Searching for Gemstones: Flawed Stegosystems May Hide Promissing Ideas

Evgnosia-Alexandra Kelesidis[1 [0000−0002−2569−5705]], Diana Maimuţ[1 [0000−0002−9541−5705]], and Ilona Teodora Ciocan[1 [0000−0003−4297−1537]]

Advanced Technologies Institute
10 Dinu Vintilă, Bucharest, Romania
{alexandra.kelesidis, diana.maimut, ilona.ciocan}@dcti.ro

Abstract. The historical domain of information hiding is alternatively used nowadays for communication security. Maybe the oldest and certainly one of the most studied field that falls in this category is steganography. Within the current paper, we focus on image steganography techniques in the case of the JPEG format. We propose a corrected and optimized version of the J3 stegosystem which, to the best of our knowledge and as shown throughout the paper, may be considered a very good choice in terms of security and efficiency as compared to current solutions. We reconstruct the entire J3 algorithm (pre-processing, message insertion and extraction) and detail all the modifications. We also present implementation optimizations and cover all practical cases while still using the maximum image insertion capacity.

Keywords: Information hiding, steganography, stegosystem, image steganography, JPEG, steganalysis.

1 Introduction

As communication methods have been constantly evolving together with the emerging technologies, the need of securing the transmitted data has grown exponentially. Closely related to this fact both cryptography and steganography have become undoubtable necessities for ensuring data protection. There is a clear difference between the goals of cryptography and steganography: cryptography's purpose is hiding the content of a message by means of various mathematical methods, while steganography's purpose consists of hiding the existence of a message or to minimize the probability of being discovered. Nonetheless, within the state-of-the-art digital stegosystems the message has to be either encrypted or at least randomized and then inserted into redundant areas of a particular file format.

Steganography [48] is used for hiding a secret message by means of an ordinary object seen as a cover. The goal is to keep the data secret even if the

cover is analysed by means of various methods[1]. Thus, a steganographic system (stegosystem) has to ensure the transportation of a message by means of a cover object. These systems can be classified as classical or modern, where those considered modern take advantage of the vast existence and usage of digital files.

Modern steganography uses different digital carriers for message hiding from simple text to images, audio or video. Unusual covers can be adopted by means of using linguistic or natural language steganography for hiding secret messages into text corpuses. Vector objects, computer generated meshes and any other graph structured data, such as XML documents, relational databases, CAD documents can also be used successfully.

In this paper we mainly focus on images as cover objects, as they are considered very common [16] and also practical from the implementation point of view. Using digital images as carriers depends on the image encoding technique. Common types of image encodings are Graphics Interchange Format (GIF), which consists of a simple encoding in red, green and blue (RGB) colors for each pixel, and Joint Photographic Experts Group (JPEG) which is further detailed in Section 2 as it represents the type of encoding that interests us for the current paper.

One of the most common methods of digital steganography is the usage of the Least Significant Bit (LSB) from a particular piece of data. More exactly, we can use the LSB in order to transport bits from the secret message. This works due to the fact that this bit is considered redundant, and its modification does not visually alter the cover object. This is most common due to the ease of implementation and the fact that the changes that appear in the image by using this technique are too small to be noticed by the human eye, so the message actually becomes hidden. However, it also has major disadvantages such as the impossibility of using a lossy compression format (the information would change with compression) and the fact that it is not resistant to steganalysis [9]. Furthermore, applying error-correction codes can improve this technique as it can be detected and fixed on the fly by various visualization software, making it impossible to detect for the human eye. Various other techniques are specifically created for a certain file type.

Another steganographic technique is the frequency hiding technique which is suitable for JPEG images.

Related Work. According to [35], until 2010, among the best known steganographic algorithms that used the technique of inserting message bits into the Discrete Cosine Transformation (DCT) coefficients of an image were F5 [49], OutGuess [41] and Steghide [26]. However, their message insertion rate is too low for practical purposes and steganalysis methods[2] have been successfully applied to break them [22, 24, 25].

[1] even though the current good practice is to delete the original cover object after inserting the message and obtaining the resulting file

[2] which allow the detection of the presence of embedded information

Also, besides the previously discussed stegosystems, the nsF5 [21] version of F5 appeared before the scheme presented in [35]. Steganalysis methods for nsF5 are publicly available [2]. In addition, [2] is a tool that applies also to OutGuess, Steghide and J-UNIWARD.

After the publication of J3, other related steganographic algorithms such as HUGO [20], WOW [27], the initial UNIWARD family [28], J2-UNIWARD [15], and NS in JPEG [14] were published. Nonetheless, various steganalysis papers regarding these stegosystems were presented in the literature: [36] for HUGO, [44] for WOW, [34] for J-UNIWARD. Moreover, various characteristics and vulnerabilites of these stegosystems are studied in detail in [16].

Motivation. As already underlined before, a variety of steganographic algorithms have been proposed in the literature and used in practice. Most of them are vulnerable to steganalysis methods. We are particularly interested in the J3 algorithm [35].

We consider that given the lack of clarity in the presentation of J3 and the fact that its implementation is not publicly available, software engineers have turned their attention especially to other stegosystems when developing a steganographic application, even though these schemes may be weaker[3] and more inefficient[4]. We strongly believe that two of the characteristics of J3 are a must for the current design of secure stegosystems: histogram restoration[5] in order to resist steganalysis methods and high embedding capacity. However, we have observed various issues of the previously mentioned algorithm and, thus, we aimed at correcting and improving it.

Moreover, the current publicly available steganography software applications use either outdated steganographic algorithms or cryptographic schemes (as the first data protection layer). Among them we mention OpenStego [5] (which supports two steganographic plugins, LSB and randomized LSB and uses DES [40] for the initial encryption step), Hide'N'Send [6], in which data hiding is done using the standard LSB algorithm and StegoShare [3], where the cover file manipulation algorithm used is based on fixed location LSB insertion that makes its output pictures detectable to most steganalysis software. Given the previously mentioned facts, we believe that new steganographic software tools are a must. We provide the reader with a sketch of such an application within this paper in Appendix B.

