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 conceals the contents of a message but also its existence. 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 different 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 allows it to restore the histogram completely, making its detection impossible by any first order steganalysis. It gives better results with higher order steganalysis when compared to F5, Outguess and Steghide. 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, J3, Histogram Compensation, 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 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 similar to watermarking but with a different purpose. 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 in 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 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 the 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. Since JPEG is the most popular image format used over the Internet and by image acquisition devices, we use JPEG as our default 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 histogram analysis of the AC components of DCT coefficients [1] in the image. Jsteg, which simply changes the LSB of a coefficient to the value desired for the next embedded data bit [2], can be detected by the effect it has of equalizing adjacent pairs of coefficient values [3]. F5 attempts to retain the general shape of the histogram [4], 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 [5]. Outguess [6] and Steghide [7] use statistical restoration schemes to embed data in the LSB coefficients. Outguess uses a threshold which determines the amount of coefficients to be preserved to restore the histogram but can be broken by second order statistical steganalysis [8], [9]. Steghide uses a graph theory approach; it swaps the values of coefficients to embed data and is more robust than Outguess. Essentially, our algorithm also swaps the coefficients in pairs but does it very efficiently so that the coefficient count is restored as in the original image while maximizing capacity.

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 [10] and hence is resistant to chi-square attacks [3]. Our algorithm can restore the histogram of any JPEG image to its original values after the embedding process completes and can carry a payload of 0.4 to 0.7 bits per non-zero coefficient. 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. For example, $(2x, 2x+1)$ form a pair. This means that 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 an 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 remaining to restore histogram, it will stop encoding that pair and store its index location in the stop point section of the header. The header gives important details about the embedded data such as stop points, data length in bytes, dynamic header length, etc. At the end of the embedding process, coefficient restoration takes place which equalizes the individual coefficient count as in the original file. Since all the stop points can only be known after the embedding process, the header bytes are always encoded last on the embedder side whereas they are decoded first on the extractor side.

We compared our results with three popular algorithms namely, F5, Steghide and Outguess. The experimental results show that J3 has a higher embedding capacity than Outguess and Steghide. F5 has a higher capacity when the file size is small but J3 outperforms F5 in the steganalysis tests. We also estimated the theoretical embedding capacity with J3 using the cover medium and the results follow closely with the experimental outcome. The theoretical analysis has not been shown in this paper due to page limitation. We have also performed steganalysis based on support vector machines [11]. The results show that J3 has a 2% lower detection rate as compared to other algorithms.

The rest of the paper is organized as follows. Section II deals with some background and related work done in JPEG steganography. In Section III and IV, we discuss our proposed J3 embedding and extraction module in detail. Section V shows some experimental results results obtained using our algorithm and compares it with F5, Outguess and Steghide. Finally, section VI concludes the paper with reference to future work in this area.

II. BACKGROUND

A. 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 [1] is likely to introduce too much noise and result in too many errors during decoding of the embedded data. 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 [12].

B. Previous Work

Jsteg [2] 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 [13] 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 [3] since they equalize pairs of coefficients in a typical histogram of the image, giving a "staircase" appearance to the histogram as shown in Figure 1(b). The histogram before the application of JSteg is shown in Figure 1(a).

(a) Histogram appearance of a cover JPEG image.

(b) Histogram appearance after application of JSteg algorithm.

Fig. 1. Histogram appearance before and after application of JSteg algorithm.

Another popular algorithm, F5 [4], 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 histogram of other coefficients and hence can be detected once an estimate of the original histogram is obtained [5].

Our algorithm falls under the category of statistical restoration or preservation schemes [6], [7], [14]. Outguess, proposed by Niels Provos, was one of the first algorithms to use statistical restoration methods to counter chi-square attacks [6]. 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. 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 [8], [9].

Yet another popular algorithm is Steghide [7], which uses graph theory techniques to preserve the histogram. Two interchangeable coefficients are connected by an edge in the graph with coefficients as vertices of the graph. The message is then 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 about 6% with respect to the cover file size. Another technique of steganography proposed by Marvel et al. [15] 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.

