A Method for Automatic Identification of Signatures of Steganography Software

Graeme Bell and Yeuan-Kuen Lee

***Abstract*—A fully automated, blind, media-type agnostic approach to steganalysis is presented here. Steganography may sometimes be exposed by detecting automatically characterized regularities in output media caused by weak implementations of steganography algorithms. Fast and accurate detection of steganography is demonstrated experimentally here across a range of media types and a variety of steganography approaches.**

***Index Terms*—Blind, media-type agnostic, steganalysis, steganography.**

I. Introduction

Steganography is the field of research that studies how secret data can be hidden in carrier media, without being detectable either to normal human observation or programmatic scrutiny. Steganalysis [1]–[5] is the field of research that (primarily) seeks methods to detect steganographic media. Broadly speaking, steganalysis can be separated into targeted steganalysis, which aims to break a particular known steganographic embedding scheme, and blind steganalysis, which attempts to identify the existence of steganography without *a priori* knowledge of the specific algorithm that was used. Universal steganalysis may be used to refer to those methods that are potentially capable of identifying many forms of steganography through a single system. Here, we add the idea of being media-type agnostic, whereby a method makes no assumptions regarding steganography, steganography algorithms, media type (e.g., pictures, music, text), or media data format (e.g., TIFF, JPG), making it suitable in principle for use against any steganography approach. The problem addressed here is that of blind, media-type agnostic steganalysis. It is well-established that it is very difficult but also very useful to develop techniques that work broadly and blindly against steganography, without knowledge of the algorithm being attacked, the secret message or the original carrier media [1], [5], [6]. For example, Fridrich notes in [5] that

> having a few candidate stego-objects without any additional information (e.g., "JPEG in the wild") is the hardest case for steganalysis.

In this paper, blind steganalysis is attempted by attacking the implementations of steganography algorithms rather than the steganography algorithms themselves. Consequently, steganalysis may be achieved by proxy, when the attack is successful. The principle of the technique is as follows.

In order to make steganographic research applicable in the real world, it is necessary to implement theoretical steganography algorithms as executable programs. In the process of realizing an abstract algorithm as a concrete piece of code, mistakes may be made by the programmer that allow the presence of steganography to be exposed. Where they exist, these implementation-induced artefacts are separate to those produced by the nature of a steganographic embedding method, such as the "pair of values" artefact [2], which the majority of research attention in steganalysis has been directed towards until now. Implementation-induced artefacts are, however, a valid and interesting target for steganalytic research, as the primary goal of steganography (covert communication) is defeated whenever steganographic and nonsteganographic media can be easily distinguished by *any means*. As Westfeld states in [2]

> The goal is to modify the carrier in an imperceptible way only, so that it reveals nothing—neither the embedding of a message nor the embedded message itself.

In the past, discovery of implementation artefacts during research has occurred infrequently and on an ad-hoc basis. This paper discusses a new steganalytic method by which the attempted discovery of these implementation-induced peculiarities can be automated. The method is applied in two stages. First, some known-stego training examples produced by a steganography tool are presented to a training algorithm, along with a larger number of examples of nonstego files of the same media type.[1] The training algorithm attempts to identify any characteristic regularities in the binary representation of the known-stego files that are not present in the nonstego files. If regularities are found, then a signature is immediately produced that may be used to help detect output from that steganography tool, and by proxy, to help detect the existence of steganography. Conversely, this means that this method will not operate at all against implementations of algorithms that do not produce characteristic regularities in their output. Tool signatures, where they are found, may be used by comparing the bits of a given file against the known bit values representing the regularities of the signature. This test of binary data values against signature values may be carried out very quickly. Unfortunately, a match against a signature does not provide a *guarantee* that a particular tool has been used, nor does it *guarantee* that steganography is present, as sources of false-positives may exist. However, a match against a stego-signature can provide a useful indication that a particular tool may have been used and consequently an indication that the file may contain steganography. This property means that this method can effectively complement existing steganalysis techniques and help to improve overall accuracy.

