Content-independent steganography and steganalysis of JPEG images

Mohammad Rezaei
Security Analysis Laboratory
Tehran, Iran
rezaei@salab.ir

M. Bagher Salahshoor
Security Analysis Laboratory
Tehran, Iran
salahshoor@salab.ir

Abstract— **This paper studies the possibilities of hiding data within a JPEG file without making any changes to the image content, and in a way that any JPEG decoder normally opens the file. These possibilities are found by careful studying the JPEG file format. A steganalysis system needs to check these simple embedding techniques before complicated analyses of DCT coefficients. We also present a number of system attacks on JPEG files, which can lead to detection of hidden data. The attacks are based on the fingerprints left in the stego file by JPEG encoders, or the steganography algorithm itself.**

Keywords—Steganography, steganalysis, data hiding, JPEG, file format

I. Introduction

Steganography is concerned with embedding secret data into an innocuous looking object such as digital image so that no one suspects the existence of hidden data. On the contrary, *steganalysis* aims at detecting the presence of hidden data [1]. JPEG is the most widely used image format that has attracted increasing attention of researchers in the fields of steganography and steganalysis [1-3]. Various steganographic methods for JPEG images have been proposed. Most of the methods embed secret data by manipulating the quantized *discrete cosine transform* (DCT) coefficients [4, 5]. In contrast, almost all JPEG steganalytic methods aim at detecting abnormal statistical artifacts resulted from data hiding in DCT coefficients. However, the stego data could be added directly to the JPEG file stream without making any changes to DCT coefficients and the content of the image. Therefore, this way, the steganalytic methods will definitely fail to find out the presence of the stego information.

Most of the JPEG steganographic methods change the *least significant bit* (LSB) of DCT coefficients [6]. One of the pioneer methods in JPEG data hiding is *JSteg*, in which the message bits are successively replaced by the LSBs of quantized DCT coefficients, skipping those coefficients with the values 0 and 1. JSteg is highly detectable because of disrupting the characteristic properties of the histogram of DCT coefficients [1]. Therefore, a number of steganographic methods such as *OutGuess* [7] were designed to preserve statistical properties of the original image [1]. The idea is to embed the message in part of the DCT coefficients and reserve the remaining part for correcting selected statistical properties such as the histogram of coefficients. *Model-based* steganography is based on a similar idea, that preserves a selected model of the DCT coefficients [8]. There are also several heuristic steganographic methods. Among those, *F5* is the most popular, which decrements the absolute value of the coefficients instead of replacing LSBs in order to maintain the histogram of coefficients. It also employs *matrix embedding* technique to minimize the number of changes required for embedding a message [9]. Another group of steganographic methods is designed in such a way that minimum distortion is introduced to the image, which intuitively makes it harder for an adversary to detect the hidden information. *Perturbed quantization* method, as an example, attempts to minimize the overall distortion by performing quantization and data embedding at the same time, so that minimum quantization error is produced [10].

Steganalytic methods can be divided into two main categories: *specific* and *universal*. A specific or targeted method constructs features for a known steganographic method and therefore, may work well only for that specific method. A universal or blind steganalytic method, in contrast, can detect different types of embedding methods [4]. A large sets of features are usually extracted, and a classifier is trained with features to distinguish between clean cover and stego images [11].

Chi-square attack is the most notable early targeted method that can detect steganography if the message is embedded sequentially. However, it was later generalized to randomly distributed messages [12]. Numerous other methods have been proposed afterwards [13-17]. Farid proposed one of the first blind methods, in which 72 features are constructed from statistics of the wavelet transform of the image [18]. Blind steganalytic methods have the advantage of potential ability to work for any steganographic method, however, they are less accurate than targeted approaches, in general [12]. Pevny and Fridrich [5] argue that extracting features directly from DCT domain provides better results for a wide spectrum of JPEG steganographic methods.