III. J3 EMBEDDING MODULE

Figure 2 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 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, and coefficient with a value of 3 will only change to 2 to

Fig. 2. Block diagram of our proposed embedding module.

encode message bit 0. 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 the 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 $(-2n, -2n-1) \cdots (-2, -3)$, $(-1, 1)$, $(2, 3) \cdots (2n, 2n+1)$, where $2n+1$ and $-2n-1$ are the threshold limits for positive and negative coefficients, respectively.

Before embedding a data bit in a coefficient, the algorithm determines whether a sufficient number of coefficients of the other member of the pair are left to balance the histogram or not. 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 coefficient pair. The unused coefficients for that pair will be used later to compensate for the imbalance. The header bits are embedded after the data bits are embedded 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.

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 $64 N_B$. N_{bSP} represents the minimum number of bits needed to represent any number between 0 to $64 N_B$, which is $\log_2(64 N_B)$. Receiver can compute this from the file itself but has been included to provide more robustness during decoding.
- N_{SP} = represents the total number of stop points present in the header.

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

TABLE I
HEADER STRUCTURE FOR J3 ALGORITHM

- $SP(x, y)$ = represents a stop point. Each stop point occupies N_{bSP} bits in the header.

Terminology:

- $Hist(x)$: Total number of coefficient x initially present in the cover image.
- $TR(x)$: Remaining number of coefficients x which remain unused and untouched during embedding.
- $TC(x \rightarrow y)$: Total number of coefficient x changed to y during embedding.
- $TC(x \rightarrow x)$: Total number of coefficient x unchanged but used for data.
- $TT(x)$: Total number of coefficient x used at any point 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 $0 \leq x \leq 64 N_B$.
- $Coef_{f_{total}}$: total number of coefficients in the image = $64 N_B$.

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:

$$\begin{aligned} Hist(2) &= 500, TC(2 \rightarrow 3) = 100, TC(2 \rightarrow 2) = 100 \\ Hist(3) &= 200, TC(3 \rightarrow 2) = 50, TC(3 \rightarrow 3) = 100 \\ \Rightarrow TR(2) &= 300, TR(3) = 50 \end{aligned}$$

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 encountered during embedding is 2013 and $C_{2013} = 3$. If $TR(3) = Unbalance(3)$, then we know that we cannot encode any more data in pair (2,3) since we have just the minimum number of 3's remaining to balance it. 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 stop decoding coefficients 2 and 3 when it reaches index 2013.

A. Embedding Algorithm

Embedding is divided into 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. The number of bits required to store header information can be calculated before the embedding process. After skipping all the coefficients which will be used for header data in the end, algorithm 3 embeds the actual message 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 are known, algorithm 4 embeds the header bits using the same index sequence traversed in algorithm 2. Since algorithm 3 and 2 modify the coefficients, algorithm 6 calculates the net change in individual coefficients and restores the histogram to its original values using the unused coefficients. Negative coefficients and the (-1,1) pairs have not been considered in the algorithms below for simplicity but can be included with a slight modification.

Let P be the password shared between the sender and the receiver. This password is used to generate the seed for pseudo-random numbers between 0 and $64N_B$. The same password is also used for encrypting and decrypting the data. Let,

- $Enc(AES, M, k)$ = Encryption of message M using k as key with AES standard.
- THR = Lower bound on the total number of a coefficient, say x , to be used for embedding data. If the total number of coefficient x 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 .
- ϕ = JPEG AC coefficient in frequency domain.

IV. J3 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 to deal only with the exact index locations to stop decoding each coefficient pair. Password P is used to