Structure of the Paper. In Section 2, we introduce the notations and briefly describe image compression with a focus on JPEG and the J3 stegosystem. We also add pseudocode for the insertion and extraction algorithms of J3 in Appendix A. The main results are discussed in Section 3, namely corrections and enhancements of the J3 algorithm. Details regarding the implementation of the previously mentioned algorithm are presented in Section 4. We conclude and provide the reader with future work ideas in Section 5.

[3] in terms of security

[4] when implemented in software

[5] the distribution of bytes in the image is not affected

2 Preliminaries

2.1 Notations

Throughout this paper, we denote by $Hist(x)$ the total number of coefficients x initially present in the cover image. Let W and H be the image width and height, respectively. We further consider that $C[i]$, $0 \leq i < W \cdot H$ is the array of DCT coefficients of the image. We denote by $||m||$ the bit-length of m.

2.2 Image Compression

An image is a collection of numerical data that constitutes different intensities of light. This numerical representation forms a two-dimensional matrix with pixel elements. A pixel consists of one byte if the image is black and white or 3 bytes if the image is color, in RGB format.

As some images are too large to be transmitted in time, there have been proposed various algorithms that use mathematical formulas to reduce the size of files. We refer to these methods as compression techniques. In the case of images, there are two types of compression algorithms: lossless and lossy.

JPEG [29,38] is a lossy compression algorithm that consists in both types of components. The lossy stages include transformations to DCT [33] coefficients by applying the DCT transform followed by quantizations, and the actual compression is done without loss with the help of Huffmann encoding. Thus, steganography can be introduced between the two stages. Through the principles derived from LSB substitution, messages can be hidden in the least significant color bits of the coefficients resulting before Huffman encoding. The message becomes very difficult to detect, being inserted at this stage in the frequency range.

Lossless compression reduces image size by using approximations and partial data deletion as opposed to lossless compression that eliminates only redundant data. Thus, lossless compression ensures complete recovery of the original data while lossy compression allows an approximation to be reconstituted.

JPEG. JPEG has been subject to various patents and standardization processes for over 30 years now and it is widely adopted.

A JPEG file may be characterized by various encoding methods. The most common one in this case is JPEG File Interchange Format (JFIF) [4]. We further recall the steps of this encoding process.

When compressing an RGB image, the following steps are performed:

1. The preprocessing step consisting of converting the image from the RGB color space to the Y, C_r and C_b (*i.e.* luminance and chrominance) color space followed by decomposing the pixel matrices corresponding to each channel into 8×8 pixel blocks;
2. The lossy compression sub-algorithm that takes place in two steps and operates on the resulted pixel blocks:

- Applying the DCT transform [8] that maps the values of a pixel block into real numbers that represent essential information about the image. More precisely, 64 functions are applied to an 8×8 pixel block, each function offering information about different patterns present in the respective block;
- Applying the quantization algorithm that keeps the lower frequency information and discards[6] the higher frequency information. This takes place by dividing each coefficient resulted after the DCT transform to a value from a quantization matrix that is set beforehand and rounding to the nearest integer.

After performing this step, an 8×8 block contains integers whose absolute values are highest in the top left corner of the matrix, as they correspond to the lowest frequency information. Thus, at this point, each channel contains numbers that represent quantized information about lower frequency patterns present in the original image. These values are the DCT coefficients of the image. For an 8×8 block, the coefficient found at the top left corner is named the DC coefficient and the rest are the AC coefficients. If the process above is reversed, a visually accurate approximation of the image is obtained;

3. The lossless compression of the resulted values is performed using either Huffman or Arithmetic Encoding. This type of compression is optimal due to the high degree of redundancy of the DCT coefficients that are smallest in absolute value.

The decoding process consists in reversing the encoding steps.

Properties of the DCT Coefficients. We briefly discuss about the distribution of the DCT coefficients. When determining the histogram of the corresponding DCT coefficients for each of the Y, C_r and C_b channels, we observe that its values peak at -1 and 1 and decrease as the absolute value of the coefficients increase.

[43] shows that the AC coefficients for both luminance and chrominance follow a Laplacian distribution of mean 0 and standard deviation σ. Hence, the Probability Density Function is characterized by

$$p(x) = \lambda/2 \cdot e^{-\lambda \cdot |x|}, \lambda = \sqrt{2}/\sigma. \tag{1}$$

As estimated in [39], σ takes values around 10. Hence, $\lambda \sim 0.14$. More precisely, the array $C[\cdot]$ of DCT coefficients[7] follows the above mentioned distribution. As it is a discrete random variable, we have that $\forall k \in \mathbb{Z}$

$$\mathbb{P}(C = k) = \int_{k-\frac{1}{2}}^{k+\frac{1}{2}} p(x)dx. \tag{2}$$

[6] maps to 0
[7] without the DC values

We are interested in examining the speed at which the histogram values decrease as the coefficient's magnitude increases. It suffices to compute the ratio $Hist(n)/Hist(n+k)$ for $n, k > 0$.

Note that $Hist(n)/Hist(n+k) = \mathbb{P}(C = n)/\mathbb{P}(C = n+k)$.

Using the Mean Value Theorem, we know that there exist $c_1 \in [n-1/2, n+1/2]$ and $c_2 \in [n+k-1/2, n+k+1/2]$ such that $\mathbb{P}(C = n) = 2/2 \cdot p(c_1)$ and $\mathbb{P}(C = n+k) = 2/2 \cdot p(c_2)$.

