Has F5 Really Been Broken?

Johann A. Briffa* and Hans Georg Schaathun* and Ainuddin Wahid Abdul Wahab*+

*Department of Computing, University of Surrey, Guildford GU2 7XH, England
+Faculty of Computer Science and Information Technology, University of Malaya, Malaysia

Keywords: Steganalysis, F5.

Abstract

The publicly-available F5 software (F5Software) implementation takes a possibly compressed cover image, decompresses it if necessary, and embeds the hidden message during a second compression process. This procedure introduces a risk that the stego image goes through 'double compression'. While this is not a problem from the embedding and extraction point of view, any steganalysis process trained on such a scheme will potentially detect artifacts caused either by the embedding process or the second compression process. In this paper we review published steganalysis techniques on F5. By re-implementing an isolated F5 embedding algorithm excluding the decompression and recompression process (F5Py), we show that published steganalysis techniques are unable to defeat F5 when its ideal operational condition is not violated. In other words, published techniques most likely detected the compression artifacts rather than the embedding process when the message size is not exceeding the optimum F5 capacity. This is an important fact that has been ignored before. Furthermore, we look for the optimum embedding rate for F5 in order for it to take advantage of matrix encoding for better embedding efficiency. From here we found that the low embedding rate considered for F5 in the previous works are actually relatively high for it. This is also important since bigger message size might degrade F5 to F4. In addition, we also verify that, as expected, steganalysis performance depends on the message size.

1 Introduction

There is a battle going on, between steganographers and steganalysts. Steganographers aim to develop ever better techniques to hide a secret message in a media file, such as an image, such that the enemy will not even suspect that there is a secret there. The steganalyst is the enemy who tries to devise techniques to detect reliably the hidden secrets. There has been a lot of activity in the last 15 years.

Current research tends to be rather ad hoc. Evaluation of new methods are generally based on simulations, and the conditions and parameters for these simulations are not always clear. When they are clear, they may be controversial.

This leads to several problems in the evaluations. For instance, simulations must be based on specific implementations which are not always correct implementations of the algorithms specified in the literature. Many authors assume that relatively long messages would be embedded, and this gives the best detection results on their part. But what message length was the stego-system designed for?

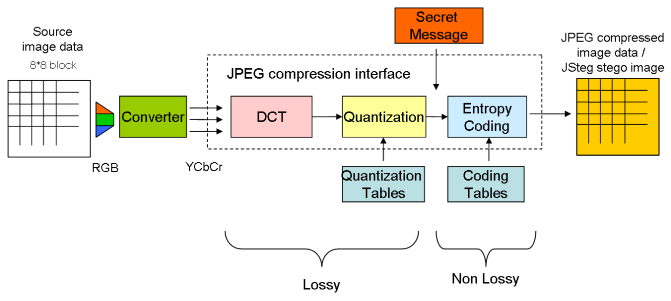

Figure 1. JPEG steganography embedding algorithm between two stages in JPEG encoding

Most papers in the literature have focused on defeating a preceeding paper. Often, too little time is spent to analyse the operating conditions where a particular technique is useful.

This is our purpose in this paper. We discuss the celebrated F5 algorithm by Andreas Westfeld [12], and analyse the operating conditions for which it can be expected to work. We demonstrate that some previous claims of breaking F5 are exaggerated, and that the respective steganalysis works due to artifacts in the F5 *software*, not the embedding algorithm.

2 JPEG Steganography

One of the most popular media for steganography is JPEG images. This makes sense because image coding and processing is relatively uncomplicated and requires much less knowledge than video and audio. JPEG is also one of the most popular image formats, and remarkable in that it has been very well known for 15–20 years across multiple architectures.

A range of stego-algorithms modulate information by modifying the JPEG coefficients, which are obtained by a blockwise DCT transform of an image and subsequent quantisation. Figure 1 shows where embedding takes place in the process. The subsequent steps of the JPEG compression are lossless, so that the stego-decoder will see the exact same JPEG coefficients as the encoder produced.

