A Study on Steganography Techniques

Deepika Bansal[1], Rita Chhikara[2]
[1,2]Department of Computer Science and Engineering,
ITM University, Gurgaon, Haryana, India

***Abstract* - Steganography is the art and science of hiding the secret data in the other file formats for ex. image, audio, video, text, etc.. In this paper we are considering the image steganography. We present a study carried out to discuss the various steganography tools. The analysis of cover and stego images is performed on the basis of two statistical analysis methods, peak Signal-to-Noise Ratio and histogram analysis. On performing the analysis we aimed at introducing a robust and high payload Steganographic algorithm.**

1. INTRODUCTION

Steganography is the art and science of hiding the existence of the communication, i.e., it hides the secret message inside the other medium like images, audio, video, text, etc. Steganography word is derived from Greek word steganos, which means covered writing and graphia means writing[1]. To embed the data in any medium requires two files. The first one is the cover file and the second one is secret message. The secret message can be any plain text, cipher text, or image. After embedding secret message in the cover file we obtains a stego- file. The existence of secret message in the stego file cannot be predicted.

Cover Image + Message = Stego Image

There are various steganographic techniques used to hide the secret message. Throughout the history, various steganography techniques were being used, for example wax covered tablets, hidden tattoos, invisible inks, microfilms, microdots, null ciphers, etc[2**]**. There are two most widely used image steganography techniques:(i) Spatial Domain & (ii) Tranform domain, shown in fig.1.

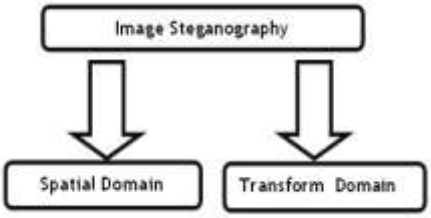

Fig.1 Image Steganography Techniques

Spatial domain technique embeds secret bits directly in the cover file. The commonly used spatial domain technique is Least Significant Bit Insertion (LSB). In LSB, the secret bits are inserted in the least significant bits of cover image. LSB is of 2 types [3]: LSB Replacement & LSB Matching. In the LSB Replacement, the least significant bit of the carrier is replaced by the message bit directly. But in LSB Matching, if the least significant bit of the cover pixel is same as the message bit then it remains unchanged, otherwise it is randomly incremented or decremented by one.

The algorithm to embed the text message using LSB technique[4] :

1: Read the cover image and text message which is to be hidden in the cover image.
2: Convert text message in binary.
3: Calculate LSB of each pixels of cover image.
Step 4: Replace LSB of cover image with each bit of secret message one by one.
5: Write stego image.

The algorithm to retrieve text message:-

1: Read the stego image.
2: Calculate LSB of each pixels of stego image.
3: Retrieve bits and convert each 8 bit into character.

Transform domain[5] hides the secret bits in significant parts of the cover file. The transform domain techniques include Discrete Cosine Transform (DCT), Discrete Wavelet Transform (DWT) and Discrete Fourier Transform (FFT) (iii) Spread spectrum technique[6]. In DCT technique, for each color component the JPEG image format uses a *discrete cosine transform* to transform successive 8 x 8 pixel blocks of the image into 64 DCT coefficients each. The DCT coefficients *F(u,v)* of an 8 x 8 block of image pixels *f(x, y)* are given by [7]

$$F(u,v) = \frac{1}{4} C(u)C(v) \left[\sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2x+1)v\pi}{16} \right]$$

Where,

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & , \text{if } u \leq 0 \\ 1 & , \text{if } u > 0 \end{cases}$$

DWT technique [8] is to store the secret data in the least important coefficients of each 4X4 Haar transformed blocks. The cover image decomposition represented by approximation and detail coefficients[9] is depicted on Fig.2 .

