Searching for the Stego-Key

[a]Jessica Fridrich*, [a]Miroslav Goljan, and [b]David Soukal
[a]Department of Electrical and Computer Engineering
[b]Department of Computer Science
SUNY Binghamton, Binghamton, NY 13902-6000, USA

ABSTRACT

Steganalysis in the wide sense consists of first identifying suspicious objects and then further analysis during which we try to identify the steganographic scheme used for embedding, recover the stego key, and finally extract the hidden message. In this paper, we present a methodology for identifying the stego key in key-dependent steganographic schemes. Previous approaches for stego key search were exhaustive searches looking for some recognizable structure (e.g., header) in the extracted bit-stream. However, if the message is encrypted, the search will become much more expensive because for each stego key, all possible encryption keys would have to be tested. In this paper, we show that for a very wide range of steganographic schemes, the complexity of the stego key search is determined only by the size of the stego key space and is independent of the encryption algorithm. The correct stego key can be determined through an exhaustive stego key search by quantifying statistical properties of samples along portions of the embedding path. The correct stego key is then identified by an outlier sample distribution. Although the search methodology is applicable to virtually all steganographic schemes, in this paper we focus on JPEG steganography. Search techniques for spatial steganographic techniques are treated in our upcoming paper.

Keywords: Steganography, forensic, steganalysis, stego key, search, key

1. STEGANOGRAPHY AND STEGANALYSIS

The purpose of steganography[1] is to hide the very presence of communication by embedding messages into innocuous-looking cover objects, such as digital images. To accommodate a secret message, the original image, also called the *cover image*, is slightly modified by the embedding algorithm to obtain the *stego image*. The embedding process may depend on a secret *stego key* K_S. The stego key is used to control the embedding process, such as the selection of pixels or coefficients carrying the message, etc. Before embedding, the message is usually pre-pended with a header H and further encoded using compression algorithms and/or encrypted using an encryption algorithm with the *encryption key* K_E.

In contrast to watermarking when the embedded message has a close relationship to the cover image supplying data, such as sender or receiver information, authentication codes, etc., in steganography, the cover image is a mere decoy and has no relationship to the hidden data. The most important requirement for a steganographic system is *undetectability*: stego images should be statistically indistinguishable from cover images. In other words, there should be no artifacts in the stego image that could be detected by an attacker with probability better than random guessing, given the full knowledge of the embedding algorithm, including the statistical properties of the source of cover images, except for the stego key (Kerckhoff's principle).

The art of discovering secret messages is called steganalysis. As pointed out in the previous paragraph, in theory, steganography is considered broken even when the only thing we have is evidence that a certain document contains hidden data without actually being able to extract it or identify the particular steganographic software that was used for hiding. On the other hand, it makes sense to include other forensic activities related to message detection and extraction to steganalysis. On a more general level, steganalysis comprises of the following phases ordered by the level of success achieved during detection:

0) Identification of web sites, Internet nodes, or computers that should be analyzed (intelligent web-crawlers).
1) Development of algorithms that can distinguish stego images from cover images.

* fridrich@binghamton.edu; phone 1 607 777-2577; fax 1607 777-4464; http://www.ws.binghamton.edu/fridrich

2) Identification of the embedding mechanism, e.g., Least Significant Bit embedding (LSB), adding 1 or –1 (±1 embedding), embedding in the frequency domain, embedding in the image palette, sequential, random, or content-adaptive embedding, etc.
3) Determining the steganographic software[9], e.g., Steganos, S-Tools, Hide&Seek, F5, OutGuess, etc.
4) Searching for the stego key K_S and extracting the embedded data
5) Deciphering the extracted data and obtaining the secret message (cryptanalysis).

Depending on the specific steganographic and detection algorithms, some of the phases listed above may merge or be skipped entirely. Most steganalytic algorithms proposed today are targeted to a specific embedding mechanism[2,3,4]. On the other hand, universal blind detectors[5,6,7] may bring a decisive answer to Phase 1 without necessarily determining the embedding method. Once the embedding operation or specific artifacts in the image are known, we may identify possible candidates for the stego software in Phase 3.

The steganalytic work does not have to necessarily follow the phases above. For example, we may have some a priori knowledge about the stego software that may drastically simplify the analysis — the suspect has downloaded a certain stego program from the net, or the stego program was found on the hard disk on a seized computer. In the latter case, it is also not unreasonable to encounter situations when the forensic investigators will have access to multiple versions of one image (to the cover and stego image). In these cases, the basic steganalysis has been simplified and the analyst may strive to recover more details, such as the stego key and, eventually, recovering the hidden data itself.

2. COMPLEXITY OF THE STEGO KEY SEARCH

In this paper, we investigate Phase 4 – the search for the stego key under the assumptions that we already know (or suspect) the steganographic algorithm under investigation. One common approach, which has also been used by Provos[8], is to apply the dictionary attack and/or brute-force the stego key while looking for a recognizable header as a sign that we have come across the correct stego key. In fact, this approach could be used as a primitive steganalytic method for Steps 1 and/or 2. This search, however, will fail if the embedded data stream does not have any recognizable structure. There are a number of ways how to avoid using recognizable headers for steganographic communication. One could simply encrypt the secret message including its header. To prevent the fixed header to be encrypted to the same sequence of bytes for different images, one can pre-pend a random "salt" to the beginning of the message prior to encryption. Another possibility is to use the PGP Stealth program (http://www.cypherspace.org /openpgp/stealth/) that was specifically developed for use with steganography.

Thus, one needs to count with the possibility that the embedded data will not exhibit any recognizable structure that could be used for stego key search. In this case, although the search for the stego key is still possible, it becomes significantly more complicated because now for each stego-key K_S, we need to try all possible encryption keys. Thus, the complexity of the brute force search becomes proportional to the product of the number of stego and crypto keys:

$$Complexity \sim |\mathsf{K}|\times|\mathsf{E}|,$$

where K and E are the sets of all stego and encryption keys, respectively, and $|X\ \ |$ denotes the cardinality of X. Even though for some stego programs the stego key space itself may be small enough to make the brute force search for the stego key plausible, if the message has been encrypted, the search may become computationally infeasible.

The main contribution of this paper is the finding that the brute force search for the stego key can be done for a very wide class of steganographic techniques in $O(|\mathsf{K}|)$ steps and is *independent* of the encryption algorithm, as long as the embedded message does not occupy 100% of the image capacity.

