J3: High Payload Histogram Neutral JPEG Steganography

Mahendra Kumar and Richard Newman
Department of Computer and Information Sciences and Engineering
University of Florida
Gainesville, FL, 32611
Email: {makumar,nemo}@cise.ufl.edu

Abstract

Steganography is the art of secret communication between two parties that not only hides the contents of a message, but does not even reveal the existence of the message. Steganalysis attempts to detect the existence of embedded data in a steganographically altered cover file. Many algorithms have been proposed, but so far each has some weakness that has allowed its effects to be detected, usually through statistical analysis of the image. In this paper, we propose a novel approach to JPEG steganography that provides high embedding capacity with zero-deviant histogram restoration. Our algorithm, named J3, uses stop points in its header structure that allow it to restore the histogram of JPEG coefficients, making it impossible for any first order steganalysis to detect it, in addition to increasing its payload compared to other algorithms. J3 can be used to embed a large amount of data with resistance to visual and first order statistical attacks. As far as we know, there is no existing algorithm that can provide as high an embedding payload with complete histogram restoration.

Index Terms

Steganography, Information Hiding, JPEG Steganography, Steganalysis.

I. Introduction

Steganography is a technique to hide data inside a cover medium in such a way that the existence of any communication itself is undetectable as opposed to cryptography where the existence of secret communication is known to everyone but is indecipherable. The word steganography originally came from a Greek word which means *"concealed writing"*. Steganography has an edge over cryptography because it does not attract any public attention, and the data may be encrypted before being embedded in the cover medium. Hence, it incorporates cryptography with an added benefit of undetectable communication.

In digital media, steganography is quite similar to watermarking but each has a different purpose for the hidden data. While steganography aims at concealing the existence of a message with high data capacity, digital watermarking mainly focusses on the robustness of embedded message rather than capacity or concealment. Since increasing capacity and robustness at the same time is not possible, steganography and watermarking have a different purpose and application in the real world. Steganography can be used to exchange secret information in a undetectable way over a public communication channel, whereas watermarking can be used for copyright protection and tracking legitimate use of a particular software or media file.

Image files are the most common cover medium used for steganography. With resolution n most cases higher than human perception, data can be hidden in the ”noisy” bits or pixels of the image file. Because of the noise, a slight change in the those bits is imperceptible to the human eye, although it might be detected using statistical methods (i.e., *steganalysis*). One of the most common and naive methods of embedding message bits is *LSB replacement* in spatial domain where the bits are encoded in the cover image by replacing the least significant bits of pixels. Other techniques might include spread spectrum and frequency domain manipulation, which have better concealment properties than spatial domain methods. JPEG is the most popular image format used over the Internet and by image acquisition devices, and therefore we use JPEG as our choice for steganography.

Steganalysis of JPEG images is based on statistical properties of the JPEG coefficients, since these are where the embedded data are usually hidden. A popular approach to steganalysis of JPEG images is based on analysis of the histogram of coefficient values in the image. Jsteg, which simply changes the LSB of a coefficient to the value desired for the next embedded data bit [18], can be detected by the effect it has of equalizing adjacent pairs of coefficient values [20]. F5 attempts to retain the general shape of the histogram [21], but can be detected by obtaining an estimate of the original histogram by re-encoding a copy of the spatial cover file offset by four rows and four columns [6]. Outguess [13] and Steghide [8] use statistical restoration schemes to embed data in the LSB coefficients. Outguess uses a threshold which determined the amount of coefficients to be preserved to restore the histogram but can be broken by second order statistical steganalysis [5], [14]. Steghide uses a graph theory approach and swaps the values of coefficients to embed data and is more robust than Outguess.

In this paper, we propose a steganography algorithm, J3, that conceals data inside a JPEG image in such a way that it preserves its first order statistical properties [4] and hence is resistant to chi-square attacks [20]. Our algorithm can restore the histogram of any JPEG image to its original values after embedding data with the added benefit of having a high data capacity of 0.4 to 0.7 bits per non-zero coefficient. It does this by manipulating JPEG coefficients in pairs, reserving enough coefficient pairs to restore the original histogram. Moreover, it can embed data in one or more component of the image depending on the user's choice.

Most of the algorithms existing today are incapable of embedding data if it exceeds the capacity of the image. J3 can embed data to its maximum capacity even if the input data file is larger that its embedding capacity. It does this by splitting the data file over several images to embed the data. This ability along with high capacity comes from the fact that J3 maintains separate header information for each component. The header information gives important details about the embedded data file such as stop points, file length, dynamic header length, etc.

Stop points are a key feature of this algorithm; they are used by the embedding module to determine the index at which the algorithm should stop encoding a particular coefficient pair. Coefficient values are only swapped in pairs to minimize detection. A coefficient with value $(2x+1)$ will only decrease to $2x$ to embed a bit while $2x$ will only increase to $(2x+1)$. Each pair of coefficients is considered independently. Before embedding data in any unused coefficient, the algorithm determines if it can restore the histogram to its original position or not. This is based on the number of unused coefficients in that pair. If during embedding, the algorithm determines that there are only a sufficient number of coefficients are remaining to restore histogram, it will stop encoding that pair and store its index location in the stop point of the header. Since all the stop points can only be known after the embedding process, the header is always encoded last on the embedder side whereas it is decoded first on the extractor side.

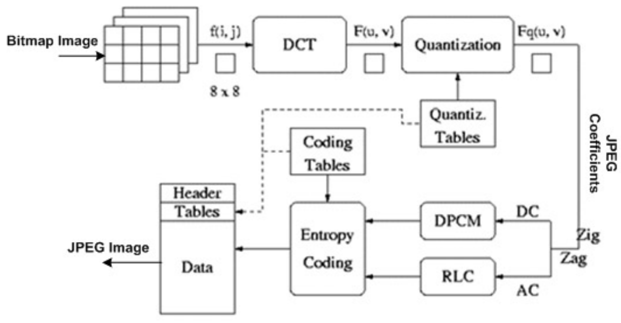
The experimental results show that J3 has a much higher embedding capacity than $F5$, *Outguess* and *Steghide* with the added advantage of complete histogram restoration. We have also estimated the theoretical capacity of the cover medium in section VI and the results follow closely with the actual capacity of the medium.

The rest of the paper is organized as follows. In Section II, we provide some background information on JPEG compression and the LSB embedding technique. Section III deals with some of the related work done in image steganography. In Section IV and V, we discuss our proposed J3 embedding and extraction module in detail while Section VI deals with the theoretical estimation of embedded data capacity and stop point calculation. Section VII shows statistical results obtained using our algorithm and compares it with F5. Finally, section VIII concludes the paper with reference to future work in this area.

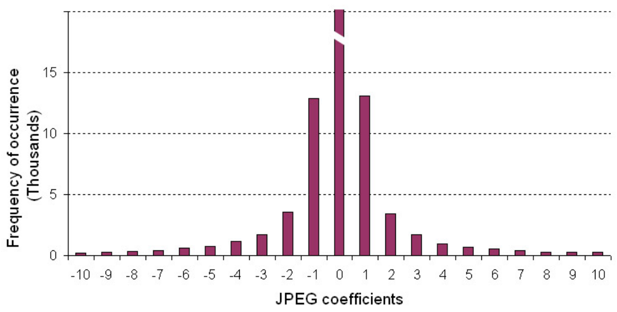
II. Background

A. JPEG Compression

Joint Photographic Expert Group, also know as JPEG, is the most popular and widely used image format for sharing and storing digital images over the Internet or any PC. The popularity of JPEG is due to its high compression ratio with good visual image quality. The file format defined by JPEG stores data in JFIF (JPEG File Interchange Format), which uses lossy compression along with Huffman entropy coding to encode blocks of pixels. Figure 1(a) shows the block diagram to compress a bitmap (BMP) image into JPEG format. First, the algorithm breaks the BMP image into blocks of 8 by 8 pixels. Then, discrete cosine transformation (DCT) is performed on these blocks to convert these pixel values from spatial domain to frequency domain. These coefficients are then quantized using a quantization table which is stored as a part of the JPEG image. This quantization steps is lossy since it rounds up the coefficient values. In the next step, Huffman entropy coding is performed to compress these quantized block of 8 x 8. The histogram in figure 1(b) shows the distribution of JPEG coefficients and their frequency of occurrence. From the histogram, we can conclude that the frequency of occurrence of coefficients decrease with increase in their absolute value. This decrease is almost by a factor of 2. We also deduce that the number of zeros is much larger than any other coefficient value. More details about JPEG compression can be found in reference [9], [10], [19].



(a) Block diagram of JPEG compression.



(b) Histogram of JPEG coefficients, Fq(u,v).

Fig. 1. JPEG encoding and histogram properties.

B. JPEG Steganography

There are two broad categories of image=based steganography that exist today: frequency domain and spatial domain steganography. The first digital image steganography was done in the spatial domain using LSB coding (replacing the least significant bit or bits with embedded data bits). Since JPEG transforms spatial data into the frequency domain where it then employs lossy compression, embedding data in the spatial domain before JPEG compression is likely to introduce too much noise and result in too many errors during decoding of the embedded data when it is returned to the spatial domain. These would be hard to correct using error correction coding. Hence, it was thought that steganography would not be possible with JPEG images because of its lossy characteristics. However, JPEG encoding is divided into lossy and lossless stages. DCT transformation to the frequency domain and quantization stages are lossy, whereas entropy encoding of the quantized DCT coefficients (which we will call the JPEG coefficients to distinguish them from the raw frequency domain coefficients) is lossless compression. Taking advantage of this, researchers have embedded data bits inside the JPEG coefficients before the entropy coding stage.

The most commonly used method to embed a bit is LSB embedding, where the least significant bit of a JPEG coefficient is modified in order to embed one bit of message. Once the required message bits have been embedded, the modified coefficients are compressed using entropy encoding to finally produce the JPEG stego image. By embedding information in JPEG coefficients, it is difficult to detect the presence of any hidden data since the changes are usually not visible to the human eye in the spatial domain. During the extraction process, the JPEG file is entropy decoded to obtain the JPEG coefficients, from which the message bits are extracted from the LSB of each coefficient.

