Hiding Data within an Image using Steganography

Shreyoshi Chatterjee, Aftab Hussain, Goutam Majumder, Ratul Paul, Dattatreya Raychowdhuri

***Abstract*— this project deals with the concept of Steganography. Steganography is the process of sending messages in a concealed manner in a particular medium. Here we intend to send messages from a sender to a receiver via image files. The message is encrypted in the image file in such a way that no one could detect any changes in the resulting image by the naked eye. It is then sent to the receiver. The message can only be decrypted by the intended receiver and no one else, because it will be password protected, which will only be known to the classified sender and receiver.**

***Index Terms*— Cover medium image, stego medium image, stego key**

I. Introduction

In this paper we discuss our project entitled "Hiding Data within an Image using Steganography". The objective of our project was to build a software that could hide data in an encrypted manner in an image file and decode the data back as and when required, in an efficient manner. The software uses the principles of Steganography.

According to Wikipedia [3], "Steganography is the art and science of writing hidden messages in such a way that no-one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity." The word "steganography" is derived from the Greek words *steganos*, which means "covered," and graphia, which means "writing." The word Steganography was first used in Johannes Trithemius's Steganographia, a treatise on cryptography and Steganography in 1499. In practice, it involves encryption of messages in a certain medium (such as a document, image, sound, or movie file) in such a manner that the existence of the message in the medium is obscured.

The software uses the following basic formula:

cover medium + hidden data + stego key = stego medium

'cover medium' denotes the Image file in which the data will be hidden. '*Hidden data'* denotes the data which is to be hidden. '*stego key'* denotes the password that is used to encrypt the '*hidden data'*. *'stego medium'* is the final image that is generated after the Steganographic encryption process.

The manner in which our project gains an advantage in the maintenance of confidentiality may be illustrated as follows. Let's say Alan sends a message to Graham, and it is to be such that only the two of them are to know about the contents of the message. In that case, Alan will encrypt the message, using any cryptographic algorithm and send it to Graham as a scrambled text message via a communication path. Now if an eavesdropper, Ryan, intercepts the message, he'll find the encrypted message. Even though all Ryan gets is a scrambled message, it would prompt Ryan that the message contains something confidential. Ryan would hence try to break the algorithm used, as hackers do, and decrypt the message.

This problem is not faced while using our software to transmit messages as it avoids any sort of suspicion. Our software encrypts the message in an image file, which could then be transferred. Hence, if Alan were to use our method, he would encrypt his message in an image file and transmit it to Graham, easily avoiding any suspicion that the image contains a message.

The method applied is simple. Any image consists of a large number of pixels, each of which consists of a certain number of bytes. The bytes of each pixel are represented by a numerical value, which corresponds to a particular color. These bytes could be manipulated to carry data. However, changing these bytes can cause changes to the overall appearance of the image, which may evoke suspicion that the image contains some data. To circumvent this issue, data is stored in the least significant bytes of each pixel of the image. This causes a slight modification to the color of the pixels, which is, however, unnoticeable to the human eye. As a result what we get is an image, with the encrypted data, visually identical to the original image, making it a lot safer for transmission via a communication path; attackers do not notice anything odd about the image being passed. This is a very popular Steganographic technique, which has been used in several Steganographic tools and applications. The technique is known as LSB embedding. This technique has also been implemented in our software.

It may however be put to notice that even this strategy has been countered by attackers. An aggressive attacker could analyze the LSB of each pixel of an image and may, eventually, bring together the message. Steganalysis is the method used by hackers to counter the techniques of Steganography. Steganalysis is the art and science of detecting messages hidden using steganography; this is analogous to cryptanalysis [17] applied to cryptography [18]. The goal of steganalysis [4] is to identify suspected packages, determine whether or not they have a payload encoded into them, and, if possible, recover that payload. So with image files, hackers are most likely to analyze the LSBs of each of the pixels, and attempt to deduce any anomaly. If they do trace any aberrance, they would be prompted with the idea that the image contains confidential data, which would encourage them to proceed with their decrypting techniques.