Modern steganalysis provides highly accurate results for most current steganographic schemes [12, 19, 20]. Almost all existing JPEG steganalytic methods work based on different statistical properties between stego and cover images as a result of embedding in the DCT domain. The statistical properties are derived directly from DCT domain or other domains such as wavelet and spatial. These steganalytic methods, however, simply fail if the stego data is added directly to the JPEG file stream because no change is made to DCT coefficients and image content. A few current steganographic programs offer this data hiding option [21]. Although, it is a difficult problem to hide messages in the JPEG stream in a secure manner [21], but a steganalysis system should first examine the JPEG file stream for detecting such kind of data embedding techniques. To achieve this goal, an in-depth understanding of JPEG file format is essential.

Inspecting the JPEG file stream for hidden data is needed even if the stego data is hidden in DCT coefficients, because there might be indications of steganography in the file protocol. For example, those steganographic methods that modify the quantization table usually produce an unusual table which is an indication of steganography. In this way, sometimes the steganalysis system can detect the hidden data much easier than complicated analyses of the image content. This study also helps steganographers to design a secure system by noticing the issues that can occur in JPEG file stream.

In this paper, we introduce several possible places in a JPEG file to hide data so that image viewers or decoders normally open the image. This study is important for steganalysis systems to consider processing a JPEG file before analyzing the content. We also introduce a few techniques, based on investigating JPEG file properties, to attack steganographic methods that leave traces behind in JPEG file. These techniques are successful even if the stego data is hidden in the content of the image, for example, in quantized DCT coefficients.

II. JPEG FILE FORMAT

JPEG, which stands for "Joint Photographic Expert Group", was created in 1992 as a standard for compressing images. The JPEG standard is so extensive, but only a small part of it is used in common [22]. We provide in this section an overview of JPEG image compression method and JPEG file format. The compression process, as shown in Fig. 1, can be summarized in five main steps:

1. The input RGB color space is typically converted to a luminance/chrominance space such as YCbCr. The next steps are applied to each component *Y*, *Cb*, and *Cr*, separately.

2. The image component is partitioned into 8×8 pixel blocks. The chrominance components may be subsampled before partitioning into blocks.

3. Each block is transformed to frequency domain using a 2-*D* DCT.

4. Each DCT coefficient in a block is quantized by dividing it by the corresponding integer number taken from an 8×8 quantization table, and the result is rounded to an integer number.

5. The 64 coefficients of each block are reordered by scanning the block in a zigzag manner. The Huffman coding is then applied, and the resulting bitstream is written to the file. The header information, that is required for decoding the compressed image, and the following bitstream construct the JPEG file.

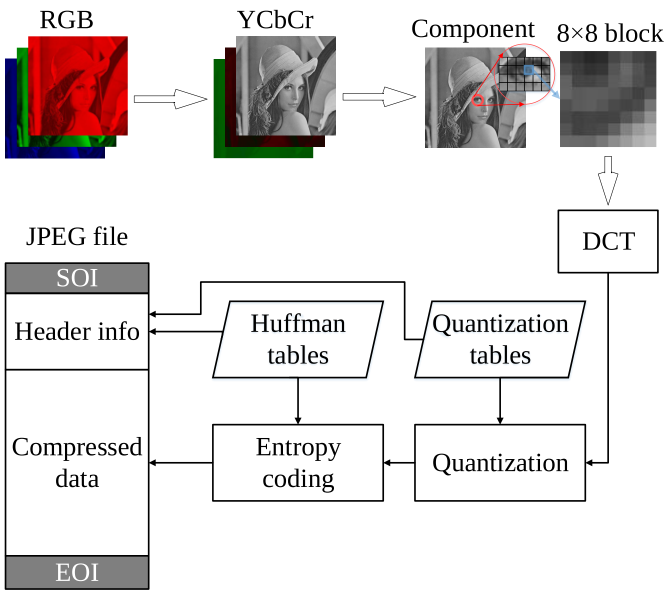

Fig. 1. Overview of JPEG encoding

A JPEG file is segmented by certain two-byte codes called markers. The first byte of a marker has the value 0xFF, and the second byte, that specifies the marker type, takes a value in the range [0x01, 0xFE]. The JPEG file starts with *Start of Image* (SOI) marker and ends with *End of Image* (EOI) marker. These are the only markers that are stand-alone with no data following. Other markers are followed by two bytes containing the number of bytes in the data field plus two. A marker with its associated data is called *marker segment*, see Fig. 2 [22]. The structure of the data in marker segment is specified based on the marker type. For example, the data structure of *Define Quantization Table* (DQT) marker contains one byte representing the quantization table number and precision of its elements followed by 64 values of the 8×8 table, see Fig. 3. The image compressed data is usually started after last marker segment of the file, and ended before EOI [22].

