F 4.9
A Distributed Approach

Table of Contents

1. Introduction

The steganographic algorithm we chose for our project is F5. It is a more advanced scheme than either the original LSB or statistical restorative approaches [8], but is still less involved than current distortion reduction approaches such as Perturbed Quantization or Bose-Chaudhuri-Hochquenghem (BCH) [6]. We aimed and succeeded at implementing most of the F5 algorithm.

Our implementation leverages the LibJPEG C/C++ library. LibJPEG is used to both extract and recompress the quantized DCT coefficients found within JPEG images. The steganographic algorithm manipulates these coefficients in a structured manner to embed a supplied stego message within a JPEG cover file. The coefficient modifications are minimal and information already present in the cover image data is taken advantage of in order to convey the stego message information instead.

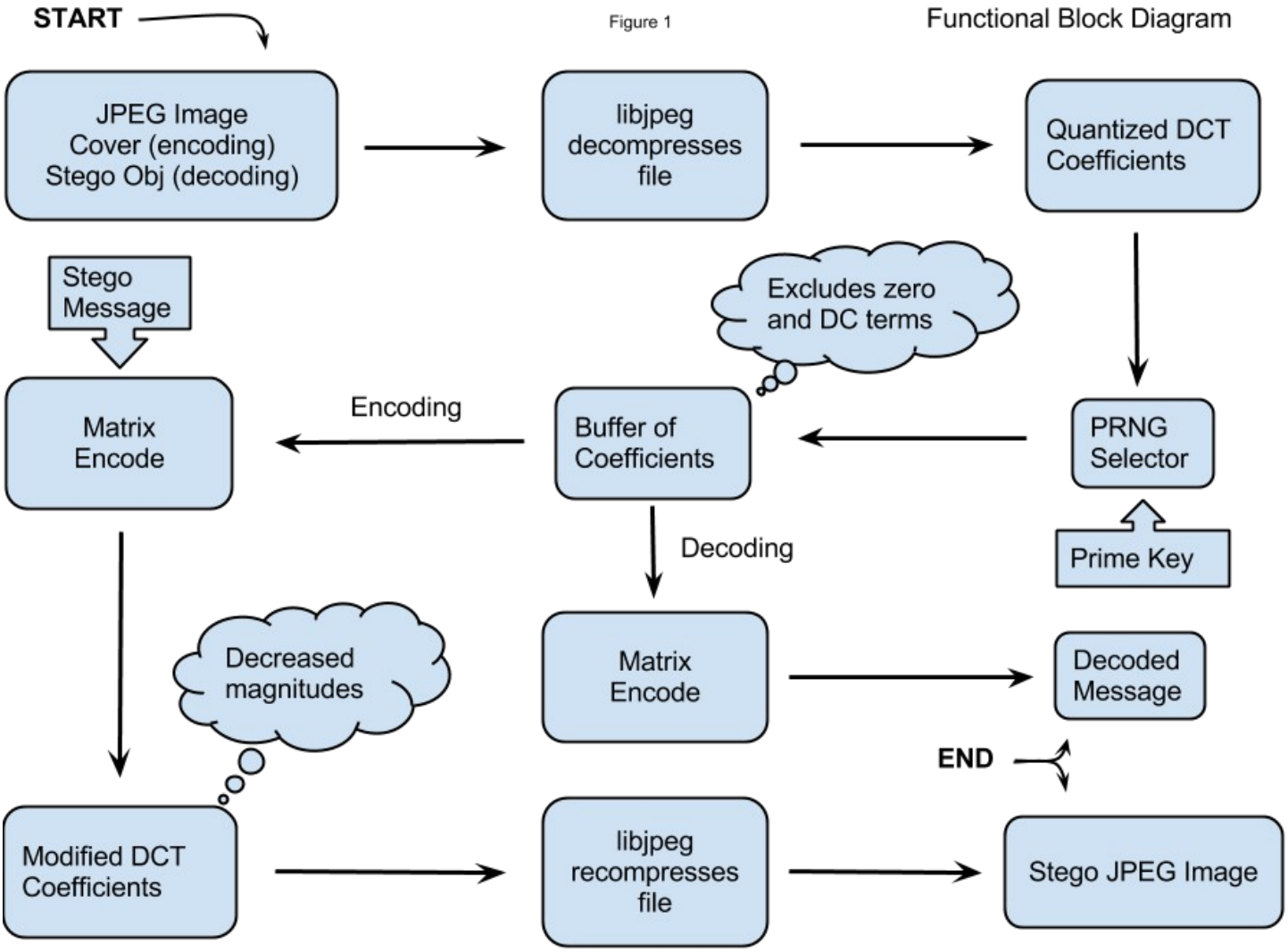
During the development of our project we used an agile workflow - integrating small iterative steps and milestones. In addition to the agile workflow we maintained an extreme programming approach during the programing portion. This approach worked well since it parallelized the development and code review process. Never meeting in person for development, we leveraged the monitor sharing, voice over IP, and chat features of Google Hangout. Our source code was stored using Git, a distributed version control system, and then shared via the BitBucket interface - a free, public, git-hosting web service. We would frequently tradeoff who wrote code and who commented by pushing commits, branching and then merging the code base.

In our tests we found an average embedding capacity of approximately twelve percent. This capacity changes based upon which matrix encoding schema is used. A tradeoff between capacity and mean squared error (MSE) or peak signal to noise ratio (PSNR) can be seen in the results section of this report. All in all, the F5 algorithm exceeded our expectations by embedding an impressive amount of data within a cover object with minimal perceptibility. As for capacity, perceptibility and robustness: it attained moderate capacity, negligible perceptibility and lacks robustness. Images produced have a low MSE and high PSNR. With low perceptibility of change and an acceptable capacity level, the project was an overall success.