The structure of this paper is as follows. Section II introduces the background and principles. Section III presents an algorithm. Sections IV and V measure and evaluate the technique experimentally. Section VI proposes developer guidelines. Section VII presents conclusions, and is followed by references.

II. Background and Motivation

The approach described in this paper was inspired by an observation regarding the implementation of F5 [7]. Provos observed [5] that the F5 implementation added a characteristic header field which was seldom seen in nonsteganographic media. The presence of this characteristic field allowed the presence of secret data to be reasonably assumed. Another well-known example is provided by the GIF steganography tool, STools [8]. It was observed that STools accidentally outputs a characteristic palette which is not commonly seen in GIF files [9]. Consequently, stego-media produced by this program can be very rapidly and reliably identified. Unfortunately, this style of steganalysis is time consuming, and requires an expert with plenty of free time, patience, and a measure of good luck. Nonetheless, these anecdotes provide a compelling motivation for the construction of a similar, but more systematic and automated "blind regularity detection" attack against steganography tools (and thus steganography, by proxy).

[1]Though it is blind, this technique still requires some output samples from a steganography tool in order to attempt an attack.

Manuscript received June 27, 2009; revised March 13, 2010; accepted March 13, 2010. Date of publication March 29, 2010; date of current version May 14, 2010. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Nasir Memon.

G. Bell is with the School of Information Technology, Murdoch University, Perth, WA 6150, Australia (e-mail: stego@graemebell.net).

Y.-K. Lee is with the Department of CSIE, Ming Chuan University, Gueishan 333, Taiwan.

Digital Object Identifier 10.1109/TIFS.2010.2046985

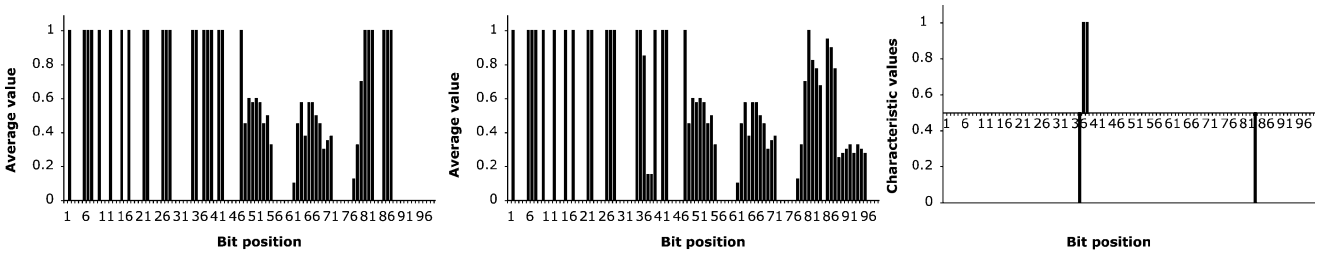

Fig. 1. (a) The average of the first 100 bits of 40 stego-GIFs produced with STools. (b) The average of the first 100 bits of 40 randomly selected nonstego GIFs. (c) The most "unusual" characteristic bits of the stego-GIFs. In (c), uninteresting bit positions are assigned the value 0.5 to distinguish them from positions that are characteristically 1 or 0 only in the stego-media.

To understand the principle of this technique, imagine that "cow," "dog," "cat," and "fox" represent the output of a steganography program given different secret messages and different carrier media. The output data clearly changes each time at the human-observable and semantic level. Even at the byte-level, there is no obvious regularity that is always present in the output. However, if we look at the bit-level representation of these ASCII characters, we see that several bit positions never change in value. Additionally, any regularity in the byte-level representation will necessarily also appear as a regularity in the bit-level representation. This means that the "obvious" types of regularities noted by human researchers in the past can be discovered along with new, more subtle vulnerabilities. This paper will, therefore, look at how the bit-representation of files may be used to quickly and effectively discover fixed, unchanging patterns of bits in files, particularly in the header data of files. Here, a fixed-bit-signature for some file might appear as "$\ldots 01100xxx0110xxx1011x\ldots$," where x is a wild-card bit, a bit-position whose value we are unconcerned about when determining the possible presence of steganography. More general attacks also exist. For example, it is possible to look for values that occur "unusually often," rather than "every time." Statistical or AI-based classification could be performed using each separate bit position of the file as a feature to be considered. Another form of attack might look for fixed patterns of values whose location in the file may vary (rather than remain fixed in one position).