This is where our software algorithm strives for complicating matters for the hacker. Instead of using the general LSB encoding technique, we have used a modified version of it. Here, instead of using the very last LSB, we have made use of the last 2 LSBs. We have implemented an alternating sequence of encryption where once the very last LSB of a pixel is altered, and then the 2nd last LSB of the next pixel is altered, and so on. This makes the encryption process less obvious and a lot less prone to attacks as compared to the general Steganographic LSB embedding technique.

In Section II we elaborate on the wide range of implementations of Steganography and on software tools which implement Steganographic techniques. Later on in Section III, the algorithm which we implemented in our software is explained. In Section IV we display and discuss the outputs of our application and also give an assessment of our project. Finally Section V gives the conclusion of the project.

II. Implementations of Steganography and related software tools used

Steganographic techniques have a wide range of applications in different fields as enlisted below:

Usage in modern printers. Steganography is implemented by many modern colour printers, such as those of HP and Xerox, which print small yellow dots or characters. These dots and characters generally denote the timestamp or a serial code and are printed in such a manner that makes it difficult to notice them by the naked eye. However, they can be noticed through a magnifying glass or blue light. Printer companies first used this technology during the 1990s mainly to ensure governments that their machines were not being used for forgery.

Usage in mobile telephony. Due to the rising needs of data security and hidden communications in mobile telephony, steganography has found significant use in WAP, or Wireless Application Protocol [8]. Here steganography has been implemented in WML by means of encoding information in the ID attribute of the WML document tags. (Wireless Markup Language is a language used for creating web pages for the WAP.) The decoder program is written in J2ME (Java 2 Micro Edition).

Usage in digital watermarking. Digital watermarking [1,5] is the process of embedding information in digital media, which can be visible or invisible. Visible watermarking can be used on digital media to represent the ownership of the media. Invisible watermarking is generally used for copyright protection of digital media. Steganography finds application in invisible watermarking where it may be necessary to communicate secret messages embedded in digital signals.

Usage in crime. The advantages of steganography technologies have also facilitated various malicious activities. Hackers use steganography along with the "chaffing and winnowing" technique to embed malicious code in emails (junk mails and spams) and various other digital media to retrieve confidential information of users connected to a network. Malicious actors communicate and pass secret information through digital media, which are very difficult to detect, thus posing a serious threat to security. Various decoding software have been built to counter such practices.

Usage in Access control system for digital content distribution. With the advent of digital content distribution & e-commerce over the Internet, the issue of access rights has become very important. As a result, many access control systems, which utilize steganographic techniques, have been designed where digital content is embedded by a steganographic technique in folder-by-folder manner and where each folder has a unique access key. The entire embedded content is then uploaded on a web page where the contents are publicized. Any customer who would wish to access the content would then have to request for the access key.

Usage in Media Database Systems. Steganography is used for non-security related purposes as well. In media database systems, problems may arise when there is a requirement to store additional information, annotations, or metadata (data about data) of the contents of the database along with the contents themselves in a unified manner. Examples of such metadata include photographer's name, date, etc for a photos database system. Common digital album software does facilitate the storage of such information, but fails to bind this additional information with the corresponding digital data. As a result, when the digital databases are transferred from one machine to another, the additional information is lost. This problem is tackled by steganography, which unifies the two types of data by an embedding operation.

Various software that make use of Steganographic techniques include:

S-Tools: S-Tools is a steganography tool that hides files in BMP, GIF, and WAV files. This is a powerful and versatile tool as it allows multiple files to be hidden in a single audio or image file. It is a Windows based application.

StegoDos: Also known as the Black Wolfs Picture Encoder version 0.90a. It works only for 320* 200 images with 256 colors

Camouflage: Allows hiding files by scrambling them and then attaching them to the file of your choice. It has found use especially in emailing where senders can send attachments without revealing the existence of the attachments.