2. High Level Overview

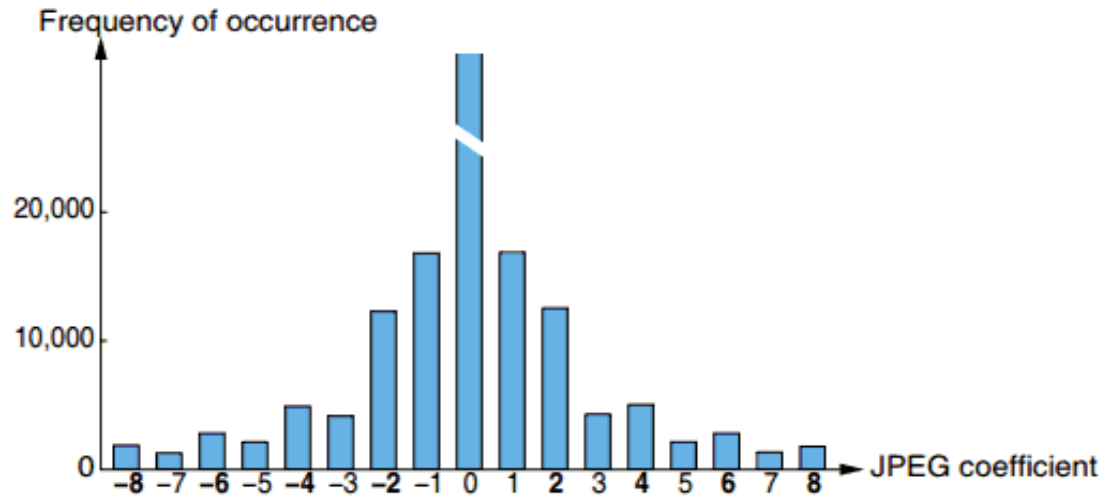
The functional block diagram contains a high level pictorial representation of our systems - both encoding and decoding. Medium to large JPEG images contain a plethora of coefficients, which are utilized as hiding space. We are using the LibJPEG library to process the cover file, attaining the quantized DCT coefficients. Our F5 algorithm processes the coefficients along with the stego message. Last, LibJPEG is invoked again to compress the data back into a JPEG. The resulting image is a stego object (cover with embedded message data). At a high level the processes of our implementation follows the steps outlined in the functional block diagram below.

Figure 1 Functional Block Diagram



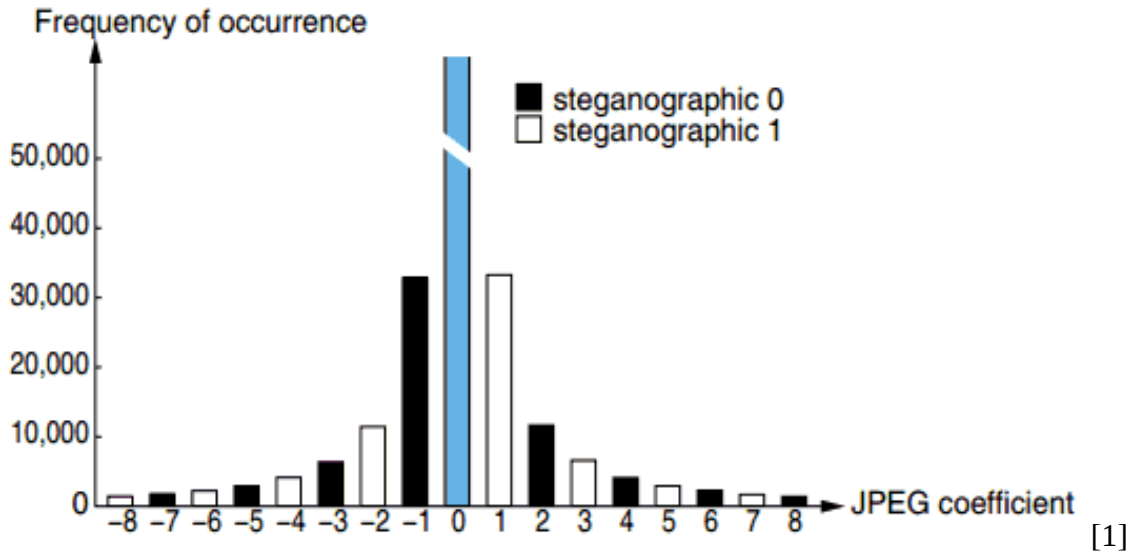
2.1 Histogram Preservation

The original F3 algorithm did not preserve the DCT coefficient histogram very well. This was a direct result of how the representation of information bits were chosen. An unmodified histogram inherently has more 1s and -1s than 2s and -2s, as well as more 3s and -3s than 4s and -4s, etc. Thus a natural distribution has more odd than even coefficients. Using the simple parity scheme in F3 to represent the information bits results in a wave-like distortion of the histogram where even coefficients become over-represented in the stego object. These artifacts are corrected in F4 and F5 by inverting the representation of information bits for negative coefficients. As can be seen in the following image the distribution, though symmetric in magnitude, creates a wave pattern in the odd to even coefficients. This is an artifact of unbalanced even and odd coefficients in the original cover.



[1]

The algorithm F4 corrects the wave distribution effect of the odd-even coefficients by inverting the steganographic value for negative coefficients as shown in the image below.



[1]