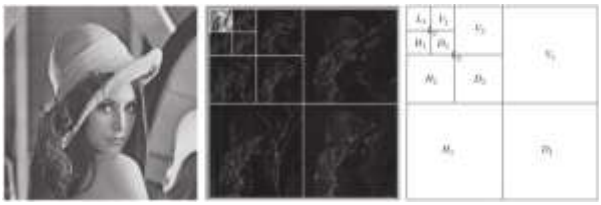

Fig.2 Image Decomposition by DWT

The DFT [10] transforms time- or space-based data into frequency-based data. The DFT of a vector *x* of length *n* is another vector *y* of length *n*:

$$y_{p+1} = \sum_{j=0}^{n-1} \omega^{jp} x_{j+1}$$

where ω is a complex *n*th root of unity:

$$\omega = e^{-2\pi i / n}$$

In the spread spectrum technique, the message is spread over a wide frequency bandwidth than the minimum required bandwidth to send the information.

2. STEGANOGRAPHY TOOLS

A. JPHide and JPSeek

JP Hide & Seek (JPHS)[11] designed by Allan Latham, is a Windows program to hide a data file in a jpeg file and to recover a file hidden in the jpeg file. JPHS uses least significant bit overwriting of the discrete cosine transform coefficients used by the JPEG algorithm. It uses Blowfish Crypto algorithm for lsb randomization and encryption to determine where to store the bits of the hidden file. In [12], the author has described all the steps for testing the JPHS program for Windows. It can be detected by X^2-test & Stegdetect.

B. S-Tools

S-Tools is a program designed by Andrew Brown. It can hide the information inside GIF, BMP and WAV files. It uses least significant bit randomization. For encryption of the data, Data Encryption Standard (DES), International data Encryption Algorithm (IDEA), Message Digest Cipher (MDC) or Triple-DES are used. In [13] author had showed a signal level comparison between a WAV carrier file before and after the airport map was hidden. The original WAV file is 178,544 bytes in length, whereas the steganography WAV file is 178,298 bytes in length. S-Tools reduce the number of colors in the image to only 32 colors. [14] shows the demo for using S-Tools program. The system interface is easy to use. It supports a drag and drop method to load images. Once the cover image is dragged in; the system will advise the user on how much data in bytes the image can hold. The presence of message can be detected by X^2-test.

C. Hide & Seek

Hide & Seek designed by Colin Maroney uses the least significant bit of each pixel to encode characters. It uses the GIF images for hiding the secret messages. The IDEA cipher is used for the encryption of the message[15]. The image of Shakespeare was used in [16] to show the limitation of Hide & Seek tool of padding the black area around the stego image if the cover image is less than the minimum size of 320 x 480 pixels. Stego-images will have different properties depending on the version of Hide and Seek used. The DOS command for the Hide and Seek software is as follows:

- hide <infile.ext> <Cover.gif> [key]
- seek <Stego.gif> <outfile.ext> [key]

D. Stella

Stella word originates from The STeganography ExLoration Lab[17]. Using Stella it is possible to embed and extract an additional message with different steganography techniques. Stella can handle the gif, bmp & jpg images. Its embedding process exploits the visually low prioritised chrominance channels; the YUV-colour system is used. It can hide an arbitrary additional message of limited length. The private key is used for the privacy of the embedded message. The embedding algorithm[18] considers only one channel and works as follows:

1. Consider the chrominance value of a given pixel.
2. Read a bit from secret message.
3. To embed a "0", decrease the chrominance value of the pixel by one.
4. To embed a "1", increase the chrominance value of the pixel by one.
5. Go to the next pixel.

E. Hide In Picture

Hide In Picture uses bmp image format for hiding the secret imformation using the password. HIP is designed by Davi Tassinari de Figueiredo in 2002. The secret message is hidden in the least significant bits of each byte of the image. If the file to be hidden is large, then more than a single bit is modified. The HIP header (containing information for the hidden file, such as its size and filename) and the file to be hidden are encrypted with an encryption algorithm, using the password given, before being written in the picture. Their bits are not written in a linear fashion; HIP uses a pseudo-random number generator to choose the place to write each bit. The values given by the pseudo-random number generator depend on your password, so it is not possible for someone trying to read your secret data to get the hidden file (not even the encrypted version) without knowing the password. According to the comparative analysis done in [15], Hide-In-Picture earned a good PSNR value but the cover image was distorted a bit after the embedding.