Before we proceed with the rest of this paper, we remark that methods that do not have any stego-keys, such as EzStego or the original version of J-Steg[9], can be reliably detected (using the chi-square attack[10] and difference histograms[4], respectively), the message easily extracted, and the steganalysis can be readily moved to Phase 5 – cryptanalysis. Thus, thanks to current advances in detection algorithms, steganalysis of these programs has been essentially reduced to cryptanalysis.

The same steganographic techniques, however, can be easily modified by embedding the message bits along a pseudo-random walk. Recently, methods, such as Pairs Analysis[11], RS Analysis[2], generalized chi-square attack[8,12], universal blind detectors[5,6,7], and JPEG steganalysis[4,13] have addressed the issue of detection of randomly spread messages. Despite the fact that most of these detection techniques can also relatively accurately estimate the unknown message length, they do not provide any information about the stego key. In order to recover the secret message, we have to go through both steganalytic Phases 4 and 5. In this paper, we show that Phase 5 can be performed without going through Phase 6 even when the embedded data is encrypted, thus drastically reducing the complexity of steganalysis.

The paper is organized as follows. In the next section, we define the embedding paradigm that will be investigated in this paper. The stego key search method is described in simple intuitive terms in Section 4. Detailed description of the search algorithm for JPEG images is given in Section 5. In Section 6, we demonstrate the method on the F5 algorithm and in Section 7, we continue with another example – the OutGuess. Finally, in Sec. 8, we elaborate on how our contribution affects the definition of steganographic security and construction of better steganographic methods. In the same section, we conclude the paper and outline future research directions.

3. EMBEDDING PARADIGM ADDRESSED

The stego-key search methods described in this paper are applicable to virtually all steganographic methods whose embedding mechanism has the following three primitives (assuming the image consists of individual units or samples):

1. To embed *m* bits, *m* samples are chosen in a pseudo-random fashion from the cover image.
2. The message bits are embedded as Parities of individual image samples.
3. If necessary, the samples' parity is changed using an Embedding Operation.

Most steganographic techniques indeed work in this manner. The random path selection is usually implemented using a Pseudo-Random Number Generator (PRNG) that is seeded with a seed derived from a user-specified stego key or a passphrase. The output of the PRNG is used to generate a pseudo-random walk through the image samples (pixels for spatial image formats or DCT coefficients for JPEGs).

The parity function assigns a binary value (parity) to every possible value of the samples. In most stego programs, the parity is the same for all samples and independent of the stego key, as it is the case for the most frequently used parity – the least significant bit (LSB) of image samples. In more sophisticated methods, the parity function can be chosen in such a manner to minimize the embedding distortion (see the optimal parity assignment[14]). Also, the parity may depend on the sample position and/or on the stego key, as in Stochastic Modulation Steganography[15].

The secret message is embedded in the form of a bit-stream as the parity of the samples along the pseudo-random walk. In order to match the image feature parity with the message bit, the sample is modified using an embedding operation. This operation could be deterministic or probabilistic. For example, for the LSB embedding, the sample parity is defined as its LSB and the embedding operation is shown in Table 1.

The program Hide[16] also hides data in LSBs of pixels (e.g., it is based on the same parity mapping), however, it uses a different probabilistic embedding operation (see Table 2). In other words, when the message bit does not match the pixel parity (its LSB), Hide either adds or subtracts 1 with equal probability 50% with the exception of values 0 and 255, which are only increased or decreased, respectively.

Sample value	Modified value when the message bit is	
	0	1
$2i$	$2i$	$2i+1$
$2i+1$	$2i$	$2i+1$

Table 1. LSB embedding operation.

Sample value	Modified value when the message bit is	
	0	1
$2i$	$2i$	$2i+1$ or $2i-1$
$2i+1$	$2i$ or $2i+2$	$2i+1$

Table 2. ±1 embedding operation.

4. PRINCIPLES OF THE STEGO KEY SEARCH

One possible approach to the stego key search could be to first identify which samples have been modified and then try to reverse-engineer the PRNG that generated the embedding path. However, this approach is infeasible for several reasons. First, it is in general very hard to identify which samples have been modified. Second, even if we were able to identify the modified samples, we will not know the order in which they were modified and we will not know the complete path because on average 50% of samples were not modified because their parity already matched the message bit. Third, most PRNG are very hard to reverse-engineer in the sense of identifying the seed from the PRN sequence. For cryptographically strong PRNGs, this task is as complex as an exhaustive search for the key. Consequently, it seems that the only way to find the stego key is to use an exhaustive search, possibly combined with the dictionary attack. The key search algorithm proposed in this paper is of this type, as well.

To avoid any confusion, we formulate the stego key search more precisely. The steganographic algorithm may employ some form of a many-to-one mapping (e.g., a hash function) to map the user passphrase (or password) to the seed for the PRNG. As a result, it may be impossible to identify the user password itself using any search method. Thus, in this paper, when we speak about searching for the stego key, we are really searching for the seed that was used to initialize the PRNG rather than the user passphrase itself. In other words, if our search is successful, we will be able to find the correct seed, the embedding path, and extract the embedded bits even though we may not be able to recover the passphrase. Having said this, due to convenience we will never the less speak about stego keys and stego key search, meaning we are in fact searching for the PRNG seeds.

Now, we are ready to briefly outline the principles of our stego key search method. Let the cover image X be represented by N samples $\{x_i\}$, $i\in\{1, \ldots, N\}=I$. Depending on the image format, the samples x_i can be shades of gray, color indices, or DCT coefficients. Let K be the space of all possible stego keys that lead to different pseudo-random paths. After embedding the message, the stego image $S=\{s_i\}$ is obtained. During embedding, $m\leq N$ samples in X are visited (and potentially modified) along the path generated from the stego key $K_0\in\mathsf{K}$. Our task is to find the embedding key K_0 given only the stego image. We proceed in the following manner.

As the first step, we may filter the stego image to improve the SNR between the cover image and the stego signal. The filtering will also decorrelate the stego image samples. This preliminary step can improve the performance of the stego key search quite dramatically[17], especially for stego schemes that work in the spatial domain. For JPEG images, we do not perform this step, because individual DCT coefficients already exhibit little inter-block correlations.