2.1 The F5 algorithm

F5 was introduced in [12], with an accompanying Java implementation [1](F5Software). It sports two key features, which should be treated separately. We'll briefly explain the purpose and result of these features. For further details we refer to the original paper.

Firstly, it uses a modulation technique, named F4, which maintains the symmetry as well as other statistical properties of the histogram of the JPEG coefficients. It effectively produces a histogram which looks like that of a JPEG image at lower quality. This modulation effectively prevents the successful attacks on its predecessors Jsteg and Outguess.

Secondly, it uses matrix encoding to minimise the distortion. This uses ideas from source coding and coding for constrained memories to represent the message as a codeword which can be embedded with a minimum of distortion.

2.2 Attacks on F5

Soon after being introduced to the steganography community, F5 has become a popular target for steganalysis. This is due to its capability to withstand visual and statistical attacks while providing a large steganographic capacity. F5's advantages are obtained by the implementation of matrix encoding and permutative straddling. Matrix encoding helps to improve the efficiency of embedding while the permutative straddling helps to uniformly spread the changes over the stego image.

The most recent algorithm claimed to defeat F5 software was our conditional probability approach [11]. With a total of 54 features extracted for each image, this steganalysis technique can defeat F5 software, particularly with longer message sizes (618 bytes, 1848 bytes and 4096 bytes).

The 324-feature Markov-process approach, which inspired our conditional probability steganalysis technique, was also shown to defeat F5 software in [11]. By using the different message sizes measured in the unit of bpc (bits per non-zero AC coefficients), the highest accuracy is 96.8% for a message size of 0.4 bpc.

Steganalysis methods based on statistical moments also claimed to defeat F5 in [3]. Generating 78 features for classification, an accuracy up to 91.1% achived by this technique for a message size of 0.4 bpc.

In [9], Pevný and Fridrich have combined their approach [6] with the markov process approach. The resultant detection accuracy (99.92% for message size of 100% embedding capacity) from this combination really shows it was better in attacking F5 compared to their individual features.

For our experiments we use the conditional probability and Markov process approaches, as well as steganalysis based on wavelet decomposition [7].

3 Double Compression Issue

Westfeld's F5 implementation, as well as many other popular stego-programs for JPEG, are not able to work directly on a given JPEG input file. An input file in JPEG format would be decompressed (using a standard API function to load the image), and represented internally in the spatial domain. Then the image is recompressed, and the F5 embedding performed during compression.

The recompression will not necessarily be with the same quantisation matrices as the original compression. In fact, the program is unable to capture the original matrices, so if the same quantisation matrices are to be used, the matrices have to bethe standard matrix and the user has to manually supply the correct quality factor.

According to Pevný and Fridrich [5], double compression occurs when a JPEG image, originally compressed with a primary quantization matrix is decompressed and compressed again with different secondary quantization matrix. In their paper, they used a primary quantization matrix estimation technique [4] to resolve the double compression issue; the estimated quantization matrix was then used for the second compression process. A similar approach, without the need for estimation, was used in [8].

Chen *et al.* [3] implemented another approach for this double compression issue. They fix the same quality factor (80) for image preparation (F5 without an embedded message) and the F5 embedding process. This assumes that the differences between stego image and cover image are only caused by data embedding and not by the JPEG compression quality.

In both solutions, double compression still occurs, and the risk of its effect is still present. Indeed, F5 with no message is actually 32 bits embedded, namely the status block saying that there is a message of length 0. It seems both approaches assume that double compression is an inherent part of F5. In the following section we discuss how our implementation avoids the double-compression altogether.

4 Optimal use of F5

The double-compression is an artifact of Westfeld's implementation of F5 (F5Software), and not of the F5 algorithm as described in Westfeld's paper [12]. In order to make a fair evaluation of F5, we reimplemented it in Python[1]. Our implementation (F5Py) reads JPEG files and does the Huffmann decoding to obtain the JPEG coefficient matrix, on which the F5 embedding operates, as well as the quantisation matrices. It never reverses the lossy compression steps.