F. Revelation

Revelation was created by Sean Hamlin. It uses the least significant bit matching. It leaves a gray value not altered if its LSB matches the bit to be hidden, otherwise a colour indexed as *2i* will be changed to *2i+1* if the embedded bit is 1, or *2i+1* is shifted back to *2i* in case of embedding a 0. Revelation program is coded in java and developed in the Eclipse IDE. In [15] the author has shown that the Revelation tool seems to do a good job in hiding any visual tamper on the cover image, but the histogram of its generated output reveals some traces.

G. Steghide

Steghide is designed by Stefan Hetzl and available at [19].It is able to hide data invarious kinds of image & audio files. The bmp, jpg, wav & au file formats are supported by Steghide. Steghide uses a graph-theoretic approach to steganography. The embedding algorithm roughly works as follows:

1. The secret data is compressed and encrypted.
2. Then a sequence of positions of pixels in the cover file is created based on a pseudo-random number generator initialized with the passphrase.
3. Of these positions those that do not need to be changed (because they already contain the correct value by chance) are sorted out.
4. Then a graph-theoretic matching algorithm finds pairs of positions such that exchanging their values has the effect of embedding the corresponding part of the secret data. If the algorithm cannot find anymore such pairs all exchanges are actually performed.
5. The pixels at the remaining positions (the positions that are not part of such a pair) are also modified to contain the embedded data (but this is done by overwriting them, not by exchanging them with other pixels).
6. The fact that (most of) the embedding is done by exchanging pixel values implies that the first-order statistics (i.e. the number of times a color occurs in the picture) is not changed

For audio files the algorithm is the same, except that audio samples are used instead of pixels. The default encryption algorithm is Rijndael with a key size of 128 bits (which is AES – the advanced encryption standard) in the cipher block chaining mode. The checksum is calculated using the CRC32 algorithm. Steghide is a windows command line tool also available for unix/linux platforms. The author in [20] has showed the usage of Steghide tool. To embed the mysecret.txt file in our cover image, cover.jpg, producing stego file stego.jpg, issue the command:

- steghide embed –pf mysecret.txt –cf cover.jpg –sf stego.jpg

To extract our secret, we use the command:

- steghide extract –sf stego.jpg

H. nsF5 (no-shrinkage F5)

The steganography algorithm nsF5 (no-shrinkage F5) was introduced in 2007 as an improved version of F5 [21]. The F5 algorithm contains two important design principles. The first one is the character of its embedding modifications chosen in such a way that the absolute value of the DCT coefficient is always decreased by one. F5 only embeds into non-zero AC DCT coefficients. If a coefficient becomes zero after embedding, which can only happen for coefficients equal to 1 or -1, so called shrinkage occurs and the same time bit is reembedded at the next coefficient. The second important element of F5 is its incorporating matrix embedding using binary Hamming codes. Matrix embedding enables embedding more bits per one embedding change and thus increases embedding efficiency. Although the embedding efficiency of F5 is low because of the additional changes introduced by shrinkage. Therefore to eliminate the shrinkage wet paper codes are applied. Wet paper codes were designed to allow the sender to use side information unavailable to the decoder. The negative effect of shrinkage is alleviated in nsF5 by using wet paper codes [22]. The theoretical bound on the embedding efficiency of the nsF5 algorithm is equal to

$$e = \frac{\alpha}{H^{-1}(\alpha)}$$

where alpha is the relative payload with respects to the number of changeable coefficients and H^{-1} stands for the inverse of the binary entropy function.

I.F5

F5 withstands visual and statistical attacks offering a large steganographic capacity. F5 implements matrix encoding to improve the efficiency of embedding. F5 employs permutative straddling to uniformly spread out the changes over the whole steganogram. It shuffles all coefficients using a permutation first. Then, F5 embeds into the permuted sequence[23]. The algorithm F5 has the following coarse structure:

1. Start JPEG compression. Stop after the quantisation of coefficients.
2. Initialise a cryptographically strong random number generator with the key derived from the password.
3. Instantiate a permutation (two parameters: random generator and number of coefficients).
4. Determine the parameter k from the capacity of the carrier medium, and the length of the secret message.
5. Calculate the code word length $n = 2k - 1$.
6. Embed the secret message with $(1, n, k)$ matrix encoding.