Algorithm 1: Function *EmbedBit*(.).

```

begin
  Function EmbedBit (DataBit bit, index x)
  if  $C_x \in \text{odd} \wedge \text{bit} \equiv 0$  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  $C_x \in \text{even} \wedge \text{bit} \equiv 1$  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 = MD5(P); /* Generate seed using MD5 hashing for PRNG */
  ME = Enc(AES, M, P); /* Encrypt message M with P as the key with AES standard */
  for i = 2 to 255 do
    if Hist(i) < Thr then /* if total number of  $i_{th}$  coeff < threshold */
      coeff_limit ← i; /* coefficient limit to consider for encoding */
      break;
    end
  end
  if coeff_limit ∈ even then /* since a pair has to end with an odd number, add the next coefficient */
    coeff_limit ← 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. */
   $HDR_{total} = 20 + 5 + 5 + SP_{total} * Dec(N_{bSP})$ ; /* total header length in bits */
  /* Skipping coefficients for header bits initially for later embedding. */
  DataIndex = 0;
  while DataIndex ≤ HDR_total do
     $x = PRNG(seed, Coeff_{total})$ ;
    if  $C_x \leq coeff\_limit \wedge C_x \neq 0 \wedge C_x \in \emptyset$  then
       $TR(C_x) \leftarrow TR(C_x) - 1$ ; /* decrease remaining number of coeff for embedding */
    end
    DataIndex ← DataIndex + 1;
  end
end

```

Algorithm 3: Embed message bits.

```

begin
  DataIndex = 0;
  while DataIndex < ME_total do
     $x = PRNG(seed, Coeff_{total})$ ;
    if  $C_x \equiv 0 \vee C_x > coeff\_limit \vee C_x \notin \emptyset$  then
      continue; /* ineligible coefficient value, so fetch next random number */
    else if EvaluateStopPoint(x) ≡ false then
      EmbedBit(Bit(ME, DataIndex), x);
       $TR(C_x) \leftarrow TR(C_x) - 1$ ;
      DataIndex ← 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 ≤ HDR_total do
     $x = PRNG(seed, Coeff_{total})$ ; /* generate same sequence for header coeff. */
    if  $C_x \equiv 0 \vee C_x > coeff\_limit \vee C_x \notin \emptyset$  then
      continue; /* ineligible coefficient value, so fetch next random number */
    else
      EmbedBit(Bit(HDR, DataIndex), x);
      DataIndex ← DataIndex + 1;
    end
  end
end

```

Algorithm 5: Function *EvaluateStopPoint*(.).

Function *EvaluateStopPoint* (index *x*)

```

begin
  if  $C_x \in \text{odd}$  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  $C_x \in \text{even}$  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

```

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

generate the random number sequence used to permute the coefficient indices for visitation order. The constant part of the header is decoded first, which in turn 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 a given coefficient pair when its stop point matches the coefficient index encountered.

Once all the header bits have been extracted, the extraction process starts decoding the message bits, taking care to stop extraction from a coefficient pair when its stop point has been reached. 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 3.

A. Extraction Algorithm

The extraction algorithm is divided into two modules. Algorithm 7 first decodes the static part of the header to recover

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  $TC(i \rightarrow i+1) > TC(i+1 \rightarrow i)$  then
             $TC(i \rightarrow i+1) \leftarrow TC(i \rightarrow i+1) - TC(i+1 \rightarrow i)$ ;
             $TC(i+1 \rightarrow i) \leftarrow 0$ ;
        else
             $TC(i+1 \rightarrow i) \leftarrow TC(i+1 \rightarrow i) - TC(i \rightarrow i+1)$ ;
             $TC(i \rightarrow i+1) \leftarrow 0$ ;
        end
         $i \leftarrow i+2$ ;
    end
    /* Calculate the total change in histogram */
     $netChange = \sum_{k=1}^{SP_{total}} (TC(2k \rightarrow 2k+1) + TC(2k+1 \rightarrow 2k))$ 
    /* 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  $C_x \in even \wedge TC(C_x+1 \rightarrow C_x) > 0$  then
             $T(C_x+1 \rightarrow C_x) \leftarrow TC(C_x+1 \rightarrow C_x) - 1$ ;
             $C_x \leftarrow C_x + 1$ ;
             $netChange \leftarrow netChange - 1$ ;
        else if  $C_x \in odd \wedge TC(C_x-1 \rightarrow C_x) > 0$  then
             $T(C_x-1 \rightarrow C_x) \leftarrow TC(C_x-1 \rightarrow C_x) - 1$ ;
             $C_x \leftarrow C_x - 1$ ;
             $netChange \leftarrow netChange - 1$ ;
        end
    end
end

```

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.

V. RESULTS

The algorithm was implemented in Java which includes code to, 1) decode a JPEG image to get the JPEG coefficients, 2) embed data in eligible coefficients, 3) balance the histogram to its original values, and finally, 4) 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 1000 different JPEG color images of varying size and texture with a quality factor of 75. We use a quality factor of 75 since this is the default quality in OutGuess. Each of the image was embedded with random data bits using a randomly generated password. The password is used to generate the pseudo random number sequence for determining the traversal sequence for coefficients. The cover and stego image of a popularly used Lena image is shown in figure 4(a) and 4(b).

The histogram of the Lena image in figure 4 is shown in figure 5. The graph shows the histogram of the image before embedding, before compensation and after compensation*. The before compensation bars show that the odd coefficients

*Although the histogram looks symmetrical, values were obtained using an experimental setup.

Algorithm 7: Extraction of header bits.