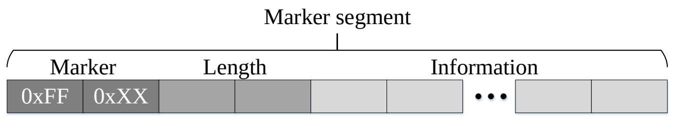

Fig. 2. Structure of marker segment

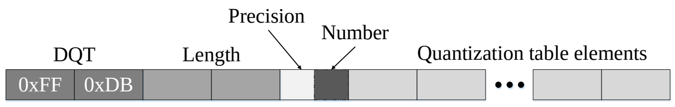

Fig. 3. DQT marker segment

III. Content-independent steganography

In this section, we introduce four possible places to hide data within a JPEG file stream.

A. *Stego data at the end of the file*

JPEG standard defines a data stream which starts and ends with specific markers SOI and EOI, respectively [22]. If the stego data is embedded before SOI, an error occurs according to standard, and therefore, JPEG decoders fail to open the file [23]. However, the standard does not specify any restrictions to add extra data after EOI, and accordingly, the JPEG decoders usually do not check the extra data, see Fig. 4. Several steganography tools use this naïve data hiding approach such as Camouflage, JpegX, and Data Stash. The stego data is lost by any image editing operations, and it is simply detected by expert steganalyzers if they check for extra data after EOI [24]. However, the steganalytic methods that only process the image content are unable to detect this simple approach.

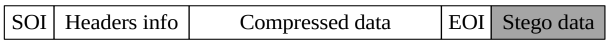

Fig. 4. Hiding data after EOI

B. *Stego data in the APP and COM marker segments*

Stego data can be embedded in *application marker segments* denoted by APP*n*, *n*=0, 1,…, 15, and *comment marker* denoted by COM, see Fig. 5. These markers except APP0 (JFIF) and APP1 (Exif), may locate anywhere within a JPEG file stream, and are ignored by the photo editor/viewer software [22]. They should be used in a way that no interfere occurs with the decoding process [23]. JPEG standard allows applications to use application marker segments (0xFFE0~0xFFEF) for defining an application specific meaning of the data. These markers are not necessary for decoding JPEG file [25].The data held in the APP*n* markers, and their format are application specific. Photo editor/viewer applications use these markers for adding more information than what the JPEG standard has specified. The marker data can be skipped by using the length field of the marker [22].

Current files with JPG or JPEG extensions are always in *JPEG file interchange format* (JFIF), which is introduced by Eric Hamilton. JFIF uses the gaps in JPEG standard to create simple JPEG encoded files that can be interchanged among applications. "JFIF" and "JPEG file" have become synonymous [22]. APP0 (0xFFE0) marker is used by JFIF immediately after SOI for inserting additional information and thumbnail image. The *extended file information format* (Exif) uses APP1 (0xFFE1) to prevent conflict with JFIF [25]. Exif stores the information about digital camera, and is preferred image format for cameras in ISO 12234-1 standard. The stego data could be replaced with the information in APP0 and APP1 markers. Another possibility is to append the data at the end of the segment, and update the length field. This way, hiding data does not change previous information in the APP marker.

We propose two methods to detect the stego data in JFIF/Exif APP markers. First, one can check for the fixed data that is expected in parts of JFIF/Exif information. For example, the following sequence should be found in a JFIF file: X'FF', SOI, X'FF', APP0, <2 bytes to be skipped>, "JFIF", X'00' [26]. Second, inserting the stego data in place of the marker segment or at the end of it can break the certain structure of the JFIF/Exif marker segment. Therefore, the steganalyzer should inspect the file to see if the structure is as expected or not. In general, the steganalyzer should always carefully inspect the APP*n* markers. The COM marker has an application specific interpretation, and is used for storing a text comment such as copyright information. A COM marker, unlike APP*n* markers, is expected to contain only plain text data [22]. Therefore, other types including encrypted data can be an indication of hidden information.