For each key $K_j\in\mathsf{K}$, let $I(j)$ denote the set of sample indices visited along the path generated from the key K_j. Assuming the message embedded in the image is a random binary stream, in the sequence $\{s_i\}_{i\in I(0)}$ on average 50% of samples will already have the correct parity and 50% of them will be modified by the embedding operation. Thus, taking the first n samples, $n<m$ along the path generated from the correct key, the expected number of modified samples is $n/2$, while the expected number of modified samples along a path generated from an incorrect key is $n\times m/(2N)<n/2$ (as long as $m<N$). Thus, if the stego image is not fully embedded, the distribution of samples $\{s_i\}_{i\in I(0)}$ along the correct path will be different from the distributions taken along the incorrect paths $\{s_i\}_{i\in I(j)}$, $j>0$. Modeling the samples s_i as realizations of an i.i.d. random variable, their Probability Density Function (PDF) is their complete statistical characterization. Therefore, it makes sense to try to identify the correct key as the one producing an "outlier distribution" of samples s_i. To identify the outlier distribution, we test that the distribution of $\{s_i\}_{i\in I(j)}$ corresponds to an incorrect key.

Assuming the embedding changes are randomly scattered in the stego image, the density of embedding modifications in the whole image is the same as along an incorrect path. Thus, the expected distribution of n samples s_i along a path generated from an incorrect key can be obtained by calculating the PDF h of samples $\{s_i\}$ from the whole stego image (total of N samples). Then, for each key, we test that the samples $\{s_i\}_{i\in I(j)}$ are drawn from h. For this purpose, we employ non-parametric statistical tests, such as the chi-square test.

In Sec. 5, we provide further details of this approach and demonstrate its feasibility on a simulated F5 algorithm. In Sec. 6, we discuss practical issues for stego key search for OutGuess.

4.1 Search speed and candidates for the correct key

Because the size of the key space varies significantly among steganographic systems and can be quite large, the most important requirement for an effective stego key search algorithm is its speed with which it processes individual keys. To maximize the processing speed *and* the probability of finding the correct key in a reasonable amount of time, one can employ several measures:

a) The stego key search should start with a dictionary attack and inspect the most likely keys first.
b) The number of samples n along each path could be varied for each key based on the evidence we collect as we add more samples (see the paper by Chandramouli[18] for details).
c) The testing may consist of several hierarchical passes. All keys are first processed using a fast detector with an extremely low probability of missing a correct key but possibly with a high false positive rate. It will produce a smaller set of keys that is further processed using another test that has higher reliability but also higher computational complexity. We can cascade several detectors in this manner to maximize the speed of the search algorithm (an example of this approach is given in Sec. 6).
d) For many steganographic techniques, it is possible to estimate the relative message length (F5, J-Steg, OutGuess, spatial LSB embedding). This estimate gives us information on how to choose n and how many false outliers we can expect during the search.

It is possible that more than one key pass Step c) above. In fact, the number of keys that are identified as potentially correct strongly depends on the relative message length $q=m/N$, the number of samples n, the properties of the cover image, and the number of inspected keys N_K. To identify the correct key, for each candidate key we can determine the whole embedding path and inspect n samples that were not visited during embedding and are thus unmodified (complement checking). For an incorrect key, we expect statistical evidence compatible with an incorrect key, while for the correct key the samples' distribution should again be an outlier.

In the next section, we describe the details of the search algorithm for the JPEG format. The stego key search for images in spatial formats is treated in our forthcoming publication[17].

5. STEGO KEY SEARCH FOR JPEG IMAGES

Let X be the set of quantized DCT coefficients of the cover JPEG image, $|X|=N$. The stego image S is obtained from X by visiting (and potentially modifying) m coefficients along the path generated from the key K_0, $m<N$. Furthermore, let $L = \min S$ and $R = \max S$ be the smallest and largest DCT coefficients in S. The histogram of DCT coefficients of the first n samples from $I(j)$ will be denoted as $h_k(K_j, n)$, $k=L, \ldots, R$. To calculate the expected value and variance of $h_k(K_j, n)$ for an incorrect key, we formulate Lemma 1.

Lemma 1. *Let $Z = \{z_1, \ldots, z_N\}$ be a set of N real numbers and $I = \{1, \ldots, N\}$. For a fixed $n \le N$, let ω_n be a random variable defined as the sum of n randomly selected elements in Z*:

$$\omega_n = \sum_{\substack{i\in I_n, I_n\subset I \\ |I_n|=n, I_n random}} z_i \quad .$$

Then, $$E(\omega_n) = \frac{n}{N}\sum_{i=1}^{N} z_i \quad and \quad Var(\omega_k) = \frac{\frac{n}{N}\left(1-\frac{n}{N}\right)}{N-1}\left(N\sum_{i=1}^{N} z_i^2 - \left(\sum_{i=1}^{N} z_i\right)^2\right).$$

Proof: (see reference[22]).

Let h_k be the value of the *normalized* histogram of DCT coefficients obtained from the whole stego image (all N samples), $\sum_k h_k=1$. For a fixed k, define $z_i = \delta_{ks_i}$ for all $i\in\{1,\ldots, N\}$ and δ_{uv} is the Dirac symbol. Then, $h_k(K_j, n) = \sum_i z_i$, where i goes through the first n indices in $I(j)$. By Lemma 1,

$$E\{h_k(K_j,n)\} = \frac{n}{N}\sum_{i=1}^{N} z_i = \frac{n}{N} h_k N = nh_k \tag{1}$$

$$Var\{h_k(K_j,n)\} = \frac{\frac{n}{N}\left(1-\frac{n}{N}\right)}{N-1}\left(Nh_k N - N^2 {h_k}^2\right) = nh_k(1-h_k)\frac{N-n}{N-1}.$$

Assuming the key K_j is an incorrect key, the variable χ^2_{d-1}

$$\chi^2_{d-1}(K_j,n) = \frac{N-1}{N-n}\sum_{k=1}^{d}\frac{\left(h_k(K_j,n)-nh_k\right)^2}{nh_k} \tag{2}$$

has the chi-square distribution with d–1 degrees of freedom. For practical calculations, the histogram values $h_k(K_j, n)$, $k = L, \ldots, R$ should be grouped into d categories $c_1, \ldots, c_d$ to make sure that each category is well populated. In this section, we assume $d = R–L+1$ and $c_k = h_k$ for all $k = 1, \ldots, d$ for simplicity.