In this paper, a simple approach is sufficient to introduce and demonstrate steganalysis based upon automated regularity discovery at the bit-level. This provides advantages in terms of a straightforward implementation and small, quick-to-check signature profiles. No attempt is made to understand media files; for example, by analyzing the meaning of data content, or the nature of data structures or metadata fields. In this way, this paper's attack can be generally and directly applicable (without modification or parameterization) against susceptible implementations of steganography utilizing any present or future form of digital media.

III. Method

To determine the locations of candidate fixed-bits in stego-media files, it is necessary to take into consideration the typical value of a bit in a particular location in the files, and how it varies from natural, nonstego media. The typical value of a bit position in a nonstego media file is not 0.5, as might be expected [Fig. 1(b)[2]]. Rather, it depends on the kind of file that is being considered. For a JPEG file, we might expect some fixed value bit-positions making up "0xFF" or "JFIF" to occur naturally within a normal file simply because it is a JPEG, without being indicative of stego-media.

The method of this paper is, therefore, to first take average (mean) of the bit value at each bit position, across several examples[3] of a particular tool's outputted steganographic files. We record the bit locations which are "unusual," i.e., vary dramatically from the expected average value of 0.5. In this implementation, we only record the bits that always have a value of "1," or always have a value of "0." In other words, *what is unusual about the bit values at each position of the average stego-file produced by a particular steganography tool (e.g., a stego-GIF)*? An example is shown in Fig. 1(a).

Next, to try to avoid accidentally recognizing nonsteganographic files, we consider the mean bit-position values of several natural, nonstego examples of the same kind of media file. In other words, *what is unusual in the average normal media file (e.g., a GIF)*? An example is shown in Fig. 1(b).

Finally, we remove from our stego fixed-bit-signature those bits that regularly have the same particular values even in nonsteganographic examples of that data type, as shown in Fig. 1(c). Any remaining fixed-value bits will represent a short, highly characteristic fixed-bit-signature, or fingerprint of the output, for a particular version of a steganography tool. Where such signatures are found, they allow steganalysts to very quickly determine if a file could or could not be an example of stego-media produced by that particular version of the tool.

This whole process can be conveniently summarized in the form of a single question: *What unusual characteristics exist in the bit-representation of the average output from a single steganographic tool, when it is compared against the average output of nonsteganographic tools that use the same medium?*

The algorithms for constructing signatures and testing media are very brief, and operate as follows:

1) First, calculate the fixed stego bits. S is an empty set. At every bit position P in a number of stego-files of a given media type:
   a) Calculate the average bit value. If position P has an average bit value of "1" or "0" (always ON–OFF), record it as a stego-media signature bit position in S: e.g., "P: fixed value: 0."
2) Next, remove weakly characteristic bits. At every bit position P in a number of nonstego-files, of the same media type:
   a) Calculate the average bit value. If bit position P "naturally" has an average value of "0" (or thereabouts[4]), and if position

[2]This example shows that the most characteristic areas may be only 1 or 2 bits wide, and difficult to observe as a regularity at the byte-level. It also shows the average for bits 87-88 and 90-95 varied somewhat between stego and nonstego GIFs.

[3]This approach can work with as few as 10–50 training examples, which is important given that many steganography programs have only a GUI interface, and no facility for convenient batch production of samples.

