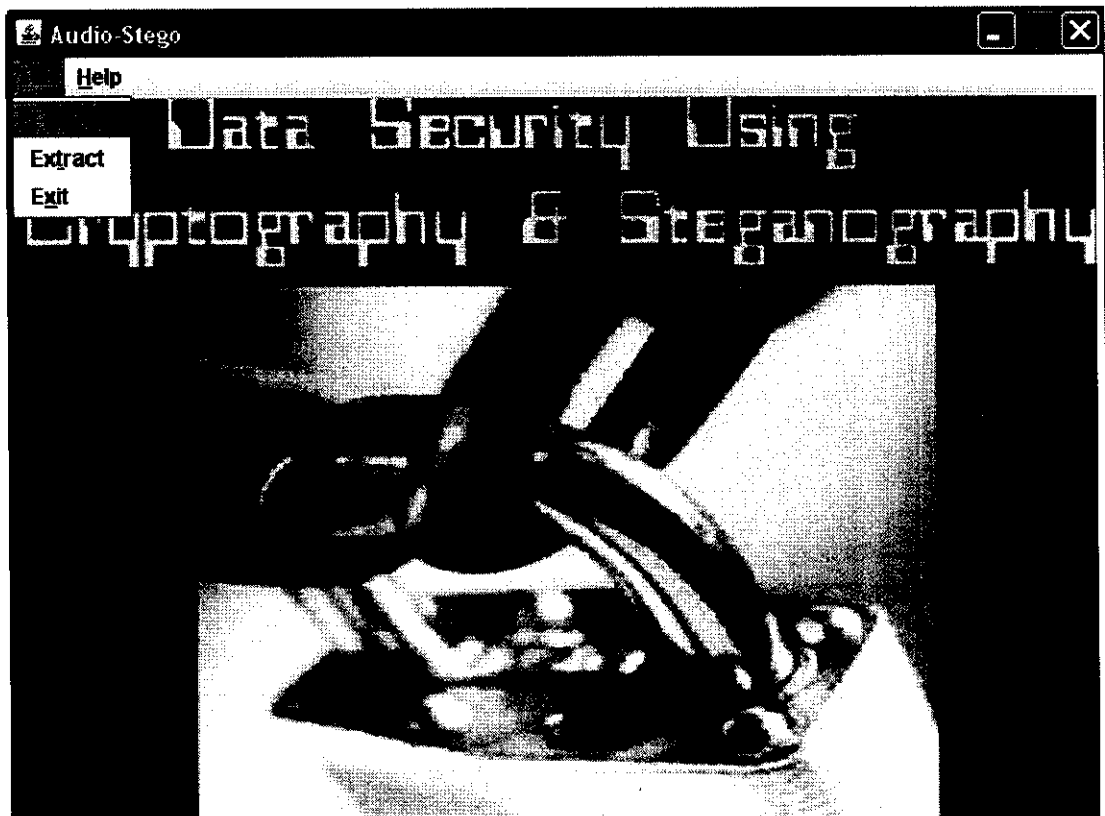


Fig A.2 Embed Action

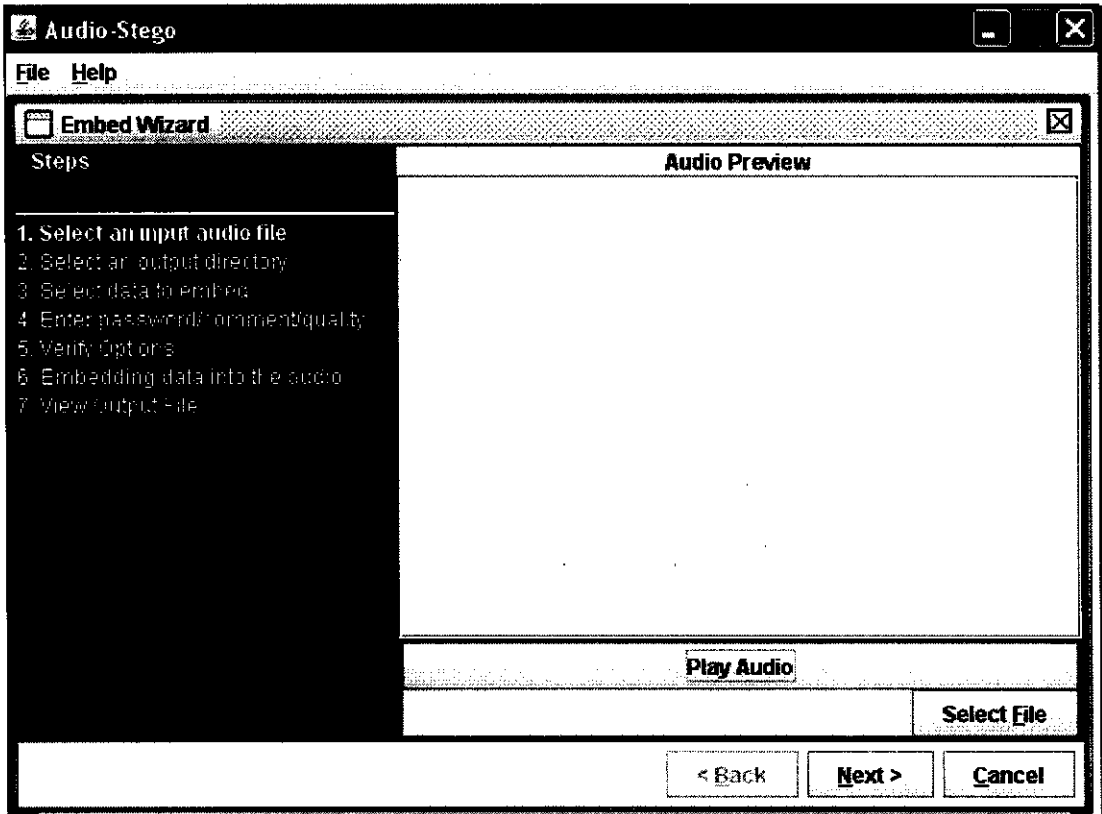


Fig A.3 Select an Input Audio File

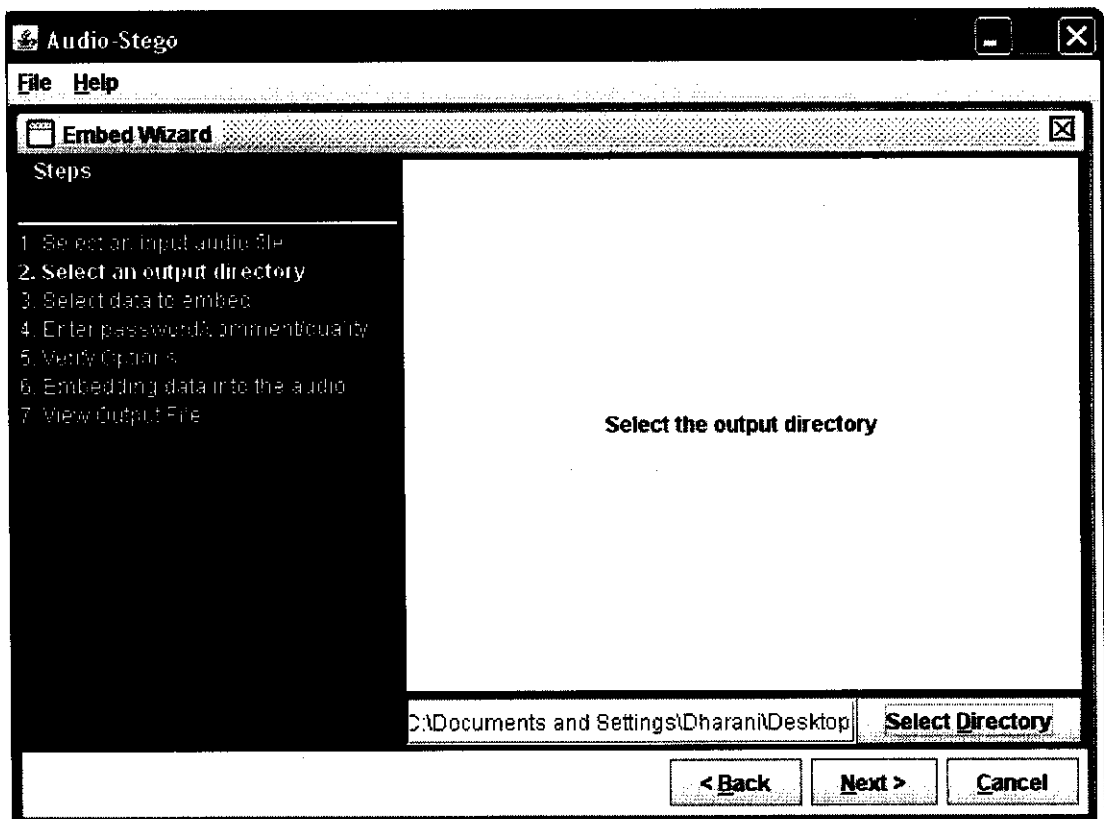


Fig A.4 Select an Output Directory