Assuming the key K_j is incorrect, the probability of obtaining the value $\chi^2_{d-1}(K_j,n) \geq t$ in the chi-square test is

$$p(j) = \frac{1}{2^{\frac{d-1}{2}}\Gamma\left(\frac{d-1}{2}\right)}\int_t^{\infty} e^{-\frac{x}{2}} x^{\frac{d-1}{2}-1}\,dx = \frac{\left(\frac{t}{2}\right)^{\frac{d-3}{2}} e^{-\frac{t}{2}}}{\Gamma\left(\frac{d-1}{2}\right)} + O\left(\frac{1}{t}\right). \tag{3}$$

During the key search, we will be focusing on those keys K_j that produce small values of $p(j)$ (or, equivalently, large values of the chi-square statistic). Such keys will be the candidates for the correct key K_0. At this point, it would be useful to know what values of $\chi^2_{d-1}(K_j,n)$ should be considered as outlier values and, also, how many "outlier" values one can expect after going through N_K keys. Obviously, this depends on m (or the message length), the properties of the cover image, the number of samples along each path n, and on N_K. To obtain a quantitative insight into this issue, first note that

$$\chi_{d-1}{}^2(K_0,n) = n\frac{N-1}{N-n}\sum_{k=1}^{d}\frac{\left(h_k(K_0,n)/n-h_k\right)^2}{h_k} \approx n\frac{N-1}{N-n}a_0\,, \tag{4}$$

where a_0 depends on the cover and stego image. This is because for the correct key K_0 $\lim_{n\to\infty} h_k(K_0, n)/n \neq h_k$. Note that the expected value of $\chi^2_{d-1}(K_j,n)$ over incorrect keys is $E[\chi^2_{d-1}(K_j,n)] = d-1$, which is independent of n.

The constant a_0 depends on the histogram of samples $\{x_i\}$ from the cover image and the relative message length $q=m/N$. The dependency on q is especially important. In Section 5.1, we derive an expression for a_0 for the F5 algorithm. Finally, assuming the key K_j is incorrect, from (3) and (4) the probability of obtaining a value $\chi^2(K_j,n) \geq \chi^2_{d-1}(K_0,n)$ is

$$P\left(\chi^2(K_j n) \geq \chi^2_{d-1}(K_0,n)\right) \approx \frac{\left(n\frac{N-1}{N-n}\frac{a_0}{2}\right)^{\frac{d-3}{2}} e^{-n\frac{N-1}{N-n}\frac{a_0}{2}}}{\Gamma\left(\frac{d-1}{2}\right)}. \tag{5}$$

To show how to further proceed with the analysis and how to calculate a_0, we will assume that the steganographic algorithm under inspection is the F5.

5.1 The F5 algorithm

Introduced by Westfeld[19] in 2001, the F5 embeds message bits as the LSBs of coefficients along a key-dependent random walk through all DCT coefficients of the cover image while skipping the DC coefficients and all coefficients that are zeros. If the coefficient's LSB does not match the message bit, the absolute value of the coefficient is always *decremented*. If the subtraction leads to a zero coefficient (the so-called *shrinkage* occurred), the same message bit must be embedded at the next coefficient, because at the receiving end, the message is extracted only from non-zero coefficients. As a special feature, the F5 algorithm employs matrix embedding to minimize the necessary number of changes to embed a message of certain length.

In this section, we discuss the key search for a modified F5 algorithm in which the matrix embedding is turned off (otherwise the F5 algorithm does not fit the paradigm of Sec. 3). Because matrix embedding essentially makes use of the whole image, the stego key search becomes much more difficult and less efficient when matrix embedding is used. In fact, we recommend matrix embedding as one of the possible mechanisms to thwart stego key search algorithms of the type described in this paper.

Let us denote the normalized histogram of the DCT coefficients in the cover image with capital letters H_k. Then, after embedding (and visiting m coefficients), the expected value (over different messages) h_k of the normalized histogram of DCT coefficients in the stego image is

$$\begin{aligned} h_k &= (1-q/2)H_k + q/2\,H_{k+1}, \quad \text{for } k>0,\\ h_0 &= H_0 + q/2\,H_{-1} + q/2\,H_1, \quad \text{for } k=0,\\ h_k &= (1-q/2)H_k + q/2\,H_{k-1}, \quad \text{for } k<0, \end{aligned} \tag{6}$$

where $q=m/N$ is the relative message length (q=1 for a fully embedded image). After substituting (1) and (6) into (4), after some algebra, we obtain

$$\chi^2_{d-1}(K_0,n) = n\frac{N-1}{N-n}\sum_{k=1}^{d}\frac{\left(h_k(K_0,n)/n - h_k\right)^2}{h_k} = n\frac{N-1}{N-n}\frac{(1-q)^2}{4}\chi_0 = n\frac{N-1}{N-n}a_0\,, \tag{7}$$

$$\text{where } \chi_0 = \sum_{k=L}^{-1}\frac{\left(H_k - H_{k-1}\right)^2}{h_k} + \frac{(H_1+H_{-1})^2}{h_0} + \sum_{k=1}^{R}\frac{\left(H_k - H_{k+1}\right)^2}{h_k}\,. \tag{8}$$

We note that χ_0^2 is still a function of q (because h_k depends on q), however the dependence on q is “weak” in the sense that $0<u_0\le\chi_0^2(q)\le u_1$ for all q, and u_0 and u_1 constants. We are now in the position to estimate the number of incorrect keys N_{out} producing outlier values of the chi-square statistic χ^2 comparable to the correct key after going through N_K keys:

$$N_{\text{out}} = N_K P\left(\chi^2 \ge \chi^2_{d-1}(K_0,n)\right) \approx N_K \frac{\left(n\dfrac{N-1}{N-n}\dfrac{(1-q)^2\chi_0}{8}\right)^{\frac{d-3}{2}} e^{-n\frac{N-1}{N-n}\frac{(1-q)^2\chi_0}{8}}}{\Gamma\left(\dfrac{d-1}{2}\right)} \tag{9}$$

5.2 Stego key search for F5

Because for natural images h_k has a sharp peak at 0 and then quickly falls off, we merged the histogram values into d=5 categories:

$$\begin{aligned} &\text{Category 1:} && c_1 = h_L + \ldots + h_{-2}\\ &\text{Categories 2–4:} && c_2 = h_{-1},\; c_3 = h_0,\; c_4 = h_1\\ &\text{Category 5:} && c_5 = h_2 + \ldots + h_R\,. \end{aligned} \tag{10}$$

The performance of the stego key search strongly depends on the parameters n, m (or q), and N_K. The number of samples n cannot be too small, otherwise the chi-square test would not have sufficient statistics. Also, smaller values of n lead to larger N_{out} necessitating “complement checking” (see Sec. 4.1) further slowing down the key search. Thus, there is a lower bound on the message length for which the stego key search is feasible.

q	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
N_{out}/N_K	10^{-43}	10^{-33}	10^{-25}	10^{-18}	10^{-12}	$10^{-7.5}$	10^{-4}	0.01

Table 3. Relative number of incorrect keys producing the chi-square value comparable with the correct key. The test image was a grayscale 864×1152, 80% quality JPEG, n=10^4, d=5.