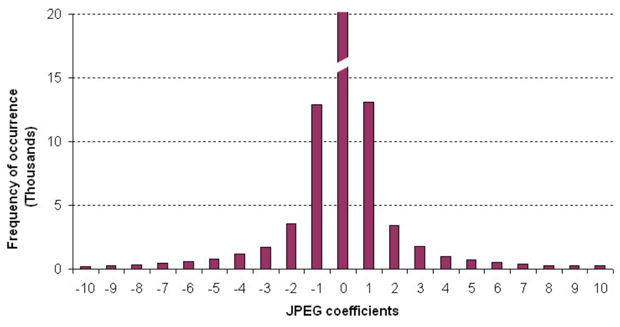
C. LSB-Based Embedding Technique

LSB embedding [22], [2], [11] is the most common technique to embed message bits DCT coefficients. This method has also been used in the spatial domain where the least significant bit value of a pixel is changed to insert a zero or a one. A simple example would be to associate an even coefficient with a zero bit and an odd one with a one bit value. In order to embed a message bit in a pixel or a DCT coefficient, the sender increases or decreases the value of the coefficient/pixel to embed a zero or a one. The receiver then extracts the hidden message bits by reading the coefficients in the same sequence and decoding them in accordance with the encoding technique performed on it. The advantage of LSB embedding is that it has good embedding capacity and the change is usually visually undetectable to the human eye. If all the coefficients are used, it can provide a capacity of almost one bit per coefficients using the frequency domain technique. On the other hand, it can provide an even greater capacity for the spatial domain embedding with almost 1 bit per pixel for each color component. The advantage of spatial domain embedding over frequency domain technique is that it can be easily applied to any raw image format such as a bitmap, and it is less prone to statistical attacks. However, sending a raw image such as a BMP to the receiver would create suspicion in and of itself, unless the image file is very small. Most of the popular formats today are compressed in the frequency domain and therefore it is not a common practice to embed bits directly in the spatial domain. Moreover, robustness techniques cannot be fully exploited in the spatial domain. Hence, frequency domain embeddings are the preferred choice for image steganography.

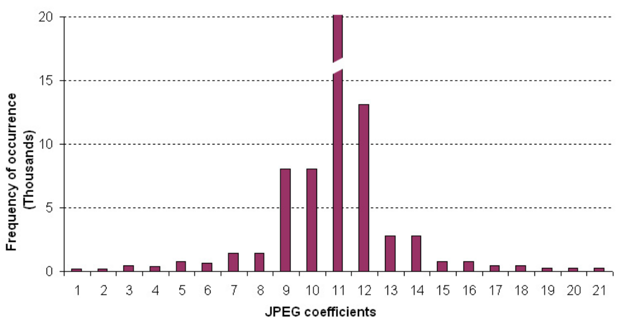
DCT coefficients resemble a typical Gaussian distribution and hence additional noise such as the message bits can be embedded in the low frequency regions without significant change in the quality of image. On the other hand, the disadvantage with this technique is that it is more susceptible to statistical attacks if the distribution curve is changed significantly due to embedding. Other advanced histogram and spread spectrum techniques of LSB embedding have been proposed which are discussed in section III.

III. Previous Work

Jsteg [18] was one of the first JPEG steganography algorithms. It was developed by Derek Upham, and embeds message bits in LSB of the JPEG coefficients. JP Hide&Seek [1] is another JPEG steganography program, improving stealth by using the Blowfish encryption algorithm to randomize the index for storing the message bits. This ensures that the changes are not concentrated in any particular portion of the image, a deficiency that made Jsteg more easily detectable. However, both of these algorithms are easily detected by the chi-square attack [20] since they equalize pairs of coefficients in a typical histogram of the image, giving a "staircase" appearance to the histogram as shown in Figure 2. F5 [21] is one of the most popular algorithms, and is undetectable using the chi-square technique. F5 uses matrix encoding along with permutating straddling to encode message bits. It also avoids making changes to any DC coefficients and coefficients with zero value. If the value of the message bit does not match the LSB of the coefficient, the coefficient's value is always decremented, so that the overall shape of the histogram is retained. However, a one can change to a zero and hence the same message bit must be embedded in the subsequent coefficients until its value becomes non-zero, since zero coefficients are ignored on decoding. However, this technique modifies the histogram of JPEG coefficients in a predictable manner. This is because of the shrinkage of ones converted to zeros increases the number of zeros while decreasing the



(a) Histogram before JSteg.



(b) Histogram after JSteg.

Fig. 2. Figure comparing the change in histogram after application of JSteg algorithm.

histogram of other coefficients and hence can be detected once an estimate of the original histogram is obtained [6].

Our algorithm falls under the category of statistical restoration or preservation schemes [13], [8], [16], [4], [7]. Outguess, proposed by Niels Provos, was one of the first algorithms to use statistical restoration methods to counter chi-square attacks [13]. The algorithm works in two phases, the embed phase and the restoration phase. After the embedding phase, using a random walk, the algorithm makes corrections to the unvisited coefficients to match it to the cover histogram. Outguess does not make any change to coefficients with 1 or 0 value and uses a error threshold to determine the amount of change which can be tolerated in the stego histogram. This means that that algorithm may not be able to restore the histogram completely to the cover image. It also compresses the stego image to a specific quality irrespective of the cover image. Our algorithm preserves all the properties of the cover image including the quality factor. Outguess makes changes to the coefficients adjacent to the modified ones to restore histogram and in turn replaces the LSBs. This property makes it detectable using second order statistics and image cropping techniques to guess the cover image [5], [14].

Another popular algorithm is Steghide [8], which uses graph theory techniques to preserve the histogram. Two inter-changeable coefficients are connected by an edge in the graph with coefficients as vertices of the graph. The message is them embedded by swapping the two coefficients connected in the graph. Since the coefficients are swapped instead of replacing LSBs, it is difficult to detect any distortion using first order statistical analysis. But the efficiency of Steghide is only 5.86% with respect to the cover file size. The results in figure 12 show that J3 has a high embedding efficiency ranging from 7% to 14% in contrast to 5.86% of Steghide algorithm.

Another technique of steganography proposed by Marvel et al. [12] uses spread spectrum techniques to embed data in the cover file. The idea is to embed secret data inside a noise signal which is then combined with the cover signal using a modulation scheme. Every image has some noise in it because of the image acquisition device and hence this property can be exploited to embed data inside the cover image. If the noise being added is kept at a low level, it will be difficult to detect the existence of message inside the cover signal. To make the detection hard, the noise signal is spread across a wider spectrum. At the decoder side, image restoration techniques are applied to guess the original image which is then compared with the stego image to estimate the embedded signal. Several other data hiding schemes using spread spectrum have been presented by Smith and Comiskey in [15]. Steganalysis techniques to detect spread spectrum steganography have been shown in [3], [17], where the authors claim to detect 70% of the embedded message bits and 95% of the images respectively.

IV. J3 EMBEDDING MODULE

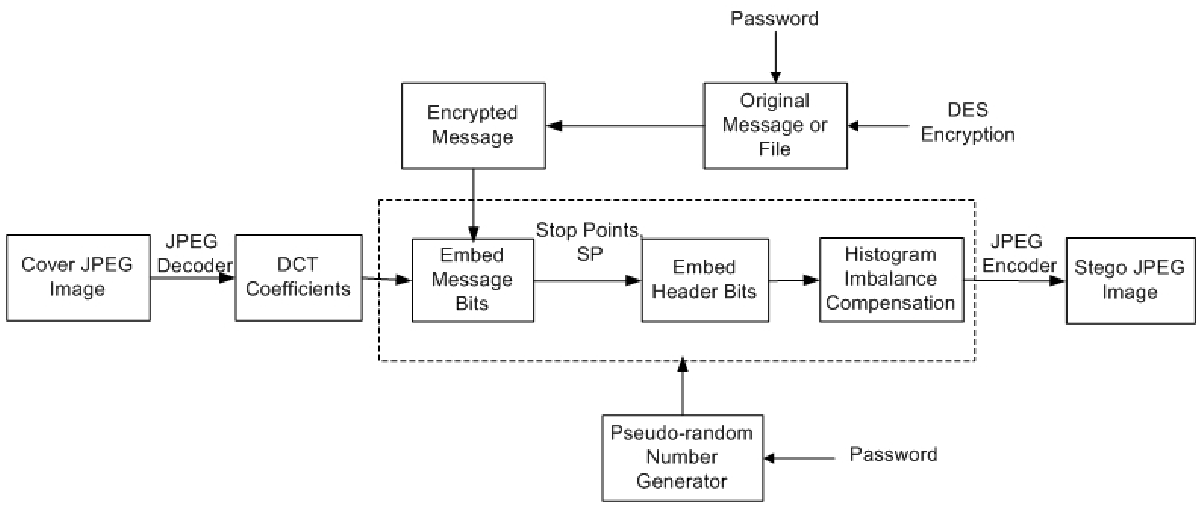


Fig. 3. Block diagram of our proposed embedding module.

Figure 3 shows the block diagram of our embedding module. The cover image is first entropy decoded to obtain the JPEG coefficients. The message to be embedded is encrypted using DES or AES. A pseudo-random number generator is used to visit the coefficients in random order to embed the encrypted message. The algorithm always makes changes to the coefficients in a pairwise fashion. For example, a JPEG coefficient with a value of 2 will only change to a 3 to encode message bit 1 in the LSB, and one with a value of 3 will only change to 2 to encode message bit 0 in the LSB. It is similar to a state machine where an even number will either remain in its own state or increase by 1 depending on the message bit. similarly, an odd number will either remain in its own state or decrease by 1. We apply the same technique for negative coefficients except that we take its absolute value to change a coefficient. Coefficients with value 1 and -1 have a different embedding strategy since their frequency is very high as compared to other coefficients. A -1 coefficient is equivalent to message bit 0 and +1 is equivalent to message bit 1. To encode message bit 0 in a coefficient with value 1, we change its value to -1. Similarly, to encode bit 1 in -1 coefficient, we change it to 1. To avoid any detection, we skip coefficients with value 0. The embedding coefficient pairs are $(-2n,-2n-1)...(-2,-3)$, $(-1,1)$, $(2,3)...$ $(2n,2n+1)$, where $2n+1$ and $-2n-1$ are the threshold limits for positive and negative coefficients, respectively.

Before embedding a bit in any coefficient, the algorithm determines if a sufficient number of coefficients of the other member of the pair are left to balance the histogram. If not, it stores the coefficient index in the header array, also known as stop point for that pair. Once the stop point for a pair is found, the algorithm will no longer embed any data bits in that pair of coefficient values. The header bits are embedded in the end since all the stop points are only known at the end of embedding.

The header stores useful information such a data length, location of stop points for each coefficient value pair, and the number of bits required to store each stop point. The structure of the header is given in table I. The formal definition of a stop point is given below.