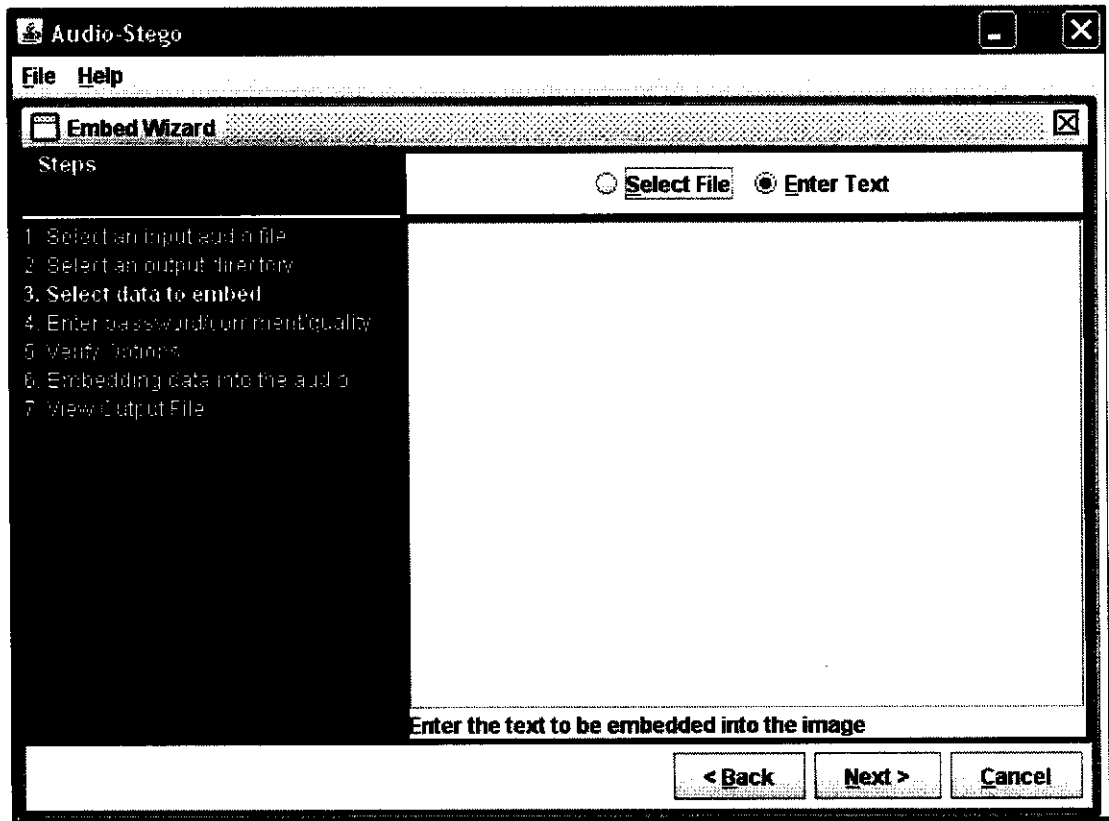


Fig A.5 Selecting the Data to Embed

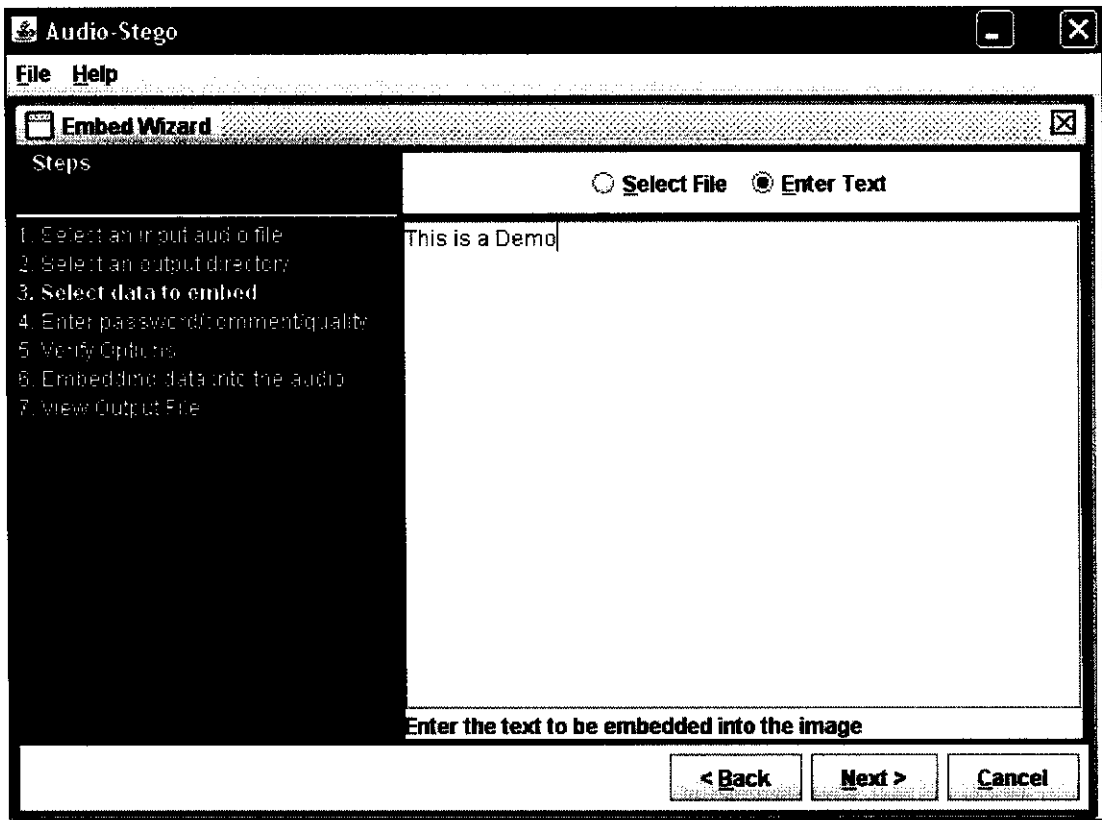


Fig A.6 Enter the Data to Embed

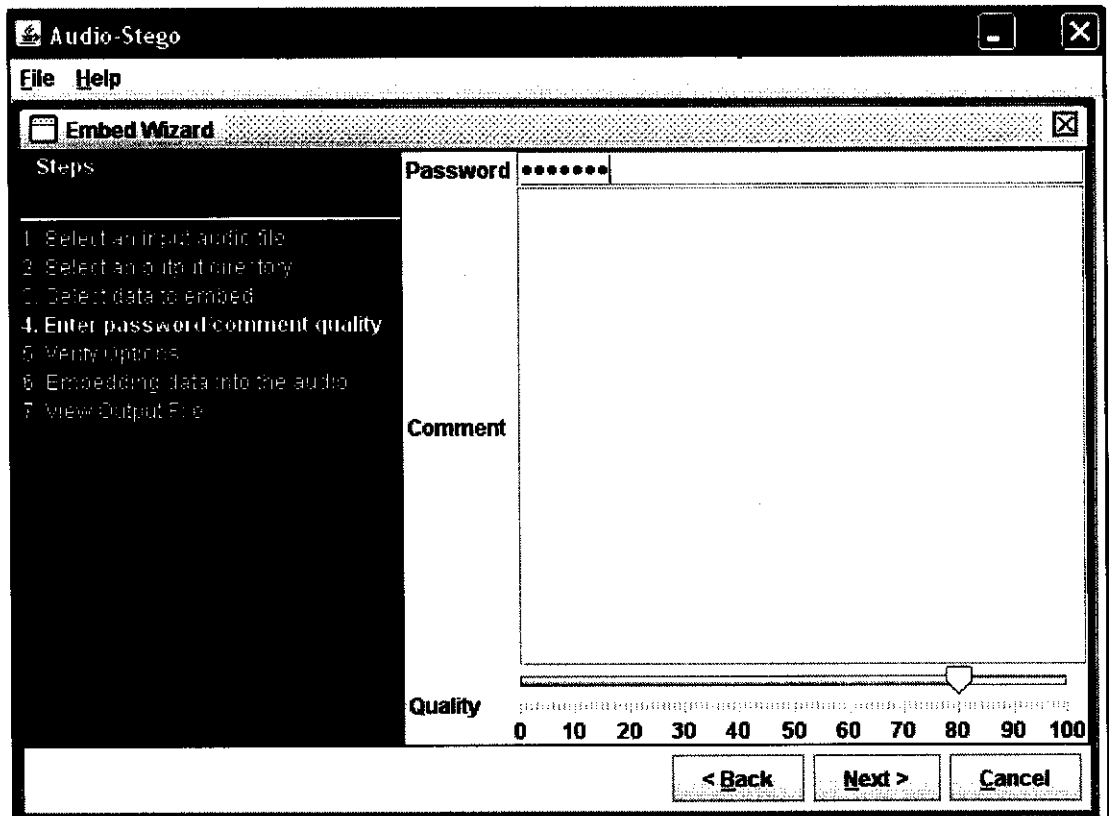