(a) Fill a buffer with n nonzero coefficients.
(b) Hash this buffer (generate a hash value with k bit-places).
(c) Add the next k bits of the message to the hash value (bit by bit, xor).
(d) If the sum is 0, the buffer is left unchanged. Otherwise the sum is the buffer's index $1 \ldots n$, the absolute value of its element has to be decremented.
(e) Test for shrinkage, i. e. whether we produced a zero. If so, adjust the buffer (eliminate the 0 by reading one more nonzero coefficient, i. e. repeat step 6a beginning from the same coefficient). If no shrinkage occurred, advance to new c coefficients behind the actual buffer. If there is still message data continue with step 6a.

7. Continue JPEG compression (Huffman coding etc.).

J. Perturbed Quantization

In Perturbed Quantization[24], the sender hides data while processing the cover object with an information reducing operation that involves quantization, such as lossy compression, down sampling, or A/D conversion. The unquantized values of the processed cover object are considered as side information to confine the embedding changes to those unquantized elements whose values are close to the middle of quantization intervals. This choice of the selection channel calls for wet paper codes as they enable communication with non shared selection channel.

Perturbed Quantization aims to achieve high efficiency, with minimal distortion, rather than a large capacity. Each coefficient in the DCT block is assigned a scalar value that corresponds to how much impact it would make to the carrier image, and then a steganographer can set a selection rule to filter out the "well behaved" coefficients[25].

K. Outguess

Outguess[26] is a universal steganographic tool that allows the insertion of hidden information into the redundant bits of data sources. The nature of the data source is irrelevant to the core of Outguess[27]. The program relies on data specific handlers that will extract redundant bits and write them back after modification. It improves the encoding step by using a pseudo-random number generator to select DCT coefficients at random. The least-significant bit of a selected DCT coefficient is replaced with encrypted message data. The χ^2-test for JSteg does not detect data that is randomly distributed across the redundant data and, for that reason, it cannot find steganographic content hidden by OutGuess 0.1. However, it is possible to extend the χ2- test to be more sensitive to local distortions in an image. Two identical distributions produce about the same χ2 values in any part of the distribution. Instead of increasing the sample size and applying the test at a constant position, we use a constant sample size but slide the position where the samples are taken over the image's entire range[28]. The Outguess algorithm is as follows:

Input: message, shared secret, cover image
Output: stego image
initialize PRNG with shared secret
while data left to embed **do**
    get pseudo-random DCT coefficient from cover image
    if DCT ≠0 and DCT ≠1 **then**
    get next LSB from message
    replace DCT LSB with message LSB
    end if
    insert DCT into stego image
end while

L. JSteg

JSteg[29] is designed by Derek Upham. JSteg[30] was the first publicly available steganographic system for JPEG images. Its embedding algorithm sequentially replaces the least-significant bit of DCT coefficients with the message's data. The algorithm does not require a shared secret; as a result, anyone who knows the steganographic system can retrieve the message hidden by JSteg. Andreas Westfeld and Andreas Pfitzmann noticed that steganographic systems that change least-significant bits sequentially cause distortions detectable by steganalysis. They observed that for a given image, the embedding of high-entropy data (often due to encryption) changed the histogram of color frequencies in a predictable way. The JSteg algorithm is as follows:

Input: message, cover image
Output: stego image
while data left to embed **do**
    get next DCT coefficient from cover image
    if DCT ≠0 and DCT ≠1 **then**
        get next LSB from message
        replace DCT LSB with message LSB
    end if
    insert DCT into stego image
end while

M. Stegodos

StegoDos is also known as Black Wolf's Picture Encoder version 0.90a [31]. This is Public Domain software written by Black Wolf (anonymous). This is a series of DOS programs that require far too much effort for the results. It will only work with 320x200 images with 256 colors.
To encode a message, one must:
1. Run GETSCR. This starts a TSR which will perform a screen capture when PRINTSCREEN is pressed.
2. View the image with a third-party image viewing software (not included with StegoDos) and press PRINTSCREEN to save the image in MESSAGE.SCR.
3. Save your message to be embedded in the image as MESSAGE.DAT.
4. Run ENCODE. This will merge MESSAGE.DAT with MESSAGE.SCR.
5. Use a third party screen capturing program (not included with StegoDos) to capture the new image from the screen.
6. Run PUTSCR and capture the image displayed on the screen. Decoding the message is not as involved but still requires a third party program to view the image.

