

J2: REFINEMENT OF A TOPOLOGICAL IMAGE STEGANOGRAPHIC METHOD

Richard E. Newman,¹ Ira S. Moskowitz² and Mahendra Kumar¹

¹CISE Department

University of Florida

Gainesville, FL 32611-6120

email: {nemo, makumar}@cise.ufl.edu

²Center for High Assurance Computer Systems, Code 5540

Naval Research Laboratory

Washington, DC 20375

email: moskowitz@itd.nrl.navy.mil

Abstract

Prior work described the baseline version J1 of a spatial domain image steganographic embedding based on manipulation of the frequency domain, and presented a topological approach to the problem. This paper modifies the embedding and extraction processes of J1 to allow for larger payload. The improved steganographic algorithm is called J2, and is more robust to steganographic detection than J1. J2 lends itself to probabilistic analysis (done elsewhere) due to the nature of the modifications to the embedding and extraction methods.

1 Introduction

An earlier paper describes a novel spatial image stego embedding “J1” based on topological concepts and using a pseudo-metric operating in the frequency domain [1]. Since the J1 stego image was produced from blocks of valid JPEG coefficients, but the embedded data were extracted from the spatial image, it could be stored in either the frequency domain as a JPEG [2] file, or in the spatial domain as a bitmap. Furthermore, even the extremely sensitive JPEG compatibility steganalysis method [4] cannot detect J1 manipulation of the spatial image. However, J1 may be detected easily by other means. J1 was meant to show a proof of concept, whereas J2 is a more pragmatic solution that let us move from the lab bench to the real world.

Aside from the stealthiness of J1 another important item remaining was estimation of the stego payload size [3] of the cover image, since it is possible that some of the blocks may not be usable to store the embedded data. That analysis was complicated by the nature of the data extraction function and the way the neighborhood of a cover image block was generated; together these rendered correlations possible in the data encoded by a block and its neighbors, so that it was difficult to predict whether or not a block would be usable. This paper suggests some modifications to the original algorithm, still retaining the topological underpinning, that rectify this situation. The refined

stego algorithm is named “J2”.

The major idea behind the extension and improvement of J1 to J2 is to make the datum embedded strongly and “randomly” dependent on all spatial bits in the block. This is done by applying a cryptographic hash to the 64 bytes of each 8×8 block¹ to produce a hash value, from which any number of bits may be extracted (limited by the ability to produce the desired bit pattern). Thus any change to the spatial block produces apparently random changes to the datum the block encodes. By randomizing the output of the extraction function, we may then legitimately analyze the embedding methods probabilistically.

2 Review of J1

This section reviews the baseline J1 algorithm version of the topological approach that encodes data in the spatial realization of a JPEG, but manipulates the JPEG coefficients themselves to do this [1]. By manipulating the image in the frequency domain, the embedding will never be detected by JPEG compatibility steganalysis [4]. The J1 system stores only one bit of embedded data per JPEG block (in 8-bit, grayscale images). Its data extraction function, Φ , takes the LSB of the upper left pixel in the block to be the embedded data. A small, fixed size length field is used to delimit the embedded data. Encoding is done by going back to the DCT coefficients for that JPEG block and changing them slightly in a systematic way to search for a minimally perturbed JPEG compatible block² that embeds the desired bit, hence the topological concept of “nearby.” The changes have to be to other points in dequantized coefficient space (that is, to sets of coefficients D_j for which each coefficient

¹We restrict our description to grayscale image in this paper, but our method is applicable to and has been extended to work on color images also.

²Spatial domain blocks that *are* the result of decoding a set of JPEG coefficients for a given quantization table are called JPEG compatible blocks, or JPEG blocks for short, while those that are not are called JPEG incompatible, or non-JPEG blocks. Note that blocks that are JPEG incompatible for one quantizing table may be JPEG compatible for a different quantizing table, and vice-versa.