[4]"Thereabouts" is intentionally imprecise. The algorithm performs similarly in practice over a wide range of values. In this paper's experiments, signature bits were those with a fixed value of 1 (or 0) in positions where nonstego pictures had average values of 0.6 or less (or 0.4 or more, respectively). Other cut-off values generally produced similar, often near-identical results.

TABLE I
EXPERIMENTAL RESULTS SHOWING EFFECTIVENESS AGAINST WELL-KNOWN STEGANOGRAPHY TOOLS

			Stego media			Non-stego media		
Program name (version)	Media type	Size of signature	True +ve rate	False -ve rate	Avg. signature bit matches	True -ve rate	False +ve rate	Avg. signature bit matches
Steganos (10.0.5)	JPEG	0 bits	No sig.	-	-	-	-	-
Inv. Secrets (4.6.3)	JPEG	22 bits	100%	0%	100%	100.0%	0.0%	53.6%
OutGuess (0.2)	JPEG	400 bits	100%	0%	100%	99.4%	0.6%	37.4%
JSteg (4.0)	JPEG	400 bits	100%	0%	100%	99.4%	0.6%	37.4%
STools (4.0)	GIF	30 bits	100%	0%	100%	99.6%	0.4%	38.7%
MP3Stego (1.1.17)	MP3	82 bits	90%	10%	100%	100.0%	0.0%	31.3%
OutGuess (Exp. 2)	JPEG	400 bits	100%	0%	100%	98.0%	2.0%	35.7%

P is noted as a "0" stego-media fixed signature bit, then discard signature bit position P from S.

b) If bit position P "naturally" has an average value of "1" (or thereabouts), and if position P is noted as a "1" stego-media fixed signature bit, then discard signature bit position P from S.

3) To test a file F, compare the signature-bit positions in S against the bits of file F using boolean AND. If every bit-signature value perfectly matches, and no values fail to match, we consider the file to be stego-media.[5] Otherwise, the file is not considered to be an example of stego-media.

IV. EXPERIMENTATION

To test this technique, output files from six steganography tools were compared with regular stego-free examples of the same kind of media. These six tools are the only steganography programs that have ever been tested by the authors using this attack, i.e., no other positive or negative known results have been excluded from this experiment. The steganography programs tested here operate on different kinds of media, use different steganography algorithms, and were implemented by different authors. Two of these programs are current-day commercial products intended for the serious purpose of real-life data hiding: "Invisible Secrets" and "Steganos Data Security Suite." These products were evaluated using trial editions from late 2008. The other programs tested were "OutGuess," "JSteg," "MP3Stego," and "STools." In all cases, the most recently available version of the program was used. A maximum signature limit of 400 fixed bits was used to avoid unnecessarily large fingerprints. Wherever possible, default settings were used for each program, to represent the most common form of use of each tool. In each case, messages embedded into the files were short, less than 50 bytes long. The message content, message length, and password selected were varied between samples. The carrier media files used here varied in file-size, dimensions, and original content, and averaged approximately 100 KB (JPEG, GIF). The relationship between the carrier file size and the secret data size allows this experiment to test whether this attack can identify steganographic media even when only a tiny fraction of the available secret data capacity has been used. In the past, specialized steganalysis techniques have had to be developed to specifically tackle this situation [10]. In other words, we seek to uncover steganography in the challenging situation of minimal secret data, where even the most famous techniques for traditional steganalysis—such as the chi-squared attack [2], the block-discontinuity attack [11], and RS-steganalysis [12]—may fail.

To build the signature for each program, 40 samples of clear-media were gathered and 40 samples of stego-media were produced. Additionally, 500 nonsteganographic JPEGs and 500 nonsteganographic GIFs were gathered from over 50 different internet sources (excluding steganography sites) and used to measure the false-positive rate. Similarly, 100 nonsteganographic MP3s were gathered from over 30 different internet sources. To measure the true-positive rate, 10 previously unencountered test samples were produced from each steganography program. Stego-media files were generated manually using program GUIs or with batch interfaces.