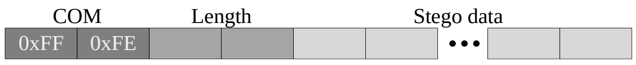

Fig. 5. Hiding data in COM marker segment

C. *Stego data between two marker segments*

According to JPEG standard, a marker should not necessarily locate immediately after the previous marker segment [22]. After the JPEG decoder completes reading a marker and its data, it seeks to find the next marker, and ignores the data in the middle. Therefore, the stego data could be embedded between two marker segments, see Fig. 6.

Adding stego data before EOI marker is a specific case. Although the standard clarifies that the EOI marker must immediately appear after Huffman encoded data of the last scan [22], many photo editor/viewer software do not consider this rule. They normally open the image ignoring the extra information between the last scan and EOI.

Gimp and Matlab JPEG Toolbox provide warnings for both types of the above embedding techniques. They specify and display the place where the extra data is added. However, they properly decode the image to be used.

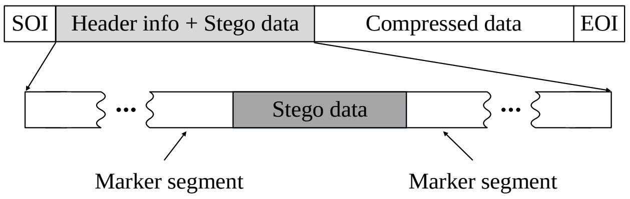

Fig. 6. Hiding data between two marker segments

D. *Stego data in quantization tables*

Multiple quantization tables are allowed by the JPEG standard to be defined, each identified by a DQT marker. A JPEG file may normally include up to four DQT tables, specified by the numbers 0 to 3. Each table is associated with a component of a scan in the image by mentioning the number of the table. The stego data can be embedded in the DQT tables that are defined but never used for decoding the image, see Fig. 7. For example, all four tables could be defined, where only one table is used for decoding and the others contain the message information.

An alternative technique is possible for hiding information in DQT tables. JPEG standard allows to redefine a DQT table, for example number 0. The new defined table is replaced with the older one, and is used for decoding, afterwards [23]. We can insert the stego data in a DQT table and immediately redefine it.

The steganalyzer can detect the hidden data by checking whether a DQT table is really referenced and used for decoding.

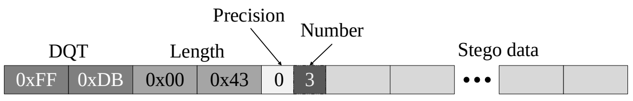

Fig. 7. Hiding data in an unused quantization table

IV. Content-independent steganalysis

Steganography software and tools use JPEG encoders to hide data in the image content such as DCT coefficients. Unintentional fingerprints left in stego files by the encoders can be used to detect the hidden data. This kind of attack, which is performed without analyzing the image content, is called system attack [11]. The system attack is performed based on protocol weaknesses in a steganography product. Inspecting files for these weaknesses is very important because it can simply lead to detecting the hidden data even if a modern steganographic method with a low embedding rate is used. In the following, we provide two examples.

A. *Suspicious COM comment*

Open source JPEG encoders available on the internet usually add a specific comment to the JPEG file in the COM marker segment. A number of images with the same comment that are not from a common photo editor software, can be an indication of steganography. For example, the JPEG encoder used for implementing the well-known F5 steganography algorithm, always adds the following comment: "JPEG Encoder Copyright 1998, James R. Weeks and BioElectroMech" [11].

B. *Unusual quantization table*

The values in the quantization tables in the DQT markers are not specified by the JPEG standard. An application is free to define its own tables. Fig. 8 shows two sample tables offered by the standard, one for luminance and the other for chrominance [22]. The tables may be scaled linearly by a quality factor (e.g. an integer number from 1 to 100) to allow the user to have a smaller file size at the cost of a lower image quality. A few existing JPEG encoders use this simple scaling with the two tables provided by the JPEG standard [27], however, several photo editor software such as *Adobe Photoshop* and *Corel Paint Shop*, and numerous digital cameras define their own quantization tables. For example, Photoshop uses 13 compression levels, which are associated with different quantization tables. Analysis of the quantization tables have been used for identifying the creator device or software [28].