$D_j(i)$, $i = 1, \dots, 64$ is a multiple of the corresponding element of the quantization table, $QT(i)$). This is depicted in Figure 1, where B' is the raw DCT coefficient set for some block F_0 of a cover image, and D' is the set of dequantized coefficients nearest to B' .³

The preliminary version changes only one JPEG coefficient at a time by only one quantization step. In other words, it uses the L_1 metric on the points in the 64-dimensional quantized coefficient space corresponding to the spatial blocks, and threshold Θ set to a maximum distance of unity. (Note that this is different from changing the LSB of the JPEG coefficients by unity, which only gives one neighbor per coefficient.) *For most blocks, a change of one quantum for only one coefficient produces acceptable distortion for the HVS.* This results in between 65 and 129 JPEG compatible neighbors⁴ for each block in the original image.

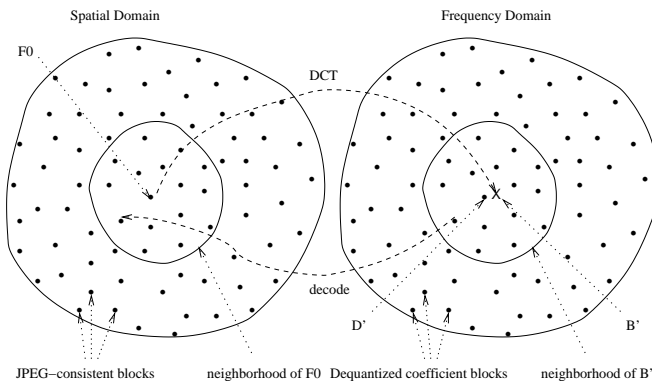


Figure 1. Neighbors of $DCT(F_0)$ in Dequantized Coefficient Space.

If there is no neighboring set of JPEG coefficients whose spatial domain image carries the desired datum, then the block cannot be used. The system could deal with this in a number of ways. In the baseline J1 system, the sender alters unusable blocks in such a way that the receiver can tell which blocks the sender could not use without the sender explicitly marking them. If there is some datum that a block cannot encode, then we call that block “poor.” Rich blocks are those that can encode *any* datum. If a block cannot encode the desired datum σ , then it must be poor, so we would like the receiver to skip these blocks. Since every block can be extracted to yield *some* datum, the receiver has to test each block to see if it is poor or not. The receiver determines if the next block to be decoded could have encoded any datum (i.e., was “rich”) or not (i.e., was “poor”).

³For quantized DCT coefficients or for DCT coefficient sets, dequantized or raw, we will use the L_1 metric to define distances. A neighbor of a block D is any other block D' that is at an L_1 metric distance of less than or equal to some threshold, Θ , from D .

⁴Changes are actually done in quantized coefficient space. Each of the 64 JPEG coefficients may be changed by +1 or -1, except those that are already extremal. Extremal coefficients will only produce one neighbor, so including the original block itself, the total number of neighbors is at most 129, and is reduced from 129 by the number of extremal coefficients.

Rich blocks are decoded and poor blocks are skipped. This means that the sender cannot just encode σ in a block and move on, but must encode σ in a rich block. If no rich block can be found that encodes σ , then the block is still not usable, even though it encodes σ ; the receiver will skip over it. Thus the sender must always encode valid data in rich blocks (after embedding) or if this is not possible, signal the receiver to skip the block by making sure the unusable block is poor (if the unusable block itself is rich it is substituted by a nearby poor block, if the unusable block is poor it is left alone).

In the first definition of usable for that system (described above), we only considered blocks that had a rich neighbor for every possible datum to be “usable.” Later, this condition was relaxed by considering what datum we desired to encode with the block, so that usability depended on the embedded data. In this case, a block was considered usable if it had some rich neighbor that encoded the desired datum. These are rather subtle points that have been addressed in [1]. What the reader should keep in mind is that J1 has several simple and subtle ways to make sure that the receiver would only decode blocks that contained stego payload.