Fig A.7 Enter the password/Comment /Quality

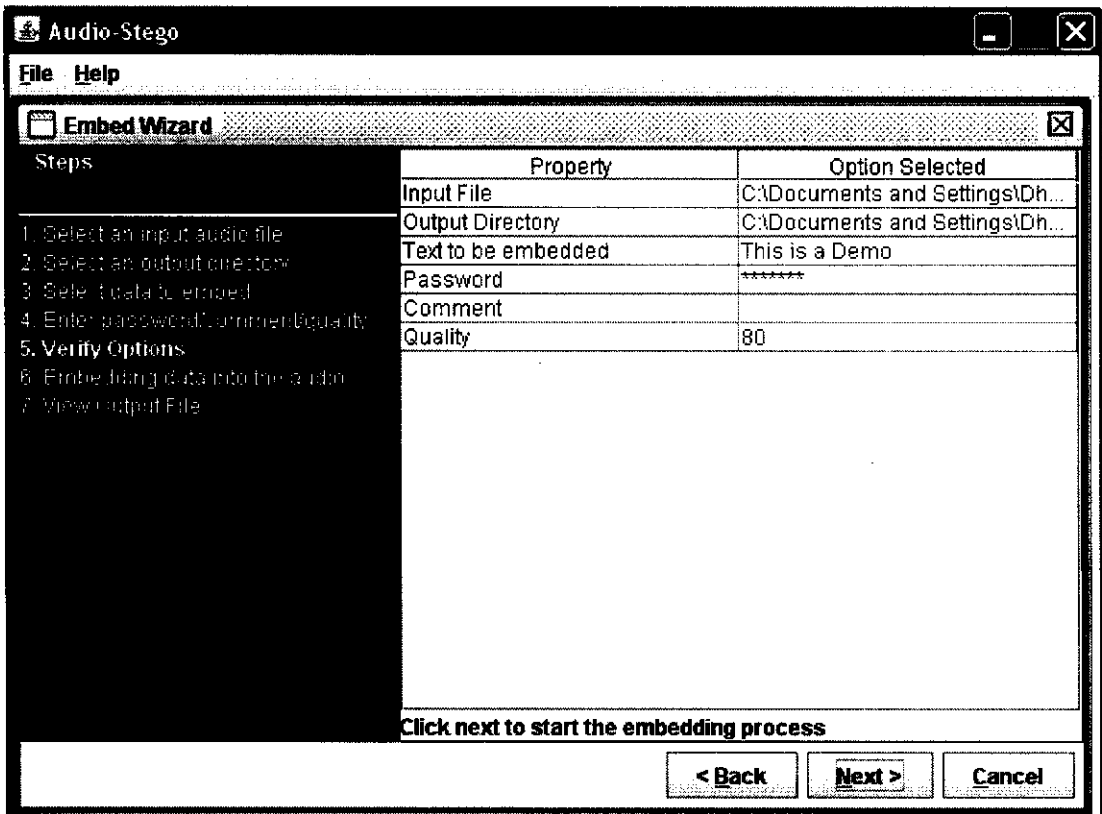


Fig A.8 Verifying the data Entered

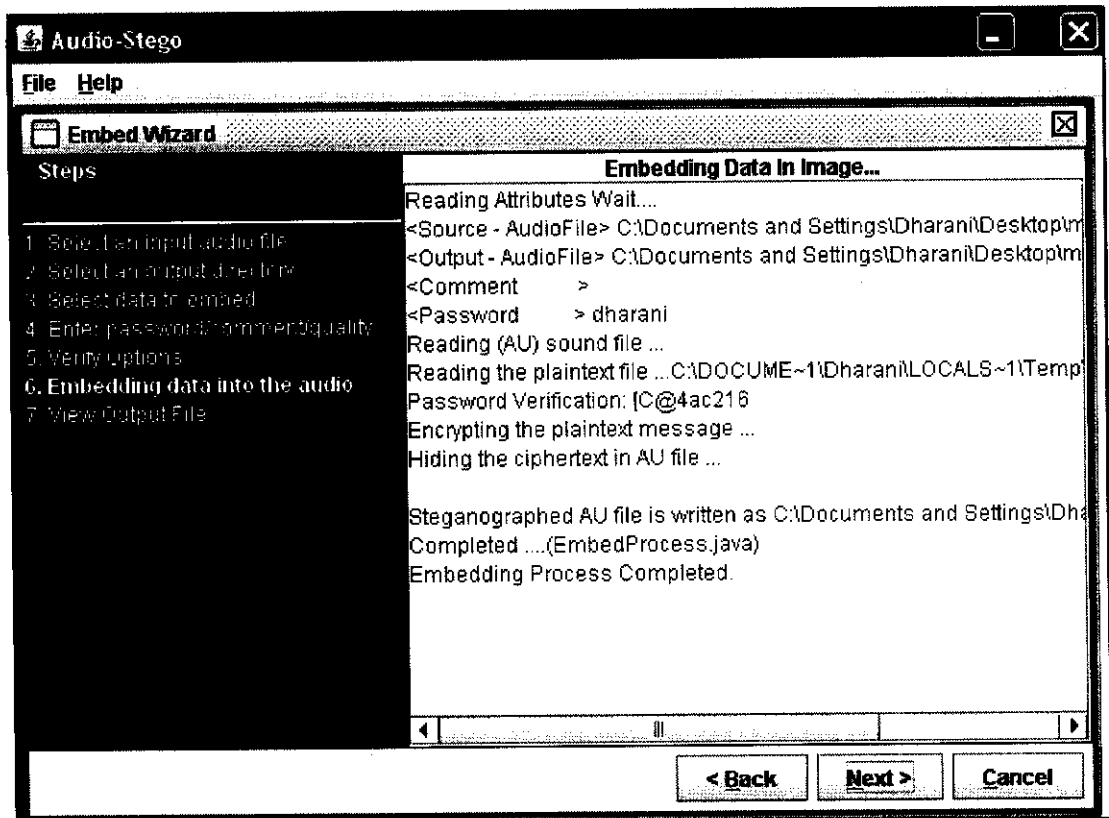


Fig A.9 Embedding data into Audio file