The bit-signature produced by the attack was then tested to see if this technique could reliably separate the 10 previously unencountered examples of stego-media from up to 500 previously unencountered non-stego-media files. At no point was human interaction, selection of features, customization, or parameter tuning necessary; the experimental software was simply presented with a set of clear images, a set of stego images for a particular tool, and a set of test images to be separated into stego and clear. Images were tested in isolation, and were not ranked to determine the likelihood of containing steganography. An image was only classified as steganographic (positive, **+ve**) if 100% of the bits in the fixed-bit-signature matched perfectly.[6] Otherwise, the image was classified as "clear" (negative, **-ve**) in this experiment.

V. DISCUSSION OF RESULTS

Table I (excluding the bottom row) shows the results of the experiment. The approach proved capable of reliably separating clear files from stego-media produced by a relatively wide range of products, including one commercial steganography product, and even when the embedded data size was less than 0.05% of the original file size. This approach can be very computationally efficient at detecting steganography, given the small number of bits to be checked per media file, and the extremely low computational cost of comparing a bit position against a known signature value.

The authors were surprised to discover that most of the tested steganography implementations, and in particular a current-day commercial product, had so many characteristic bit values present in all outputted stego-media. It seems that the practically minded, real-world steganalyst can sometimes side-step the need to break theoretical steganographic algorithms when aiming to detect steganography.

The authors note that the true-positive rate found in the five successful cases (against previously unencountered samples) was 100% in almost all studied cases, with the exception of a single "near-miss" result for MP3Stego, where one stego-file resulted in a 99% match against the signature rather than 100%. The false-positive rate seen here across all tested signatures is also good, and in two of the studied cases, optimal within these experiments. A low false-positive rate is an extremely important characteristic for steganalytic attacks [4].

Considering the results for each tool in turn: Steganos is a modern-day commercial steganography tool. Steganos did not prove vulnerable to this attack. Examination of the media produced by Steganos indicated that it does not alter the information present at the

[5]Requiring a perfect match is strict, but it is essential for steganalytic techniques to target a very low false-positive rate [4].

[6]For general use, we recommend further classification steps, i.e., this technique is best used to complement rather than replace other methods—as there is no guarantee that a signature corresponds uniquely to a stego-tool.

start of files (the header), but instead keeps the original file's header data as much as possible. Consequently, this paper's attack technique was unable to identify any characteristic bits within the file at fixed positions. This negative result demonstrates that well-implemented steganography tools can overcome the attack of this paper.

Invisible Secrets is a modern-day commercial steganography tool. However, here it was possible to reliably and correctly separate 100% of all tested examples of previously unencountered regular media and stego-media by analyzing only 22 bits of data from each file. The authors feel this result effectively demonstrates the potential of this steganalysis technique.

STools is an older GIF-based steganography program. It was straightforward to reliably and correctly separate all tested examples of previously unencountered regular media and stego-media. This demonstrates that the method of this paper operates successfully across different media types besides JPEG. The result for STools was quite interesting to the authors, because STools is known to produce characteristic output [1], previously discovered by human researchers. However, this technique automatically identified several areas of the stego-media that were unusually characteristic in STools output, that were unmentioned in steganalysis literature previously. A visual example is provided in Fig. 1 of this paper. This result is perhaps unique within steganalysis research, in that it appears to be an example of a software steganalysis system improving upon a previous human research finding, by automatically discovering a selection of further new regularities that are only observable at the bit-level.