3 Motivation for Probabilistic Spatial Domain Stego-embedding

The baseline J1 version of the embedding algorithm hid only one bit per block, and so the payload size was very small. Further, although it is likely that the payload rate (in bits per block) could have been increased, there remained two difficulties. First, use of a simple extraction function renders the encoded data values unevenly distributed over the neighbors of a block, and so there could be considerable nonuniformity in the data encoded by the blocks of a neighborhood. This made it difficult to predict whether or not a block would be usable, and hence made analysis complicated. This effect was most problematic when small quanta were used in the quantizing table, when small changes to the spatial data might not produce any change in the extracted data.

Second, both the sender and the receiver had to perform a considerable amount of computation per block in order to embed and to extract the data, respectively. The sender had to test each block for usability, that is, make sure that each block could be used to encode any datum in a rich block. This in turn meant that each block’s neighbors had to be produced, decoded, and the datum extracted, and the block tested for richness, until a rich block for ever datum was found. To test for richness, a block’s neighbors had to be produced, decoded, and the datum extracted to see if every datum could be encoded by at least one of the block’s neighbors. This process continued until a rich neighbor for each datum was found, (i.e., the block was usable) or all the neighbors had been tested (i.e., there was some datum that could not be encoded by a rich neighbor block, so the

block was not usable). Likewise, the receiver had to test each block to determine if it was rich or not, by producing, decoding, and extracting the datum from each neighbor until it was either determined that the block was rich or all the neighbors had been tested. For a small data set (e.g., binary for the baseline J1 as discussed above), this could be fairly fast, but to extend J1 to deal with larger data sets it could be quite costly (the sender’s number of change, decode, extract operations grows roughly as 2^{2k} for k bits per datum, and as 2^k for the receiver).

Both of these limitations created significant problems when the data set became larger. The first caused the likelihood of finding a usable block to decrease and for this to become unpredictable. The second meant that the computational burden would become too great as the neighborhood size increased (by increasing the threshold Θ for the neighborhood radius) to accommodate larger payloads. To overcome these problems, we modified the baseline J1 approach as described in the following section.

4 J2 Stego Embedding Technique

In order to provide a block datum extraction mechanism that is guaranteed to depend strongly and randomly on each bit of the spatial block, we apply a keyed, secure hash function $H_K(\cdot)$ to each spatial block to produce a large number of bits, from which we may extract as many bits as the payload rate requires. (Note that the key, K , is known through some out of band means to the sender and the receiver.) This causes the set of data values encoded by a neighborhood to be, in effect, a random variable with uniform distribution. Not only does this make it more likely that a neighbor block encoding the desired datum will be found, but it makes probabilistic analysis possible, so that this likelihood can be quantified. In addition, it makes it easy to hide the embedded data without encrypting it first. Finally, we use the keyed hash and a block of the file to produce a random number sequence that is used to permute the order in which the blocks are visited, making it more difficult to detect changes by comparing different pieces of the image file.

The problem of computational load remained, and with larger data sets it promised to make the method impractical. To circumvent this problem, we observed that, with a sufficiently small data set and sufficiently large threshold, and hence neighborhood, the likelihood of encountering unusable blocks was very small. If this probability is small enough, then the receiver can assume that all blocks are usable and the sender just has to guarantee that all the blocks used to send the embedded data do, in fact, encode the correct datum.

Computationally, both the sender’s and the receiver’s jobs are made much simpler. The receiver’s job is simplified, as the receiver just extracts the embedded data from the blocks of the stego image in the permuted order until the embedded string is recovered. The sender’s job is also made simpler: the sender just has to find a neighbor of

each block in the permuted order that encodes the desired datum, or start over again if this can’t be done. In particular, the sender does not have to test for a given block if it is rich, much less usable, so the search of neighboring blocks is much faster and can stop as soon as a neighbor that encodes the desired datum is found.