Mp3 Stego: Hides information using Steganography in MP3 files during the compression process.

III. Algorithms used

While designing our application, we have kept in mind our primary goal, which is to maintain the identicalness of the original image and the image, which bears the message.

Our algorithm falls under private key or symmetric key cryptography where a shared key is used for both encryption and decryption. The key is to be shared between the sender and the receiver. In our design, in order to decrypt a message from a particular image, one must have our application as well as the shared key.

A. ENCRYPTION ALGORITHM

Step.1: Store the path of the *textfile*, which contains the message to be encrypted.

Step.2: Read and store contents of the *textfile* in ASCII format in a string variable, *ascmsg*. (Putting an 'A' after the ASCII code of each character; for example, the message "abc" would be stored in a string as, "97A98A99A".

Step.3: Take the password as input and store it in another string variable, *ascpwd* in ASCII format.

Step.4: Store a new string variable, ascContent, such that it contains *ascpwd*+*length* of *ascmsg* + 'F' + *ascmsg* (concatenated)

Step.5: Load the image, on which the message is supposed to be encrypted, in the buffer.

Step.6: Put the 1st character of *ascContent* in the 6th double byte of the first pixel (top left) of the loaded image.

Step.7: Put the next character of *ascContent* in the 5th double byte of the next pixel (in the same row) of the loaded image. Move to the next row if the end of row is reached.

Step.8: Put the next character of *ascContent* in the 6th double byte of the next pixel of the loaded image. Move to the next row if the end of the row is reached.

Step.9: Repeat steps 7 and 8 until all characters of *ascContent* have been encrypted in the image.

Step.10: Save the modified image from the buffer in a location specified by the user. (This is the Stego Medium image).

B. DECRYPTION ALGORITHM

Step.1: Load the Stego Medium image (which contains the encrypted message) in the buffer.

Step.2: Take password as input and store it in string variable, *ascpwd*, in ASCII format.

Step.3: Extract the 6th double byte code of the 1st pixel of the loaded image. Compare it with the 1st character of *ascpwd*. If they do not match, print "incorrect password" and exit algorithm.

Step.4: Extract the 5th double byte code of the next pixel of the loaded image. Compare it with the next character of *ascpwd*. If they do not match, print "incorrect password" and exit algorithm". Set Flag=1.

Step.5: Extract the 6th double byte code of the next pixel of the loaded image. Compare it with the next character of *ascpwd*. If they do not match, print "incorrect password" and exit algorithm". Set Flag=0.

Step.6: Repeat steps 4 and 5 until the last character of *ascpwd* is compared.

Step.7: If Flag=1, extract the 6th double byte code of the next pixel of the loaded image. Concatenate it with string variable *msglen*. Set Flag=0.

Step.8: If Flag=0, extract the 5th double byte code of the next pixel of the loaded image. Concatenate it with *msglen*. Set Flag=1.

Step.9: Repeat steps 7 and 8 while at any point the extracted code is not equal to "F".

Step.10: Set a counter c=0.

Step.11: If Flag=1, extract the 6th double byte code of the next pixel of the loaded image. Concatenate it with string variable *msgcode*. Set Flag=0. Increment counter c.

Step.12: If Flag=0, extract the 5th double byte code of the next pixel of the loaded image. Concatenate it with string variable *msgcode*. Set Flag=1. Increment counter c.

Step.13: Repeat steps 11 and 12 until c= *msglen*.

Step.14: Convert the *msgcode* to standard characters and save the result in string variable msg.

Step.15: Print *msg*.

IV. DISCUSSION

A. *Verification and Validation*

Here we show the operation of our application and test it by checking whether the key (password) actually works or not.

Encrypting the message in an image file:

We put the following message in a text file by the name of docu.txt, “This is a secret message”. Next we encrypt this text inside an image file—123.png with a password. The new image with the message encrypted is generated and we name this image file 123new.png.