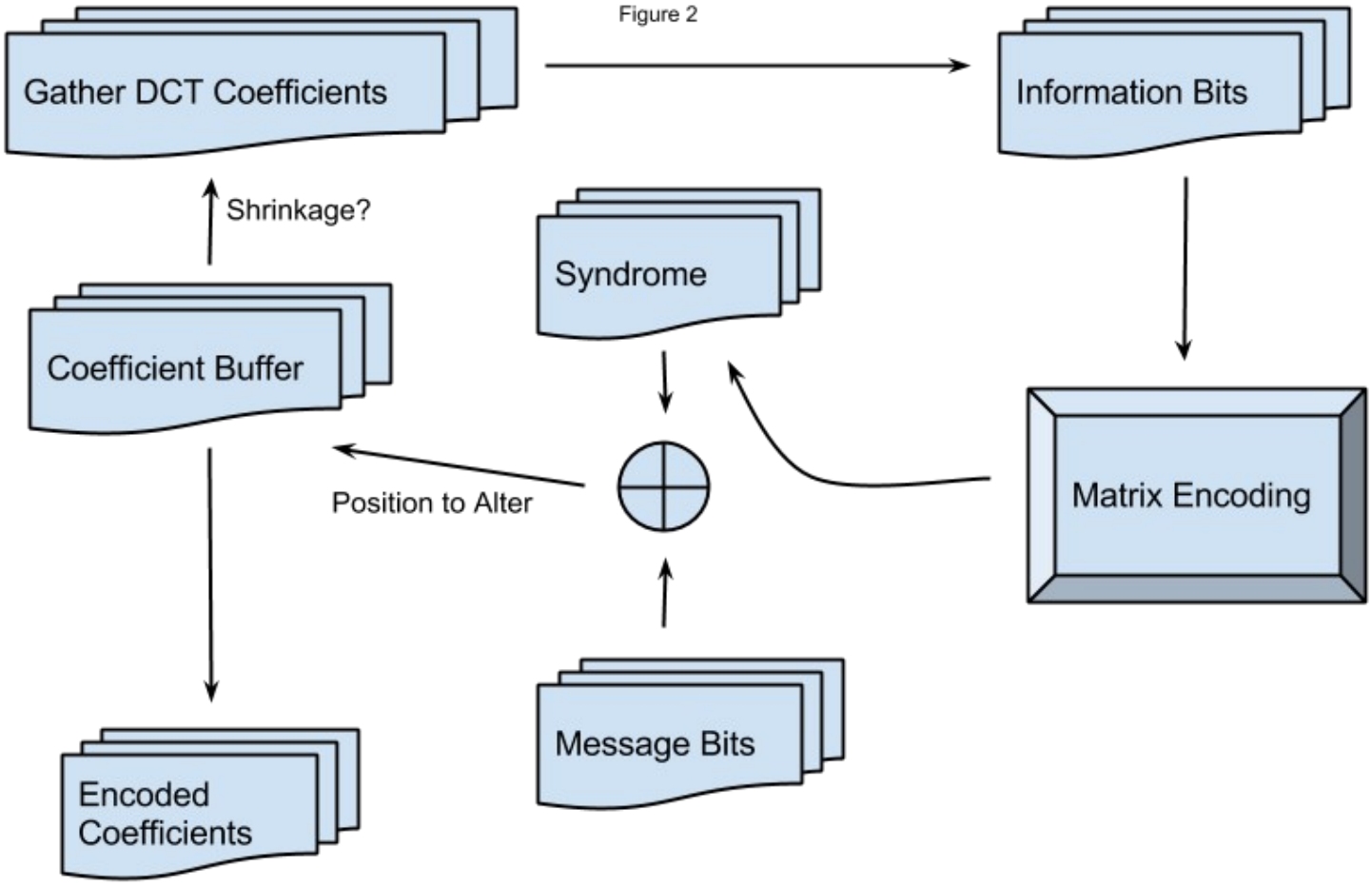
2.2 Modification of DCT Coefficients

2.2.1 Encoding

The DCT coefficients are modified in the following manner:

1. Decide the matrix encoding K-N scheme to use
2. Attain a stream of the DCT coefficients
    i. use the same order for encoding and decoding
    ii. exclude zero and DC coefficients
3. Put N coefficients into a buffer
4. Compute the corresponding set of N information bits (steganographic 1s and 0s)
5. Compute the K bit syndrome via matrix encoding
6. Compute a buffer position by XORing the syndrome and message bits together
7. Apply the appropriate DCT coefficient modification
    i. if the position is zero, then do not modify the coefficient (syndrome matches the message)
    ii. otherwise decrease the magnitude of the coefficient (flips the corresponding information bit)
8. Check for shrinkage and recalculate with a new coefficient if necessary
    i. shrinkage coefficients ARE reduced to zero

Figure 2

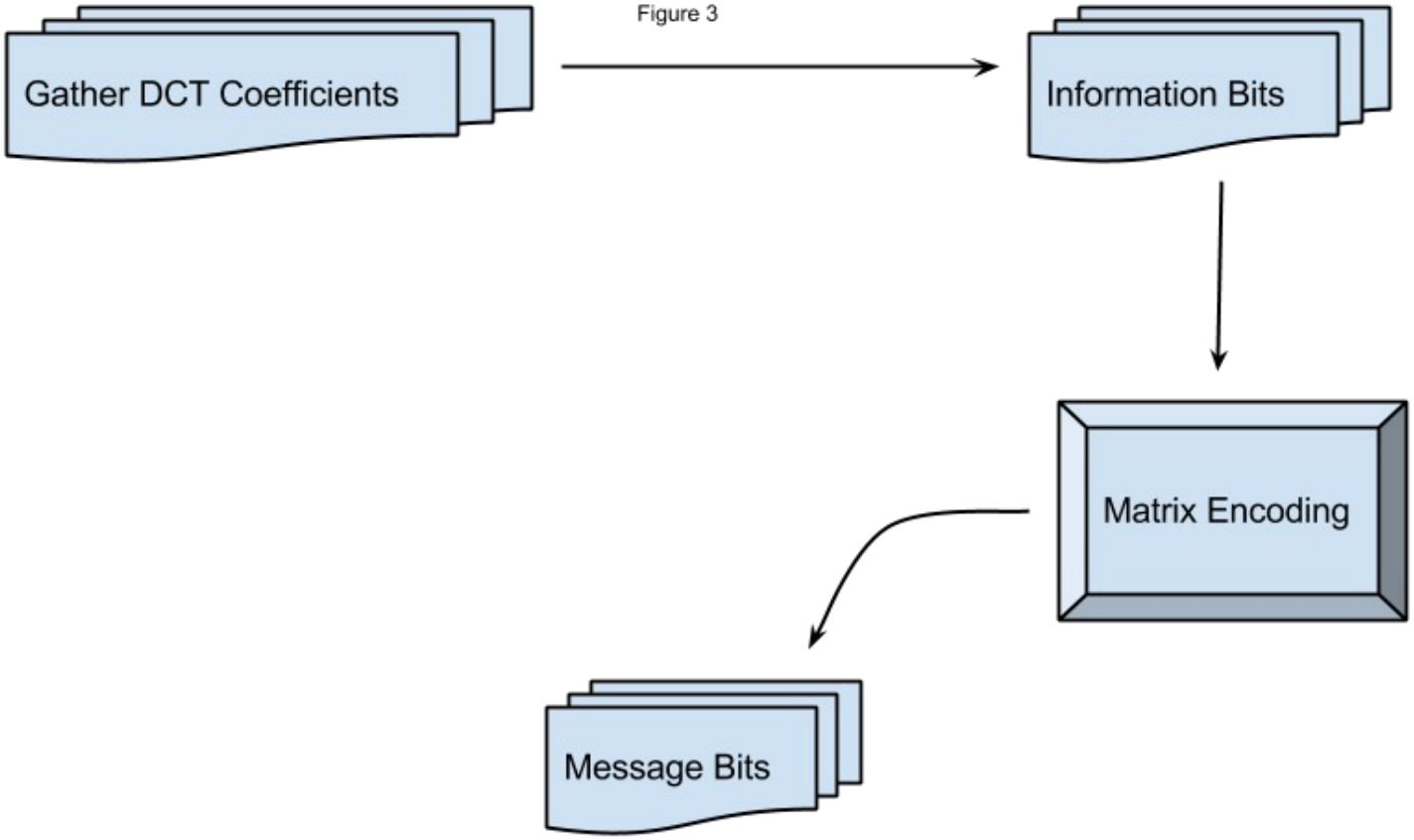


2.2.2 Decoding

The DCT coefficients are modified in the following manner:

1. Use the same matrix encoding K-N scheme which was used to encode
2. Attain a stream of the DCT coefficients
    ii. use the same order as encoding (same prime key)
    iii. exclude zero and DC coefficients
3. Put N coefficients into a buffer
4. Compute the corresponding set of N information bits (steganographic 1s and 0s)
5. Compute the K bit syndrome via matrix encoding
6. The syndrome is the message bits which were encoded

Figure 3



2.3 Matrix Encoding Explained

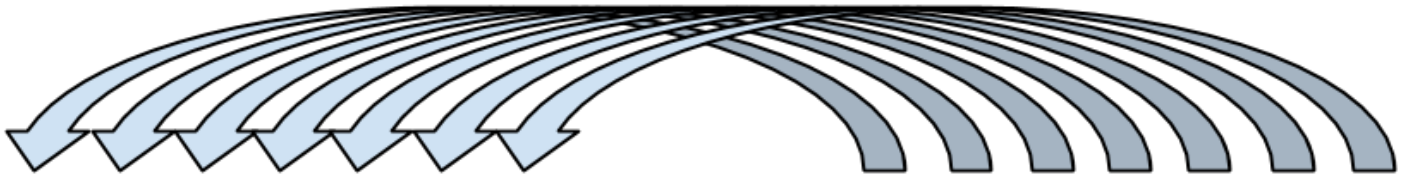
The Matrix Encoding is carried out by some fancy bit magic. This section explains how to derive which information bits each syndrome bit depends upon. The figure below depicts a 3,7 scheme. The information, syndrome, and message bits are labelled b1-b7, s1-s3, and m1-m3 respectively. Dependencies are marked by an x.

A XOR of the syndrome and message bits tells which syndrome bits must be corrected to match the message bits. All possible results are shown to the right of the message bits in the figure. If the result is all zero bits, then nothing needs to be changed. However, if the result is non-zero, then it represents the position of the information bit that requires updating.

For each information bit position column, a syndrome dependency exists for every 1 that appears in the corresponding result bits column. If each dependency is written as a 1 and each non-dependency is written as a zero, then the dependency table exactly matches the table of result bits excluding the column of all zeros. Thus the dependency table is trivially generated by placing in each column that column's position. The pattern scales for any K-N scheme.

Each syndrome bit can then be calculated by XORing together the information bits it depends on. The information bits are picked out by the rows of the dependency table. The final result of this 3,7 scheme is listed in the box below the tables.

Notice how in a K bit scheme, the number of possible messages, or syndromes, or results by XORing them together, is 2^K. But the number of information bits is one less (the zero position was excluded). So in any K-N scheme, N is fixed to $2^K - 1$ and varies exponentially in relation to K.



b1	b2	b3	b4	b5	b6	b7	
x		x		x		x	s1
	x	x			x	x	s2
			x	x	x	x	s3

⊕ ⊕ ⊕

m1	0	1	0	1	0	1	0	1
m2	0	0	1	1	0	0	1	1
m3	0	0	0	0	1	1	1	1

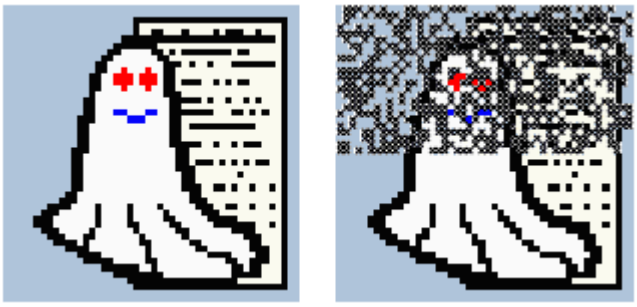
$$s_1 = b_1 \oplus b_3 \oplus b_5 \oplus b_7$$
$$s_2 = b_2 \oplus b_3 \oplus b_6 \oplus b_7$$
$$s_3 = b_4 \oplus b_5 \oplus b_6 \oplus b_7$$

3. Project Review

As outlined in the previous sections the technique we implemented is a generic form of the F5 algorithm. Being one of the first algorithms to foil statistical steganalysis, we were interested. It foils attacks such as chi square by preserving the DCT coefficient symmetry whilst uniformly distributing the message. It also maintains low visual perceptibility while preserving good hiding capacity. All of these attributes intrigued us so we decided to implement them.

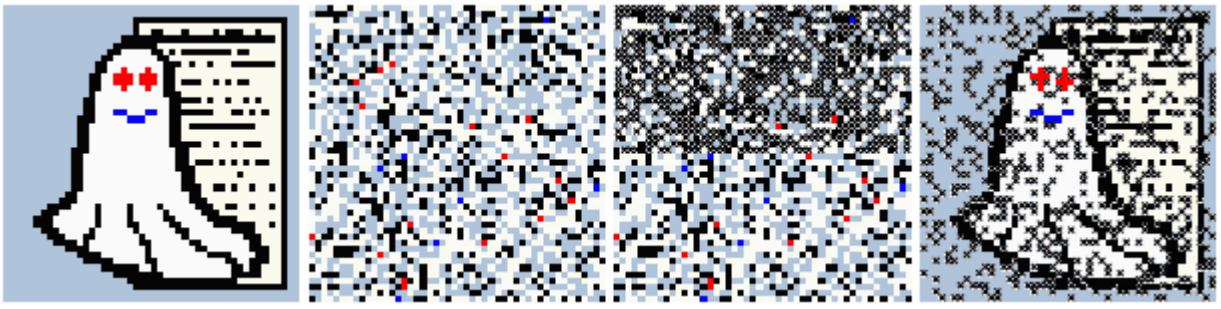
Since LibJPEG allows for granular control over the coefficients with minimal overhead we decided to utilize the library. F5 Incorporates the stego message within the DCT coefficients after they have been quantized but before they are entropy encoded. LibJPEG is used to decompress a JPEG, extract the DCT coefficients, modify, re-encode and then compress them producing a new image.