Thus, $H = Hist(n)/Hist(n+k) = p(c_1)/p(c_2) = e^{\lambda \cdot (c_2 - c_1)}$ which is in the interval $[e^{\lambda \cdot (k-1)}, e^{\lambda \cdot (k+1)}]$, with $\lambda \sim 0.14$. When $k \geq 8$, $H > 2$ and, therefore, we can conclude that $Hist(n) > 2 \cdot Hist(n+8)$ and, in particular, $Hist(1) > 2 \cdot Hist(9)$. The values in between are closer to each other.

2.3 The J3 Steganographic Algorithm

The J3 algorithm operates on the DCT coefficients of a JPEG image. For inserting or extracting data into or from such image, the first step that occurs is Huffman decoding for obtaining the coefficients array $C[\cdot]$. We will further consider that the array $C[\cdot]$ consists only in the AC coefficients, as the DC values represent the coefficients with zero frequency in both dimensions.

In the following we describe the J3 algorithm applied on the DCT coefficients array for any of the channels Y, C_r and C_b.

Remark 1. For more clarity, besides the following description of J3, we refer the reader to Algorithms 1 and 2 which represent the pseudocode for the insertion and the extraction methods.

Setup. The central technique used by the J3 Algorithm is LSB replacement of the DCT coefficients of an image. More specifically, a message bit b is inserted into a DCT coefficient i by replacing the LSB of i with b. The resulting coefficient is either in $\{i, i+1\}$ if i is even or $\{i-1, i\}$, if i is odd.

Let $THr \in \mathbb{N}^*$ be a threshold value chosen in advance and let $\mathscr{P} = \{-C_L, \ldots, C_L\} \setminus \{0\}$ be the set of coefficients used for data embedding, where C_L is the last odd coefficient with $\text{Hist}(C_L) > THr$. In order to perform the insertion, we use the set of ordered sets $\mathscr{C} = \{\{-C_L, -C_L+1\}, \ldots, \{-3, -2\}, \{-1, 1\}, \{2, 3\}, \ldots, \{C_L - 1, C_L\}\}$.

The hidden component of J3 consists in embedding the message bits into DCT coefficients from $\mathscr{P}$ whose indices are determined by a Pseudorandom Number Generator (PRNG) which is initialised with a secret seed.

Remark 2 (Histogram Restoration). Data embedding is done under the condition that restoring the histogram of the resulting image is possible. Before inserting a bit into a pair of coefficients, the algorithm verifies if there are enough unused coefficients in the image from that pair in order to use them for restoring the original histogram.

Definition 1 (*Stop Point*). *Let* $SP(x, y)$ *be a* Stop Point *of the pair* $\{x, y\}$. *For each valid pair of coefficients, its corresponding* Stop Point *represents the first index (generated by the PRNG) such as bit insertion in the respective pair stops.*

Definition 2 (Notations). *To check if a generated index is a* Stop Point[8]*, we define two counters in Table 1 for* $\{x, y\} \in \mathscr{C}$ *and* $x \in \mathscr{P}$.

Counter	Definition
$TC(x, y)$	The number of coefficients x changed to y.
$TR(x)$	The number of unused x coefficients.

Table 1. Definitions

Remark 3 (Properties). Note that $TC(x, x)$ is the number of coefficients x that remained unchanged during the insertion of bits and $TR(x) = Hist(x) - TC(x, y) - TC(x, x)$. For each $x \in \mathscr{P}$, $TR(x)$ is initialised with $Hist(x)$ and it decreases at every insertion into x. For each pair $\{x, y\} \in \mathscr{C}$, both $TC(x, y)$ and $SP(x, y)$ are initialised with 0. $TC(x, y)$ increases for every bit insertion that maps x into y.

Both insertion and extraction start by computing C_L using the value THr and initialising the PRNG with the secret *seed*.

Insertion. For inserting a message bit b, a secret index i is generated using the PRNG such that $C[i] \in \mathscr{P}$. We map the obtained $C[i]$ into $\{x, y\} \in \mathscr{C}$. We further assume that $C[i] = x$.

First, it is verified whether there are enough coefficients with the value x that were not accessed in order to use them for restoring the histogram. $Hist(x)$ increases every time a coefficient y is mapped into x and decreases when x is mapped into y, so restoring the original histogram is only possible when $TR(x) > TC(y, x) - TC(x, y)$. In consequence, $SP(x, y)$ becomes i once the inequality

$$TR(x) \leq TC(y, x) - TC(x, y) \tag{3}$$

occurs for the first time[9].

If $SP(x, y) \neq 0$, then the bit b is not inserted into any of the coefficients $\{x, y\}$ and another index i is generated. Else, we insert b into $x = C[i]$, decrease $TC(x)$, and if x becomes y, $TC(x, y)$ increases.

We consider $NSP = C_L$ as the number of *Stop Points* and $NbSP = \lfloor \log_2(W \cdot H) \rfloor + 1$ as the length in bits of a *Stop Point*. Before inserting any message bits,

[8] which is equivalent to checking if inserting a message bit into that pair is still possible or not

[9] which means that we have reached the minimum number of untouched coefficients used for histogram restoration

one must make space for inserting the *Stop Points* into the cover image, as they are necessary for extraction. In consequence, the PRNG is used for generating a set $\mathcal{D}$ of indices d, with $C[d] \in \mathscr{P}$ and $||\mathcal{D}|| = NSP \cdot NbSP$. We denote the set of indices for embedding the message bits by $\mathcal{M}$, with $\mathcal{M} \cap \mathcal{D} = \emptyset$ and $|\mathcal{M}| = ||m||$. Note that when constructing the $SP(\cdot, \cdot)$ array, one must allocate exactly $NbSP$ bits for each element.

The $SP(\cdot, \cdot)$ array is identified as the *Dynamic Header*. The actual first step of the embedding component of J3 is building a *Static Header* as follows: the message length is stored on the first 2 bytes of the array, and $NbSP$ and NSP on the last two bytes, respectively.