5 Formalizations of Our Stego Embedding Technique

This section describes formally how our modified method hides an arbitrary embedded data string in the spatial realization of a JPEG image. The embedded data must be self-delimiting in order for the receiver to know where it ends, so at least this amount of preprocessing must be done prior to the embedding described. In addition, the embedded data may first be encrypted (although this seems unnecessary if a keyed secure hash function is used for extraction), and it may have a frame check sequence (FCS) added to detect transmission errors.

Let the embedded data string (after encryption, end delimitation, frame check sequence if desired, etc.) be $s = s_1, s_2, \dots, s_K$. The data are all from a finite domain $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_N\}$, and $s_i \in \Sigma$ for $i = 1, 2, \dots, K$. Let $\tau : \Sigma^* \rightarrow \{0, 1\}$ be a termination detector for the embedded string, so that $\tau(s_1, s_2, \dots, s_j) = 0$ for all $j = 1, 2, \dots, K - 1$, and $\tau(s_1, s_2, \dots, s_K) = 1$. Let $S = [0..255]^{64}$ be the set of 8×8 spatial domain blocks with 8 bits per pixel (whether they are JPEG compatible or not), and let $S_{QT} \subseteq S$ be the JPEG compatible spatial blocks for a given quantization table QT .⁵ Let Φ extract the embedded data from a spatial block F , $\Phi : S \rightarrow \Sigma$. In J1, the extraction function is $\Phi_{n,bas}(F) = \text{LSB}_n(F[0, 0])$, that is, the n LSBs of the upper, leftmost pixel, $F[0, 0]$. (In our proof-of-concept program, $n = 1$ [1].) For the probabilistic algorithms, the extraction function is $\Phi_{n,prob}(F) = \text{LSB}_n(H(F|X))$, the n LSBs of the hash H of the block F concatenated with a secret key, X .

Let μ be a pseudometric on S_{QT} , $\mu : S_{QT} \times S_{QT} \rightarrow \mathbb{R}^+ \cup \{0\}$. In particular, we will use a pseudometric that counts the number of places in which the quantized JPEG coefficients differ between two JPEG blocks, if that difference is at most one; if differences greater than one are scaled so that two blocks whose JPEG coefficients differ by at most x are always closer than two blocks with even one coefficient that differs by more than x . For example, let B and B' be two JPEG blocks, with $B[i, j]$ denoting the (i, j) th AC coefficient. If $B[1, 3] = B'[1, 3] + 1$, $B[2, 2] = B'[2, 2] - 1$, and all other coefficients are exactly the same, then $\mu(B, B') = 2$. If all differences are no greater than one, then the maximum distance between two JPEG blocks is 64. Now if

⁵Here, the notation $[a..b]$ denotes the set of integers from a to b , inclusive,

$$[a..b] \stackrel{\text{def}}{=} \{x \in \mathbb{Z} \mid a \leq x \leq b\},$$

and as usual, for a set S , S^n denotes the set of all n -tuples taken over S .

$B[1, 3] = B'[1, 3] + 2$ and all other coefficients are exactly the same, then $\mu(B, B') = 65$. This is done so that many, less perceptible changes will be made before a single, more perceptible change is made. In practice, we will constrain ourselves from ever changing any coefficient by more than unity.

Let $N_{\Theta}(F)$ be the set of JPEG compatible neighbors of JPEG compatible block F according to the pseudometric μ and threshold Θ based on some acceptable distortion level (μ and Θ are known to both sender and receiver),

$$N_{\Theta}(F) \stackrel{\text{def}}{=} \{F' \in S_{QT} \mid \mu(F, F') < \Theta\},$$

where QT is the quantizing table for the image of which F is one block. Θ is chosen small enough so that the HVS cannot detect our stego embedding technique. Neighborhoods can likewise be defined for JPEG coefficients and for dequantized coefficients for a particular quantizing table (by pushing the pseudometric forward).