On the other hand, messages with q close to 1 will also pose a problem because the difference in histograms between a correct and an incorrect path will be increasingly small as q approaches 1. This qualitative feeling can clearly be seen in (9) and is illustrated in Table 3.

To obtain the same number of outliers N_{out} for two different message lengths q_1 and q_2, the number of samples n_1 and n_2 must be chosen so that

$$\frac{n_1}{N-n_1}(1-q_1)^2 = \frac{n_2}{N-n_2}(1-q_2)^2 \tag{11}$$

However, this in turn imposes a lower bound on the message lengths for which we can perform the stego key search because in all cases $n \leq qN$.

To test our stego key search algorithm, a simple Matlab routine was implemented to search through 10^6 keys with $n=10^4$ samples for $q=0.3$ ($m=298600$) and the same test image as the one used to obtain Table 3. A Pentium IV, 2.7GHz PC went through all keys in 8.5h, which corresponds to 32 keys per second. The correct key was identified as an outlier (see Fig. 1.).

We acknowledge that the search could be *substantially* sped up by programming it in C rather than Matlab. However, the most decisive factor influencing the speed of the key search is the PRNG used for generating the random paths. Steganographic algorithms that generate a random permutation of all samples will lead to slower key searches than algorithms for which only a small portion of each path can be generated without having to produce the whole embedding path. The results for the F5 are given here for illustration of the search algorithm. Practical implementation issues, ideas for speeding up the search, and various tricks one can use for specific steganographic schemes are elaborated upon in the next section where we discuss OutGuess.

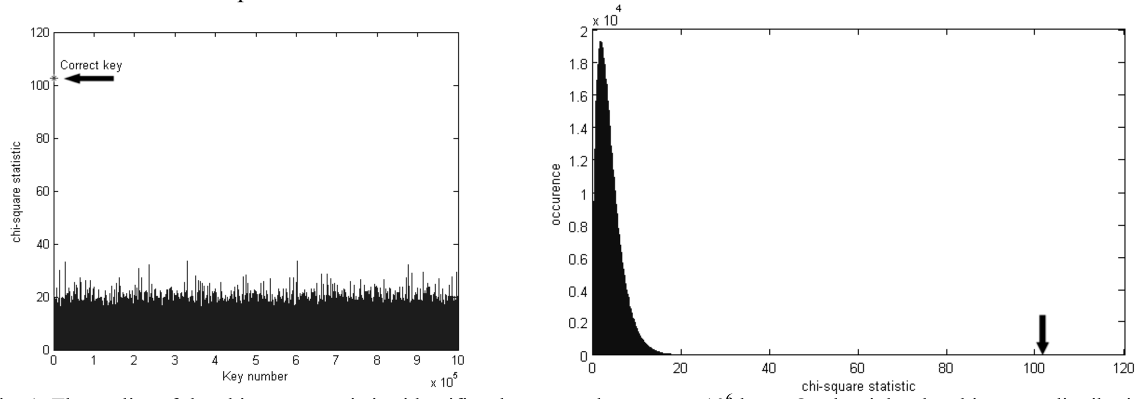

Fig. 1. The outlier of the chi-square statistics identifies the correct key among 10^6 keys. On the right, the chi-square distribution for $d=5$ degrees of freedom obtained from the experimental data (outlier denoted with arrow).

6. STEGO KEY SEARCH FOR OUTGUESS

In this section, we study the key search for the OutGuess algorithm[20]. We explore some important implementation issues in more detail, paying attention to the speed of the key search. Also, this study is an example of how one can proceed for a specific steganographic program and how to utilize specific embedding features to our advantage.

OutGuess was proposed by Provos[20] to counter the steganalytic attack proposed by Westfeld[10]. The embedding process proceeds in two passes. In the first pass, similar to J-Steg, OutGuess embeds message bits along a random walk into the LSBs of coefficients while skipping 0's and 1's. A pre-calculated number of DCT coefficients are left unmodified with the intention to adjust (after embedding) the histogram of the stego image to its original state. Thus, after embedding the image is processed again using a second pass. This time, corrections are made to the coefficients that were not visited during the first pass to make the stego image histogram match the cover image histogram. The program is freely available at http://www.outguess.org.

The embedded data always consist of two parts – a header and the message data. The header is embedded in coefficients determined by the secret shared stego key. It contains information about the message length and a session key. The embedding path in OutGuess is a function of the message length *and* the session key. The pseudo-random selection of coefficients proceeds sequentially through the image while the offset between selected coefficients is subjected to pseudo-random variations tuned so that OutGuess goes through the whole image during the first pass.

The sequential character of the path generation creates some difficulties for our stego key search. The first n coefficients from each path do not form a random sample of all coefficients. Thus, (2) will not have the chi-square distribution. This can be simply remedied by generating the complete embedding path (instead of the first n samples) and then randomly selecting from it n samples. While this approach indeed works as expected, it is slow because the whole path must be generated for each key.