To decode a message:
1. Run GETSCR. This starts a TSR which will perform a screen capture when PRINTSCREEN is pressed.
2. View the image containing a message with a third-party image viewing software (not included with StegoDos) and press PRINTSCREEN to save the image in MESSAGE.SCR.
3. Run DECODE. This will extract the stored message from MESSAGE.SCR.

N. White Noise Storm

White Noise Storm is designed by Ray Arachelian [32]. White Noise Storm is based on spread spectrum technology and frequency hopping, which scatters the message throughout the image. Instead of having *x* channels of communication that are changed with a fixed formula and passkey, White Noise Storm spreads eight channels within a random number generated by the previous window size and data channel. Each channel represents 1 bit, so each image window holds 1 byte of information and many unused bits. These channels rotate, swap, and interlace among themselves to yield a different bit permutation. For instance, bit 1 might be swapped with bit 7, or both bits may rotate one position to the right. The rules for swapping are dictated by the stego-key and by the previous window's random data (similar to DES block encryption). Scattering and encryption helps protect against hidden message extraction but not against message destruction through image processing. A scattered message in the image's LSBs is still as vulnerable to destruction from lossy compression and

image processing as is a clear-text message inserted in the LSBs [23].The main disadvantage of applying the WNS encryption method to steganography is the loss of many bits that can be used to hold information. Relatively large files must be used to hold the same amount of information other methods provide.

O. YASS

YASS is Yet Another Steganographic Scheme)[24], a method based on embedding data in randomized locations so as to disable the self calibration process (such as, by cropping a few pixel rows and/or columns to estimate the cover image features) popularly used by blind steganalysis schemes. YASS can successfully resist recent blind steganalysis methods, in addition to surviving distortion constrained attacks. The errors induced in the embedded data due to the fact that the stego signal must be advertised in a specific format such as JPEG, are dealt with by the use of erasure and error correcting codes. For the presented JPEG steganographic scheme, it is shown that the detection rates of recent blind steganalysis schemes are close to random guessing, thus confirming the practical applicability of the proposed technique. We also note that the presented steganography framework, of hiding in randomized locations and using a coding framework to deal with errors, is quite simple yet very generalizable.

The original algorithm can be summarized using the following five steps[25]:

1. The entire message is encoded using Repeat-Accumulate (RA) error correction code.
2. The cover image in its spatial-domain representation is divided into big blocks of B × B pixels, B > 8.
3. In each big block, an 8 × 8 block is pseudo-randomly selected using a secret key.
4. For every such selected 8 × 8 block, the embedding follows the algorithm originally described in:

(a) The block is transformed using a two-dimensional DCT.

(b) Every real-valued DCT coefficient is divided by the corresponding quantization step from a predefined quantization table (corresponding to the hiding quality factor QFh). No rounding of coefficients is performed at this stage.

(c) A fragment of the encoded message is embedded in a predetermined band of 19 low-frequency AC DCT coefficients using QIM, while skipping all coefficients that quantize to zero by the JPEG quantizer.

(d) The block is decompressed back to the spatial domain and put in its original position within the big block.

5. Finally, the image is compressed using JPEG with the advertising quality factor QFa to obtain the stego image. The extraction algorithm first decompresses the stego JPEG image to the spatial domain, identifies the same 8 × 8 blocks within the big blocks as during the embedding, extracts the encoded (noisy) message bits, concatenates them, and finally applies the RA error-correcting algorithm to extract the secret message. In [26], the authors introduced the following two extensddions of YASS whose primary purpose was to increase the embedding capacity:

1. Mixture-based QFh approach. Here, the hiding quality factor QFh varies across the 8 × 8 blocks either randomly or adaptively based on the block variance or the coefficient count.
2. Attack-aware iterative embedding. The embedding process is repeated several times with the hope to obtain a lower error rate, which, in turn, would increase the embedding capacity.