If $F' \in N_{\Theta}(F)$, we say that F' is a (μ, Θ) -neighbor or just *neighbor* of F (the Θ is usually understood and is not explicitly mentioned for notational convenience). Being a neighbor is both reflexive and symmetric.

The first modification that we make to the baseline J1 encoding is to change the data extraction function, Φ . If it has been decided to use n bits per datum, then Φ takes the n least significant bits of the hash of the spatial block, taken as a string of bytes in row-major order⁶, concatenated with a secret X (X is just a passphrase of arbitrary length - it will always be hashed to a consistent size for later use). This has the effect of randomizing the encoded values, so that probabilistic analysis is possible. It also has the effect of hiding and randomizing the embedded data, so that they do not need to be encrypted first. Lacking the secret X , the attacker will not be able to apply the data extraction function and so will not be able to discern the embedded data for any block, so it will be impossible for the attacker to search for patterns in the extracted data. Further, even if the embedded data are known, the attacker will have to try to guess a passphrase that causes these data to appear in the outputs of the secure hash function $H(\cdot)$, which is very hard. In all other respects, the algorithm is the same as the baseline J1 algorithm.

A second modification we make is to randomize the order in which the blocks are visited, further confounding the attacker. To do this, the hash of the secret passphrase K is used with a block from the stego image to generate a pseudorandom number sequence that is then converted into a permutation of indices of the remaining blocks. This permutation defines the walk order in which the blocks are visited for encoding and decoding. Without the the walk order, the attacker does not even know which blocks may hold the embedded data, and so statistics must be taken on the image as a whole, making it easier to hide the small changes we make.

⁶That is, the bytes of a row are concatenated to form an 8-byte string, then the 8 strings corresponding to the 8 rows are concatenated to form a 64-byte string.

6 Results

We have implemented the described stego algorithm, and have tested it on a number of images with the number of bits per block ranging from one to eight. A value of $\Theta = 2$ sufficed. MD5 was used as the hash function, and the images and histograms shown here are for eight bits of data embedded per block. A log file was used for embedded data, although it really does not matter what the nature of the embedded data are (they could be all zeros) due to the way extraction works. The images were perceptually unaltered, and the histograms of the stego image were nearly identical to those of the cover image. Typical results for all quantized JPEG coefficients are shown in Figures 3 (omitting zero coefficients since these dominate the other coefficient values to the point of obscuring the differences) and 2 (which highlights the "interesting" changes).

Not unexpectedly, the number of zero coefficients is decreased slightly (less than 3%) and the numbers of coefficients with value -1 or 1 is accordingly increased (by 20-30% in this case) as shown in Figure 2. This is because the vast majority of quantized JPEG coefficients have zero value, so randomly changing a coefficient by $+/- 1$ can be expected to remove many more zeros than it adds. Of course, the values of +1 and -1 are increased accordingly, with a relatively small number of +1 and -1 coefficients changed to zero or $+/- 2$. All other coefficient values with reasonable occurrence were changed by less than $+/- 10\%$, most by less than $+/- 5\%$ (see Figure 3).