We further generate a set $\mathcal{S}$ of secret coefficients s, with $C[s] \in \mathscr{P}$ and $||\mathcal{S}|| = 32$, used for embedding the *Static Header* bits. Note that when generating the sets $\mathcal{D}$ and $\mathcal{M}$, the conditions $\mathcal{D} \cap \mathcal{S} = \emptyset$ and $\mathcal{S} \cap \mathcal{M} = \emptyset$ must also hold.

Given the previous notations and observations, the following steps take place during the insertion.

1. Build the *Static Header* and generate the set $\mathcal{S}$ as previously described. The *Static Header* bits are inserted into the coefficients $C[s]$, for $s \in \mathcal{S}$;
2. Generate the set $\mathcal{D}$ and store it separately;
3. Insert the message bits following the previously mentioned rules. Each index from the set $\mathcal{M}$ is generated once accessing a message bit b. The *Stop Points* array is built in the process;
4. Insert the *Dynamic Header* into the coefficients $C[d]$, for $d \in \mathcal{D}$;
5. Restore the original histogram by using coefficients from a set $\mathcal{R}$, with $\mathcal{R}[i] \in \mathscr{P}$ with the sets $\mathcal{R}, \mathcal{M}, \mathcal{D}$ and $\mathcal{S}$ being pairwise disjoint. For each set of coefficients $\{x, y\} \in \mathscr{C}$, proceed as follows: the value of $Hist(x)$ after insertion became $Hist(x) - TC(x, y) + TC(y, x)$, and $Hist(y)$ became $Hist(y) - TC(y, x) + TC(x, y)$. Without loss of generality, we suppose $TC(x, y) - TC(y, x) > 0$. Then, the $TC(x, y) - TC(y, x)$ additional y coefficients must become x. In consequence, we generate enough indices from $\mathcal{R}$ to restore the original histogram values for each $\{x, y\} \in \mathscr{C}$.

Note that when generating the sets $\mathcal{S}$ and $\mathcal{D}$, for each $i \in \mathcal{S} \cup \mathcal{D}$, the $TR(C[i])$ value decreases by 1. Moreover, the $TC(\cdot, \cdot)$ array is updated accordingly during any of the insertion steps.

Extraction. For extracting data bits from a cover image, one must know the secret *seed* in order to determine the indices used for insertion along with the *Stop Points*.

The following steps occur during the extraction algorithm.

1. Generate the indices from $\mathcal{S}$ and use them for extracting the *Static Header* and verify whether the last 2 bytes of the obtained header are equal to $NbSP$ and NSP, respectively. If one of the equalities doesn't hold, then either the values of the image coefficients where the static header bits were inserted are changed (the picture has been tampered with), either the seed entered is wrong or the picture has no message inserted.

2. Generate the indices from the set $\mathcal{D}$ and extract $SP(x, y)$, for $\{x, y\} \in \mathscr{C}$.
3. Generate the indices from $\mathcal{M}$ by taking into account the message length and the *Stop Points* found and use them to extract the message.

3 Main Results

Note that when recalling J3 in Section 2 we already presented some of the corrections we applied to the original algorithm. Next, we emphasize these modifications.

1. We have corrected the mechanism of finding *StopPoints*. In the original article [35], the *StopPoint* value was updated each time Equation (3) was fulfilled. Therefore, the *StopPoint* became the last index of the coefficient with that value that was used when inserting, which led to the incorrect extraction of the message.
   Let $\{x, y\} \in \mathscr{C}$ be a pair of coefficients such that $SP(x, y) = i \neq 0$. We know i is not the first value for which Equation (3) occurred. Let j be the first index for which Equation (3) holds. Then, embedding in the pair $\{x, y\}$ has stopped once reaching j, but the $SP(x, y)$ value keeps changing until i is generated. During extraction, when generating j, a message bit is extracted automatically, which is incorrect, as no message bit was embedded into $C[j]$. Moreover, bit extraction occurs for each index k generated between i and j with $C[k] \in \{x, y\}$, which leads to obtaining a bit string that was not inserted in the first place.
2. Another fundamental observation that was omitted in [35] regarding the building blocks of the algorithm (*i.e.* bit insertion and extraction) is the fact that the sets $\mathcal{S}, \mathcal{D}$ and $\mathcal{R}$ must be pairwise disjoint.
   Supposing $i \in \mathcal{S} \cap \mathcal{M}$ such as the *Static Header* bit inserted into $C[i]$ was 0, and the message bit inserted was 1. Let $C[i] = x, \{x, y\} \in \mathscr{C}$. Then $C[i]$ becomes y after the two insertions. The first bit extracted is the *Static Header* bit, as i is firstly generated for this purpose. Simultaneously, the first bit extracted is the message bit, as the value of $C[i]$ corresponds to the last insertion made. In consequence, for both occurrences of i, the same bit is extracted which is a contradiction.
3. Within the extraction algorithm, we added the extracted header check. In the original version, the *Static Header* was built only for extracting the message length, the number of *Stop Points* and the number of bits on which a *Stop Point* is stored.
   The inadequacy of this approach resides in the fact that the *NbSP* and *NSP* obtained are used for further extraction, when they could be bogus values. Their use is only for verification as they depend only on the image and its histogram (which is preserved), so they are computed before any step of insertion/extraction and further compared to the extracted values.

In the following we detail other modifications and enhancements that we brought to the J3 stegosystem.

3.1 Adding a Pre-Processing Step

A preprocessing step was added prior to each sub-algorithm of our proposed version of J3 (insertion and extraction). The outcomes of this component are obtaining an adequate C_L value that maximizes the embedding capacity depending on the distribution of the image histogram and determining a total embedding capacity of the image for assuring the robustness of the algorithm.