To improve the search speed, we have decided to use a different approach. Because the embedding operation in OutGuess is LSB flipping (see Table 1), we expect $h_i(K_0, n) = h_{i+1}(K_0, n)$, $i = 2, 4, 6, \ldots$ and $i = -2, -4, -6, \ldots$ along the path from the correct key K_0. This is because LSB flipping equalizes the histogram values of coefficients that form an LSB pair. This observation is the basis of the "chi-square attack[10]" and is also the starting point for our stego key search for OutGuess. Again, denoting the normalized histogram of the cover image with capital H, the expected values of the stego image histogram h_k are

$$h_{2i} = q/2H_{2i+1} + (1-q/2)H_{2i},\ h_{2i+1} = (1-q/2)H_{2i+1} + q/2H_{2i}. \tag{12}$$

Each variable $h_k(K_j, n)$ follows a binomial distribution with the following mean and variance:

$$\begin{aligned} \text{Correct key:}\quad & E(h_k(K_j, n)) = n(h_{2i}+h_{2i+1})/2 = n(H_{2i}+H_{2i+1})/2 = n\,\bar{h}_{2i},\ k \in \{2i, 2i+1\} \\ & Var(h_k(K_j, n)) = n\,\bar{h}_{2i}(1-\bar{h}_{2i})(N_{01}-n)/(N_{01}-1), \\ \text{Incorrect key:}\quad & E(h_k(K_j, n)) = nh_k, \\ & Var(h_k(K_j, n)) = nh_k(1-h_k)(N_{01}-n)/(N_{01}-1), \end{aligned} \tag{13}$$

where N_{01} is the number of all DCT coefficients different from 0 and 1 and $\bar{h}_{2i} = (h_{2i}+h_{2i+1})/2$. The stego key search proceeds by testing for each key K_j that the histogram $h_k(K_j, n)$ corresponds to the *correct* key. The chi-square test is used to test that $h_{2i} = \bar{h}_{2i}$,

$$\chi^2_{d-1}(K_j, n) = n\frac{N_{01}-1}{N_{01}-n}\sum_{2i=L}^{R}\frac{\left(h_{2i}(K_j, n)/n - \bar{h}_{2i}\right)^2}{\bar{h}_{2i}}\ . \tag{14}$$

For the correct key K_0, (14) has a chi-square distribution with 4 degrees of freedom (over all random messages). For an incorrect key, from (12) and (13) we obtain

$$E[h_{2i}(K_j, n)/n - \bar{h}_{2i}]^2 = (H_{2i}-H_{2i+1})^2(1-q)^2/4 + (N_{01}-1)/(N_{01}-n)h_{2i}(1-h_{2i})/n\ .$$

In the expression above, we used the fact that $E(\xi^2) = E(\xi)^2 + Var(\xi)$ for any random variable ξ. Thus, for an incorrect key K_j and for large n, we can write:

$$\begin{aligned} E(\chi^2_{d-1}(K_j, n)) &= n\frac{N_{01}-1}{N_{01}-n}\sum_{i=1}^{d}\frac{(H_{2i}-H_{2i+1})^2(1-q)^2/4 + (N-n)/(N-1)h_{2i}(1-h_{2i})}{\bar{h}_{2i}} \cong \\ &\cong \frac{n(1-q)^2}{2}\frac{N_{01}-1}{N_{01}-n}\sum_{i=1}^{d}\frac{(H_{2i}-H_{2i+1})^2}{H_{2i}+H_{2i+1}}. \end{aligned} \tag{15}$$

Notice that the outlier value is again proportional to the product of the sample length n and $(1-q)^2$. The stego key search now proceeds by calculating (14) for each key while identifying outlier distributions that produce small values of the chi-square statistics. In sections below we describe in detail how the stego key search was customized for OutGuess.

For the chi-square test, we grouped the histogram values into d=5 categories consisting of four pairs (–4,–3), (–2,–1), (2,3), (4,5), and the remaining coefficients

$$\begin{aligned}&\text{Category 1–4: } c_1 = h_{-4},\ c_2 = h_{-2},\ c_3 = h_2,\ c_4 = h_4\\ &\text{Category 5: } c_5 = \ldots\ h_{-8}+h_{-6}+h_6+h_8\ldots .\end{aligned} \tag{16}$$

6.1 OutGuess random path generator

There are two published versions of OutGuess – version 0.13b and 0.2. Because both versions are very similar from the point of view of this paper, we describe them at the same time while pointing out their differences throughout the text. The embedding algorithm has the following features:

1. The pseudo-random path is determined by a user-specified pass phase. After the pass phrase is hashed using the MD5 hash function, the hash is used to initialize an RC4 PRNG.
2. The message is stored in LSBs of DCT coefficients whose selection depends on the OutGuess version
   Version 0.2 uses all DCT coefficients different from 0 or 1.
   Version 0.13b uses all *non*-DC coefficients whose values are neither 0 nor 1.
3. The message can be optionally "wrapped" using an error correcting code (ECC).
4. OutGuess tries many stego (session) keys and embeds the message bits while measuring the embedding distortion to the cover image. The stego key that gives the least distortion is finally selected for embedding. The program communicates the session key and the message length in a fixed 32-bit header along with the message.
5. The version 0.2 allows for correction of distortion of the global histogram so that the stego image histogram of DCT coefficients is virtually identical to the cover image histogram.

We have modified the original source code of the program (for both versions) to implement the following stego key search algorithm. The choice of the parameters T and n is discussed later.

6.2 Stego key search algorithm for OutGuess 0.13

```
0. Read the stego image under inspection and extract all DCT coefficients. Calculate the critical value $\chi^2_c$ of $\chi^2$ corresponding to 4 degrees of freedom for a given significance level (we used $\alpha$ = 0.001). Before embedding, OutGuess calculates the maximal message length $N_{\max}$ ($N_{\max}=N_{01}$ for version 0.13, and $N_{\max}=min\{N_{01}/2, 2N_{01}h_{-2}/(h_{-2}+h_{-1})\}$ for version 0.2).
1. Select a stego key $K$ to be tested.
2. Initialize the PRNG using $K$.
3. Read the header and extract the message length $m$ and the session key $K_S$.
4. Test the validity of the header
    a. If the option of using ECC has been selected, check the consistency of the ECC code. If it is not valid, $K$ cannot be the right stego key. Go to Step 9.
    b. If $m>N_{\max}$, reject $K$ as being wrong and go to Step 9.
    c. (Consistency check) If $m$ is not consistent with our a priori information about the message length, reject the key and go to Step 9.
    d. If $m<n$, reject the key and go to Step 9.
5. Seed the generator with the extracted session key $K_S$ and generate the embedding path.
6. Extract the first $n$ DCT coefficients from the path and test them as follows:
    a. Evaluate the chi-square statistics $\chi^2_{d-1}(K,n)$.
    b. If $\chi^2_{d-1}(K,n) < \chi^2_c$, reject the key and go to Step 9, otherwise continue.
7. $K$ is a potential key – it has passed the test based on $n$ coefficients. We repeat Step 6 with $n = m$ – the length of the alleged message extracted from the header. At this point, we could also engage the complement checking as described in Sec. 4.1.
8. If the sequence of $m$ coefficients passes the test, $K$ is declared the correct stego key and the search algorithm stops.
```

9. If there are any stego keys to be tested, go to Step 1. If there are no stego keys left, the search did not find any correct key.