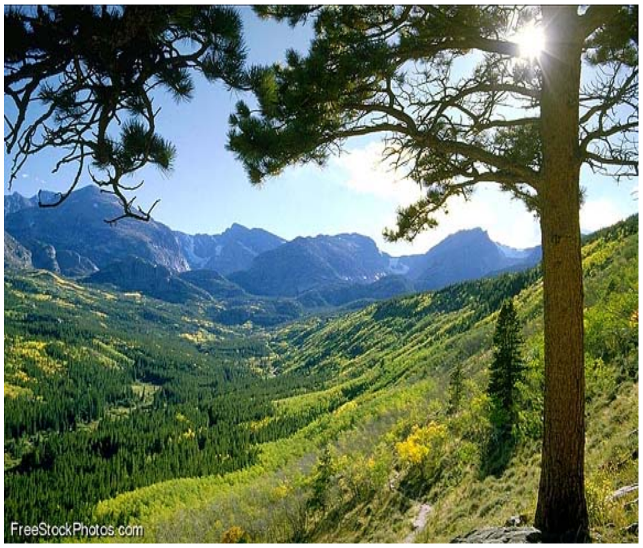


Figure 1: Original Image – cover medium image (123.png)

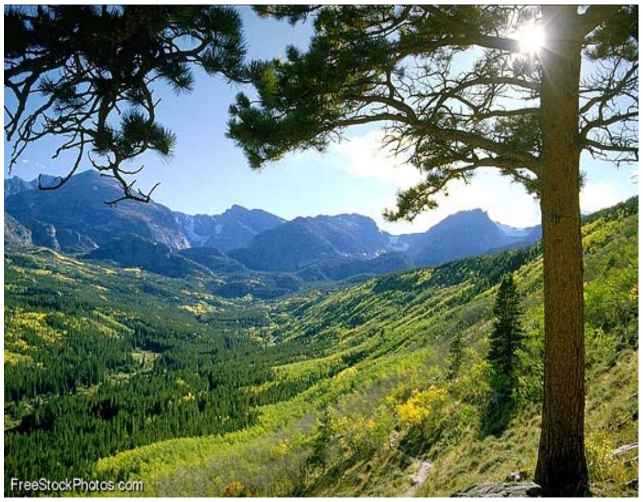
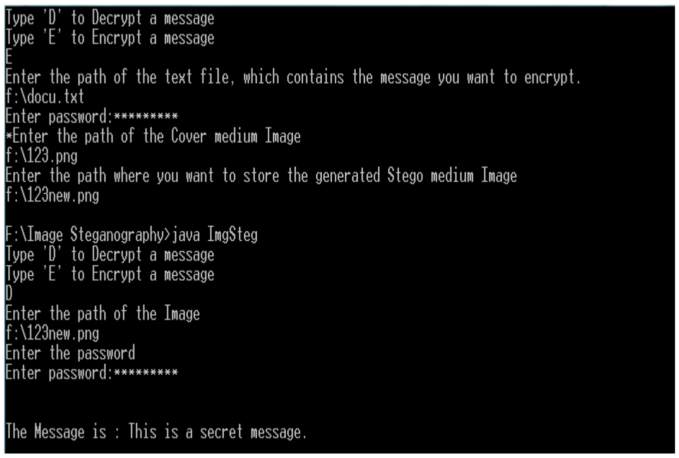


Figure 2: Image with hidden data- Stego medium image (123new.png)

Decrypting the message from the newly generated image:

In our application, we enter the path of the new image (123new.png). Then we enter the password that had been used to encrypt the message. A successful match of the password would return the hidden text message on the command prompt.