Definition 1 [Stop Points] *A stop point, $SP(x,y)$ in J3 stores the index of DCT coefficient matrix and directs the algorithm to ignore any coefficients with value x or y that have an index value $\geq SP(x,y)$ during embedding or extraction process.*

16 Bits	5 Bits	5 Bits	$N_{SP} * N_{bSP})$ Bits
Data Length in Bytes, M_L	No. of bits required to store a stop point, N_{bSP}	No. of stop points, N_{SP}	Stop points, $SP(-2n,-2n-1)...SP(-2,-3)$, $SP(-1,1)$, $SP(2,3)...SP(2n,2n+1)$

TABLE I
HEADER STRUCTURE FOR J3 ALGORITHM

Explanation of Header fields:

$M_L =$ Represents the total message length in bytes. It does not include the length of header.

$N_{bSP} =$ Represents the total number of bits required to store a stop point. Let N_B be the total number of

blocks in the cover file. The total number of coefficients is then N_B*64. N_{bSP} represents the minimum number of bits needed to represent any number between 0 to N_B*64, which is $log_2(N_B*64)$.
$N_{SP} =$ represents the total number of stop points present in the header.
$SP(x,y) =$ represents a stop point. Each stop point is represented using N_{bSP} bits.

Terminology:
$Hist(x)$: Total number of coefficient x initially present in the cover image.
$TR(i)$: Remaining number of coefficients with value i that are unused.
$TC(x \rightarrow y)$: Total number of coefficient x changed to y and used for data.
$TC(x \rightarrow x)$: Total number of coefficient x unchanged and used for data.
$TR(x)$: Total number of coefficient x which remain untouched and unchanged.
$TT(x)$: Total number of coefficient x used to store data. $TT(x) = TC(x \rightarrow y) + TC(x \rightarrow x) = Hist(x) - TR(x)$
$Unbalance(x)$: Represents the unbalance in coefficient x as compare to $Hist(x)$.
N_B: Total number of blocks in the cover image
C_x: Value of coefficient at index location x in the cover image where $0 \leq x \leq N_B*64$
$Coeff_{total}$: total number of coeff in the image.
$Coeff_{total} = N_B*64$
$C_x =$ Value of coefficient at index x.

Example 1 *At the start of embedding process $Hist(x) = TR(x)$, since none of the coefficient x have been used for data. Assume the following scenario during embedding:*
$Hist(2) = 500, TC(2 \rightarrow 3) = 100, TC(2 \rightarrow 2) = 100$
$Hist(3) = 200, TC(3 \rightarrow 2) = 50, Tc(3 \rightarrow 3) = 100$
$\Rightarrow TR(2) = 300, TR(3) = 50$
Since 100 2's have been changed to a 3 and 50 3's have been changed back to 2, we have an imbalance in the histogram.
$Unbalance(2) = TC(3 \rightarrow 2) - TC(2 \rightarrow 3) = -50.$
$Unbalance(3) = TC(2 \rightarrow 3) - TC(3 \rightarrow 2) = -Unbalance(2) = 50.$
This means we have 50 more 3's than required and 50 fewer 2's than needed to balance the histogram pair (2,3) to its original values. Hence, we need at least 50 3's to balance the pair (2,3).

Let's assume that the next coefficient index is 2013 and $C_2013 = 3$. If $TR(3) = Unbalance(3)$, then *we know that we cannot encode any more data in this pair since we have just the minimum number of 3's remaining to balance the coefficient pair(2,3). Hence, we store the index location in* $SP(2,3)$*, i.e.,* $SP(2,3) = 2013$*. This directs the algorithm to stop embedding any more data in this pair after index 2013. This stop point is also used during the extraction process to locate the index to stop encoding pair(2,3).*

A. Embedding Algorithm

Embedding is divided in to various smaller subtasks. Algorithm 2 calculates the coefficient limit to consider for embedding. If a coefficient value is larger than the coefficient limit, it ignores it and selects the next one in sequence. It also skips the coefficients for embedding header bits since these will be embedded only after all the stop points are known. After skipping the header coefficients, algorithm 3 embeds the actual data bits. It calls function 1 to update the TC tables and function 5 to evaluate if sufficient number of coefficients are still remaining to balance the histogram. Once the message bits have been embedded and all the stop points known, algorithm 4 embeds the header bits using the same index sequence traversed in algorithm 2. Algorithm 3 and 2 modify the coefficients, and hence algorithm 6 calculates the net change in individual coefficients and restores the histogram to its original values using the unused coefficients. Negative coefficients and (-1,1) pair have not been considered in the algorithm to keep it short and simple but this pair is handled easily with a slight modification.

Let P be the password shared between the sender and the receiver. This password is used to generate the seed and also the sequence of pseudo-random numbers between 0 and $64N_B$.
$Enc(DES, M, k) =$ Encryption of Message M using k as key with DES standard.
$THr =$ Threshold to consider a coefficient for embedding data. If the total number of x coefficients is less than THr, we ignore that coefficient during embedding and extracting. This THr is a preset constant.
$PRNG(seed, x) =$ Pseudo-random number generating a number between 0 and x
$Bit(M, i) = i_{th}$ bit in message M
ME_{total} = Total number of bits in encrypted message, ME
$\phi =$ represents an AC coefficient

Algorithm 1: Function *EmbedBit()*.

```
begin
    Function EmbedBit (DataBit bit, index x)
    if $C_x \in odd \wedge bit \equiv 0$ then
        $TC(C_x \rightarrow C_x - 1) \leftarrow TC(C_x \rightarrow C_x - 1) + 1$ ;
        $C_x \leftarrow C_x - 1$ ;
    else if $C_x \in even \wedge bit \equiv 1$ then
        $TC(C_x \rightarrow C_x + 1) \leftarrow TC(C_x \rightarrow C_x + 1) + 1$ ;
        $C_x \leftarrow C_x + 1$ ;
    end
end
```

Algorithm 2: Calculate the threshold coefficient value to consider for embedding.

Input: (i) C – Input DCT coefficient array, (ii) M – the message to be embedded, and (iii) P.

Output: C– Modified DCT coefficient array.

```
begin
    $seed = k = MD5(P), ME = Enc(DES, M, k)$ ;          /* Encrypt message M with key k and DES standard */
    for $i = 2$ to 255 do
        if $Hist(i) < THr$ then                              /* if total number of $i_{th}$ coeff < threshold */
            $Coeff\_Limit \leftarrow i$ ;                   /* coefficient limit to consider for encoding */
            break ;
        end
    end
    if $Coeff\_Limit \in even$ then                          /* since a pair always ends in odd number */
        $Coeff\_Limit \leftarrow Coeff\_Limit + 1$;
    end
    /* Calculate $SP_{total}$, number of stop points                                                  */
    $SP_{total} \leftarrow (Coeff\_Limit - 1)/2$;                 /* number of pairs to store stop points. */
    $HDR_{total} = 16 + 5 + 5 + SP_{total} * Dec(N_{bSP})$;                /* total header length in bits */
    /* Skipping coefficients for header bits initially for later embedding.                        */
    $DataIndex = 0$;
    while $DataIndex \leq HDR_{total}$ do
        $x = PRNG(seed, Coeff_{total})$;
        if $C_x \leq Coeff\_Limit \wedge C_x \neq 0 \wedge C_x \in \phi$ then
            $TR(C_x) \leftarrow TR(C_x) - 1$ ;              /* decrease remaining number of coeff for embedding */
        end
    end
end
```

Algorithm 3: Embed message bits.

```
begin
    $DataIndex = 0$;
    while $DataIndex < ME_{total}$ do
        $x = PRNG(seed, Coeff_{total})$;
        if $C_x \equiv 0 \vee C_x > Coeff\_Limit \vee C_x \notin \phi$ then
            continue ;          /* ineligible coefficient value, so fetch next random number */
        else if $EvaluateStopPoint(x) \equiv false$ then
            $EmbedBit\Big(Bit(ME, DataIndex), x\Big)$;
            $TR(C_x) \leftarrow TR(C_x) - 1$ ;
            $dataIndex \leftarrow dataIndex + 1$ ;
        end
    end
end
```

Algorithm 4: Embed header bits in the coefficients.

```
begin
    /* Assume that the header data is stored in HDR array                                */
    $DataIndex = 0$ ;
    while $DataIndex \leq HDR_{total}$ do
        $x = PRNG1(seed, Coeff_{total})$;              /* generate same sequence for header coeff. */
        if $C_x \equiv 0 \vee C_x > Coeff\_Limit \vee C_x \notin \phi$ then
            continue ;          /* ineligible coefficient value, so fetch next random number */
        else
            $EmbedBit\Big(Bit(HDR, DataIndex), x\Big)$;
            $dataIndex \leftarrow dataIndex + 1$ ;
        end
    end
end
```

Algorithm 5: Function *EvaluateStopPoint()*.

```
Function EvaluateStopPoint (index x)
begin
    if $C_x \in odd$ then
        $Unbalance = TC(C_x - 1 \rightarrow C_x) - TC(C_x \rightarrow C_x - 1)$;
        if $Unbalance >= TR(C_x)$ then                     /* stop encoding the pair */
            $SP(C_x - 1, C_x) \leftarrow x$ ;                 /* store the stop point */
            return $true$;
        end
    else if $C_x \in even$ then
        $Unbalance = TC(C_x + 1 \rightarrow C_x) - TC(C_x \rightarrow C_x + 1)$;
        if $Unbalance >= TR(C_x)$ then                     /* stop encoding the pair */
            $SP(C_x, C_x + 1) \leftarrow x$ ;                 /* store the stop point */
            return $true$;
        end
    end
    return $false$;
end
```

Algorithm 6: Compensate histogram for changes made in algorithm 3 and 4.

```
begin
    /* Calculate net change in coefficient pairs                                    */
    for $i = 2$ to $Coeff\_Limit$ do
        if $TC(i \to i+1) > TC(i+1 \to i)$ then
            $TC(i \to i+1) \leftarrow TC(i \to i+1) - TC(i+1 \to i)$ ;
            $TC(i+1 \to i) \leftarrow 0$;
        else
            $TC(i+1 \to i) \leftarrow TC(i+1 \to i) - TC(i \to i+1)$ ;
            $TC(i \to i+1) \leftarrow 0$ ;
        end
        $i \leftarrow i+2$;
    end
    /* Calculate the total change in histogram                                      */
    $netChange = \sum_{k=1}^{SP_{total}} \Big(TC(2k \to 2k+1) + TC(2k+1 \to 2k)\Big)$
    /* Make changes to the unused coefficients to balance                           */
    while $netChange > 0$ do
        $x = PRNG(seed, Coeff_{total})$ ;
        if $C_x = 0 \vee C_x > Coeff\_Limit \vee C_x \notin \phi$ then
            continue;
        else if $C_x \in even \wedge TC(C_x + 1 \to C_x) > 0$ then
            $T(C_x + 1 \to C_x) \leftarrow TC(C_x + 1 \to C_x) - 1$;
            $C_x \leftarrow C_x + 1$;
            $netChange \leftarrow netChange - 1$ ;
        else if $C_x \in odd \wedge TC(C_x - 1 \to C_x) > 0$ then
            $T(C_x - 1 \to C_x) \leftarrow TC(C_x - 1 \to C_x) - 1$;
            $C_x \leftarrow C_x - 1$;
            $netChange \leftarrow netChange - 1$ ;
        end
    end
end
```