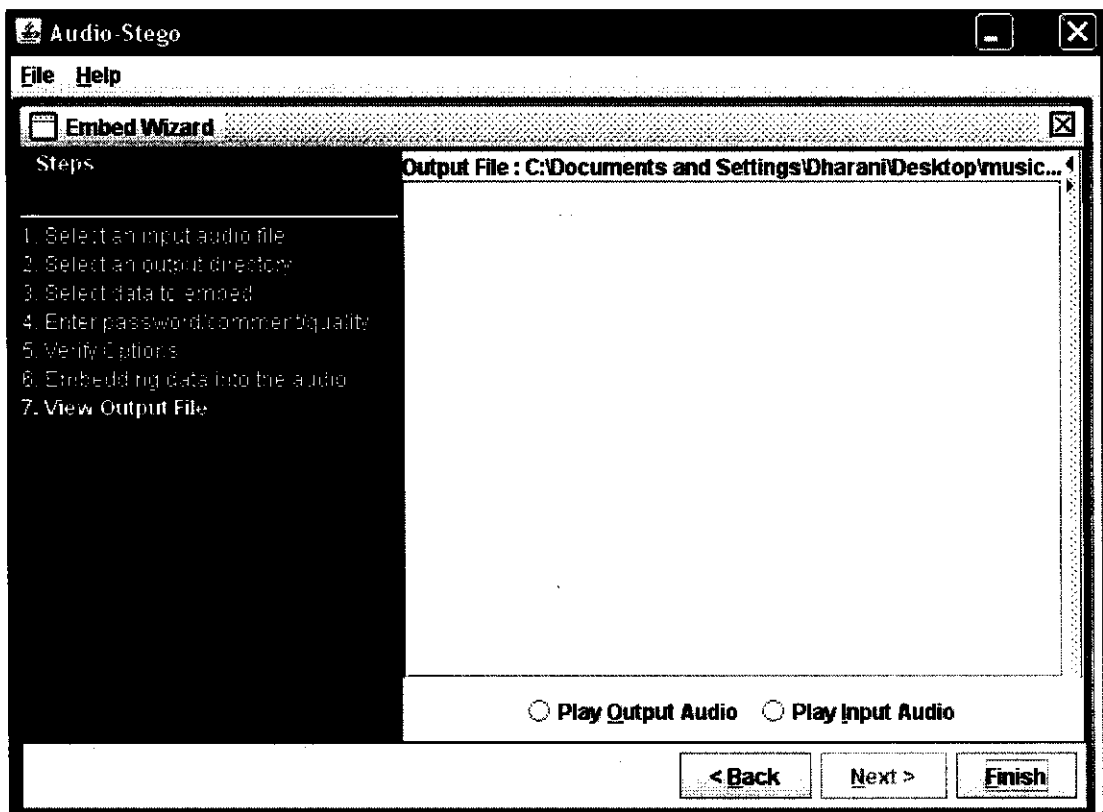


Fig A.10 View output file after Embedding

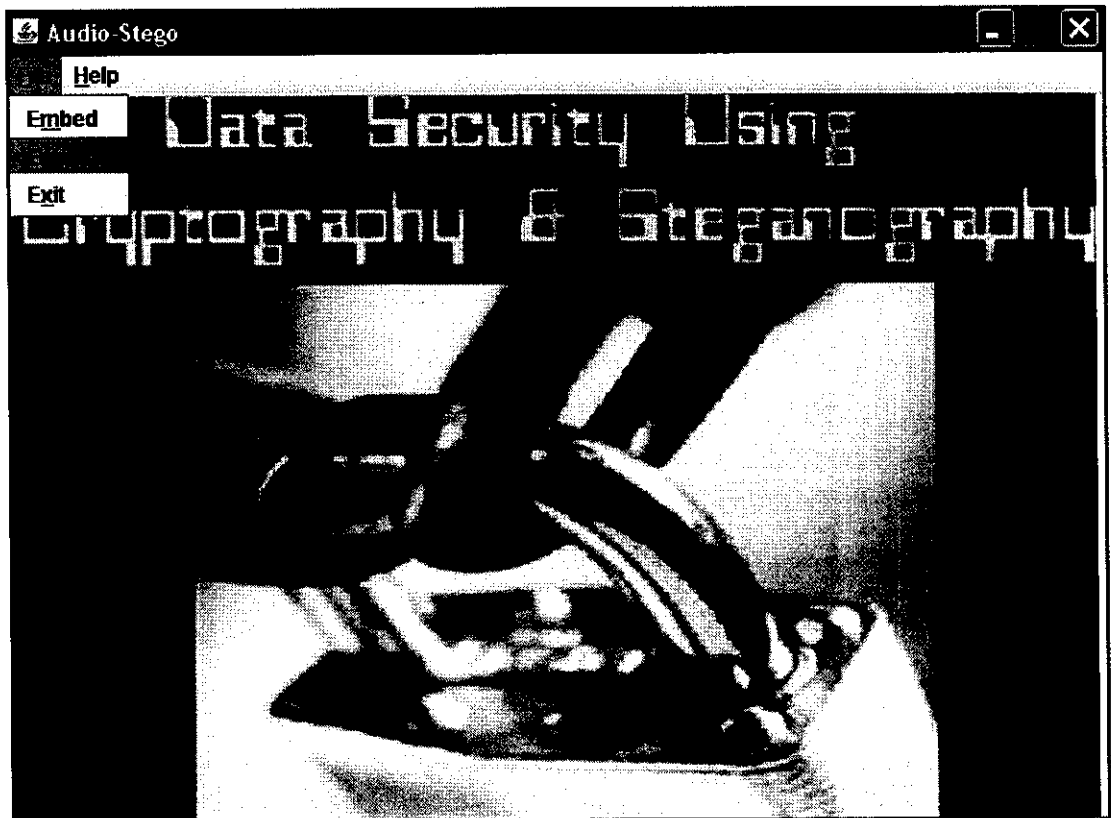


Fig A.11 Extract Action Screen

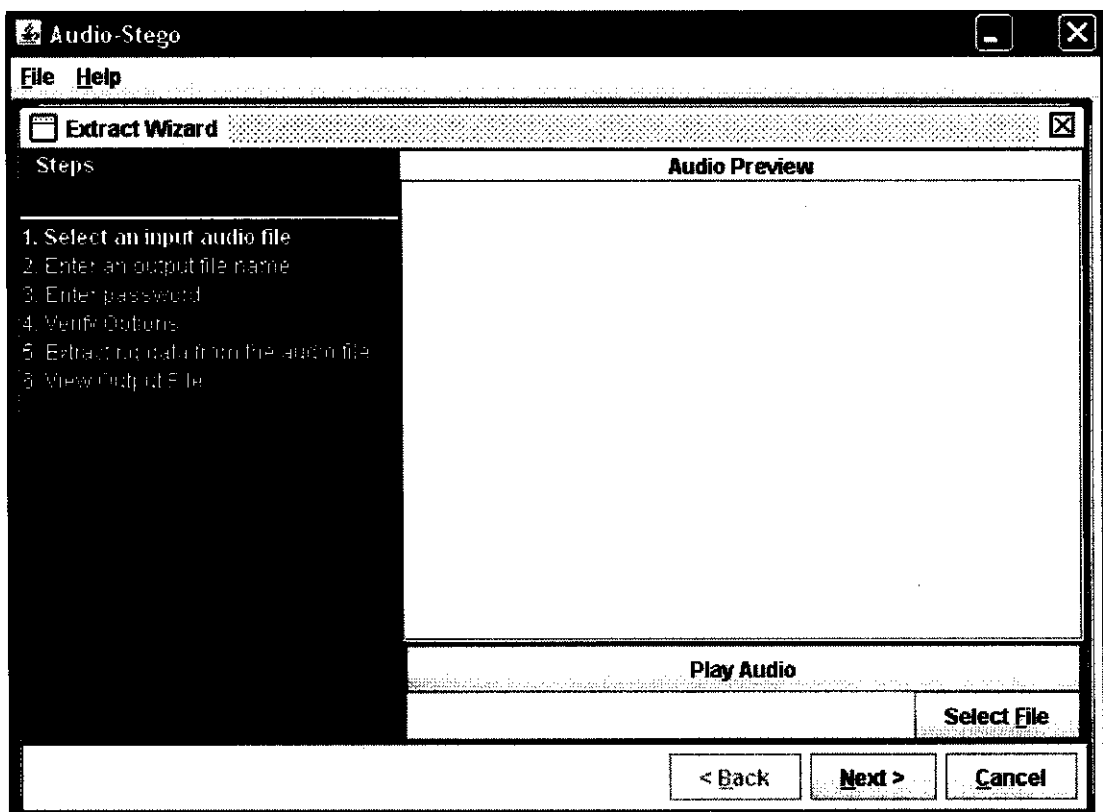