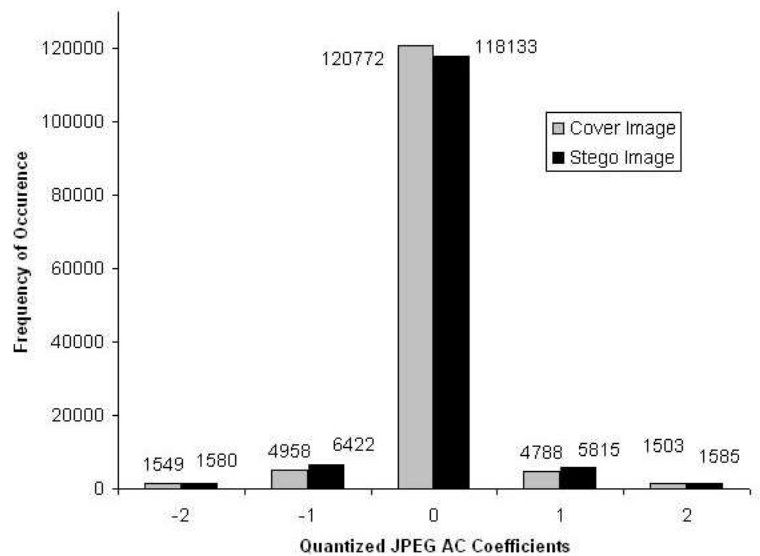


Figure 2. Histograms of cover and stego file: zero, ± 1 , ± 2 coefficients

An example image is also included here as a demonstration. The image in Figure 4 is an unaltered cover file, while the image in Figure 5 is the same file with embedded data encoded at a rate of eight bits per block, using almost all the blocks.

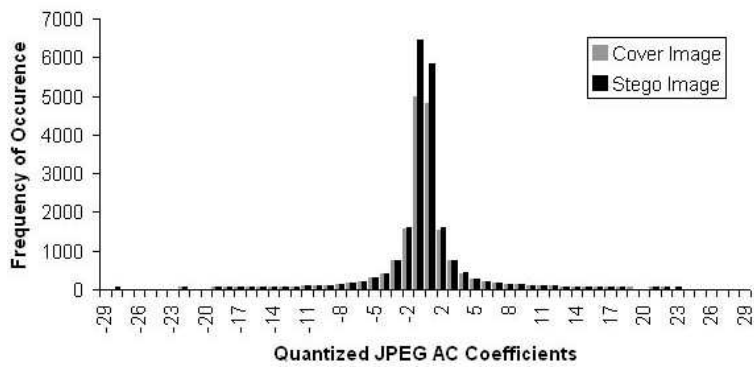


Figure 3. Histograms of cover and stego file ignoring zero coefficients



Figure 4. Cover file

7 Conclusions

This paper has briefly discussed the baseline stego embedding method J1 introduced in prior work to circumvent detection by the JPEG compatible steganalysis method. It then discussed some shortcomings of the baseline approach, and described a modified version that overcomes these problems (to some extent). Our new method still cannot be detected by JPEG-compatibility steganalysis, and the changes to the spatial domain and to the JPEG coefficient histograms are so small that without the original, it would be very difficult to detect any abnormalities. It would be relatively easy to modify the program to avoid changing zero coefficients, and to bias changes to non-zero coefficients toward zero to reduce the effects this simple version has.

The method is quite fragile, and any change to a spatial domain block (or to a JPEG block) will certainly randomize the corresponding extracted bits. Hence, we expect



Figure 5. Stego file

that the method will be very difficult to detect, but relatively easy to “scrub” [5] using active measures.

References

- [1] R. E. Newman and I. S. Moskowitz and L. Chang and M. M. Brahmadessam. A steganographic embedding undetectable by JPEG compatibility steganalysis. In F.A.P. Petitcolas, editor, *Information Hiding, Proc. 5th*, volume LNCS 2578, pages 258–277. Springer, 2002.
- [2] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1993.
- [3] I. S. Moskowitz and L. Chang and R. E. Newman. Capacity is the wrong paradigm. In *Proc. New Security Paradigms Workshop*, pages 114–126, 2002.
- [4] J. Fridrich and M. Goljan and R. Du. Steganalysis based on JPEG compatibility. In Jr. A. Tescher and B. Vasudev and V.M. Bove, editor, *SPIE Vol. 4518, Special session on Theoretical and Practical Issues in Digital Watermarking and Data Hiding, SPIE Multimedia Systems and Applications IV*, pages 275–280, Denver, CO, 20–24 August 1998.
- [5] I.S. Moskowitz and P. Lafferty and F. Ahmed. Stego scrubbing—a new direction for image steganography. to appear: *Proc. IEEE IAW 2007*, June 2007.