V. J3 Extraction Module

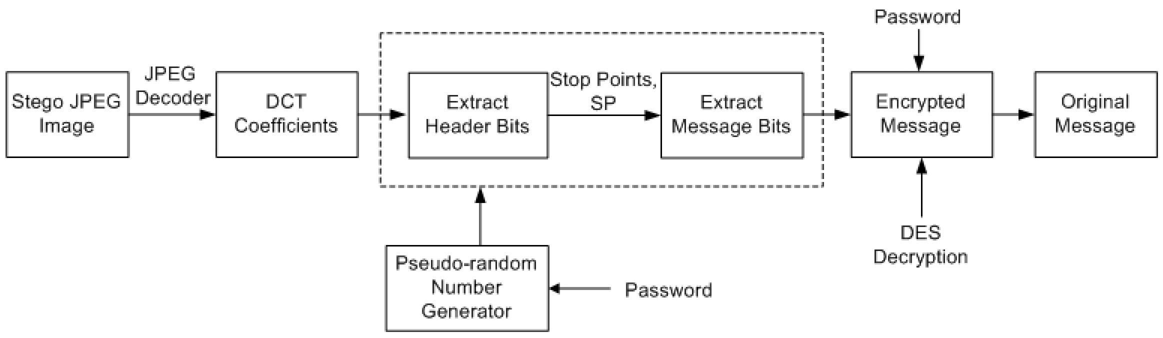


Fig. 4. Block diagram of our proposed extraction module.

This section deals with the extraction of a message *M* from a given stego image. The extraction algorithm is simple, as the receiver has only to deal with the exact index locations to stop decoding each value pair. Password *P* is used to generate the random number sequence used to permute the coefficient indices for visitation order. The constant part of the header is decoded first, and reveals the length of the dynamic portion of the header. The dynamic portion of the header contains the stop points, which are necessary to stop decoding where required. Once all the header data have been extracted, the extraction process starts decoding the message bits, taking care to stop extraction from a pair of coefficients values when the stop point has been reached. As usual, (-1,1) are decoded in a different way as explained above. The rest are decoded in pairs (2,3), (4,5), etc. The decoding algorithm is given below. As explained earlier, we will only show the algorithm for positive coefficients. Similar rules apply to the negative coefficients and the (-1,1) pair, with slight modification. A block diagram of our extraction module is given in figure 4.

A. Extraction Algorithm

Similar to the embedding algorithm, the extraction algorithm is also divided into two smaller modules. Algorithm 7 first decodes the static part of header to recover the message length, the number of stop points, and the number of bits needed to store each stop point. Using the static header part, the algorithm determines the length and interpretation of the dynamic portion of header to finally decode all the stop points. Finally, algorithm 8 extracts the encrypted message bits, which are then decrypted to recover the actual message.

Algorithm 7: Extraction of header bits.

```
begin
    $Dec(DES, M, k)$ = Decryption of Message M using k as key with DES standard.;
    Input: (i) $C$ – Modified DCT coefficient array and (ii) $P$ – shared password between the sender and receiver.
    Output: $M_{out}$– Output Message
    $seed = k = MD5(P)$;
    /* Assume $HDR$ array to be empty initially. Extract static header part first */
    $HDR_{static} = 16+5+5$ ;                                   /* static header length in bits */
    Let $HDR_i = i_{th}$ bit of $HDR$ array;
    $i = 0$ ;
    while $i \leq HDR_{static}$ do
        $x = PRNG(seed, Coeff_{total})$ ;          /* PRNG to generate random indices for coeff. */
        /* $Coeff\_Limit$ is calculated the same way as in the embedding algorithm */
        if $C_x \equiv 0 \vee C_x > Coeff\_Limit \vee C_x \notin \phi$ then
            continue;
        else if $C_x \in odd$ then
            $HDR_i \leftarrow 1$;
            $i++$ ;
        else if $C_x \in even$ then
            $HDR_i \leftarrow 0$ ;
            $i++$ ;
        end
    end
    /* Decode data Length in Bytes, $M_L$ and No. of bits required to represent a coeff location,
       $N_{bSP}$ from $HDR$ array. */
    /* Decode No. of stop points and $SP_{total}$ from $HDR$ array */
    /* Now calculate the dynamic header length using the number of stop points, $SP_{total}$ and $N_{bSP}$ */
    /* Traverse the coefficients and decode the stop points from dynamic header array. */
    /* Store the values in $SP(x, y)$ array from decoded bits. */
end
```

Algorithm 8: Extraction of message bits.

```
begin
    $M_{total} = M_L * 8$ ;                                  /* total message length in bits */
    $i = 0$;
    while $i \leq M_{total}$ do
        $x = PRNG(seed, Coeff_{total})$ ;           /* PRNG to generate random indices for coeff. */
        if $C_x \equiv 0 \vee C_x > Coeff\_Limit \vee C_x \notin \phi$ then
            continue;                                /* ineligible coeff for data extraction */
        else if $C_x \in even \wedge SP(C_x, C_x + 1) \neq x$ then    /* current index doesn't match stop point */
            $M_i \leftarrow 0$ ;                                  /* $i_{th}$ bit of Message array, $M$ */
            $i \leftarrow i + 1$ ;
        else if $C_x \in odd \wedge SP(C_x - 1, C_x) \neq x$ then     /* current index doesn't match stop point */
            $M_i \leftarrow 1$ ;                                  /* $i_{th}$ bit of Message array, $M$ */
            $i \leftarrow i + 1$ ;
        end
    end
    $M_{out} = Dec(DES, M, k)$ ;                 /* Decrypt message M using key k and DES standard */
end
```

VI. Theoretical Estimation of Embedding Capacity and Stop Point

This section shows how to estimate the expected embedding capacity of a file and the stop point indices for each coefficient pair. We show the calculation for positive coefficients only. The calculation for the negative coefficients and the (-1,1) pair are similar with slight modifications.

$p_{m,0}$ =Probability of bit 0 in the message.

$p_{m,1} = (1 - p_{m,0})$ = Probability of bit 1 in message.

$p_{c,2x+1}$ = Probability of encountering an odd number with value (2x+1) in traversing the coefficients.

$p_{c,2x}$ = Probability of encountering an even number with value 2x in traversing the coefficients.

k_{total} = Total number of coefficients in the input image.

$$p_{m,0} = \frac{\sum M_0}{\sum M_0 + \sum M_1} \tag{1}$$

$$p_{m,1} = \frac{\sum M_1}{\sum M_0 + \sum M_1} \tag{2}$$

$$k_{total} = \sum_{x=2}^{n} Hist(x) \tag{3}$$

$$p_{c,2x+1} = \frac{Hist(2x+1)}{k_{total}} \tag{4}$$

$$p_{c,2x} = \frac{Hist(2x)}{k_{total}} \tag{5}$$

An odd coefficient can only decrease or retain its value to embed a data bit. Similarly, an even number can only increase or retain its value to embed a data bit, as explained in embedding module.

$$Probability(2x+1 \rightarrow 2x) = p_{m,0} \cdot p_{c,2x+1} \tag{6}$$

$$Probability(2x \rightarrow 2x+1) = p_{m,1} \cdot p_{c,2x} \tag{7}$$

$$Probability(2x+1 \rightarrow 2x+1) = p_{m,1} \cdot p_{c,2x+1} \tag{8}$$

$$Probability(2x \rightarrow 2x) = p_{m,0} \cdot p_{c,2x} \tag{9}$$

Let $\gamma_{2x,2x+1}$ = Total number of eligible coefficients visited so far at any instant.
Let $TC_{Ex}(x \rightarrow y)$ be the expected number of coefficients with value x changed to y to embed a data bit.
Let $TR_{Ex}(x)$ be the expected number of coefficients with value x remaining unchanged and unused.

$$TC_{Ex}(2x+1 \rightarrow 2x) = \gamma_{2x,2x+1} \cdot Probability(2x+1 \rightarrow 2x) \tag{10}$$

$$TC_{Ex}(2x+1 \rightarrow 2x+1) = \gamma_{2x,2x+1} \cdot Probability(2x+1 \rightarrow 2x+1) \tag{11}$$

$$TC_{Ex}(2x \rightarrow 2x+1) = \gamma_{2x,2x+1} \cdot Probability(2x \rightarrow 2x+1) \tag{12}$$

$$TC_{Ex}(2x \rightarrow 2x) = \gamma_{2x,2x+1} \cdot Probability(2x \rightarrow 2x) \tag{13}$$

$$TR_{Ex}(2x+1) = Hist(2x+1) - \Big[TC_{Ex}(2x+1 \rightarrow 2x) + TC_{Ex}(2x+1 \rightarrow 2x+1)\Big] \tag{14}$$

$$TR_{Ex}(2x) = Hist(2x) - \Big[TC_{Ex}(2x \rightarrow 2x+1) + TC_{Ex}(2x \rightarrow 2x)\Big] \tag{15}$$

Let $Unbalance_{Ex}(x)$ be the expected net unbalance of coefficients with value x.
Since we have estimated $TR_{Ex}(i)$ for all the coefficients, we can now calculate the condition when the coefficient pair will no longer be used to embed data, since we will be left with the exact amount of

coefficient to balance the histogram after the embedding process. The condition is:

$$\begin{aligned} Unbalance_{Ex}(2x+1) &= TC_{Ex}(2x \rightarrow 2x+1) - TC_{Ex}(2x+1 \rightarrow 2x), \\ & TC_{Ex}(2x \rightarrow 2x+1) \geq TC_{Ex}(2x+1 \rightarrow 2x) \end{aligned} \tag{16}$$

$$\begin{aligned} Unbalance_{Ex}(2x) &= TC_{Ex}(2x+1 \rightarrow 2x) - TC_{Ex}(2x \rightarrow 2x+1), \\ & TC_{Ex}(2x+1 \rightarrow 2x) \geq TC_{Ex}(2x \rightarrow 2x+1) \end{aligned} \tag{17}$$