Setting TH_r and Computing C_L. In the original algorithm [35], C_L is computed using the value THr[10]. Note that the authors do not offer any optimal possible values for this threshold. More precisely, THr is hard-coded as a constant that intuitively determines a nonempty set $\mathscr{P}$ for images that have a histogram which takes smaller values. Usually, it is set somewhere near 300 for the luminance component and near 100 for the chrominance channels. Consequently, for higher quality images[11], the set of coefficients to be used for performing J3 is large.

In addition, the majority of images existing nowadays have a higher quality. *E.g.*, for smartphone camera images even in the portrait mode, $Hist(1)$ and $Hist(-1)$ are surpassing 10000. Also, $Hist(2)$ and $Hist(3)$ exceed 7000 on the Y channel. Therefore, $\mathscr{P}$ is large: usually $C_L > 50$ on the Y channel and $C_L > 10$ on the Cr and Cb channels. What stands out about the distribution of the determined coefficients is the fact that the values closer to C_L have their $Hist(\cdot)$ values by definition near THr but disproportionately small compared to the $Hist(\cdot)$ values of the coefficients closer to 0. Even though apparently this implication has no impact on message insertion (the high frequency of the coefficients from $\mathscr{P}$ implies the possibility of inserting large messages), there are very often cases in which the algorithm loops as it is incapable of finding coefficients closer to C_L for restoring the histogram.

Moreover, before reaching this step of the J3 insertion, the algorithm speed is decreased compared to the case when we set a large THr (only for inserting into reasonable quality images) such as C_L is small. On the other hand, a large C_L implies a large *Stop Points* array, which means a big additional number of bits to be inserted into the image as the *Dynamic Header*. In most cases, $SP(x, y) = 0$ for all $\{x, y\} \in \mathscr{C}$ as $TR(\cdot)$ automatically contains large values, being initialised by $TR(i) = Hist(i)$. This means that for a large set of messages $||\mathcal{D}|| = C_L \cdot NbSP$ additional zeroes are inserted. For a common JPEG image of 1 MB and reasonable quality, there are $\mathcal{D} \approx 50 \cdot 20 = 1000$ null bits of *Dynamic Header*. As the image size increases, $||\mathcal{D}||$ increases, and this leads to the incapacity of embedding messages of reasonable length into the image because for coefficients closer to C_L, the corresponding *Stop Point* remains 0, but when inserting into the *Dynamic Header*, the coefficients that must remain untouched for restoring the histogram are used here for embedding the *Stop Points* array.

[10] set in advance

[11] *i.e.* their histogram contains larger values

To sum up, THr must be a value that depends on each image so it is computed according to the image histogram. The main property that THr must fulfill is to determine a C_L so as for an i with the absolute value close to C_L, $Hist(i)$ is proportional to $Hist(1)$ and $Hist(-1)$.

Remark 4 (Optimal THr). By taking into account the distribution of the DCT coefficients as exposed in 2, we concluded that $THr = Hist(1)/2$ is an optimal value for both performance and robustness. The choice was confirmed in practice, as both parameters substantially increased.

The Limit E_C. With the value C_L computed, one must know how many message bits can be safely embedded into an image (channel). The limit we propose is

$$E_C = \Sigma_{\{x,y\}\in\mathscr{C}} \min(Hist(x), Hist(y)) - ||\mathcal{S}|| - ||\mathcal{D}||.$$

The choice of E_C enables us to take advantage of the maximum embedding capacity of every pair $\{x, y\} \in \mathscr{C}$ while keeping exactly the minimum number of unused coefficients for restoring the histogram. This occurs in the worst case scenario which is for every pair $\{x, y\} \in \mathscr{C}, Hist(x) \geq Hist(y)$, every message bit to be inserted maps x into y.

Indeed, we start with $TR(x) = Hist(x), TR(y) = Hist(y), TC(x, y) = 0$ and $TC(y, x) = 0$. When insertion transforms x into y, $TR(x)$ becomes $Hist(x) - 1$ and $TC(x, y)$ becomes 1. After k successive identical steps, $TR(x)$ becomes $Hist(x) - k$ and $TC(x, y)$ becomes k. We know that in order to reach a *Stop Point* for the pair $\{x, y\}$, Equation (3) must hold either for x or y. For y, we have $Hist(y) \leq k$. Thus, the number of steps needed for updating the value of $SP(x, y)$ is $k = Hist(y)$. In conclusion, the message piece to be inserted into $\{x, y\}$ must have the length at most $Hist(y)$ in order for the restoration of the histogram to occur.

If the message is constructed such that for each pair $\{x, y\} \in \mathscr{C}$ insertion occurs as above, then the longest message that can be inserted while keeping enough untouched coefficients for histogram restoration is $\Sigma_{\{x,y\}\in\mathscr{C}} \min(Hist(x), Hist(y))$. As the *Static* and *Dynamic* headers are also inserted, the maximum embedding capacity becomes E_C.

Remark 5 (Pre-processing Motivation). In conclusion, the pre-processing step is necessary for an accurate embedding process, as it reveals the optimal coefficient pairs for a correct and efficient insertion component and a safe upper bound for the size of the data to be inserted.

3.2 Taking Advantage of Each Channel's Embedding Capacity

In the original J3 algorithm, the embedding component is applied on all three channels Y, C_r and C_b: if the message length exceeds the total capacity of the

Y channel[12], then the remaining bits are inserted into C_r. If message bits still remain, they are inserted into C_b.

Our proposal is to take advantage of each channel's embedding capacity (as defined in the previous paragraph) by splitting the message into pieces proportionate to the E_C of each channel.

We denote by $E_C(Y), E_C(C_r)$ and $E_C(C_b)$ the embedding capacities of Y, C_r and C_b respectively. We split the message into pieces of length $\lfloor \frac{E_C(Y)}{E_C(Y)+E_C(C_r)+E_C(C_b)} \cdot ||m|| \rfloor$, $\lfloor \frac{E_C(C_r)}{E_C(Y)+E_C(C_r)+E_C(C_b)} \cdot ||m|| \rfloor$ and $\lfloor \frac{E_C(C_b)}{E_C(Y)+E_C(C_r)+E_C(C_b)} \cdot ||m|| \rfloor$.