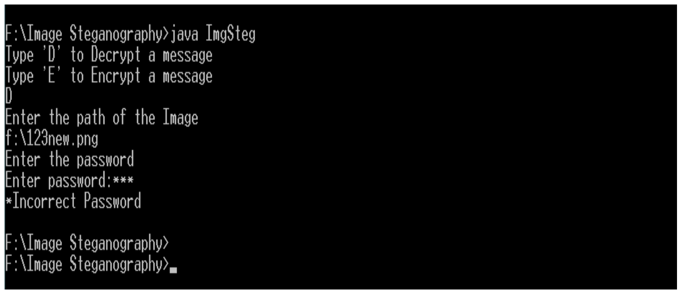


```
Type 'D' to Decrypt a message
Type 'E' to Encrypt a message
E
Enter the path of the text file, which contains the message you want to encrypt.
f:\docu.txt
Enter password:*********
*Enter the path of the Cover medium Image
f:\123.png
Enter the path where you want to store the generated Stego medium Image
f:\123new.png

F:\Image Steganography>java ImgSteg
Type 'D' to Decrypt a message
Type 'E' to Encrypt a message
D
Enter the path of the Image
f:\123new.png
Enter the password
Enter password:*********


The Message is : This is a secret message.
```

Figure 3: Snapshot of the working of our application

As can be seen the images, the cover medium image and the stego medium image (Figure 1 and Figure 2 respectively) are visually identical and hence successfully avoid any suspicion from potential attackers.

Now we try to use this application to retrieve the message from the same image (123new.png) by deliberately using an incorrect password. The result is displayed in the snapshot below.



```
F:\Image Steganography>java ImgSteg
Type 'D' to Decrypt a message
Type 'E' to Encrypt a message
D
Enter the path of the Image
f:\123new.png
Enter the password
Enter password:***
*Incorrect Password

F:\Image Steganography>
F:\Image Steganography>_
```

Figure 4: Application rejects an incorrect password

Our application successfully rejects the password and denies retrieval of the encrypted message. Hence it becomes vital in the whole process to protect the password from any person who’s not supposed to know about the message being passed.

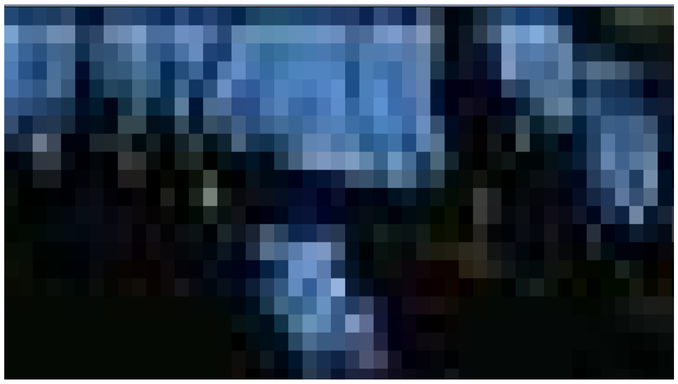


Figure 5: Original Image (Zoomed)

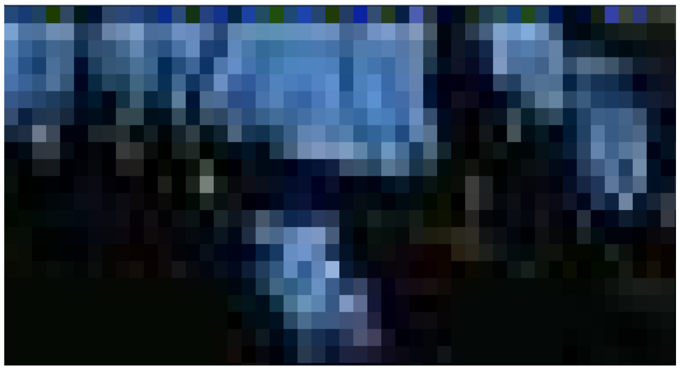


Figure 6: Image with hidden data (Zoomed)

B. Project Assessment and Future Developments

The project application has been built using Java and hence carries along with it all the advantageous features of Java such as platform independence and security. Besides, the application also has a very short runtime. This is due to the fact that the Java compiler first compiles the program code into bytecode (an optimized set of instructions designed to be executed by the Java run-time system), which is then executed by the java run-time system (the Java Virtual Machine) with the help of the Just in Time Compiler [2]. The Just in Time compiler only compiles a selected portion of the code on a demand-basis instead of compiling the entire code unnecessarily. This saves a lot of time in program execution.