The stop condition is:

$TR_{Ex}(x) = Unbalance_{Ex}(x)$

Replacing LHS of equation 16 with RHS of equation 14, we get

$$\begin{aligned} & Hist(2x+1) - \Big[TC_{Ex}(2x+1 \rightarrow 2x) + TC_{Ex}(2x+1 \rightarrow 2x+1)\Big] \\ & = TC_{Ex}(2x \rightarrow 2x+1) - TC_{Ex}(2x+1 \rightarrow 2x) \end{aligned} \tag{18}$$

Using equation 10, 11 and 12, we get:

$$Hist(2x+1) - \gamma_{2x,2x+1} \cdot Probability(2x+1 \rightarrow 2x+1) = \gamma_{2x,2x+1} \cdot Probability(2x \rightarrow 2x+1) \tag{19}$$

Solving for $\gamma_{2x,2x+1}$ using equation 7 and 8, we get:

$$\gamma_{2x,2x+1} = \frac{Hist(2x+1)}{p_{m,1} \cdot (p_{c,2x} + p_{c,2x+1})} \tag{20}$$

Simplifying using equation 2, 3, 4 and 5, we get:

$$\gamma_{2x,2x+1} = \frac{Hist(2x+1) \cdot \sum_{i=2}^{n} Hist(i) \cdot \Big(\sum M_0 + \sum M_1\Big)}{\sum M_1 \cdot \Big(Hist(2x) + Hist(2x+1)\Big)} \tag{21}$$

If we solve equation 15 in a similar way, we get another value of $\gamma_{2x,2x+1}$ as:

$$\gamma_{2x,2x+1} = \frac{Hist(2x) \cdot \sum_{i=2}^{n} Hist(i) \cdot \Big(\sum M_0 + \sum M_1\Big)}{\sum M_0 \cdot \Big(Hist(2x) + Hist(2x+1)\Big)} \tag{22}$$

Let equation 21 be represented as $\gamma^{\alpha}_{2x,2x+1}$ and equation 22 as $\gamma^{\beta}_{2x,2x+1}$ for convenience.

Theorem 1 *The estimated stop point for pair(2x,2x+1), $\gamma^{est}_{2x,2x+1}$, is the minimum of $\gamma^{\alpha}_{2x,2x+1}$ and $\gamma^{\beta}_{2x,2x+1}$.*

$\gamma^{est}_{2x,2x+1} = min\Big\{\gamma^{\alpha}_{2x,2x+1}, \gamma^{\beta}_{2x,2x+1}\Big\}$

Proof: Let the maximum coefficient index be represented by $Index_{max}$. The maximum index value is equal to the maximum number of eligible coefficients in the image. Hence,
$Index_{max} = \sum_{i=2}^{n} Hist(i)$
Any stop point, $\gamma_{2x,2x+1}$ cannot exceed the value of maximum coefficient index. Lets assume
$\gamma^{\alpha}_{2x,2x+1} \leq Index_{max}$
$\gamma^{\alpha}_{2x,2x+1} \leq \sum_{i=2}^{n} Hist(i)$
Using equation 20 and 21 and substituting for $\gamma^{\alpha}_{2x,2x+1}$, we get

$$\frac{Hist(2x+1) \cdot \sum_{i=2}^{n} Hist(i)}{p_{m,1} \cdot \Big(Hist(2x) + Hist(2x+1)\Big)} \leq \sum_{i=2}^{n} Hist(i) \tag{23}$$

Simplifying equation 23, we get

$$\frac{Hist(2x+1)}{(1 - p_{m,0}) \cdot \Big(Hist(2x) + Hist(2x+1)\Big)} \leq 1 \tag{24}$$

Further simplifying,

$$p_{m,0} \cdot \Big(Hist(2x+1) + Hist(2x)\Big) \leq Hist(2x) \tag{25}$$

$$\Rightarrow \frac{Hist(2x)}{p_{m,0}) \cdot \Big(Hist(2x) + Hist(2x+1)\Big)} \geq 1 \tag{26}$$

Multiplying both sides by $\sum_{i=2}^{n} Hist(i)$, we get

$$\frac{Hist(2x) \cdot \sum_{i=2}^{n} Hist(i)}{p_{m,0}) \cdot \Big(Hist(2x) + Hist(2x+1)\Big)} \geq \sum_{i=2}^{n} Hist(i) \tag{27}$$

From equation 22, L.H.S. of the above equation is $\gamma^{\beta}_{2x,2x+1}$ and R.H.S. is $Index_{max}$.
$\Rightarrow \gamma^{\beta}_{2x,2x+1} \geq Index_{max}$, which is not vaild.
Similarly, using $\gamma^{\beta}_{2x,2x+1}$ as the starting point for proof, we get
$\gamma^{\alpha}_{2x,2x+1} \geq Index_{max}$
Hence, $\gamma^{est}_{2x,2x+1}$ can be written as

$$\gamma^{est}_{2x,2x+1} = min\Big\{\gamma^{\alpha}_{2x,2x+1}, \gamma^{\beta}_{2x,2x+1}\Big\} \tag{28}$$

Hence proved. ■

From the calculations, we conclude that the stop point for the pair $(2x, 2x+1)$ would likely be the coefficient index at which the current value of $\gamma_{2x,2x+1}$ satisfies either equation 21 or 22.

The estimated embedding capacity for coefficient pair $(2x, 2x+1)$ is:

$$\begin{aligned} Capacity_{Ex}(2x,2x+1) &= TC_{Ex}(2x \rightarrow 2x+1) + TC_{Ex}(2x \rightarrow 2x) \\ &\quad + TC_{Ex}(2x+1 \rightarrow 2x) + TC_{Ex}(2x+1 \rightarrow 2x+1)\, Bits \end{aligned} \tag{29}$$

Simplifying equation 29 using equation 1 to 13, we get

$$Capacity_{Ex}(2x,2x+1) = \gamma_{2x,2x+1} \cdot \Big(p_{c,2x} + p_{c,2x+1}\Big)\, Bits \tag{30}$$

Total expected capacity including negative coefficients and (-1,1) pair is:

$Capacity_{total}$ = Negative Coefficient Capacity + $(-1,1)$ Capacity + Positive coefficient capacity.

$$\begin{aligned} Capacity_{total} = &\Big(\sum_{x=-1}^{-coeff\_limit} \gamma_{2x,2x-1} \cdot (p_{c,2x} + p_{c,2x-1}) \Big) \\ &+ \Big(\gamma_{-1,1} \cdot (p_{c,-1} + p_{c,1}) \Big) \\ &+ \Big(\sum_{x=1}^{coeff\_limit} \gamma_{2x,2x+1} \cdot (p_{c,2x} + p_{c,2x+1}) \Big)\, Bits \end{aligned} \tag{31}$$

Using 31 and replacing the value of $\gamma_{2x,2x+1}$ from equation 21 and 22, we get

$$Capacity1_{total} = \frac{\sum_{x=1}^{coeff\_limit} \Big(Hist(2x) + Hist(-2x)\Big) + Hist(-1)}{p_{m,0}}\, Bits \tag{32}$$

$$Capacity2_{total} = \frac{\sum_{x=1}^{coeff\_limit} \Big(Hist(2x+1) + Hist(-2x-1)\Big) + Hist(1)}{p_{m,1}}\, Bits \tag{33}$$

Let $Capacity_{max}$ = maximum capacity possible.

Now $Capacity_{max}$ will be equal to the total number of coefficients within $coeff\_limit$ range.

$Capacity_{max} = \sum_{x=1}^{coeff\_limit} \Big(Hist(2x) + Hist(2x+1) + Hist(-2x) + Hist(-2x-1)\Big) + Hist(1) + Hist(-1)$

Simplifying using equation 33 and 33, we get

$$Capacity_{max} = Capacity1_{total} \cdot p_{m,0} + Capacity2_{total} \cdot p_{m,1} \tag{34}$$

Theorem 2 *The estimated capacity Est_Capacity is the minimum of* $Capacity1_{total}$ *and* $Capacity2_{total}$.

$Est\_Capacity = min\Big\{Capacity1_{total}, Capacity1_{total}\Big\}$

Proof:

$$Let\ \ Capacity1_{total} \leq Capacity_{max} \tag{35}$$

Substituting value of $Capacity1_{total}$ from 34, we get

$$\begin{aligned}
&\frac{\Big(Capacity_{max} - Capacity2_{total} \cdot p_{m,1}\Big)}{p_{m,0}} \leq Capacity_{max}\\
&\Big(Capacity_{max} - Capacity2_{total} \cdot p_{m,1}\Big) \leq \Big(Capacity_{max} \cdot p_{m,0}\Big)\\
&Capacity_{max} \cdot (1 - p_{m,0}) \leq Capacity2_{total} \cdot p_{m,1}
\end{aligned}$$

since $(1 - p_{m,0}) = p_{m,1}$

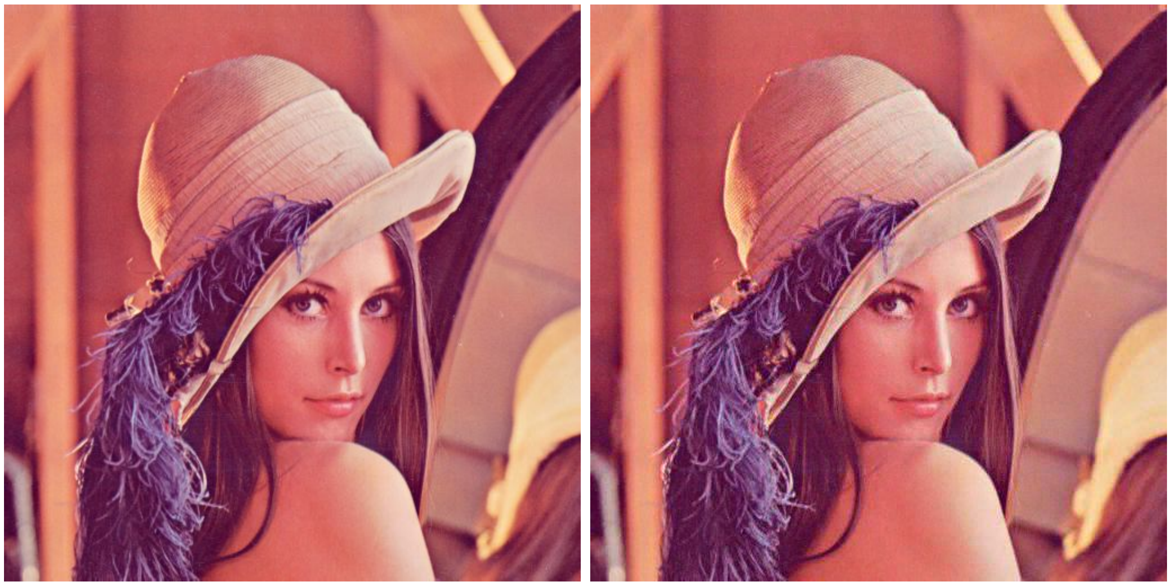
$$\begin{aligned}
&Capacity_{max} \cdot p_{m,1} \leq Capacity2_{total} \cdot p_{m,1}\\
&Capacity2_{total} \geq Capacity_{max} \qquad (36)
\end{aligned}$$