JSteg and OutGuess, two older but famous JPEG-based steganography tools, proved very vulnerable to the attack and have identical fingerprint results. This is perhaps unsurprising, as both of these programs are based on the same IJG reference source code. Both of these tools were used with a 75% quality setting, as this is the default value automatically provided by OutGuess. However, the size of the signature may suggest that (in part) it is the "75%" default quality setting of these programs that is being attacked, which produces a large characteristic pattern near the start of the file. Very few natural images were found that had been output by programs developed from the IJG source code, using a quality setting of 75%. This result raises an important issue—the default settings of a steganography tool should not make steganalysis straightforward![7] Consequently, a second experiment was conducted to investigate the consequences of varying quality settings. OutGuess allows values in the range 75–100, so 26 different files were generated. Training was conducted using 16 randomly chosen files, and the resulting signature was used to try to detect the remaining 10 unencountered files with previously unencountered quality settings. The result is shown in the bottom row of the table. The false-positive rate increased, but the technique successfully generalized in this situation where a program parameter was varying. This allowed detection of steganography in cases where previously unencountered nondefault parameters were used.

Finally, MP3Stego is a steganography program that embeds data in the frames of MP3 music files. Here, the signature that was automatically produced resulted in a 90% true-positive rate and a 0% false-positive rate during experiments. A single case was observed where the signature resulted in a 99% match (81 bits rather than 82) which was classed as a false-negative. The MP3Stego result demonstrates that this paper's technique is effective for blind steganalysis of other media types besides image files.

These results represent all of the programs that the authors have tested so far against this new technique. There remain hundreds of other steganography programs that have never been tested for such vulnerabilities. The only factor that seemed to distinguish vulnerable implementations from nonvulnerable implementations in these experiments was the issue of whether a "fresh" file was generated by the software, or whether the steganography software "meddled" with an original file. If all programmers took care to retain the original headers and structure of cover media files, it would be hard for the attack in this paper to be so successful. Generating a fresh output file with predictably structured headers and data fields is no doubt convenient and pleasant for programmers, but the price is the failure of steganography when attacked by the method of this paper. It was also found that basing a steganography tool upon a particular reference implementation can be dangerous in its own right. If a tool is not updated, and the reference code becomes less commonly used, that style of output may become characteristically associated with the stego-tool. It appears that the use of default values may make steganographic output from tools easier to detect via the method of this paper. We also note, as a warning, that other media tools will exist (now or in future) whose output is not represented within the media samples used here. Such tools could in principle raise or lower false-positive rates. This is a danger that all steganalysis experiments face: false-positives may be generated in the real world by causes that are not represented in experimental sample sets. We suggest real-world steganalysts can most effectively address the problem of unanticipated causes of false-positives by combining many steganalysis techniques.

Finally, the authors wish to report a further result that came to their attention during peer review. An anonymous referee has privately produced an independent implementation of this technique based upon the text of this paper, and has tested it on a privately selected data-set, independently and without involving the authors. The referee noted that they have confirmed the results of this paper.

VI. Proposed Guidelines

The authors suggest the following guidelines for those implementing steganography tools.

1) Avoid creating fresh new media files. Instead, "meddle" with a copy of the carrier file directly.
2) Avoid altering carrier file metadata wherever possible.
3) Produce varied output rather than having a fixed output format.
4) Do not give default parameter values for the user's convenience.
5) Avoid output that "merges with the crowd" by explicitly mimicking particular nonstego programs, or by using popular reference code. In future, the output may not resemble "the crowd" at all.
6) Automatically retest each new version of your program against the method of this paper.
7) If your implementation is only intended to demonstrate an algorithm, rather than for serious use in hiding data, then make this very clear to all potential users.

VII. Conclusions and Future Work

This paper has presented an automated bit-signature-building steganalytic attack that can be applied against any susceptible implementations of any type of steganography algorithm. This is achieved here in a blind and media-type agnostic manner, under traditionally challenging conditions (steganography-only, across multiple media types, against tiny data payloads, against multiple different unknown algorithms, against multiple different implementations, without parameterization, and without the assistance of stego-features provided by human steganalysts). This attack has demonstrated clear vulnerabilities in popular, current-day steganography software, circa late 2008, and in a range of older, famous software. Future research might involve

[7]One way for steganography tool implementers to address this challenge may be to remove default settings from their programs, and instead force users to provide values, or auto-configure from a range of acceptable values.

the implementation of unexplored ideas that were mentioned in this paper—such as header-parsing-based analysis, as opposed to flat-file, fixed-bit, media-type agnostic analysis.