Note that when adding the three lengths, we obtain a value in the set $\{||m||, ||m|| + 1, ||m|| + 2\}$. In consequence, if the sum is $||m|| + 1$, then one of the lengths is decreased by 1, and if it is $||m|| + 2$, then two of the lengths are decreased.

In summary, we maximize the relative embedding capacity of each channel. If a channel has a capacity much higher than the others, then the remaining channels will have the corresponding message lengths equal to 0 and no embedding will occur (only in their *Static Headers*, and each will have their first two bytes equal to 0). If a channel has a small capacity relatively to the other two channels, then its *Static Header* will begin with two null bytes.

3.3 Security Aspects

Cryptographic Security. A cryptographic component is necessary for a stegosystem for at least two reasons:

1. Given solely an image that hides the existence of a message m, if one knows beforehand the indices of the coefficients in which the bits of m were inserted, then recovering them can be done by simply applying the inverse of the LSB embedding. In consequence, the message insertion must be performed into coefficients of secret indices, known only by the sender and the receiver. A feasible method to ensure this is using a PRNG initialised with a secret seed (as currently done in most of the modern stegosystems).
2. If an adversary obtains a cover image both before and after insertion, then part of the message bits can be recovered by simply observing the DCT coefficients that differ. Thus, in general, half of the message bits can be obtained (the bit and the coefficient used for insertion must have different parities). Therefore, prior to insertion, the messages have to be encrypted with a key agreed between the sender and the recipient[13].

Steganographic Security. We further asssume that the cryptographic security requirements are fulfilled.

If an adversary has the two versions of an image (before and after insertion), recovering parts of the original message becomes infeasible due to the encryption

[12] defined in [35] as $\Sigma_{\{x,y\}\in\mathscr{C}} Hist(x) + Hist(y)$

[13] Symmetric key cryptosystems are customarily used.

step. Even so, by analyzing the differences in histograms in the case of various stegosystems using specific tools, it can be observed which of the two images hides information.

When the J3 stegosystem is used, both images have the same histogram due to the histogram compensation step. Therefore, an image that hides data is indistinguishable from an image that was not used for insertion. Thus, the security is theoretically assured by default.

We applied steganalysis techniques implemented in specific tools to our enhanced J3 algorithm (*e.g.* StegoHunt [7]) and obtained the expected result.

Remark 6. Regarding the cryptographic security, we propose the use of state-of-the-art cryptographic schemes (see Appendix B): the better the encryption scheme, the more secure the stegosystem. Nonetheless, without affecting the efficiency of the implementation.

4 Implementation

We ran the code for our algorithm on a standard laptop using both Windows 10 and Ubuntu 20.04.5 LTS OS[14], with the following specifications: Intel Core i7-10510U with 4 cores and 16 Gigabytes of RAM. The programming language we used for implementing our algorithms was C++.

To extract the DCT coefficients from an image and restore the image from them independently of the platform on which the application runs, it is necessary to use a library that implements the **ITU T.81 JPEG Compatible** standard. The main library that is used for encoding and decoding JPEG is libjpeg. It is written in the C programming language and distributed under the Custom BSD license (free software) for any platform (cross-platform) [46].

The latest released version is libjpeg 9e[15]. It underlies the OpenCv, torchjpeg, and JasPer libraries, implementing basic JPEG operations [46].

We refer the reader to [1] for the source code representing the implementation of our proposed results.

4.1 Implementation Results

Taking into account the fact that J3 was already compared to the other stegosystems mentioned in Section 1 and its performance and security were proven superior both theoretically and in practice, we directly compare our proposal to the original version of J3.

When given an image for extraction, the original J3 algorithm first obtains the *Static Header* and then uses it for further extracting data which leads to events such as obtaining bogus values or remaining stuck in a loop. By using our additional *Static Header* checkup, if an image has nothing embedded into it or if it was altered (for example by sending it through a channel that applies

[14] The Ubuntu OS was installed on a virtual machine.
[15] as of January 16, 2022

additional compression algorithms), then an exception is thrown that warns the user regarding the state of the image.

On the other hand, if the message inserted was long enough for at least a *Stop Point* to change its value, then the extraction using J3 is incorrect for the reasons previously exposed (in Section 3). Note that our proposal works with long messages, extraction taking place as expected.

For large images (especially for those taken with the camera of an Android/IOS phone, their size being over 1 MB), the original J3 algorithm loops even for very short messages (under 20 bytes), or lasts for an unreasonable amount of time (more than 5 minutes) as searching for indices for histogram restoring was time consuming (for pairs of the form $\{2k, 2k+1\}$, for $2k+1$ close to C_L). The reason of this occurrence is the fact that the search for indices such as the corresponding $C[i]$ is in $\{2k, 2k+1\}$ and i was not previously used takes too much time. This is caused by the fact that when generating i, $C[i]$ has a much higher probability to be closer to 0.

Remark 7. We present a set of results regarding the speed of our enhanced J3 algorithm in Table 2 for an image of $3,4$ MB, taken with an Android camera having the quality 300×400. The metrics used were the average execution time (mean) and the standard deviation from the mean (std) of the obtained timings. We generated 50 random messages for each length. It can easily be observed that not only the average time of our proposal is considerably smaller, but also the execution times are closer to the average time than in the case of the original algorithm.

Message length (bytes)	**Original J3 insertion (seconds)**		**Enhanced J3 insertion (seconds)**	
	mean	**std**	**mean**	**std**
20	8.203	0.120	0.21	0.005
200	10.782	3.764	0.346	0.018
1000	21.740	6.036	2.615	0.126
5000	136.098	35.078	60.108	2.045
10000	234.663	49.002	139.67	6.008

Table 2. Original vs. enhanced J3 implementation results

For lower quality images, J3 not only has a very low embedding capacity, but it is also prone to being unable to completely insert a message of a reasonable length relatively to the total embedding capacity of the image.