Several steganographic methods change the quantization table in order to reduce the distortion caused by data embedding [29-32]. However, this makes the quantization table unusual, and the stego file becomes vulnerable to the system attack. For example, the steganalyzer can compare the quantization table with a database of quantization tables used in common devices or software. The JPEG file is suspicious of carrying hidden data if its quantization table is not equal to any table in the database.

The steganographer might use a JPEG encoder to produce a stego file, and then modify some of the parameters in the file header to the parameters of a well-known photo editor software such as Photoshop to deceive the steganalyzer. However, inspecting all properties of the software might reveal the fact that the file has not been really produced by that software. For example, suppose that the steganographer has changed the comment and APP markers information to that of Photoshop. Checking the quantization table of the file can show that the table does not belong to Photoshop, and therefore, the JPEG file is a fake Photoshop file.

Luminance quantization table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Chrominance quantization table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Fig. 8. Sample quantization tables for luminance and chrominance offered by the JPEG standard.

V. Experiments

We next examine five well-known photo editor/viewer software applications to find out how they react to input JPEG images containing hidden information embedded directly to the file bitstream. The applications include Gimp, Photoshop, Win10 Photos, Matlab JPEG toolbox, and Win10 Paint. The stego data is embedded in seven different places in JPEG files including:

- End of file after EOI marker
- Immediately before EOI marker

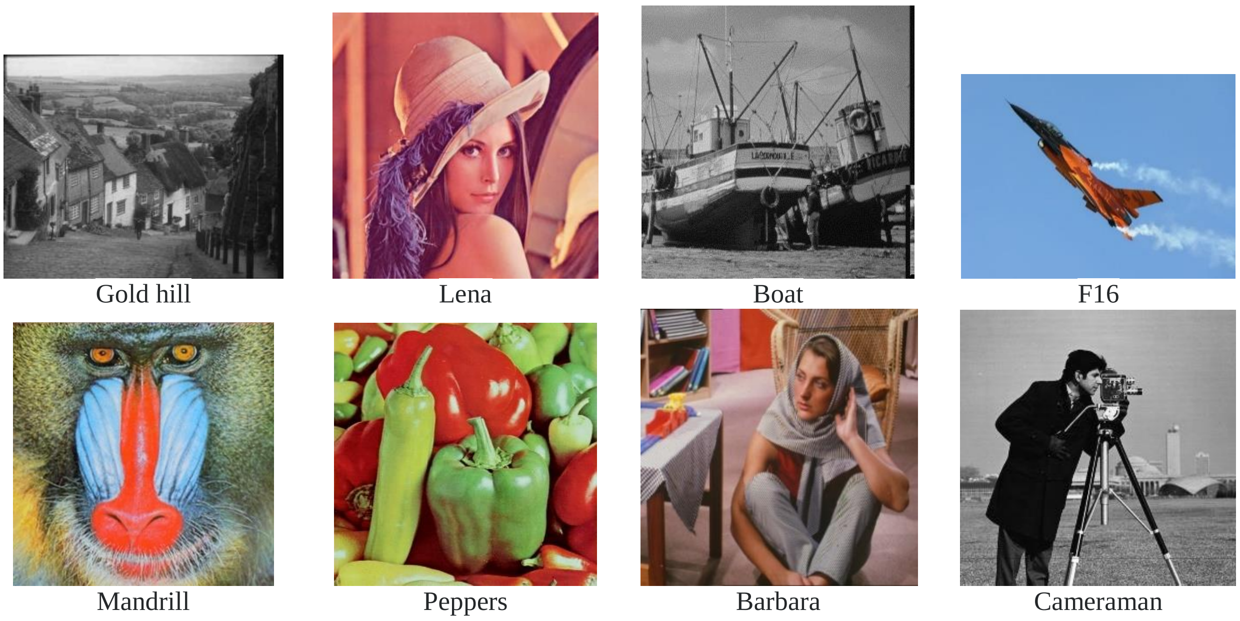

Fig. 9. JPEG image database

TABLE I. REACTION OF SELECTED SOFTWARE TO CONTENT INDEPENDENT STEGANOGRAPHY

Hiding method	**software**				
	Gimp	Photoshop	Win10 Photos	Matlab JPEG Toolbox	Win10 Paint
End of file	OK	OK	OK	OK	OK
Before EOI marker	Warning	OK	OK	Warning	OK
APP marker segment	OK	OK	OK	OK	OK
Beginning of file	Error	Error	Error	Error	Error
JFIF/Exif segment	OK	OK	OK	OK	OK
Between marker segments	Warning	OK	OK	Warning	OK
DQT marker segment	OK	OK	OK	OK	OK