```

begin
     $Dec(AES, M, k)$  = Decryption of Message M using k as key with AES 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 */
     $HDR_{static} = 20 + 5 + 5$ ; /* 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
             $HDR_i \leftarrow 1$ ;
             $i \leftarrow i + 1$ ;
        else if  $C_x \in even$  then
             $HDR_i \leftarrow 0$ ;
             $i \leftarrow i + 1$ ;
        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
     $M_{out} = Dec(AES, M, k)$ ; /* Decrypt message M using key k and AES standard */
end

```

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 increase 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. The "after compensation" bars show the status of the histogram after compensation is done. We thus verify experimentally that there is zero deviation in the

(a) Lena Cover, File Size = 44KB (b) Lena Stego, File Size = 44KB, Payload= 5019 Bytes

Fig. 4. Comparison of Cover image with Stego image

Fig. 5. Comparison of Lena histogram at different stages of embedding process.

histogram after the compensation process is completed.

A. Estimated Capacity vs Actual Capacity

The graph in figure 6 compares the estimated capacity with the actual capacity for 1000 different images. In conclusion, our estimated capacity is almost equal to the actual capacity, which supports the correctness of the theoretical analysis of capacity estimation using J3. As mentioned before, we have not shown the theoretical analysis due to page limitation, but interested readers can request a copy of it separately. 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 is 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. Embedding Efficiency of J3

Graph in figure 7 shows the embedding efficiency with respect to the number of data bits embedded per pixel(bpp) and

Fig. 6. Comparison of estimated capacity with actual capacity using J3

bits embedded per non-zero coefficients($bpnz$). The general trend in the graph shows that the average bpp is around 0.16. Since a sudden increase in the number of pixels will not lead to the same amount of increase in embedded bits, a peak in the number of pixels will result in a valley in the bpp curve and vice-versa. The other part of the graph shows that $bpnz$

Fig. 7. Embedding efficiency of J3 for bits per pixel and bits per non-zero coefficient

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 and valleys in the curve are due to the variation in the number of non-zeros which is also shown. Since an increase in the number of non-zeros will not lead to the same amount of increase in number of embedded bits, it will result in a decrease in $bpnz$. Hence, the peaks in the non-zero will result in valley in the $bpnz$ curve.

C. Comparison of J3 with other algorithms

In this experiment, we took the same 1000 JPEG images of various size and texture for embedding data to its maximum capacity using J3, F5, Steghide and OutGuess algorithms. The comparison graph is shown in figure 8. 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 a better data capacity than Outguess and Steghide when the image size is small, and it performs better than F5 in some cases with larger image size. 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. 8. Comparison of embedding capacity of J3 with other algorithms

D. Steganalysis Results

Our steganalysis experiments are based on Support Vector Machines (SVM) for classification of images embedded with the following stego algorithms: OutGuess, F5, Steghide and J3 along with the cover images. We use soft margin SVM (C-SVM) with RBF (Radial Basis Function) kernel, which is one of the most popular choices of kernel type for support vector machines. The experiments use a feature extractor which extracts 274 merged Markov and DCT features for multi-class steganalysis as mentioned in [11]. The results show that J3 has 2% lower detection rate than the other three algorithms with maximum embedding capacity. In fair steganalysis, where we embedded equal amount of data using each algorithm, J3 has a 3%-4% lower detection rate than the other three algorithms. Detailed results are not shown here due to page limitation.