The program algorithm also contributes to the minimization of program runtime. As mentioned earlier, our program algorithm uses a symmetric key algorithm, which is much less computationally complex as compared to asymmetric key algorithms. Typically, symmetric key algorithms are hundreds to thousands times faster than asymmetric key algorithms. However, an issue of key management arises in using a symmetric key algorithm. Here, the security of the message is based on one shared key. Hence distribution of the shared key becomes a serious issue, especially if the content of the message is to be shared by a large number of users. To overcome this problem a hybrid cryptosystem, such as pretty good privacy (PGP), could be implemented in a future version of the application. Hybrid cryptosystems combine symmetric and asymmetric key algorithms and hence carry the advantages of both the encryption systems.

At the moment, our application only works with PNG image files. Hence, the functionality of our project could be improved by enabling it to work with more popular image files such as JPG.

V. CONCLUSION

Steganography is a widely studied and highly researched field today as it bears significant importance in many key areas, such as digital forensics, network security, copyright protection schemes, crime, etc. This paper illustrates a software project which utilizes a Steganographic technique to encrypt data into image files, without deteriorating the image with respect to its view at plain sight, and consequently also decodes the data successfully on entering the correct key.

VI. REFERENCES

1. Multimedia security - Steganography and digital Watermarking techniques for protection of intellectual property by Chun Shien Lu Encryption Threat
2. The Complete Reference – Java by Herbert Schildt Seventh Edition
3. “Steganography”
URL: http://en.wikipedia.org/wiki/Steganography *(accessed on 07.01.2009)*
4. “Steganalysis”
URL: http://en.wikipedia.org/wiki/Steganalysis *(accessed on 07.01.2009)*
5. “Digital Watermarking”
URL: http://en.wikipedia.org/wiki/Digital_watermarking *(accessed on 07.01.2009)*
6. “Steganography”
URL: http://www.jjtc.com/Steganography/ *(accessed on 07.01.2009)*
7. “Printer Steganography”
URL: http://en.wikipedia.org/wiki/Printer_steganography *(accessed on 09.01.2009)*
8. “Steganography in wireless application protocol”
URL: http://portal.acm.org/citation.cfm?id=1169183 *(accessed on 09.01.2009)*
9. “Learning Java 2D”
URL: http://java.sun.com/developer/technicalArticles/GUI/java2d/java2dpart1.html *(accessed on 15.01.2009)*
and
http://java.sun.com/developer/technicalArticles/GUI/java2d/java2dpart2.html *(accessed on 15.01.2009)*
10. “SampleModel Class”
*URL:*http://java.sun.com/j2se/1.5.0/docs/api/java/awt/image/SampleModel.html *(accessed on 20.01.2009)*
11. “BufferedImage Class”
*URL:*http://java.sun.com/j2se/1.4.2/docs/api/java/awt/image/BufferedImage.html *(accessed on 20.01.2009)*
12. “Applications of Steganography”
URL: www.datahide.com/BPCSe/applications-e.html *(accessed on 09.02.2009)*
13. “Writing/Saving an Image”
*URL:*http://java.sun.com/docs/books/tutorial/2d/images/saveimage.html *(accessed on 11.02.2009)*
14. Digital Watermarking and Steganography 2nd Edition by Morgan Kaufmann
15. Data Communications and Networking 4th Edition by Behrouz A Forouzan
16. Armin Bahramshahry, Hesam Ghasemi, Anish Mitra, and Vinayak Morada, “Design of a Data Hiding Application Using Steganography” April 2007
17. “Cryptanalysis” http://en.wikipedia.org/wiki/Cryptanalysis
18. “Cryptography” http://en.wikipedia.org/wiki/Cryptanalysis