A simple pseudo random number generator (PRNG) is used to iterate over the DCT coefficients. The iteration jumps by a predetermined prime number (passed in at runtime) which serves as an encoding or selection key. If the total number of coefficients is divisible by this prime then one fewer coefficients is used for hiding. Due to time constraints we decided not to implement a more sophisticated PRNG. The actual F5 algorithm selects coefficients at random using permutative straddling (PS). This distributes the message bits across the carrier medium attaining balanced encoding density. In contrast continuous embedding places all of the data from the beginning of the image in sequential order. This linear distribution of the message within the image leads to simplified detection. An example of a non distributed, sequential message and then one where permutative straddling was utilized is seen below in the images supplied by “Andreas” in his paper [1].



[1]

Continuous Embedding [1]



[1]

Permutative Embedding [1]

While encoding, the DCT coefficients represent steganographic zeros and ones. One possible way to assign the information bits is to take the parity of the coefficients. This was done in F3. However, since there is an abundance of odd to even coefficients, this will result in a wave-like distortion of the histogram. This problem was corrected in F4 by inverting the scheme for negative coefficients. F5 incorporates this technique in addition to permutative straddling and matrix encoding.

At the heart of the F5 algorithm is a process that utilizes Ron Crandall's idea - matrix encoding (ME). ME relies on a K-N scheme by which small groupings of K stego-message bits are embedded into a collection of $N = 2^K - 1$ DCT coefficients (excluding the zero and DC terms). The steganographic bits corresponding to these coefficients are XORed together in a precise manner to produce what's called a syndrome. The K syndrome bits are XORed with the stego-message bits to calculate an index. Finally, this index identifies which DCT coefficient must have its magnitude decreased so that the syndrome matches the stego message. This syndrome is what will be calculated during the decoding process and is how the algorithm recovers the embedded stego message.

ME is used to minimize the number of altered coefficients when embedding a group of stego-message bits. At most one coefficient need be modified per group of N in order to convey K message bits. Some of the selected coefficients may be reduced to zero. When this happens it is called shrinkage due to the inability of these coefficients to convey meaningful data. As a result they are discarded and new coefficients are brought into the grouping of N coefficients. The decoder simply skips the shrinkage coefficients since they are all set to zero.

The DCT histogram is preserved in a balanced fashion by reducing the magnitude of selected DCT coefficients by one. Cumulatively, this shrinks the magnitude of the entire coefficient distribution uniformly instead of introducing the immediately detectable asymmetry caused by LSB bit flipping. Our project implements the majority of the F5 algorithm.

3.1 Technical Difficulties

3.1.1 Shrinkage and Buffering

Our original interpretation of Andreas Westfeld's approach[1] to correcting shrinkage was incorrect. During the matrix encoding process we didn't modify any DCT coefficients that caused shrinkage. The correct (and obvious) method is to set the coefficient to zero before shifting in a new one. Another misinterpretation was how to modify the buffer when shrinkage occurs. Instead of replacing the coefficient in-place, the new coefficient should be shifted in at the end of the buffer to preserve their natural ordering. This is critical since the decoder reads linearly with no knowledge of the buffer used to encode. We resolved the misinterpretation of the shrinkage issues after two days of hammering keyboards and reading his paper over - and - over.

3.1.2 Byte Management

Big and little endian issues abounded. We had to create a wrapper which allows for bit level access to bytes. We wrote the bits in a Big Endian manner while encoding and Little Endian while decoding. (Due to the byte being read in Big Endian and written sequentially instead of reversed and then written.) This is was resolved by manually tracing select data with print statements. Initially we pushed random data into a file and were trying to deduce the correctness of our output. Not wanting to deal with the headache and overhead introduced by the horrible test data we settled on the character "U" (capitalized). This was due to the fact that in binary the letter "U" is represented in ascii by 01010101. This pattern allowed for quick and easy debugging and was used to reveal our initial endianness issue.

3.1.3 LibJPEG's Complex Data Structures

The most complicated use of the library is found in our ***getCoefficient*** utility method. We needed to make use of *jpeg_component_info*, *JBLOCKARRAY*, *j_common_ptr*, *JDIMENSION*, and *JCOEFPTR* in addition to the parent structures *j_compress_ptr*, *j_decompress_ptr*, *jvirt_barray_ptr* to access a single DCT coefficient. The structures were passed around as parameter references and used with other libJPEG library methods. Multi-level pointers and the library's memory management also caused difficulties. All native arrays were eliminated in favor of pointers and all library interaction was eventually encapsulated.

3.1.4 The Handling of Messages

Without a flag of some sort the decoder cannot deduce when to stop. As far as it is concerned all of the input is valid message data. To resolve this issue we decided to sacrifice 4 bytes of capacity and reserve them for the original cover's file size. Once the message size is known it is encoded with the message data following. Considering the unpredictable nature of the message data, it was deemed more reliable than terminating with a few signal codes such as EOT (0x04) or ETB (0x17). Although we sacrificed some capacity, the simplicity of this approach was more desirable than the failure of decoding data incorrectly.

3.2 Bugs in the system

3.2.1 JPEG Quality

The quality method of libJPEG utilizes a zealous and optimized routine which results in an image with a skewed quantization table. The resulting image also has higher saturation levels than the original. This seems to be due to a lack of standardization of the JPEG standard in regards to quality levels. However, we didn't research the root cause due to time constraints - an example of our findings can be seen below.

Input Image

Resultant Image with a 95% quality setting

- *The image found above doesn't have a source and seems to be very popular online -*
- *If we knew the artist we would give them attribution -*

3.3 Features

3.3.1 Adaptive

Our F5 algorithm implementation utilizes pointers and short-circuit logic to process images quickly minimizing encode and decode times. All DCT coefficients and data structures are managed via pointers. The length of the original message is encoded in the first four bytes to tell the decoder when to stop assuming max capacity isn't reached. In addition we allow the user to specify the following command line options: the PRNG prime key and K-N scheme to use during the encoding process, and generation of stego object statistics. Unfortunately we were not able to incorporate image quality.