VI. CONCLUSION

J3 is a new JPEG steganography algorithm that uses stop-point technique with LSB encoding to embed data efficiently and histogram compensation to balance all the coefficients changed during the embedding process. J3 only swaps the non-zero coefficients in pairs, which ensures 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 higher than Outguess and Steghide, with

the added benefit of an unchanged histogram. The embedding rate of J3 ranges between 0.16 *bpp* and 0.65 *bpnz*, which is significant for a good steganography algorithm. Steganalysis results based on higher order statistics using SVM shows that J3 has 2% lower detection rate than F5, Outguess and Steghide which is a significant improvement for steganography.

In the future, we plan to improve on this algorithm to increase its stealthiness and perform detailed higher order statistical steganalysis using machine learning techniques. We also plan to use matrix encoding techniques to improve embedding efficiency when the message length is 50% or smaller than the maximum capacity.

ACKNOWLEDGMENT

The authors would like to thank Dr. Josè Fortes (CISE Department at University of Florida) and Ira Moskowitz (U.S. Naval Research Labs) for their valuable comments and feedback.

REFERENCES

- [1] A. C. Hung. (1993, November) Pvrge-jpeg codec 1.1. [Online]. Available: <http://www.dclunie.com/jpegge/jpegpvrg.pdf>
- [2] D. Upham. Jpeg-jsteg. [Online]. Available: <http://www.funet.fi/pub/crypt/steganography/jpeg-jsteg-v4.diff.gz>
- [3] A. Westfeld and A. Pfitzmann, "Attacks on steganographic systems," *Lecture notes in computer science*, pp. 61–76, 2000.
- [4] A. Westfeld, "F5-a steganographic algorithm," in *IHW '01: Proceedings of the 4th International Workshop on Information Hiding*. Springer-Verlag, 2001, pp. 289–302.
- [5] J. Fridrich, M. Goljan, and D. Hoge, "Steganalysis of JPEG images: Breaking the F5 algorithm," *Lecture Notes in Computer Science*, pp. 310–323, 2003.
- [6] N. Provos, "Defending against statistical steganalysis," in *Proceedings of the 10th conference on USENIX Security Symposium-Volume 10*. USENIX Association Berkeley, CA, USA, 2001, pp. 24–24.
- [7] S. Hetzl and P. Mutzel, "A graph-theoretic approach to steganography," *Lecture Notes in Computer Science*, vol. 3677, p. 119, 2005.
- [8] J. Fridrich, M. Goljan, and D. Hoge, "New methodology for breaking steganographic techniques for JPEGs," *Submitted to SPIE: Electronic Imaging*, 2003.
- [9] Y. Shi, C. Chen, and W. Chen, "A Markov process based approach to effective attacking JPEG steganography," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 4437, p. 249, 2007.
- [10] E. Franz et al., "Steganography preserving statistical properties," *Lecture notes in computer science*, pp. 278–294, 2003.
- [11] T. Pevny and J. Fridrich, "Merging Markov and DCT features for multi-class JPEG steganalysis," *Security, Steganography, and Watermarking of Multimedia Contents IX*, pp. 1–13, 2007.
- [12] R. Chandramouli and N. Memon, "Analysis of LSB based image steganography techniques," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, vol. 3, 2001.
- [13] Jp hide&seek. [Online]. Available: <http://linux01.gwdg.de/~alatham/stego.html>
- [14] K. Solanki, K. Sullivan, U. Madhow, B. Manjunath, and S. Chandrasekaran, "Statistical restoration for robust and secure steganography," in *IEEE International Conference on Image Processing, 2005. ICIP 2005*, vol. 2, 2005.
- [15] L. Marvel, C. Boncelet Jr, and C. Retter, "Spread spectrum image steganography," *IEEE Transactions on Image Processing*, vol. 8, no. 8, pp. 1075–1083, 1999.