In this algorithm, we first test whether the header corresponding to the key K could belong to a real message. For example, if the user specified the usage of an ECC code, this fact itself forms a constraint helping us reduce the number of operations because most keys will lead to an incorrect header, and thus the main bulk of computations can be eliminated (in our experiments, only about 15–25% of keys lead to a valid header).

Then, we extract n coefficients from the message and subject them to the chi-square test. If the test is negative, we pronounce the key in question as incorrect. However, if this test is successful, we retrieve all m DCT coefficients corresponding to the message and run the same test again for the whole sequence of m bits. If this test is positive, we have found the right key. At this point, we could also use the idea of complement checking (see Sec. 4.1).

The choice of the parameter n is quite important. If n is too small, the chi-square statistics may still be in its "transient phase" due to insufficient statistics. Thus, the correct key may not pass through Step 6 or, on the other hand, an incorrect key may be passed to Step 7, which unnecessarily increases the computational load. On the other hand, if n is too large, we waste our time by computing the chi-square statistics for large amounts of data where a smaller amount would have been sufficient. Even though we did not implement it, it would be possible to estimate the optimal value of the parameter n dynamically based on the percentage of keys that pass Step 6 and the average time spent in Steps 6 and 7. Balancing the parameter n during the test could improve the performance for large key sets. In our experiments, n = 700 (for the default quality factor in OutGuess) usually gave us the best ratio of tested keys per second. For larger messages, it is necessary to increase n as can be seen in (16).

The consistency check 2c) represents some a priori information we might have about the message length. For example, the steganalytic tool that drew our attention to the image might have provided us with an estimate m_e of the message length[2,3,4,1113]. If m is "significantly" different from m_e, we reject the key. The consistency check can further increase the speed of the stego key search.

The validity test in Step 4 has a major influence on the running speed. Typically, only about 15–25% percent of keys lead to a valid header. Similarly, the usage of the ECC code helps reduce the number of operations. For large n, excluding keys based on compatibility with the ECC improved the speed by a factor of 50. This gain becomes insignificant with smaller n (e.g., around 500).

The key search algorithm was tested on a variety of images with dimensions 1024×768. In each image, we have embedded messages of length 25%, 50%, and 100% of the maximal histogram-correctable message. Outguess was set to use the quality factor 75 (its default). The search rate of 20000–35000 keys per second was achieved on a Pentium IV machine HT (hyper threading) running at 2.4GHz, 512MB, 3200 DDR RAM. This rate would allow us to search through all numerical passwords of length up to 9 digits (~ 30-bit keys) in about 12 hours. The distribution of the statistic (15) for 1.1×10^6 keys and different values of n is shown in Fig. 2.

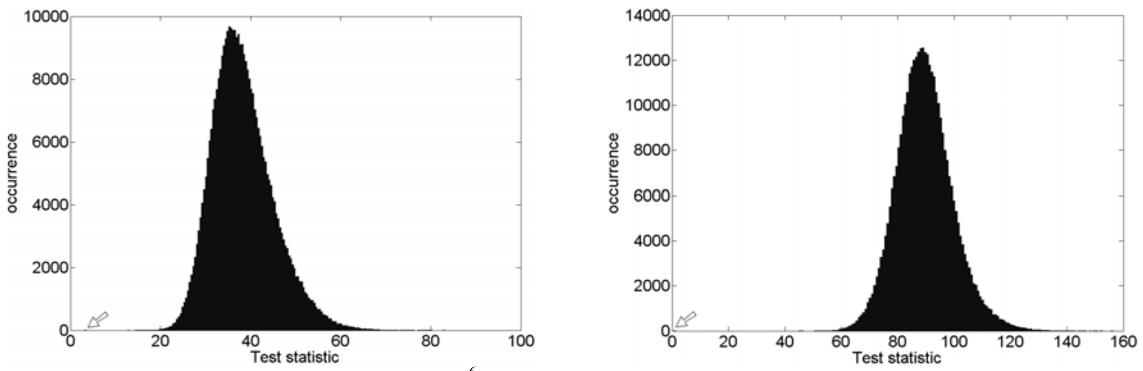

Fig. 2. Test statistic (15) for n=700 and 1500 for 10^6 keys. The outlier correct key is indicated with an arrow.

The key search speed is mostly influenced by the quality of the image, not by the length of the message. Because OutGuess never uses all possible DCT coefficients (Outguess limits the maximal message length to 50% of all available coefficients), we can detect stego keys even for fully embedded messages.

There are other aspects of Outguess that are relevant for the stego key search. First of all, the program uses the RC4 stream cipher to encrypt messages. Thus, *any* message is transformed into a pseudo-random stream and this makes the chi-square test extremely powerful and accurate. Second, Outguess uses a very fast algorithm to generate the pseudo-random embedding path. It goes through the image in a row-by-row manner and at each place it chooses the next coefficient by calculating a random shift from the current position; the maximum length of the shift is dynamically adapted every 8 bits according to the number of bits that remain to be embedded and the number of bits in the free space. Thus, the embedding path depends on the message length, not only on the key. This way, the message is spread uniformly. This path generation makes our search run much faster than if the embedding path was generated by permuting all DCT coefficients.

We close this section with one more observation. Any counter-measures that Outguess uses to fool the detectors do not affect our key search because we follow the decoding algorithm exactly as if we tried to recover the message and we stop before we encounter coefficients modified due to the histogram-correction step. This is why the stego key search can be performed in the same manner for both versions of OutGuess.

7. CONCLUSIONS

In this paper, we present a methodology for identifying the stego key in key-dependent steganographic schemes. Previous approaches for stego key search were exhaustive searches looking for some recognizable structure (header) in the extracted bit-stream. However, if the message is encrypted, the search becomes much more expensive because for each stego key, all possible encryption keys would have to be tested. In this paper, we show that for a very wide range of steganographic schemes, the complexity of the stego key search is determined only by the size of the stego key space and is independent of the encryption algorithm. The correct stego key can be determined through an exhaustive stego key search by quantifying statistical properties of samples along portions of the embedding path. The correct stego key leads to an outlier sample distribution. The search methodology is applicable to virtually all steganographic schemes, but this paper focuses on JPEG steganography.