3. COMPARATIVE ANALYSIS & RESULTS

In this section, all the steganography tools discussed above are categorized in Table 1, on the basis image format used, the hiding technique used & the encryption algorithm used.

Tool	**Image Format**	**Hiding Technique**	**Encryption Algorithm**
JP Hide & Seek	JPEG	DCT	Blowfish
S-Tools	GIF, BMP	LSB	DES, IDEA, MDC, Triple-DES
Hide & Seek	GIF	LSB	IDEA
Stella	BMP, GIF, JPEG	Chrominance Value	Private Key
Hide in Picture	BMP	LSB	Using Password
Revelation	BMP	LSB	-
Steghide	BMP, JPEG	LSB	AES
nsF5	JPEG	DCT	-
F5	JPEG	DCT	Huffmann Coding
Perturbed Quantization	PNM, JPEG	DCT	-
Outguess	JPG	DCT	-
JSteg	JPEG	DCT	No Encryption
Stegodos	GIF	LSB	-
White Noise Storm	PCX	LSB	DES
YASS	JPEG	DCT	Repeat Accumulative Error Correcting Algorithm

Table 1 Image Steganography Tools

3.1 Statistical Analysis

There are two main types of statistical analysis methods investigated in this paper for comparative analysis. These are the peak signal-to-noise ratio and image histograms. These are discussed below:

(a) Peak Signal-to-noise ratio

Peak Signal-to-noise ratio is the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. As a performance measurement for image distortion, the well known Peak-Signal-to-Noise Ratio (PSNR) which is classified under the difference distortion metrics can be applied on the stego images. It is defined as:

$$PSNR = 10log_{10}(\frac{C_{max}^2}{MSE})$$

The mean square error (MSE) for an N _ N gray-level image is defined as follows:

$$MSE = (\frac{1}{N})^2 \sum_{i=1}^{N}\sum_{j=1}^{N}(x_{ij} - \overline{x_{ij}})^2$$

Here x_{ij} denotes the original pixel value, and $\overline{x_{ij}}$ denotes the decoded pixel value.

The Table 2 shows different PSNR values obtained using cover and stego images on applying the different 5 steganography tools choosen from the tools discussed in Section 2. The cover image and stego images obtained using various steganography algorithms are shown below:

Fig. 3 Cover Image

Fig. 4 F5 Stego Image

Fig. 5 JPHS Stego Image

Fig. 6 nsF5 Stego Image

Fig. 7 Outguess Stego Image

Fig. 8 PQ Stego Image

Fig. 9 S-Tools Stego Image

Tools	PSNR (in db)
F5	33.18
JPHS	46.82
nsf5	51.37
Outguess	50.55
S-Tools	52.87

Table 2 PSNR values obtained from different stego tools.

(b) Image Histogram

An image histogram is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance. In this study we can trace any abnormalities in the Stego image's histogram. The histogram of cover image and various stego images is shown below:

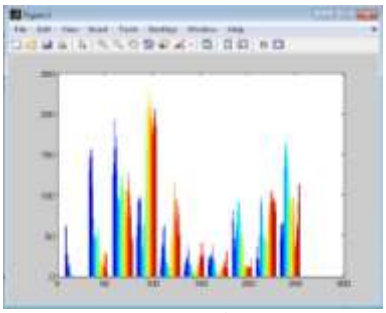

Fig. 10 Histogram of cover image

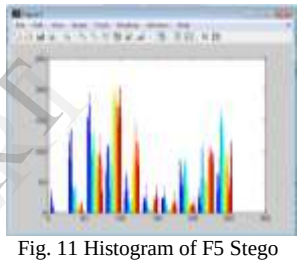

Fig. 11 Histogram of F5 Stego Image

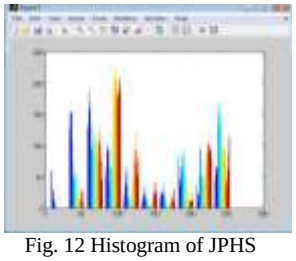

Fig. 12 Histogram of JPHS Stego Image

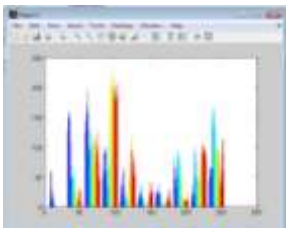

Fig. 13 Histogram of nsf5 Stego Image

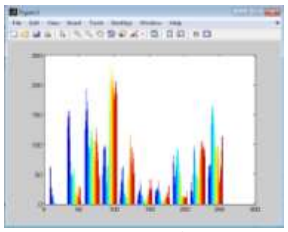

Fig.14 Histogram of Outguess Stego Image

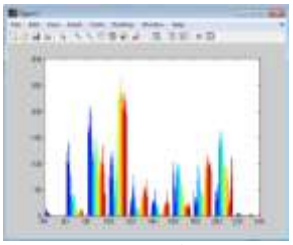

Fig. 15 Histogram of PQ Stego Image

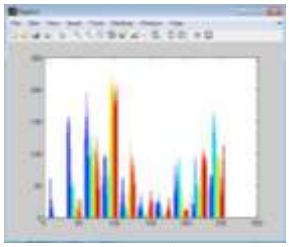

Fig. 16Histogram of S-Tools Stego Image

3.2 Results

According to the statistical analysis, we can see that S-Tools algorithm has the highest performance and its software provides a better graphical interface in the LSB domain. Among the DCT technique, nsF5 shows a high PSNR value (high quality of image). On analyzing the histogram of

cover image & all the stego images we can find that there are no visual differences in histogram of nsF5 and Outguess, very slight differences can be noticed in the histogram of JPHS & S-Tools, and noticeable differences are present in the histogram of F5 and PQ.

4. CONCLUSION

Steganography is the art of hiding secret messages in the other file formats. The Image steganography is discussed in this paper. The author has discussed various steganography tools available for embedding the secret data in image. After analyzing various tools, 6 different tools are used to perform the analysis of images using 2 statistical analysis methods i.e., Peak Signal-to-Noise Ratio and Histogram analysis. From the PSNR values obtained we can conclude that S-Tools give a high quality stego image after embedding the text file in the lsb domain and nsF5 gives high quality image among the dct domain tools. On the basis of histogram analysis we can conclude that there are no visual differences in the cover and stego image of nsF5 and Outguess. So, we can say that S-Tools, Outguess and nsF5 are more robust to steganalytic attacks.