- In APP marker segment (0xFFE8)
- Before SOI marker
- JFIF/Exif marker segment
- Between two marker segments
- In DQT marker segments

The stego data contains 106 encrypted bytes, which is embedded in each of the eight popular images shown in Fig. 9. Considering seven places for hiding data and eight images, 56 stego images were produced. We used the software Hex Editor (HxD) 1.7.7.0 to embed data in the JPEG files.

The results of decoding the stego images by the software applications are shown in Table 1. Three cases may happen when decoding:

- Normally decode the file (OK)
- Decode the file but with warning, which indicates extra data (Warning)
- Cannot decode (Error)

All the applications normally open the images that have embed data in four places including end of file, APP marker segment, JFIF/Exif segment, and DQT marker segment. Gimp and Matlab JPEG toolbox give warning and specify the position and length of the stego data, while other software applications normally decode the images. Stego data before the SOI marker is the only case in which all the software applications fail to decode the images and give error.

VI. CONCLUSIONS

Inspecting JPEG file bitstream is important both for steganography and steganalysis. Before analyzing the image content, the steganalyzer should consider the possibility that the stego data is embedded within the file stream. On the other hand, the steganographer should be careful that the embedding process and creating JPEG file do not leave traces in the JPEG file, which can be indications of steganography. This paper presents four data hiding methods that embed the data within the file bitstream. Furthermore, two content-independent steganalysis techniques are proposed by inspecting the file properties: Unusual comments left by JPEG encoders and unusual quantization tables created by some steganography algorithms.

REFERENCES

[1] J. Fridrich, T. Pevný, and J. Kodovský, "Statistically undetectable jpeg steganography: dead ends challenges, and opportunities," in *Proceedings of the 9th workshop on Multimedia & security*, 2007, pp. 3-14: ACM.

[2] V. Holub and J. Fridrich, "Low-complexity features for JPEG steganalysis using undecimated DCT," *IEEE Transactions on Information Forensics and Security,* vol. 10, no. 2, pp. 219-228, 2015.

[3] T. Pevný and J. Fridrich, "Towards multi-class blind steganalyzer for JPEG images," in *International Workshop on Digital Watermarking*, 2005, pp. 39-53: Springer.

[4] B. Li, J. He, J. Huang, and Y. Q. Shi, "A survey on image steganography and steganalysis," *Journal of Information Hiding and Multimedia Signal Processing,* vol. 2, no. 2, pp. 142-172, 2011.

[5] T. Pevny and J. Fridrich, "Merging Markov and DCT features for multi-class JPEG steganalysis," in *Electronic Imaging 2007*, 2007, pp. 650503-650503-13: International Society for Optics and Photonics.

[6] Y. Q. Shi, C. Chen, and W. Chen, "A Markov process based approach to effective attacking JPEG steganography," in *International Workshop on Information Hiding*, 2006, pp. 249-264: Springer.

[7] N. Provos, "Defending Against Statistical Steganalysis," in *Usenix security symposium*, 2001, vol. 10, pp. 323-336.

[8] P. Sallee, "Model-based steganography," in *International Workshop on Digital Watermarking*, 2003, pp. 154-167: Springer.

[9] A. Westfeld, "F5—A Steganographic Algorithm," in *Information Hiding: 4th International Workshop, IH 2001, Pittsburgh, PA, USA, April 25-27, 2001. Proceedings*, 2001, vol. 2137, p. 289: Springer Science & Business Media.

[10] J. Fridrich, M. Goljan, and D. Soukal, "Perturbed quantization steganography," *Multimedia Systems,* vol. 11, no. 2, pp. 98-107, 2005.