Fig A.12 Screen to select the input file for extracting the data

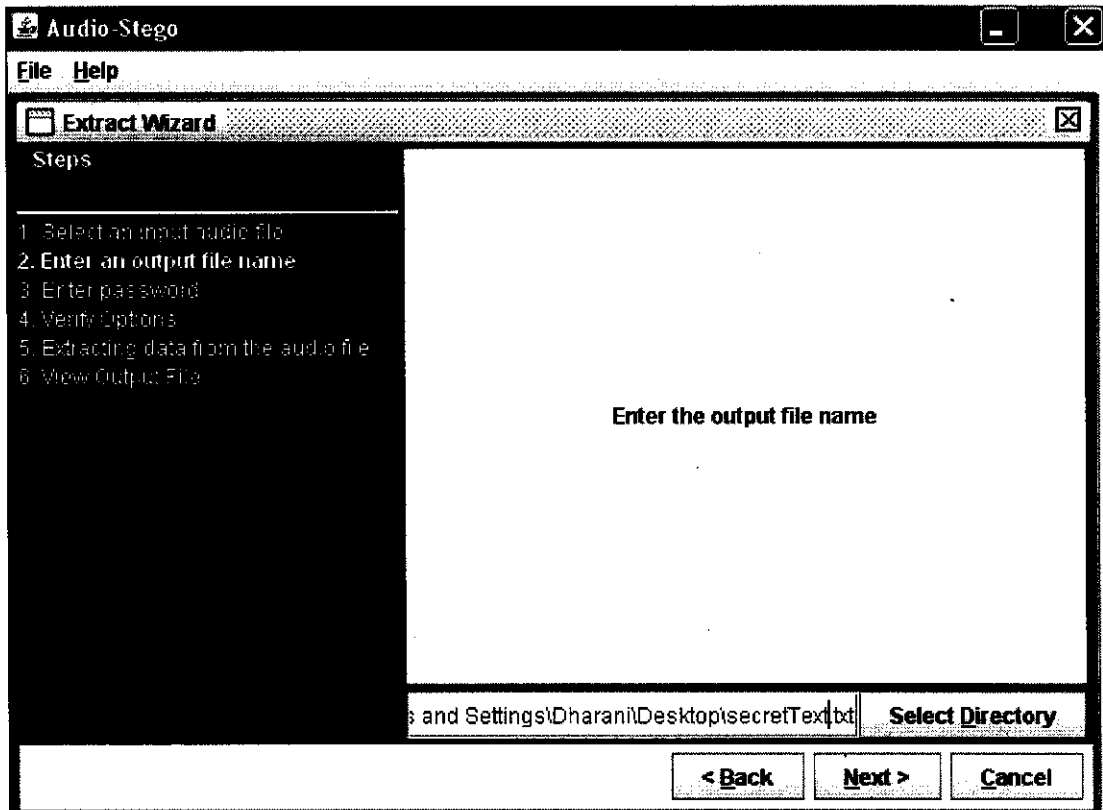


Fig A.13 Enter the output directory to save the secret text

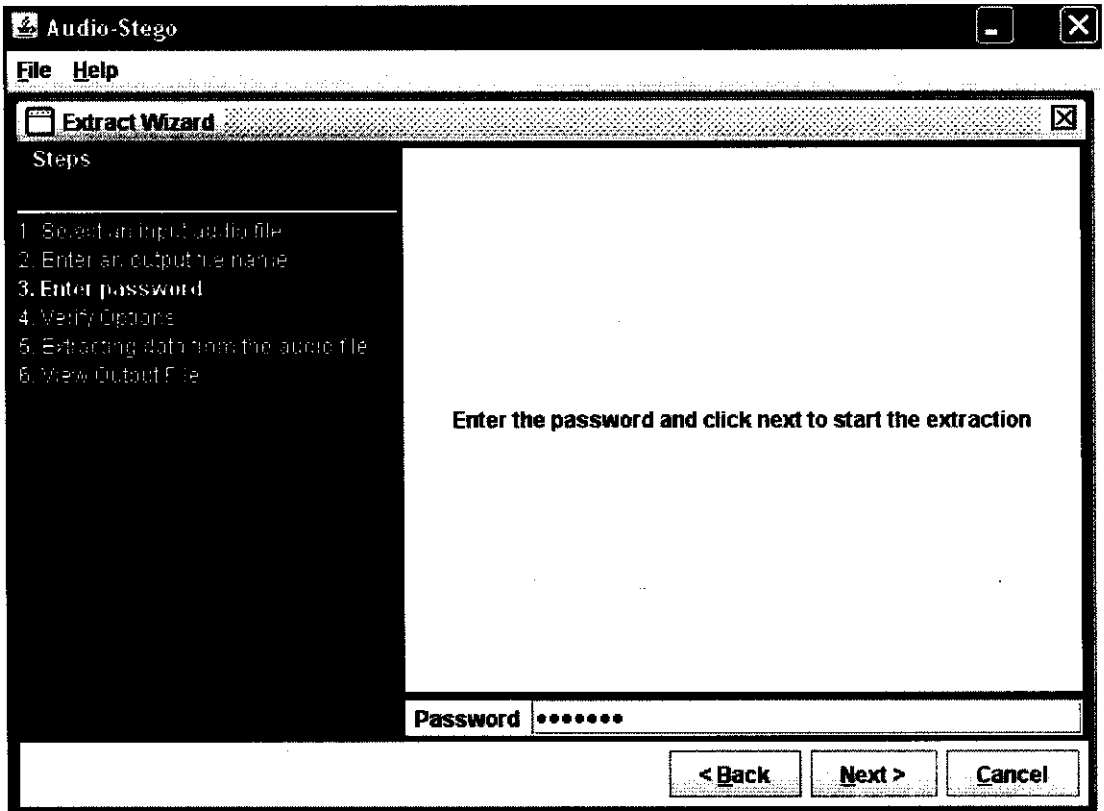


Fig A.14 Entering the password for Extraction