Our approach performs well for very low quality images. For an image of 4.5 kB of a very small resolution (256×194, 96 dpi on 96 dpi), grayscale, with $Hist(1) = 1870$ on Y we could embed messages of 140 bytes in under 0.2 seconds.

In all the previously mentioned measurements, extraction was also performed and the message was correctly recovered in each case.

5 Conclusions and Future Work

We recalled the J3 algorithm, which differs from other stegosystems working with DCT coefficients both by its high ability to hide data and by keeping the histogram of the original image (and, thus, preventing statistical attacks). The main result of our work was the reconstruction of the entire J3 algorithm including all its sub-algorithms: pre-processing, message insertion and extraction. We also optimized the stegosystem and covered all practical cases while still using the maximum image insertion capacity.

Future Work. In the near future, a major goal of our research is building an application based on our short description in Appendix B that will enable the user to perform data insertion and extraction within different platforms, a feature not yet available in the case of current steganography applications.

A first step for continuing our research in this direction is to present a concrete comparison (especially in terms of efficiency) between our proposed software application and the publicly available applications.

Another idea is to apply more powerful steganalysis techniques in order to check the security of our proposed scheme.

References

1. https://github.com/cryptocrew601
2. https://github.com/daniellerch/aletheia
3. https://stegoshare.soft112.com
4. https://www.iso.org/standard/54989.html
5. https://www.openstego.com
6. https://www.softpedia.com/get/Security/Encrypting/Hide-N-Send.shtml
7. https://www.wetstonetech.com/products/stegohunt-steganography-detection
8. Ahmed, N., Natarajan, T., Rao, K.: Discrete Cosine Transform. IEEE Transactions on Computers **C-23**(1), 90–93 (1974)
9. Arun Kumar Singh, Juhi Singh, D.H.V.S.: Steganography in Images Using LSB Technique **5**(1), 425–430 (January 2015)
10. Aumasson, J.P.: Password Hashing Competition (2013-2015)
11. Aumasson, J.P.: Serious Cryptography: A Practical Introduction to Modern Encryption. USA (2017)
12. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: the Memory-Hard Function for Password Hashing and Other Applications. University of Luxembourg (2015)
13. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition. The MIT Press, 3rd edn. (2009)
14. Denemark, T., Bas, P., Fridrich, J.: Natural Steganography in JPEG Compressed Images. In: Proc. IS&T, Electronic Imaging, Media Watermarking, Security, and Forensics (2018)
15. Denemark, T., Fridrich, J.: Steganography with Multiple JPEG Images of the Same Scene. In: IEEE TIFS. pp. 2308–2319 (2017)
16. Denemark, T.: Side-Information for Steganography Design and Detection. Ph.D. thesis, Binghamton University–SUNY, New York, USA (2018)

17. Dworkin, J.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC (2007-11-28 2007)
18. Dworkin, J., Barker, B., Foti, J., Bassham, E., Roback, E.: Announcing the Advanced Encryption Standard (AES). Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, MD (2001)
19. Fabien A. P. Petitcolas, Ross J. Anderson, M.G.K.: Information Hiding—A Survey **87**(7), 1062–1078 (1999)
20. Filler, T., Fridrich, J.: Gibbs Construction in Steganography. In: IEEE Transactions on Information Forensics and Security (2010)
21. Fridrich, J., Pevný, T., Kodovský, J.: Statistically Undetectable JPEG Steganography: Dead Ends, Challenges, and Opportunities. In: Proceedings of the 9th ACM Multimedia & Security Workshop, Dallas, TX. pp. 3–14 (2007)
22. Fridrich, J., Goljan, M., Hogea, D.: Attacking the OutGuess. In: Proc. ACM: Special Session on Multimedia Security and Watermarking (2002)
23. Fridrich, J., Goljan, M., Hogea, D.: Steganalysis of jpeg images: Breaking the f5 algorithm. pp. 310–323 (10 2002)
24. Fridrich, J., Goljan, M., Hogea, D.: New Methodology for Breaking Steganographic Techniques for JPEGs. In: SPIE: Electronic Imaging 2003, Security and Watermarking of Multimedia Contents (2003)
25. Fridrich, J., Goljan, M., Hogea, D.: Steganalysis of JPEG Images: Breaking the F5 Algorithm. In: Information Hiding. pp. 310–323. Springer (2003)
26. Hetzl, S., Mutzel, P.: A Graph-Theoretic Approach to Steganography. In: Communications and Multimedia Security (2005)
27. Holub, V., Fridrich, J.: Designing Steganographic Distortion using Directional Filters. In: IEEE International Workshop on Information Forensics and Security (WIFS) (2012)
28. Holub, V., Fridrich, J., Denemark, T.: Universal Distortion Function for Steganography in an Arbitrary Domain. In: EURASIP Journal on Information Security, (Section:SI: Revised Selected Papers of ACM IH and MMS 2013) (2014)
29. International Organization for Standardization: Information Technology — Digital Compression and Coding of Continuous-Tone still Images: Requirements and Guidelines. ISO/IEC 10918-1 (1994)
30. Isnanto, R.R., Septiana, R., Hastawan, A.F.: Robustness of steganography image method using dynamic management position of least significant bit (lsb). In: 2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI). pp. 131–135 (2018)
31. Jamil, T.: Steganography: The Art of Hiding Information Is Plain Sight **18**(01) (1999)
32. Kaliski, B.: RFC2898: PKCS5: Password-Based Cryptography Specification Version 2.0 (2000)
33. Ken Cabeen, P.G.: Image Compression and Discrete Cosine Transform (2013)
34. Koshkina, N.V.: J-UNIWARD Steganoanalysis. In: Cybernetics and Systems Analysis (May 2021)
35. Kumar, M., Newman, R.: J3: High Payload Histogram Neutral JPEG Steganography. In: 2010 Eighth International Conference on Privacy, Security and Trust. pp. 46–53 (2010)
36. Luo, X., Song, X., Li, X., Zhang, W., Lu, J., Yang, C., Liu, F.: Steganalysis of HUGO steganography based on parameter recognition of syndrome-trellis-codes. Multimedia Tools and Applications **75**, 13557–13583 (2015)