From equation 35 and 36, $Est\_Capacity$ can be written as:
$Est\_Capacity = Capacity1_{total} = min\Big\{Capacity1_{total}, Capacity2_{total}\Big\}$
$Capacity2_{total}$ is not valid since $Capacity2_{total} \geq Capacity_{max}$. Similarly, assuming that $Capacity2_{total} \leq Capacity_{max}$, we get the result:
$Est\_Capacity = Capacity2_{total} = min\Big\{Capacity1_{total}, Capacity2_{total}\Big\}$
Hence proved. ■

VII. Results

The code was written in Java, and includes code to decode a JPEG image to get the JPEG coefficients, embed data in eligible coefficients, balance the histogram to its original values, and finally re-encode the image in JPEG format with modified coefficients while preserving the original quantization tables and other properties of the image. Tests were performed on 107 different JPEG color images of varying size and texture. In our experiments, we encode all three components of the image, with each component having its own separate header and part of the message. The extraction algorithm then reads and extracts the data from all three components one by one, and finally combines them into one single message. The histogram remains unchanged in the stego JPEG file.

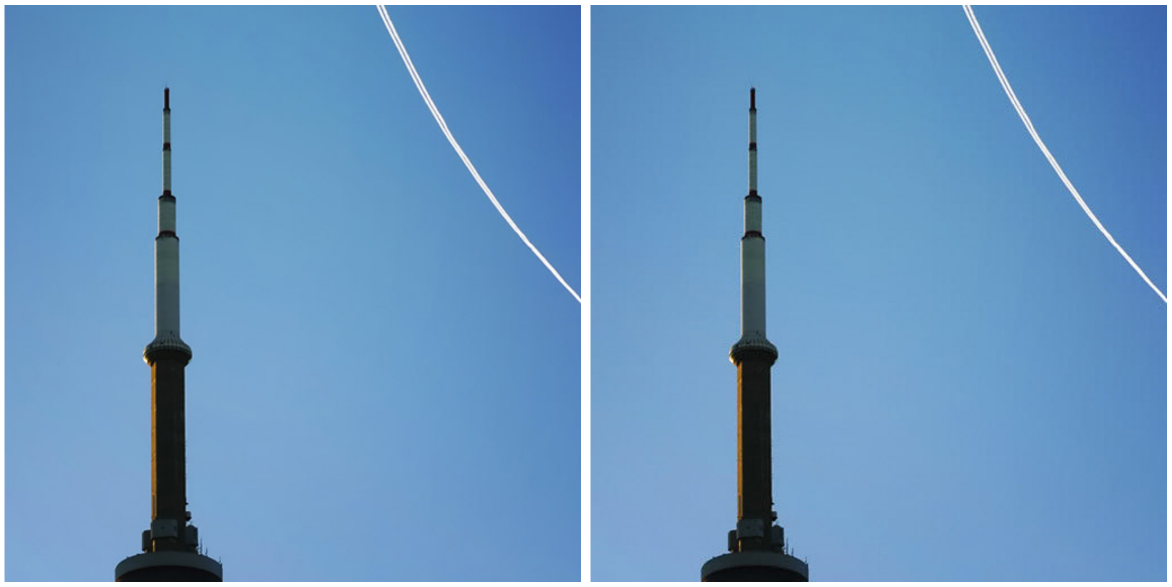
The cover and stego image of a popularly used Lena image are shown in figure 5(a) and 5(b). Another cover and stego image of a tower is given in figure 6 to show that there is no visual change even if most of the image is plain without texture.



(a) Lena Cover Image, File Size = 44KB, 512 x 512 pixels (b) Lena Stego Image, File Size = 44KB, 512 x 512 pixels, Embedded Data Size= 5019 Bytes

Fig. 5. Comparison of Lena Cover image with Stego image

The histogram of the Lena image in figure 5 is shown in figure 7. The graph shows the histogram of the image before embedding, before compensation and after compensation. The before compensation bars shows that the odd coefficients have increased in number as opposed to the even coefficients, which are reduced. This is because of the embedding scheme. Since we make changes in pairs(2x, 2x+1), and $Hist(2x) \approx 2Hist(2x+1)$, the number of changes from $2x$ to $2x+1$ will be more than number of changes from $2x+1$ to $2x$. Hence, even coefficients decrease and odd coefficients decrease in their overall number. After the embedding process, there is an imbalance in the histogram as a result of embedding data in the JPEG coefficients. After compensation bars show the status of the histogram after compensation is done. We thus verify experimentally that there is zero deviation in the histogram after the compensation process is completed.



(a) Cover Image, File Size = 23KB, 512 x 512 pixels (b) Stego Image, File Size = 20.1KB, 512 x 512 pixels, Embedded Data Size= 1220 Bytes

Fig. 6. Comparison of Cover image with Stego image containing a clear blue sky

A. *Estimated Capacity vs Actual Capacity*

In section VI, we estimated the theoretical capacity of the embedded data in the image. The graph in figure 8 compares the estimated capacity with the actual capacity for 107 different images. In conclusion, our estimation is almost equal to the actual capacity, which supports the correctness of the theoretical analysis of capacity estimation. The slight variation between the actual and theoretical capacity is because $p_{m,0}$ and $p_{m,1}$ are calculated based on the total message bits to be embedded, which is much larger than the maximum capacity of the image. The algorithm only embeds data in the image up to its maximum capacity until which it can balance the histogram. Also, the header data in all the three components are not accounted in the calculations which makes another contribution in the slight difference between the two graphs. Moreover, the random number generator is a pseudo-random number generator and not a true random number generator, which also makes difference between actual and theoretical embedding capacity.

B. *Estimated Stop-Point vs Actual Stop-Point*

It is that no matter what the visitation order, it is likely that there will be some deviation from the expected in visitation order for each pair, so we will have to stop sooner than expected. Graph in figure

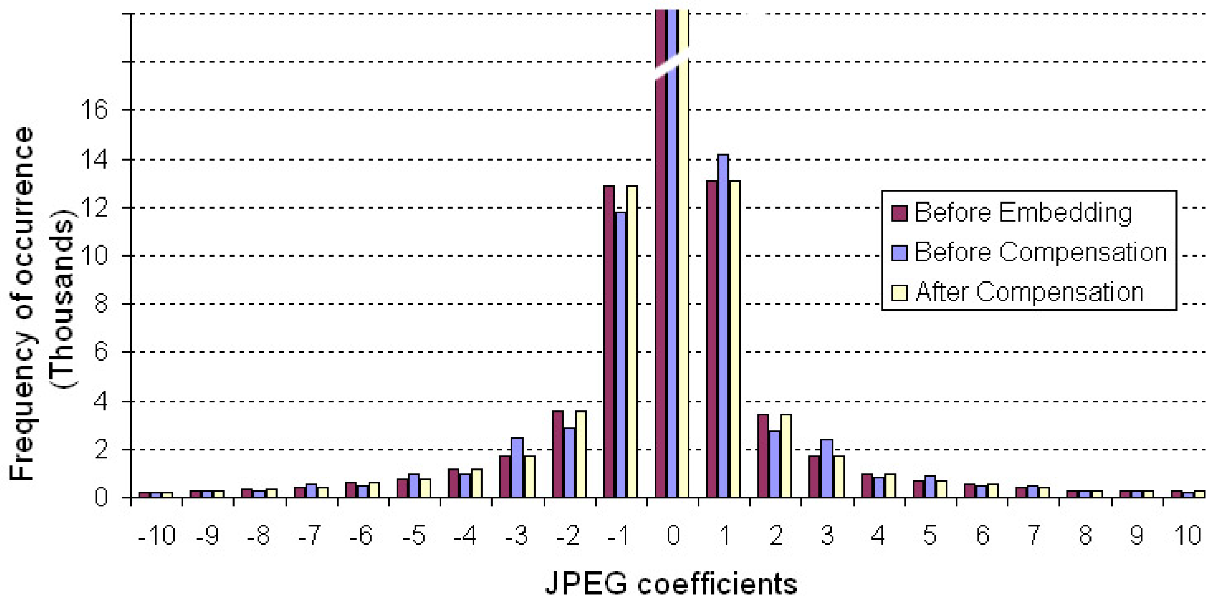


Fig. 7. Comparison of Lena histogram at different stages of embedding process.

10 proves this corollary and shows that the actual stop point index occurs before the theoretical stop point. Images in figure 9 along with the Lena image in figure 5(a) have been used to demonstrate this result for different pairs. The higher order coefficient pairs have not been used since the frequency of occurrence is very low and our algorithm ignores these coefficients while embedding data.

C. Embedding Efficiency of J3

Graph in figure 11 shows the embedding efficiency with respect to the number of data bits embedded per pixel(bpp) and bits embedded per non-zero coefficients(bpnz). The general trend in the graph shows that bpp varies between 0.05 to 0.55 and increases with increase in file size. This is due to the reason that greater file size gives more data to be embedded in other two components of the image which increases the bit capacity. The peaks and valleys in the graph are due to the texture of the image. Some images have a large number of zeros which result in low *bpp* value. The other part of the graph shows that *bpnz* varies from 0.45 to 0.75. This demonstrates that our algorithm has a very high capacity, since we are able to use almost 40%-70% of non-zero coefficients to embed data. We have found no other existing algorithm with as high a *bpnz* value as J3. The peaks in the graph are due to images with a large number of zero coefficients, which gives a high *bpnz* and low *bpp* value.