[11] J. Fridrich, *Steganography in digital media: principles, algorithms, and applications*. Cambridge University Press, 2009.

[12] J. Fridrich, "Feature-based steganalysis for JPEG images and its implications for future design of steganographic schemes," in *International Workshop on Information Hiding*, 2004, pp. 67-81: Springer.

[13] J. Fridrich, M. Goljan, and D. Hogea, "Steganalysis of JPEG images: Breaking the F5 algorithm," in *International Workshop on Information Hiding*, 2002, pp. 310-323: Springer.

[14] J. Fridrich, M. Goljan, and D. Hogea, "Attacking the outguess," in *Proceedings of the ACM Workshop on Multimedia and Security*, 2002, vol. 2002: Juan-les-Pins, France.

[15] B. Li, J. Huang, and Y. Q. Shi, "Steganalysis of YASS," *IEEE Transactions on Information Forensics and Security,* vol. 4, no. 3, pp. 369-382, 2009.

[16] K. Lee, A. Westfeld, and S. Lee, "Category attack for LSB steganalysis of JPEG images," in *IWDW*, 2006, pp. 35-48: Springer.

[17] K. Lee, A. Westfeld, and S. Lee, "Generalised category attack—improving histogram-based attack on JPEG LSB embedding," in *International Workshop on Information Hiding*, 2007, pp. 378-391: Springer.

[18] S. Lyu and H. Farid, "Detecting hidden messages using higher-order statistics and support vector machines," in *International Workshop on Information Hiding*, 2002, pp. 340-354: Springer.

[19] M. Goljan and J. Fridrich, "CFA-aware features for steganalysis of color images," in *SPIE/IS&T Electronic Imaging*, 2015, pp. 94090V-94090V-13: International Society for Optics and Photonics.

[20] M. B. Desai and S. Patel, "Survey on Universal Image Steganalysis," *International Journal of Computer Science and Information Technologies,* vol. 5, no. 3, pp. 4752-4759, 2014.

[21] J. Fridrich, M. Goljan, and R. Du, "Steganalysis based on JPEG compatibility," in *ITCom 2001: International Symposium on the Convergence of IT and Communications*, 2001, pp. 275-280: International Society for Optics and Photonics.

[22] J. Miano, *Compressed image file formats: Jpeg, png, gif, xbm, bmp*. Addison-Wesley Professional, 1999.

[23] T. Recommendation, "CCITT T. 81," 1993.

[24] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt, "Digital image steganography: Survey and analysis of current methods," *Signal processing,* vol. 90, no. 3, pp. 727-752, 2010.

[25] T. Tachibanaya, "Description of Exif file format," *URL http://park2. wakwak. com/tsuruzoh/Computer/Digicams/exif-e. html*, 2001.

[26] E. Hamilton, "JPEG File Interchange Format Version 1.02," 1992.

[27] R. Neelamani, R. De Queiroz, Z. Fan, S. Dash, and R. G. Baraniuk, "JPEG compression history estimation for color images," *IEEE Transactions on Image Processing,* vol. 15, no. 6, pp. 1365-1378, 2006.

[28] H. Farid, "Digital image ballistics from JPEG quantization: A followup study," *Department of Computer Science, Dartmouth College, Tech. Rep. TR2008-638,* vol. 7, pp. 1-28, 2008.

[29] H. W. Tseng and C. C. Chang, "High Capacity Data Hiding in JPEG-Compressed Images," *Informatica,* vol. 15, no. 1, pp. 127-142, 2004.

[30] H.-W. Tseng and C.-C. Chang, "Steganography using JPEG-compressed images," in *Computer and Information Technology, 2004. CIT'04. The Fourth International Conference on*, 2004, pp. 12-17: IEEE.

[31] H. Kobayashi, Y. Noguchi, and H. Kiya, "A method of embedding binary data into JPEG bitstreams," *Systems and Computers in Japan,* vol. 33, no. 1, pp. 18-26, 2002.

[32] A. Almohammad, R. M. Hierons, and G. Ghinea, "High capacity steganographic method based upon JPEG," in *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, 2008, pp. 544-549: IEEE.