We have followed Westfeld's version 12 beta, which means that the algorithm ignores, in addition to all zero coefficients, every other coefficient of value ± 1. This feature is not documented in [12], but our simulations show that it gives a slight improvement over using all ± 1 coefficients.

Another issue which requires some consideration is the embedding capacity of F5. It is not uncommon in the literature to estimate the capacity of F5 by the number of non-zero AC coefficients, and work on embedding rates of 0.25-0.8 bpc (bits per non-zero coefficient). Firstly, this over-estimates the total capacity, both because F5 creates new zeros which are then ignored, and because the software ignores half of all the ± 1 coefficients.

[1]We plan to make this code publicly available when the paper is published.

Table 1. F5 Capacity in our test images.

	Range	Mean	Variance
Capacity	$(1579, 8719)$	3710.77	1643133
Changes	$(517, 2621)$	1100.10	98747
BPC	$(0.00335, 0.01849)$	0.00816	7.74×10^{-6}
Length	$(430416, 471429)$	—	—

More importantly, F5 *is not designed* to operate close to the maximum capacity. The main idea in F5 is matrix coding, which means that we *use more* coefficients, but *change fewer*, per embedded bit. By changing fewer coefficients, we reduce the distortion and the steganographic modification becomes harder to detect. Operating close to capacity, matrix coding is not possible and F5 degenerates to F4.

The codes used in F5 are Hamming codes of co-dimensions $k = 1, 2, \ldots, 7$, giving block lengths of $n = 2^k - 1$. The case $k = 1$ is degenerate and gives no coding at all. At $k = 1$, we can typically embed 1.4 bits per change. At $k = 4$ we can almost always embed more that 2 bits per change (2.1–2.5 typically). At $k = 7$, we can typically embed 3.5–4 bits per change.

In order to reduce the distortion by 50%, it is necessary to use a matrix code of $k \geq 5$, which gives a rate of less than $1/6$. There is no doubt that many application may require embedding rates of 0.25 bpc and higher, but this is not the problem F5 set out to solve.

To give an impression of the capacity of F5, we have run a simulation at $k = 7$ over all the 1745 images in our test database. The result is shown in Table 1. We generated a long random message which we attempted to embed in every image. Then we recorded the number of bits successfully embedded, including the 32-bit status block which is embedded without matrix coding.

In our simulation, the rate which can be achieved with $k = 7$ is around 0.01 bpc. We did not run any simulations with $k = 5$, but it is expected that it would give a rate approximately three times higher, corresponding to the ratio of the code rates. This means that 0.05 bpc, considered by many authors as an extremely low rate, should actually be considered as a relatively high rate for F5.

5 Experimental Methodology

A significant difficulty in any attempt to compare steganalysis techniques arises from the many variables involved in setting up an experiment to determine the classification accuracy of such a scheme. A non-exhaustive list includes:

- Image source: sensor type; sensitivity setting; lens used; aperture; shutter speed; mounted/handheld; temperature & other environmental conditions
- Image subject: illumination; detail; etc.
- Image processing: conversion from raw sensor data; compression for storage; conversion to grayscale; cropping; possible re-compression
- Steganography: embedding rate/message size; strength or other parameters of embedding scheme; randomization of message; randomization of embedding location
- Classifier used (this is often not necessarily determined by the algorithm)
- Classifier training: selection of cover images; selection of stego images (should their covers be part of the training set?); proportion of cover to stego images; proportion of stego algorithms/settings
- Classifier tuning: parameter tuning for balancing false positive/negative errors
- Classifier testing: selection of images; statistical accuracy of results

This is further complicated by the lack of a comprehensive guide to the approach that should be used. While the general approach is accepted, the many details of a best-practice methodology are scattered in the literature. This makes it very difficult for new researchers, resulting in even more publications using divergent approaches. The lack of consistency makes it practically impossible to make a meaningful comparison between techniques. Sometimes, missing details in published work compromise the reproduction of experimental results.