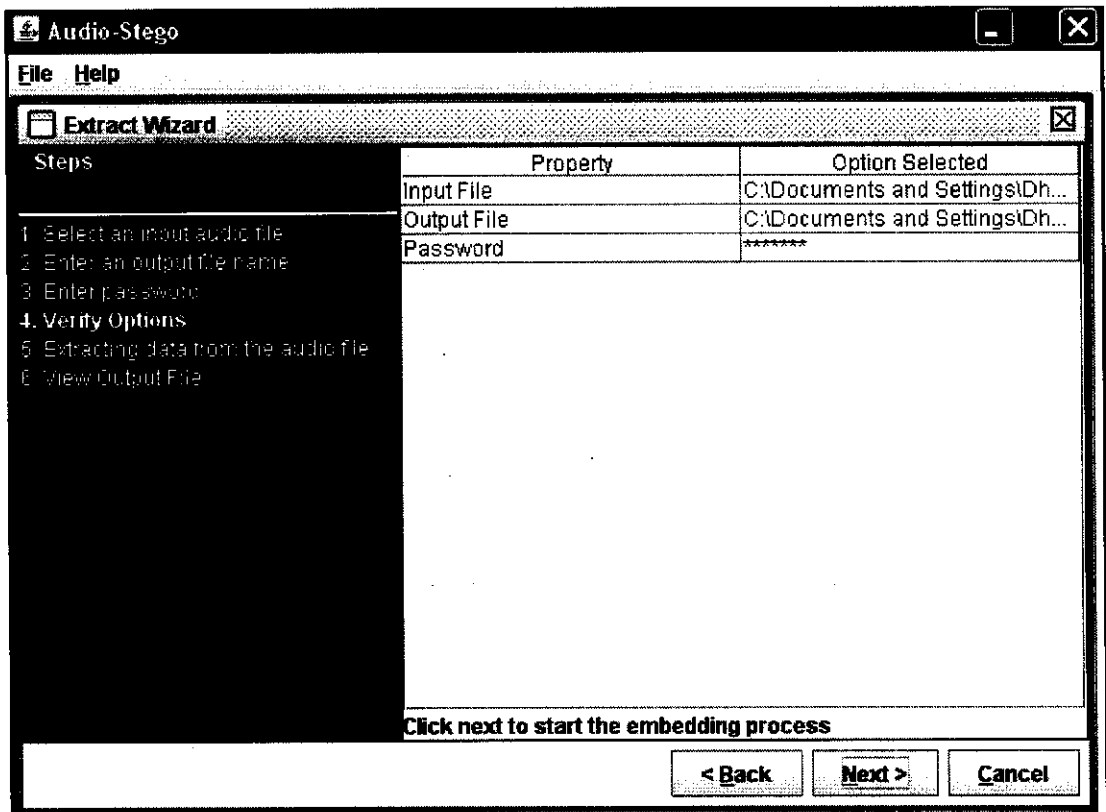


Fig A.15 Screen that verifies options entered

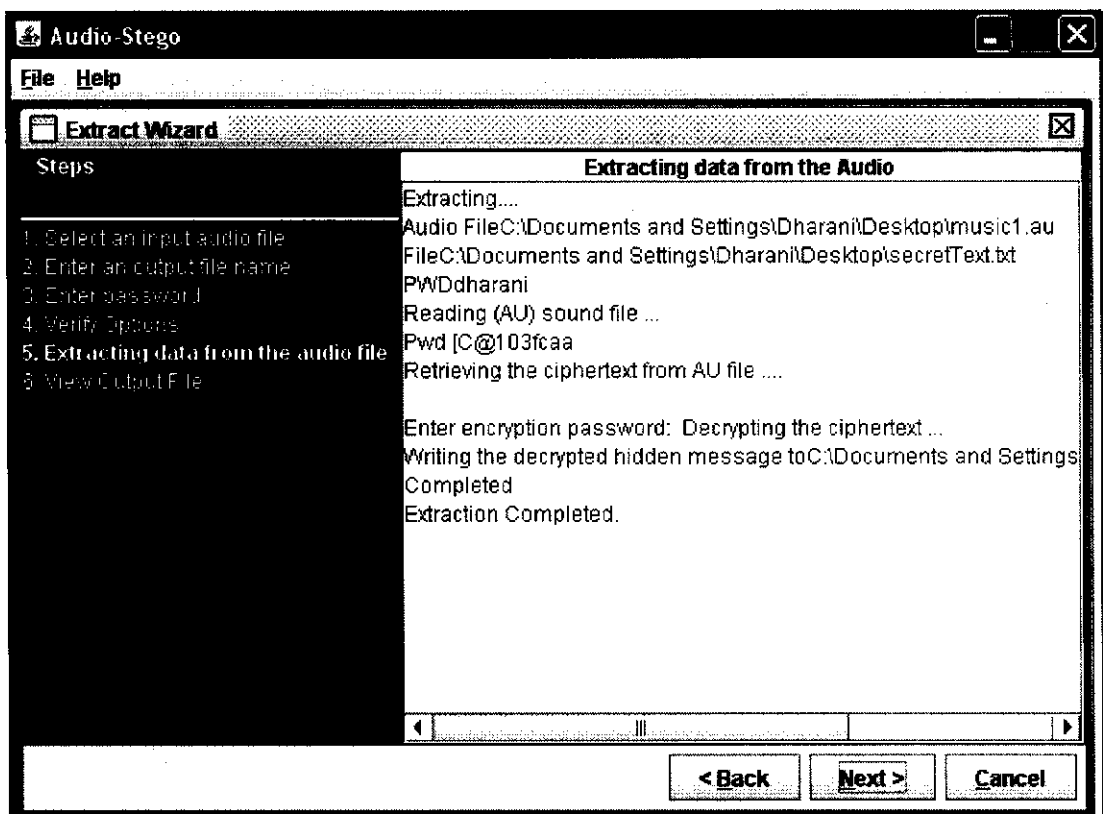


Fig A.16 performing the extraction

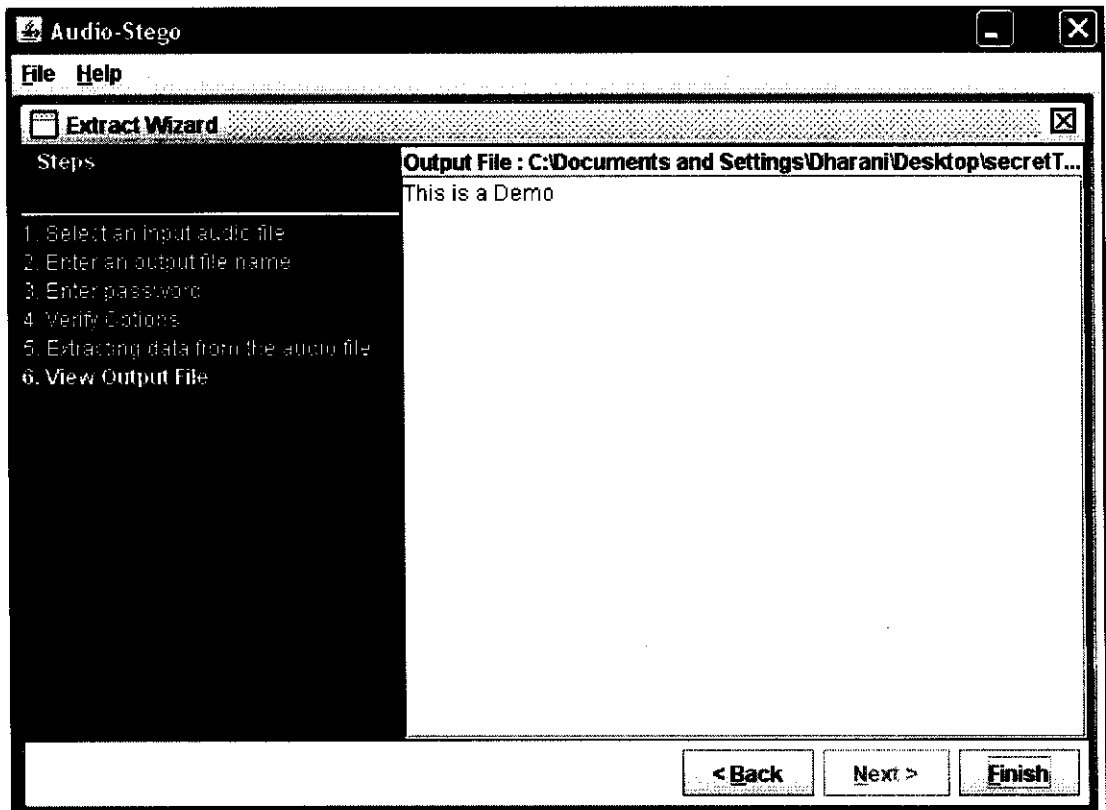


Fig A.17 View the Secret data

CHAPTER-9

BIBLIOGRAPHY

1. Herbert Schildt, "Java 2 - The Complete Reference", IVth Edition.
2. Java Sound API – <http://java.sun.com/products/java-media/sound/>
3. Java™ Cryptography Extension - <http://java.sun.com/products/jce/>
4. Bender W., Gruhul D., Morimoto N., "Techniques for data hiding", IBM Systems Journal, Vol 35, Nos 3&4, 1996.
5. Core Java –Cay S.Horstmann, Gary Cornell.