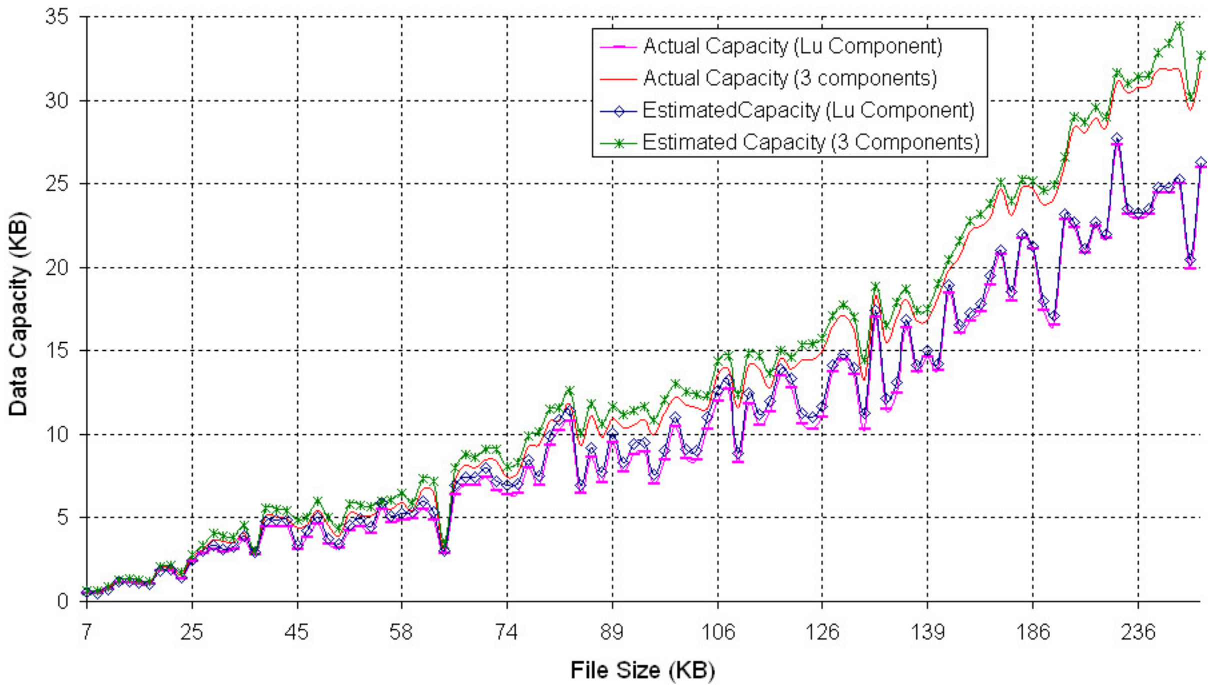


Fig. 8. Comparison of estimated capacity with actual capacity

(a) Lotus.jpg (b) plane.jpg (c) apple.jpg

Fig. 9. JPEG images used for comparison of stop point indices

D. Embedding Percentage of J3 compared to Cover Image

Graph in figure 12 shows the data embedding ratio with respect to the cover image size. The graph shows that we are able to embed almost 6-14% data compared to its file size. The graph also shows that our method outperforms F5 if the file size is larger. The capacity with respect to file size is low because the majority of the coefficients in any JPEG image are zeroes. Our method does not embed any data in

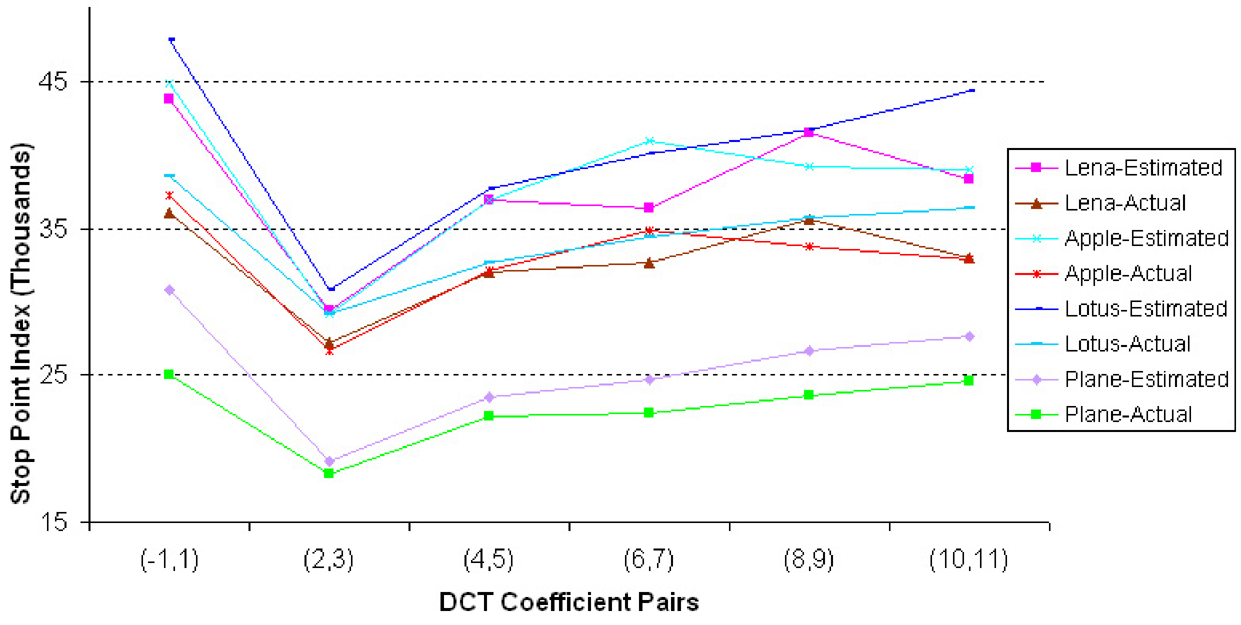


Fig. 10. Comparison of estimated stop point index vs actual stop point index

zero coefficients to help avoid detection.

E. Comparison of J3 with other algorithms

In this experiment, we took the same 107 JPEG images of various size and texture for embedding data to it maximum capacity using J3, F5, Steghide and Outguess algorithms. The comparison graph is shown in figure 13. From the graph, we can conclude that our algorithm performs better when the image size is large. Peaks and valleys in the graph are due to the varying texture of images. Valleys occur when images don't contain much variation in them and are usually plain textured. This leads to good compression ratio and hence a large number of zero coefficients, which doesn't leave many coefficients in which to embed data. J3 has nearly the same data capacity as compared to other algorithms when the image size is small, but it outperforms the others when the image size is large. Even using single component embedding it performs better than the other three algorithms. This is due to the fact that J3 uses stop points to minimize the wastage of any unused coefficients and leaves just the right amount to balance the histogram. Outguess performs the worst in embedding capacity since it stops embedding data when a certain threshold is reached.

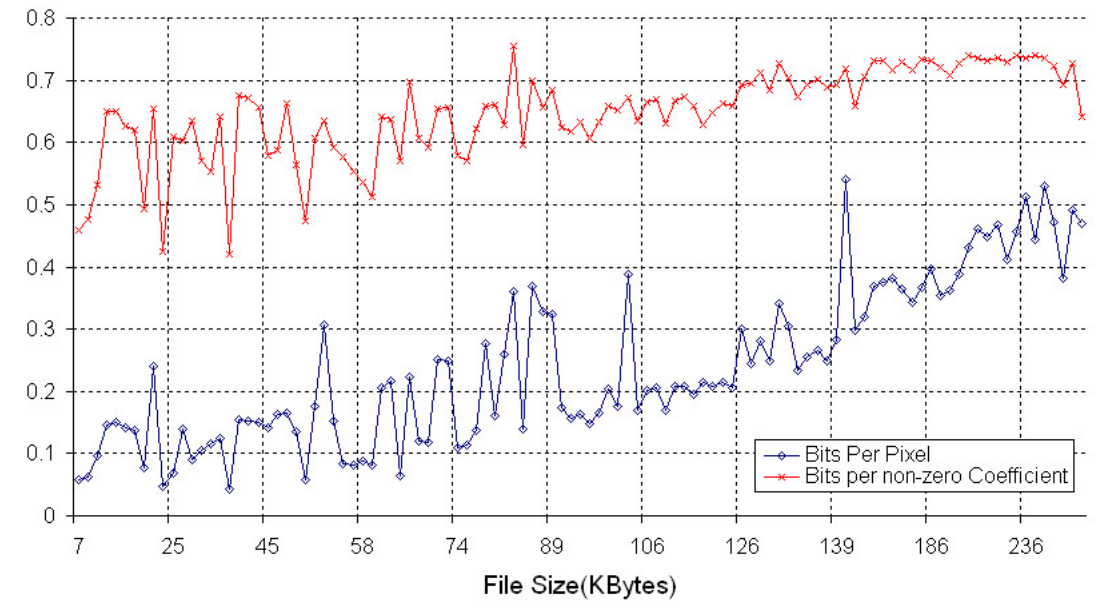


Fig. 11. Embedding efficiency of J3 for bits per pixel and bits per non-zero coefficient