5. REFERENCES

1. N.F.Johnson,S.Jajodia, Exploring steganography: Seeing the Unseen, IEEE Computer 31 (2) (1998) 26-34.
2. Kh. Manglem Singh, S.Birendra Sigh and L. Shyam Sundar Singh, Hiding Encrypted Message in the Features of Images, IJCSNS, Vol.7 No. 4, April 2007, pp 302-307.
3. W.-N. Lie and L.-C. Chang, Data hiding in images with adaptive numbers of least significant bits based on human visual system, in Proc.,IEEE Int. Conf. Image Processing, 1999, Page(s): 286–290.
4. K.B.Shiva Kumar, K.B. Raja, R.K.Chhotaray, Sabyasachi Pattnaik, "Coherent Steganography using Segmentation and DCT", IEEE-978-1-4244-5967-4/10/$26.00 ©2010.
5. S. Katzenbeisser and F. A. P. Petitcolas, Information Hiding Techniques for Steganography and Digital Watermarking. Norwood, MA: Artech House, 2000.
6. L. M. Marvel, C. G. Boncelet Jr., and C. T. Retter, Spread spectrum image steganography, IEEE Trans. Image Process., vol. 8, no. 8, Aug. 1999, Page(s): 1075–1083.
7. D.R. Denslin Brabin, Dr.V.Sadasivam, QET Based Steganography Technique for JPEG Images.
8. Nadiya P v, B Mohammed lmran, Image Steganography in DWT Domain using Double-stegging with RSA Encryption, 2013 International Conference on Signal Processing, Image Processing and Pattern Recognition [ICSIPR].
9. *Vladimír BÁNOCI, Gabriel BUGÁR, Dušan LEVICKÝ,* A Novel Method of Image Steganography in DWT Domain, 978-1-61284-324-7/11/$26.00 ©2011 IEEE.
10. http://www.mathworks.in/help/matlab/math/discrete-fourier-transform-dft.html#brenxmh.
11. JPHS: http://linux01.gwdg.de/~alatham/stego.html
12. JPHS:http://io.acad.athabascau.ca/~grizzlie/Comp607/programs.htm
13. S-Tools: http://www.garykessler.net/library/fsc_ Stego.html
14. S-Tools: http://www.cs.vu.nl/~ast/books http:/http://www.cs.vu.nl/~ast/books/mos2/zebras.html
15. Abbas Cheddad, Joan Condell, Kevin Curran and Paul McKevitt, A Comparative Analysis of Steganographic Tools
16. Narinder Kehar, Jaspreet Kaur, A Smart Technique: Stegnography, IJCST Vol. 2, Issue 1, March 2011.
17. Stella : http://vcg.informatik.unirostock.de/~sanction/stella/
18. René Rosenbaum, Heidrun Schumann, A steganographic framework for reference colour based encoding and cover image selection. Steghide : http://steghide.sourceforge.net/index.php
19. Erin Michaud, Current Steganography Tools and Methods, GSEC Practical, Version 1.4b April, 2003
20. J. Fridrich, T. Pevný, and J. Kodovský, *Statistically undetectable JPEG steganography: Dead ends, challenges, and opportunities*. In J. Dittmann and J. Fridrich, editors, Proceedings of the 9th ACM Multimedia & Security Workshop, pages 3–14, Dallas, TX, September 20–21, 2007.
21. J. Fridrich, M. Goljan, and D. Soukal, W*et paper codes with improved embedding efficiency*. IEEE Transactions on Information Forensics and Security, 1(1):102–110, 2006.
22. A. Westfeld, High capacity despite better steganalysis (F5 – a steganographic algorithm). In I. S. Moskowitz, editor, Information Hiding, 4th International Workshop, volume 2137 of Lecture Notes in Computer Science, pages 289–302, Pittsburgh, PA, April 25–27, 2001. Springer-Verlag, New York.
23. J. Fridrich, M. Goljan, and D. Soukal, Perturbed quantizationsteganography. ACM Multimedia System Journal, 11(2):98–107,2005.
24. Abbas Cheddad, Joan Condell, kevin Curran, paul Mc Kevitt, Digital image steganography: Survey and analysis of current methods, Signal Processing 90 (2010) pp 727-752.
25. Niels provos, www.outguess.org.
26. Provos, N. Defending Against Statistical Steganalysis. Proc. 10th USENIX Security Symposium. Washington, DC, 2001.
27. N. Provos, P. Honeyman, Hide and seek: an introduction to steganography, IEEE Security and Privacy 1 (3) (2003) 32–44.
28. JSteg : http://zooid.org/~paul/crypto/jsteg/
29. JSteg: http://csis.bits-pilani.ac.in/faculty/murali/netsec-09/seminar/refs/anuroopsrep.pdf
30. Stegodos : http://www.jjtc.com/stegdoc/sec310.html
31. WhiteNoise Storm : http://www.jjtc.com/stegdoc/sec315.html
32. N.F.Johnson, S.Jajodia, Exploring steganography: Seeing the Unseen, IEEE Computer 31 (2) (1998) 26-34.
33. K. Solanki, A. Sarkar, and B. S. Manjunath. YASS: Yet another steganographic scheme that resists blind steganalysis. In T. Furon, F. Cayre, G. Doërr, and P. Bas, editors, Information Hiding, 9th International Workshop, volume 4567 of Lecture Notes in Computer Science, pages 16–31, Saint Malo, France, June 11–13, 2007. Springer-Verlag, New York.
34. Jan Kodovskýa, Tomáš Pevnýb, Jessica Fridricha,Modern Steganalysis Can Detect YASS,
35. A. Sarkar, K. Solanki, and B. S. Manjunath. Further study on YASS: Steganography based on randomized embedding to resist blind steganalysis. In E. J. Delp and P. W. Wong, editors, Proceedings SPIE, Electronic Imaging, Security, Forensics, Steganography, and Watermarking of Multimedia Contents X, volume 6819, pages 16–31, San Jose, CA, January 27–31, 2008.