Search techniques for spatial steganographic techniques are treated in our upcoming paper[17]. For spatial formats, we first preprocess the stego image to increase the SNR between the stego signal and the cover image. Denoising filters, together with the chi-square test can be used to detect stego keys even for stego schemes for which there is no realible steganalysis, such as +–1 embedding or stochastic modulation[15].

The existence of fast stego key search algorithms underlines the need for long steganographic keys and secure PRNGs. Combining a strong encryption algorithm with an insufficient stego key space may actually lead to successful attacks on the scheme. If the stego key search can be searched in a reasonable time, this method could be used as a detection method.

Besides making the stego key space large, there is one simple countermeasure that effectively prevents stego key searches similar to those described in this paper. If the embedding scheme used every sample in the image with the same probability, independently of the message length, our stego key search would fail. For example, to embed a message of relative length q, one can embed each message bit into disjoint groups of $floor(1/q)$ samples as their combined parity (XOR of parities of individual samples). At most one arbitrarily chosen sample needs to be changed in each group. The message can be padded if necessary to make sure all image samples were potentially used. Now, the samples along the correct path will have the same properties as samples along an incorrect key.

Another effective measure against the key search is the matrix embedding[19]. It essentially achieves the same effect as the group parity embedding and in addition minimizes the number of embedding changes, which further increases the security of the steganographic scheme. However, in the JPEG format, one can still determine the correct key when the groups in the matrix embedding are small, e.g., (1,2,4)-matrix embedding. This is because the DCT coefficients are distributed unevenly (generalized Gaussian) and bits extracted along an incorrect path exhibit slight bias by which one can identify the correct key, which always produces a random bit-stream (the messages in

F5 are encrypted). Using groups of 8 or more pixels in matrix embedding did not produce any measurable bias in the extracted bit-stream that one could use for stego key search.

ACKNOWLEDGEMENTS

The work on this paper was supported by the Air Force Research Laboratory, Air Force Material Command, USAF, under research grant number F30602-02-2-0093. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation there on. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Air Force Research Laboratory, or the U. S. Government.

REFERENCES

1. R.J. Anderson and F.A.P. Petitcolas, "On the Limits of Steganography", *IEEE Journal of Selected Areas in Communications*, Special Issue on Copyright and Privacy Protection, vol. 16(4), pp. 474–481, 1998.
2. J. Fridrich, M. Goljan, and R. Du, "Detecting LSB Steganography in Color and Gray-Scale Images", *Magazine of IEEE Multimedia*, *Special Issue on Security*, October-November issue, pp. 22–28, 2001.
3. S. Dumitrescu, Wu Xiaolin, and Z. Wang, "Detection of LSB Steganography via Sample Pair Analysis", In: *LNCS*, vol. 2578, Springer-Verlag, New York, pp. 355–372, 2002.
4. T. Zhang and X. Ping, "A New Approach to Reliable Detection of LSB Steganography in Natural Images", Signal Processing, vol. 83, No. 10, pp. 2085–2094, 2003.
5. H. Farid and L. Siwei, "Detecting Hidden Messages Using Higher-Order Statistics and Support Vector Machines", In: LNCS, vol. 2578, Springer-Verlag, New York, pp. 340–354, 2003.
6. J.J. Harmsen and W.A. Pearlman, "Steganalysis of Additive Noise Modelable Information Hiding", *Proc. EI SPIE Electronic Imaging*, Santa Clara, January 21–24, pp. 131–142, 2003.
7. R. Tzschoppe, R. Bäuml, J.B. Huber, and A. Kaup, "Steganographic System based on Higher-Order Statistics", *Proc. EI SPIE Electronic Imaging*, Santa Clara, January 21–24, pp. 156–166, 2003.
8. N. Provos and P. Honeyman, "Detecting Steganographic Content on the Internet", CITI Technical Report 01-11, 2001.
9. Steganography software for Windows, http://members.tripod.com/steganography/stego/software.html
10. A. Westfeld and A. Pfitzmann, "Attacks on Steganographic Systems", In: *LNCS*, vol.1768, Springer-Verlag, New York, pp. 61−75, 2000.
11. J. Fridrich, M. Goljan, and D. Soukal, "Higher-Order Statistical Steganalysis of Palette Images", *Proc. EI SPIE: Electronic Imaging*, Santa Clara, January 21–25, pp. 178–190, 2003.
12. A. Westfeld, "Detecting Low Embedding Rates", In: *LNCS*, vol. 2578, Springer-Verlag, New York, pp. 324–339, 2003.
13. J. Fridrich, M. Goljan, and D. Hogea, "New Methodology for Breaking Steganographic Techniques for JPEGs", In *Proc. EI SPIE Electronic Imaging*, Santa Clara, January 21–24, pp. 143–155, 2003.
14. J. Fridrich and R. Du, "Secure Steganographic Methods for Palette Images", In: *LNCS*, vol. 1768, Springer-Verlag, New York, pp. 47–60, 2000.
15. J. Fridrich and M. Goljan, "Digital Image Steganography Using Stochastic Modulation", *Proc. EI SPIE Electronic Imaging*, Santa Clara, January 21–24, pp. 191–202, 2003.
16. T. Sharp, "An Implementation of Key-Based Digital Signal Steganography", In: *LNCS* vol. 2137, Springer-Verlag, New York, pp. 13–26, 2001.
17. J. Fridrich, M. Goljan, D. Soukal, and T. Holotyak, "Forensic Analysis: Determining the Stego Key from Stego Images", submitted to the 6th Information Hiding Workshop, Toronto, May 23–25, 2004.
18. R.Chandramouli and N.D. Memon, "On sequential watermark detection", to appear in *IEEE Transactions on Signal Processing, Special Issue on Signal Processing for Data Hiding in Digital Media and Secure Content Delivery,* 2003.
19. A. Westfeld, High Capacity Despite Better Steganalysis (F5–A Steganographic Algorithm). In: *LNCS*, vol. 2137, Springer-Verlag, New York, pp. 289–302, 2001.
20. N. Provos, "Defending Against Statistical Steganalysis", 10th USENIX Security Symposium, Wash. DC, 2001.
21. F. Alturki and R. Mersereau, "A Novel Approach for Increasing Security and Data Embedding Capacity in Images for Data Hiding Applications", *Proc. of ITCC*, Las Vegas, Nevada, pp. 228–233, 2001.
22. J. Fridrich and M. Goljan, "On Estimation of Secret Message Length in LSB Steganography in Spatial Domain", to appear in *EI SPIE Electronic Imaging*, San Jose, January 18–22, 2004.