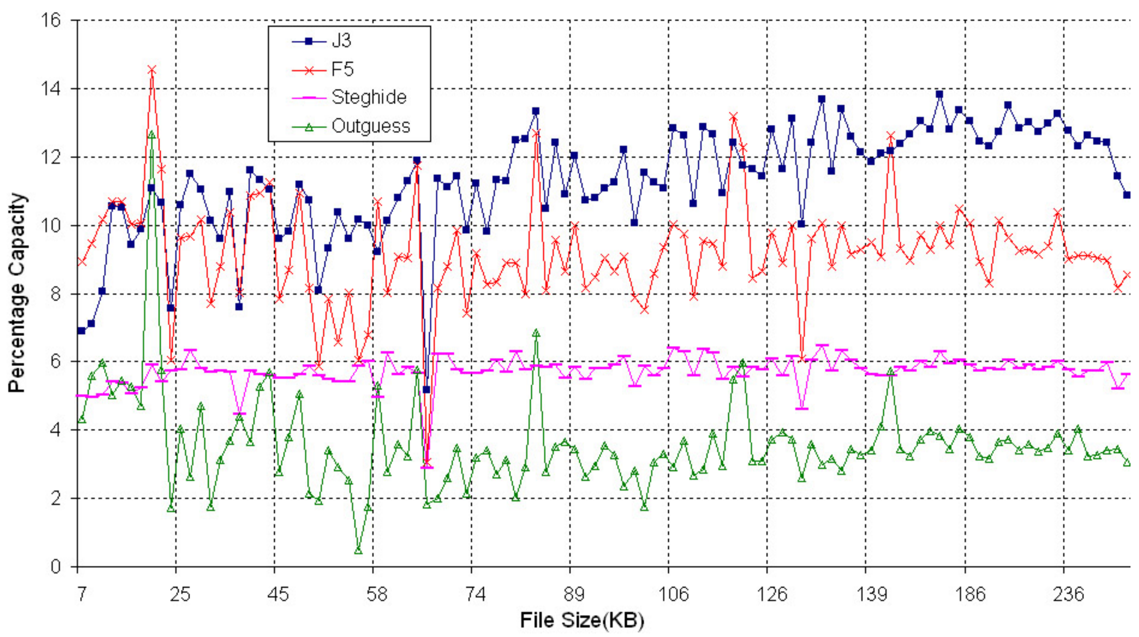
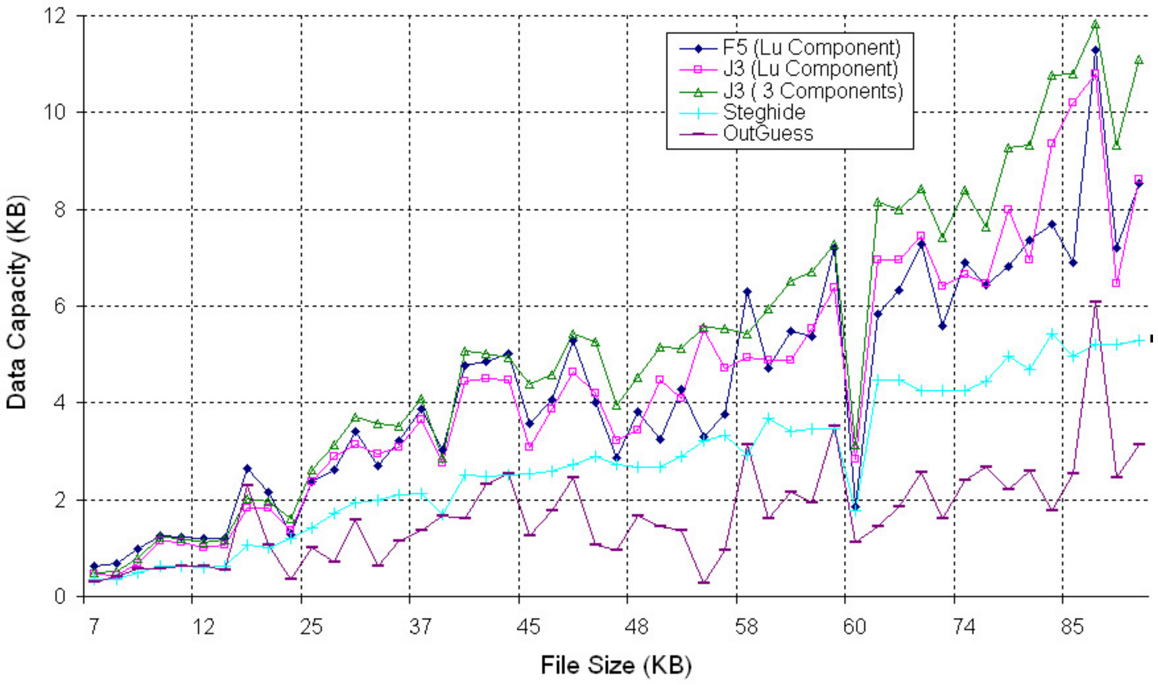
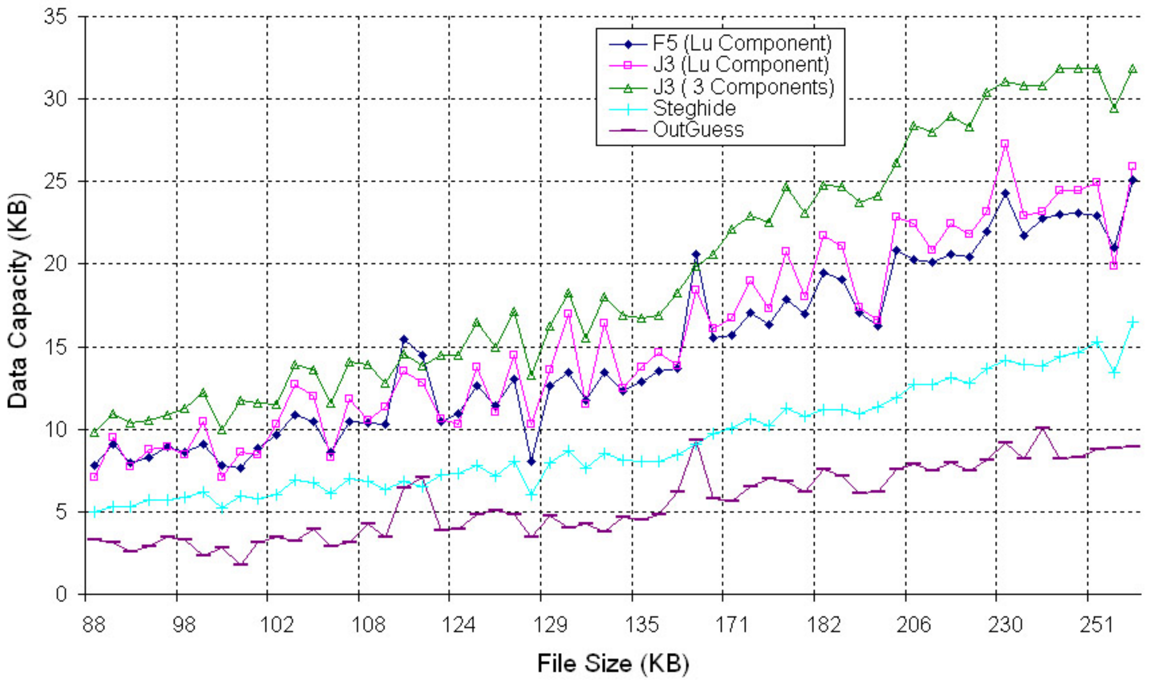


Fig. 12. Capacity percentage of data embedded in each image of J3 and F5



(a) Embedding capacity for file size $\leq$ 88 KB



(b) Embedding capacity for file size $\geq$ 88 KB

Fig. 13. Comparison of embedding capacity of J3 with other algorithms

VIII. Conclusion

J3 is a new JPEG steganography algorithm that uses LSB encoding to embed data and histogram compensation to balance all the coefficients changed during the embedding process. J3 only makes changes to the non-zero coefficients in pairs, which ensures that that the coefficients are only changed by a +1 or -1, except for the (-1,1) pair. We compared our scheme to the popular F5, Steghide, and Outguess algorithms, and the results show that the capacity of J3 is much larger than these algorithms with the added benefit of a perfectly unchanged histogram. The cover and the stego images retain all their first order statistical properties including their quantization tables, height, width, file size, and the net value of individual coefficients. As far as we are know, there is no existing algorithm that provides such a high data capacity with a perfect histogram restoration. The embedding rate of J3 ranges between 0.35 bpp and 0.65 $bpnz$, which is quite high for a JPEG image.

In the future, we plan to improve on this algorithm to increase its data capacity and perform second order statistical steganalysis.

References

[1] Jp hide&seek. http://linux01.gwdg.de/~alatham/stego.html.

[2] R. Chandramouli and N. Memon. Analysis of LSB based image steganography techniques. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, 2001.

[3] R. Chandramouli and KP Subbalakshmi. Active steganalysis of spread spectrum image steganography. In *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, volume 3, 2003.

[4] E. Franz et al. Steganography preserving statistical properties. *Lecture notes in computer science*, pages 278–294, 2003.

[5] J. Fridrich, M. Goljan, and D. Hogea. New methodology for breaking steganographic techniques for JPEGs. *Submitted to SPIE: Electronic Imaging*, 2003.

[6] J. Fridrich, M. Goljan, and D. Hogea. Steganalysis of JPEG images: Breaking the F5 algorithm. *Lecture Notes in Computer Science*, pages 310–323, 2003.

[7] J. Fridrich, T. Pevnỳ, and J. Kodovskỳ. Statistically undetectable jpeg steganography: dead ends challenges, and opportunities. In *Proceedings of the 9th workshop on Multimedia & security*, pages 3–14. ACM New York, NY, USA, 2007.

[8] S. Hetzl and P. Mutzel. A graph-theoretic approach to steganography. *Lecture Notes in Computer Science*, 3677:119, 2005.

[9] Andy C. Hung. PVRG-JPEG CODEC 1.1. www.dclunie.com/jpegge/jpegpvrg.pdf, November 1993.

[10] ITU-T. ITU-T T.81 (JPEG-1)-based still-image coding using an alternative arithmetic coder, September 2005.

[11] YK Lee and LH Chen. High capacity image steganographic model. *IEE Proceedings-Vision, Image and Signal Processing*, 147(3):288–294, 2000.

[12] LM Marvel, CG Boncelet Jr, and CT Retter. Spread spectrum image steganography. *IEEE Transactions on Image Processing*, 8(8):1075–1083, 1999.

[13] N. Provos. Defending against statistical steganalysis. In *Proceedings of the 10th conference on USENIX Security Symposium-Volume 10*, pages 24–24. USENIX Association Berkeley, CA, USA, 2001.

[14] Y.Q. Shi, C. Chen, and W. Chen. A Markov process based approach to effective attacking JPEG steganography. *LECTURE NOTES IN COMPUTER SCIENCE*, 4437:249, 2007.

[15] J. Smith and B. Comiskey. Modulation and information hiding in images. *Lecture Notes in Computer Science*, 1174:207–226, 1996.

[16] K. Solanki, K. Sullivan, U. Madhow, BS Manjunath, and S. Chandrasekaran. Statistical restoration for robust and secure steganography. In *IEEE International Conference on Image Processing, 2005. ICIP 2005*, volume 2, 2005.

[17] K. Sullivan, U. Madhow, S. Chandrasekaran, and B.S. Manjunath. Steganalysis of spread spectrum data hiding exploiting cover memory. In *Proc. SPIE*, volume 5681, pages 38–46, 2005.

[18] Derek Upham. Jpeg-jsteg. http://www.funet.fi/pub/crypt/steganography/jpeg-jsteg-v4.diff.gz.

[19] G.K. Wallace et al. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

[20] A. Westfeld and A. Pfitzmann. Attacks on steganographic systems. *Lecture notes in computer science*, pages 61–76, 2000.

[21] Andreas Westfeld. F5-a steganographic algorithm. In *IHW '01: Proceedings of the 4th International Workshop on Information Hiding*, pages 289–302. Springer-Verlag, 2001.

[22] H.C. Wu, N.I. Wu, C.S. Tsai, and M.S. Hwang. Image steganographic scheme based on pixel-value differencing and LSB replacement methods. *IEE Proceedings-Vision, Image and Signal Processing*, 152(5):611–615, 2005.