In our experiments we use a combination of public image databases and our own captured images, ensuring a sufficient number of training and testing images. Source images are greyscale, mostly stored in JPEG format, and have not been processed other than to remove color information and store in compressed format. For each source image, we decompress, crop to the central 640×480 region, and store in JPEG-compressed format.The cropping technique ensures that our source images have the same size, so we can compare the embedded message size in a meaningful way.

After image preparation, we create stego-images corresponding to each cover image, using both Westfeld's F5 software implementation (F5Software) and our implementation that avoids double-compression (F5py). In each case, we embed messages of size ranging from 1 byte to 500 bytes, creating five different sets of stego images for each implementation.

For each of the cover and stego images, we extract the features for every steganalysis technique we are analysing, in preparation for the subsequent classification process. The freely available LibSVM [2] is then used as the classifier. The soft margin and γ parameters are determined using the 'grid.py' parameter selection tool, available as part of the LibSVM package. Classifier training is based on a random selection of $\frac{2}{3}$ of the cover and stego image pairs. The remaining $\frac{1}{3}$ of the image pairs are used for classifier testing. The complete process is shown schematically in Figure 2.

6 Results

Classification results for our conditional probability steganalysis and for the Markov process based technique are shown respectively in Tables 2–3. Additionally, results for the wavelet

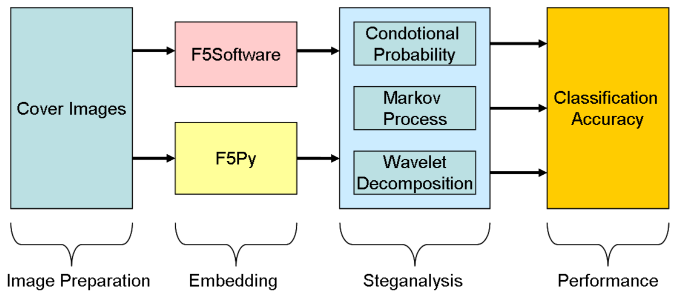

Figure 2. Experiment tasks

Table 2. Classification accuracy for conditional probability technique (optimal soft margin and γ parameters)

	Message Size (bytes)				
	1	10	50	250	500
F5Software	97.0%	97.6%	97.4%	98.2%	99.2%
F5py	50.0%	50.0%	50.0%	54.6%	66.8%

decomposition technique are shown in Table 4. It is clear from the presented results that classification accuracy is significantly reduced when double compression is excluded (F5Py). In fact, none of the steganalysis techniques tested have any detection ability at small message sizes that allows F5 to operate on its optimal performance. The Markov technique is only able to detect F5 at a message length of 500 bytes, and even then the accuracy is still rather low. Conditional probability steganalysis seems to perform only slightly better. Finally, as already observed in [11] and [10], wavelet decomposition steganalysis is unable able to defeat F5 at any message size.

F5Py also provide a new approach on how to deal with double compression issue. Comparing our new implementation with previous approaches, we can see that:

- Using stego image with zero message [3] resolved the problem with different quality factor in preparation and embedding process. One disadvantage is that there will be more time needed in order to prepare the F5 with no message set.
- Using F5Py did also resolved the above problem. With an advantage, it save the time by not involving another image preparation process. For example, the process needed to estimate the primary quantization matrix in [5].
- Using F5Py resolved the issue with the unknown additional JPEG compression with cover image. This is a very important factor when we consider the real world implementation of steganalysis.