The primary contributions of this paper are first, drawing attention to similarity detection at the bit-representation level as a way of attacking weak implementations of steganography, and second, showing how "fixed bits" provide a means to rapidly construct small and effective signature profiles that allow fast, effective, cross-media, blind steganalysis. The authors observe that if this form of steganalytic attack had ever been described previously, it is unlikely that so many programs—in particular, at least one commercially produced steganography product, circa late 2008—would today be so vulnerable to the attack. Consequently, the authors consider this approach to be a simple, novel, and useful contribution to the science of steganalysis.

ACKNOWLEDGMENT

The authors wish to thank the anonymous referee who independently implemented and verified this technique during peer review.

REFERENCES

[1] N. F. Johnson and S. Jajodia, "Steganalysis of images created using current steganography software," in *Proc. Int. Workshop Information Hiding, ser. LNCS*, London, U.K., 1998, vol. 1525, pp. 273–289.

[2] A. Westfeld and A. Pfitzmann, "Attacks on steganographic systems," in *Proc. Int. Workshop Information Hiding, ser. LNCS*, London, U.K., 2000, vol. 1768, pp. 61–76.

[3] J. Fridrich and M. Goljan, "Practical steganalysis of digital images: State of the art," *Proc. SPIE, Security and Watermarking of Multimedia Contents IV*, vol. 4675, pp. 1–13, 2002.

[4] N. Provos and P. Honeyman, "Hide and seek: An introduction to steganography," *IEEE Security Privacy*, vol. 1, no. 3, pp. 32–44, May/Jun. 2003.

[5] J. Fridrich, "Steganalysis," in *Multimedia Security Technologies for Digital Rights Management*. Amsterdam, The Netherlands: Elsevier, 2006, ch. 14, p. 352.

[6] K. Curran and K. Bailey, "An evaluation of image-based steganography methods," *Int. J. Digital Evidence*, vol. 2, no. 2, 2003.

[7] A. Westfeld, "F5–a steganographic algorithm: High capacity despite better steganalysis," in *Proc. Int. Workshop Information Hiding, ser. LNCS*, 2001, vol. 2137, pp. 289–302.

[8] A. Brown, STools v4.0 [Online]. Available: ftp://ftp.demon.net/pub/mirrors/crypto/idea/code/s-tools4.zip

[9] N. F. Johnson and S. Jajodia, "Exploring steganography: Seeing the unseen," *Computer*, vol. 31, no. 2, pp. 26–34, 1998.

[10] A. Westfeld, "Detecting low embedding rates," in *Proc. Int. Workshop Information Hiding, ser. LNCS*, 2002, vol. 2578, pp. 324–329.

[11] J. Fridrich, M. Goljan, and D. Hogea, "Attacking the OutGuess," in *Proc. ACM Workshop on Multimedia and Security*, 2002, pp. 1–4.

[12] J. Fridrich, M. Goljan, and R. Du, "Detecting LSB steganography in color and gray-scale images," *Special Issue on Security, IEEE Multimedia*, vol. 8, no. 4, pp. 22–28, Oct./Dec. 2001.

[13] G. Bell, "The dangers of webcrawled datasets," *First Monday* vol. 15, no. 2, Feb. 1, 2010 [Online]. Available: http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/2739/2456

[14] T. Pevný and J. Fridrich, "Novelty detection in blind steganalysis," in *Proc. ACM Workshop on Multimedia and Security*, 2008, pp. 167–176.

[15] M. Kharrazi, H. T. Sencar, and N. Memon, "Benchmarking steganographic and steganalysis techniques," *Proc. SPIE, Security, Steganography, and Watermarking of Multimedia Contents VII*, vol. 5681, pp. 252–263, 2005.