37. Morkel, T., Eloff, J.H.P., Olivier, M.S.: An Overview of Image Steganography. In: ISSA (2005)
38. Muzhir Shaban AL-Ani, F.H.A.: The JPEG Image Compression Algorithm **6**(3), 1055–1062 (May 2013)
39. Narayanan, G.: A study of probability distributions of dct coefficients in jpeg compression (2010)
40. National Institute of Standards and Technology: Data Encryption Standard (DES) (October 1999)
41. Provos, N.: Defending Against Statistical Steganalysis. In: Usenix security symposium. vol. 10, pp. 323–336 (2001)
42. Rif, D.: Re-encoding Persistent Video Steganography (2018)
43. Smoot, S.R., Rowe, L.A.: DCT coefficient distributions. In: Human Vision and Electronic Imaging. vol. 2657, pp. 403–411. International Society for Optics and Photonics, SPIE (1996)
44. Tang, W., Li, H., Luo, W., Huang, J.: Adaptive Steganalysis against WOW Embedding Algorithm. In: Proceedings of the 2nd ACM Workshop on Information Hiding and Multimedia Security. p. 91–96. IH&MMSec Í4, Association for Computing Machinery, New York, NY, USA (2014)
45. Trithemius, J.: Steganographia: Hoc est: Ars Per Occultam Scripturam Animi Sui Voluntatem Absentibus aperiendi certa. Berner (1606)
46. Vollbeding, I.J.G.G.: Libjpeg (January 16 2022), http://ijg.org/
47. Wang, H., Wang, S.: Cyber Warfare: Steganography vs. Steganalysis **47**(10), 73–83 (October 2004)
48. Wayner, P.: Disappearing Cryptography. In: Information Hiding: Steganography and Watermarking (3rd Edition - December 3, 2008)
49. Westfeld, A.: High Capacity Despite Better Steganalysis (F5 – a Steganographic Algorithm). In: I. S. Moskowitz, editor, Information Hiding, 4th International Workshop, volume 2137 of Lecture Notes in Computer Science. pp. 289–302 (2001)
50. Winarno, A., Arrasyid, A.A., Sari, C.A., Rachmawanto, E.H., et al.: Image Watermarking Using Low Wavelet Subband Based on 8× 8 Sub-Block DCT. In: 2017 International Seminar on Application for Technology of Information and Communication (iSemantic). pp. 11–15. IEEE (2017)

A Pseudocode

Algorithm 1 Insertion of message bits

$x \leftarrow PRNG(seed)$
if C_x is *even* and $b = 1$ **then**
  if $C_x > 0$ **then**
    $C_x \leftarrow C_x + 1$
    $TC(C_x, C_x + 1) \leftarrow TC(C_x, C_x + 1) + 1$
  else
    $C_x \leftarrow C_x - 1$
  end if
end if
if C_x is $odd \neq 1$ and $b = 0$ **then**
  if $C_x > 0$ **then**
    $C_x \leftarrow C_x - 1$
    $TC(C_x, C_x - 1) \leftarrow TC(C_x, C_x - 1) + 1$
  else
    $C_x \leftarrow C_x + 1$
  end if
end if
if $C_x = -1$ and $b = 1$ **then**
  $C_x \leftarrow 1$
  $TC(C_x, C_x + 1) \leftarrow TC(C_x, C_x + 1) + 1$
end if
if $C_x = 1$ and $b = 0$ **then**
  $C_x \leftarrow -1$
  $TC(C_x, C_x - 1) \leftarrow TC(C_x, C_x - 1) + 1$
end if

Algorithm 2 Extraction of message bits

$x \leftarrow PRNG(seed)$
if C_x is *even* **then**
  $b \leftarrow 1$
end if
if C_x is $odd \neq 1$ **then**
  $b \leftarrow 0$
end if
if $C_x = -1$ **then**
  $b \leftarrow 0$
end if
if $C_x = 1$ **then**
  $b \leftarrow 1$
end if

B A Steganography Application Proposal

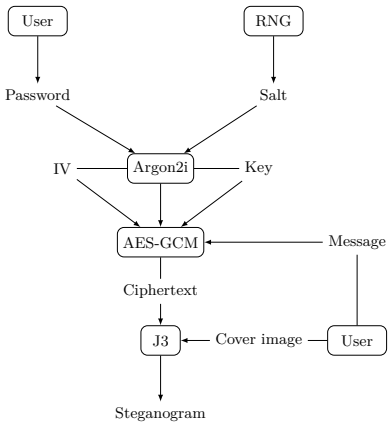

Fig. 1. The flow of our steganography application proposal

A natural continuation of our proposed results is putting them in practice. Thus, we briefly describe a steganography software application. The flow of the application is shortly presented in Figure 1.

The user enters a password from which the secret key for AES-256 [18] is derived. AES is used in the GCM authenticated encryption mode of operation [17]. Thus, the plaintext (the message entered by the user) is encrypted and authenticated. Then, our proposed J3 variant is used to hide the cryptogram into a cover JPEG image.

Within the application, the Argon2i function [12] is used (as the winner of the Password Hashing Competition [10]) to derive both the key and the IV[16].

The keys should only be temporarily stored in the RAM of the smartphone/PC (while the cryptographic and steganographic operations are performed).

[16] of size 96 bits