7 Conclusion

Our discussion of reasonable operating conditions for F5 have shown two things. Firstly, we have shown that F5 is not designed to operate at high embedding rates, and we recommend

Table 3. Classification accuracy for Markov process technique (optimal soft margin and γ parameters)

	Message Size (bytes)				
	1	10	50	250	500
F5Software	99.8%	99.6%	99.6%	99.6%	99.9%
F5Py	50.0%	50.0%	50.0%	50.0%	62.4%

Table 4. Classification accuracy for wavelet decomposition technique (optimal soft margin and γ parameters)

	Message Size (bytes)				
	1	10	50	250	500
F5Software	50.2%	50.0%	50.6%	50.4%	50.2%
F5Py	50.0%	50.4%	50.6%	50.4%	50.2%

to restrict use of F5 to less than 0.05 bpc. This is to ensure F5 can take full advantage the of matrix encoding and its optimal embedding efficiency is not violated by the access message.

Secondly, we have demonstrated the possibility that Markov model based attack by Shi *et al.* detects double compression artifacts, which is an artifact of the implementation and not of the algorithm in F5 software. It is not nearly as good a classifier for stegograms from an improved implementation of F5 (F5Py) which does not give double compression.

We stress the importance of clarifying the test conditions when stego-systems and attacks are specified and evaluated. It is not very useful to attack a system under conditions where it never was supposed to be used.

It remains to be seen whether double-compressed images actually exist in the wild, and if so, which image processing operations tend to produce them. If such images do exist, the problem of developing and tuning a steganalysis algorithm may be even more complex than is generally believed; allowing double-compressed images as cover and stego-images would mean that the artifacts produced by double compression would have to be ignored by the classifier.

It is an interesting open problem to see how effective other attacks are on our implementation of F5 with reasonable embedding rates.

Another open problem, and this is of course well-known, is how we can design stego-systems with higher capacity, without becoming more detectible.

References

[1] A.Westfeld. F5 version 12beta. Software available at `http://www.inf.tu-dresden.de/~westfeld/f5`.

[2] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] Chunhua Chen, Yun Q. Shi, Wen Chen, and Guorong Xuan. Statistical moments based universal steganalysis using JPEG 2-d array and 2-d characteristic function. In *Proceedings of the International Conference on Image Processing, ICIP 2006, Atlanta, Georgia, USA*, pages 105–108, October 8-11, 2006.

[4] J. Fridrich and J. Lukas. Estimation of primary quantization matrix in double compressed JPEG images. *in International Conference on Image Processing*, September 2002.

[5] Jessica Fridrich, Tomáš Pevný, and Jan Kodovský. Statistically undetectable JPEG steganography: dead ends challenges, and opportunities. In *MM&Sec '07: Proceedings of the 9th workshop on Multimedia & security*, pages 3–14, New York, NY, USA, 2007. ACM.

[6] Jessica J. Fridrich. Feature-based steganalysis for JPEG images and its implications for future design of steganographic schemes. In *Information Hiding*, 2004.

[7] Siwei Lyu and Hany Farid. Detecting hidden messages using higher-order statistics and support vector machines. In *5th International Workshop on Information Hiding*, Noordwijkerhout, The Netherlands, 2002.

[8] Siwei Lyu and Hany Farid. Steganalysis using color wavelet statistics and one-class support vector machines. *Proc. SPIE*, 2003.

[9] Tomáš Pevný and Jessica Fridrich. Merging Markov and DCT features for multi-class JPEG steganalysis. In *Proceedings SPIE, Electronic Imaging, Security, Steganography, and Watermarking of Multimedia Contents IX*, January, 2007.

[10] Yun Q. Shi, Chunhua Chen, and Wen Chen. A Markov process based approach to effective attacking JPEG steganography. In *Information Hiding*, 2006.

[11] Ainuddin Wahid Abdul Wahab, Johann A. Briffa, Hans Georg Schaathun, and Anthony TS Ho. Conditional probability based steganalysis for JPEG steganography. *2009 International Conference on Signal Processing System (ICSPS 2009)*, May 15 -17 2009.

[12] Andreas Westfeld. High capacity depsite better steganalysis: F5 – a steganographic algorithm. In *Fourth Information Hiding Workshop*, pages 301–315, 2001.