3.3.2 Pseudo Random Number Generator

The PRNG is based on two relatively prime numbers. This ensures that every coefficient is iterated once and only once. One of these numbers is a secret prime key passed into the program. The other is the total number of DCT coefficients in the cover image. If the prime key divides the number of coefficients, then one fewer is used to encode and decode, ensuring relative primeness.

3.3.3 Variable K-N Scheme

F5 allows variance in coefficient group size during the matrix encoding portion based upon a flexible scheme which allows one to adjust capacity for perceptibility. The matrix encoding group size is determined by the K value specified at run time. F 4.9 adjusts the matrix encoding based upon the K parameters supplied utilizing the equation $N = 2^k - 1$ to determine coefficient group size. This flexibility lends to simplified testing and usability since the program doesn't need to be edited and then recompiled.

3.3.4 Capacity, MSE, and PSNR Statistics

Our program incorporates three statistical methods: capacity, mean square error (MSE), and peak signal to noise ratio (PSNR). The capacity is calculated in two different ways. The first is a ratio of the stego message file size to the stego object file size and the second is a ratio of the stego message file size to only the size of all the DCT coefficients. Ultimately, the former ratio carries practical significance whereas the latter ratio is more a measure of how efficiently the algorithm embeds (without header or file format information in the mix). MSE is a measure of how much the coefficients differ between cover and stego images. Since we are decreasing the magnitude of only 1 out of every N coefficients, the MSE is very small (it is actually slightly larger due to shrinkage). PSNR is a measure of how much information is being carried per unit error. Since the cover conveys the majority of our message with little modification, the PSNR is quite large (it would be infinity if they matched exactly).

4. Future Work and Ideas for extension

4.1 Built in Quality Settings

We did not have enough time to debug quality settings passed to the program. It would also be nice to have some clarification on LibJPEG's quality level specification for the current release. LibJPEG uses the IJG standard, however image editor programs seem to have no consensus on quality level. The quality level can be specified at runtime, however, isn't set (the line is commented out).

4.2 A Truly Random PRNG

Our current PRNG follows a very simplistic iteration by prime number. The actual F5 algorithm selects coefficients at random and the key is a value which seeds a generator. This would yield additional security towards extraction. It would also distribute the encoded message more randomly throughout the stego image.

4.3 Encryption and Advanced Algorithms

Implementing any of the FIPS recognized algorithms for encryption of the message data would be a huge security bonus (libgcrypt). Also, F5 is becoming outdated as newer steganalysis algorithms are released. Many of these newer algorithms can detect F5. Other more recent steganographic algorithms offer less detectable means of hiding. A couple we encountered while we were reading research articles were Perturbed Quantization (PQ), Yet Another Steganographic Scheme (YASS), and Bose-Chaudhuri-Hochquenghem (BCH) [6].

4.4 Add support for MP4 video files

If one were to implement a hook into the FFMPEG library, it would be possible to pull DCT coefficients from MP4 video frames. Jeff wrote some preliminary work with this library but sadly there was not enough time to add it to our project (we were able to integrate FFMPEG with libJPEG, however, due to the limitation in FFMPEG pertaining to coefficient extraction we weren't able to utilize it). Our code base is built modularly enough to allow effortlessly extension once a coefficients hook is provided.

4.5 Advance the Encoder/Decoder Capabilities

In the event that a cover's capacity is maxed out, it would be nice if the encoder allowed the program to embed the remaining message portion into additional covers. This would be simple to add but due to time constraints and the scope of the project it was left out.

5. Examples and Their Results

5.1 Example Images

All of the JPEG images which we tested on were 24 bit. Below the images are layout as follows:

*- **Left**: Original image; **Right**: stego image -*

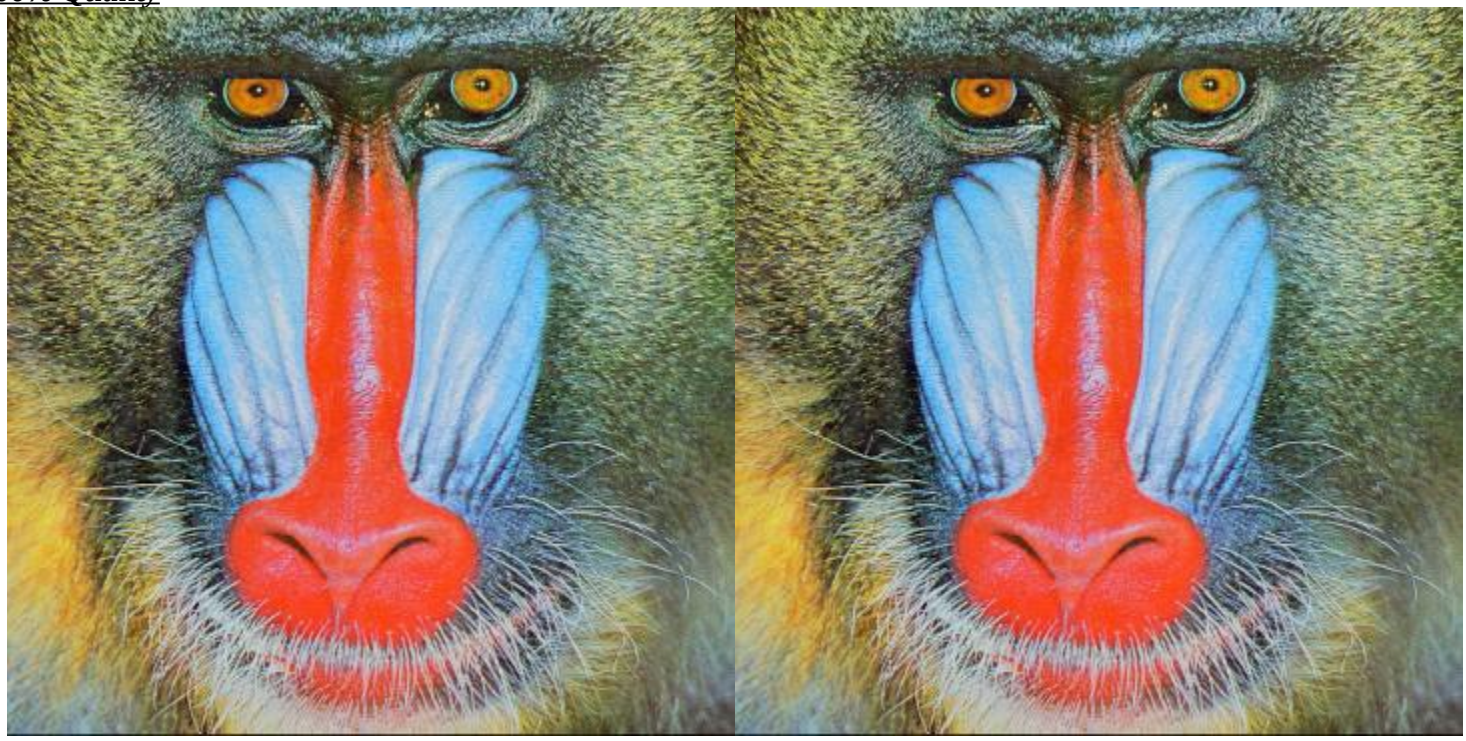
5.1.1 Lena

100% Quality

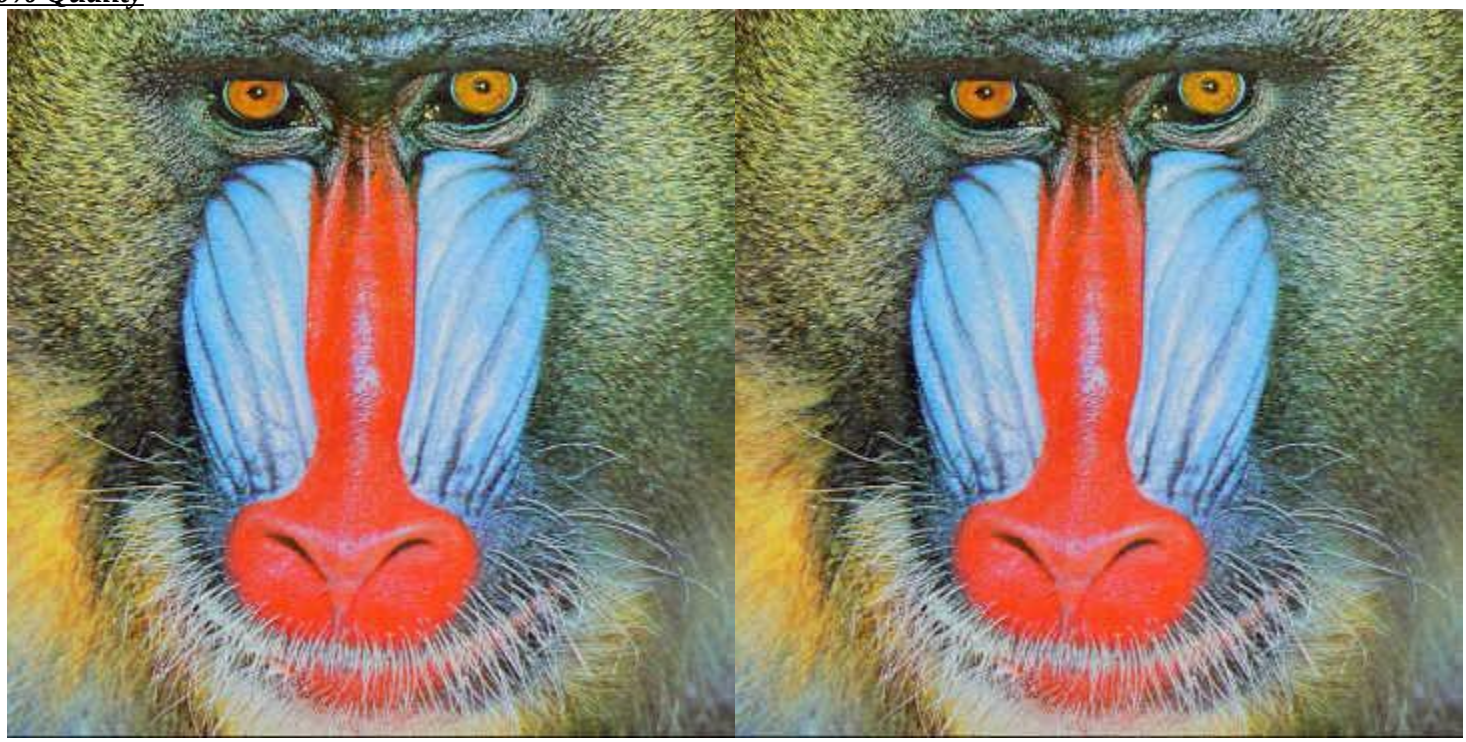
50% Quality

5.1.2 Baboon

100% Quality



50% Quality



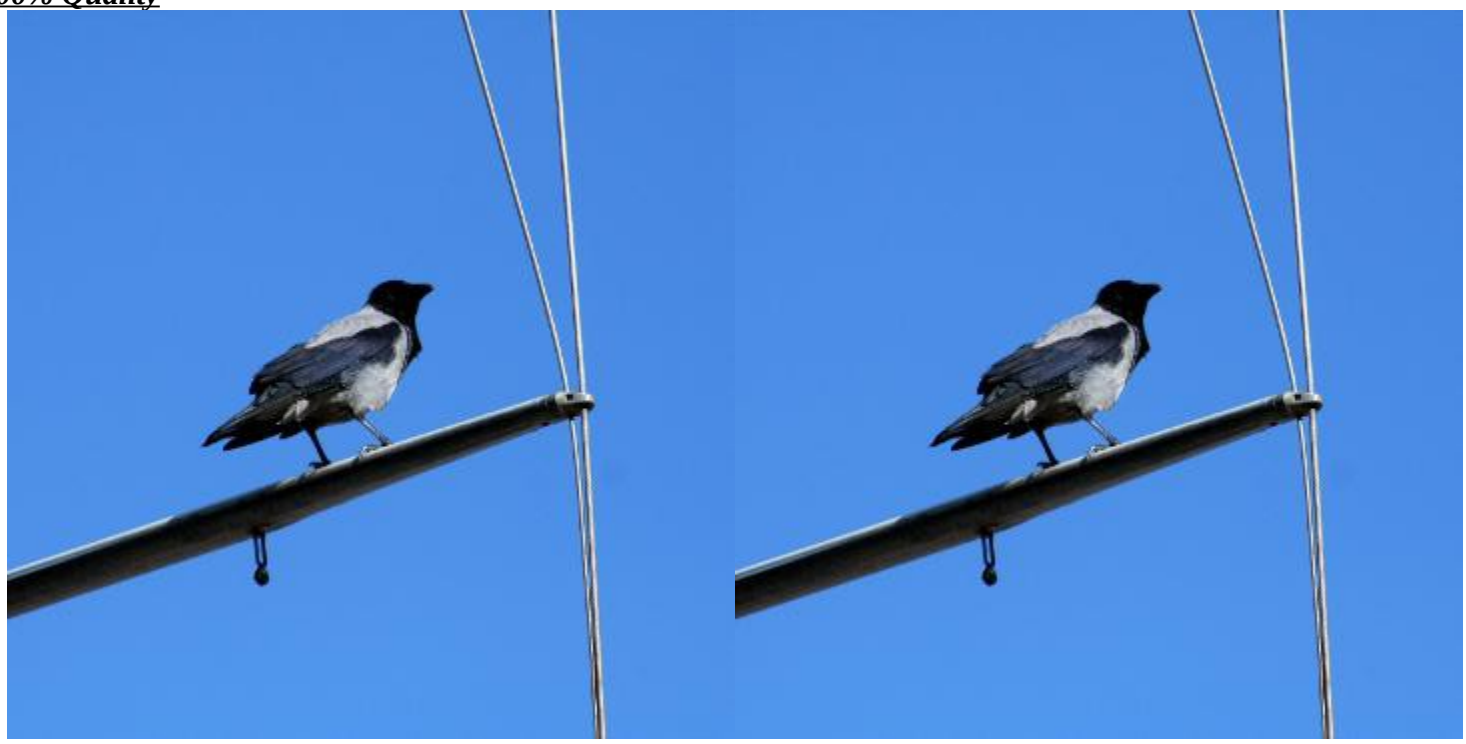
5.1.3 Peppers

100% Quality

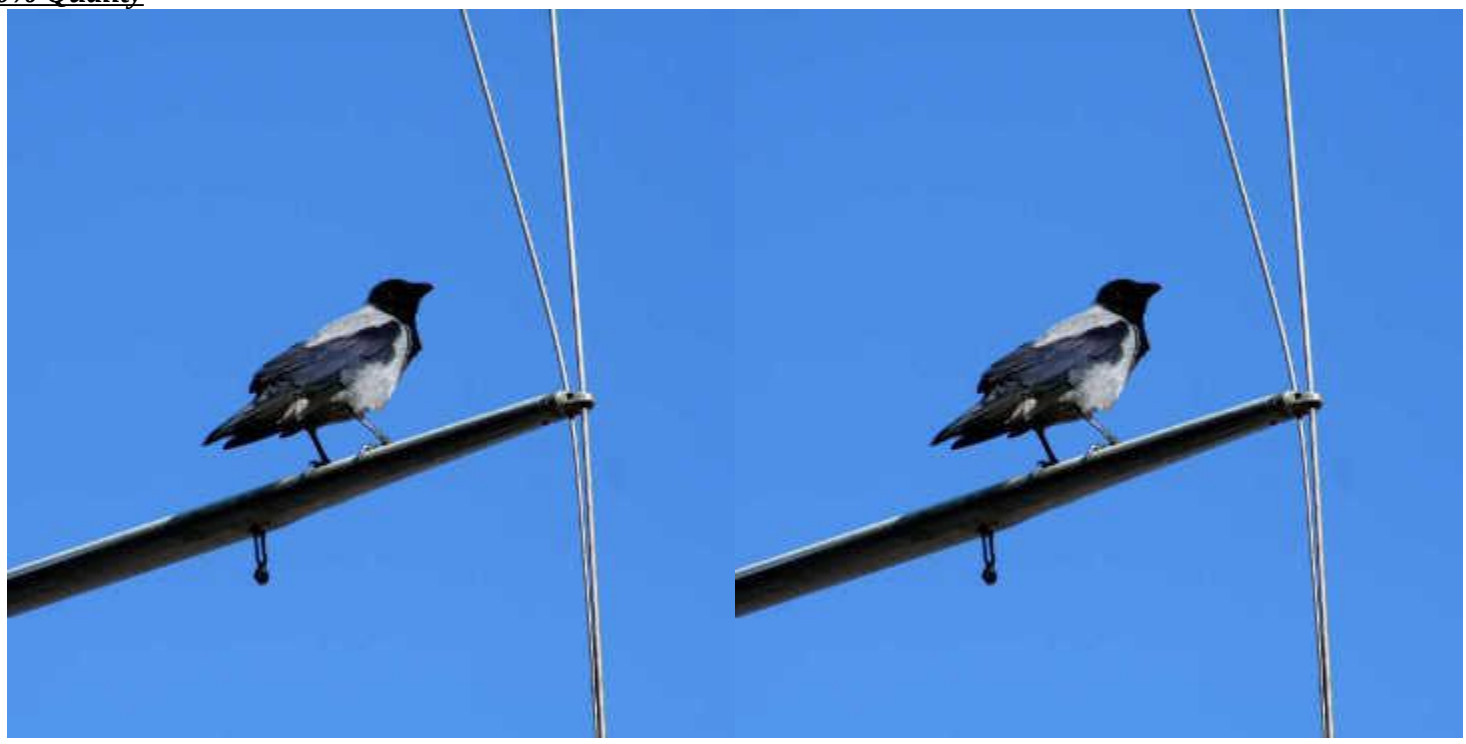
50% Quality

5.1.4 Bird

100% Quality



50% Quality



5.2 Results From Statistical Analysis

The images listed above are only a small portion of the data which we collected in total. (See the zip file for the images and the spreadsheet for all of the results.) Above - in order - are encoded to the max capacity with the following k values: 1, 3, 6, 9. Each set was individually tested with K values ranging from 1 to 9 at three quality levels: 50%, 75% and 100%. Since this creates an abundance of data listed are only the edge cases. Listed below in a table is the related datums per image set above. All data sizes in bytes.

Image	Quality	K	MSE	PSNR	Embed File Size	Written	Total DCT%	Total File %
Lenna	100	1	0.416285	119.58	276982	34636	5.12	16.35
Lenna	50	1	0.023866	148.17	276982	1681	4.33	9.28
Baboon	100	3	0.127237	131.44	276982	19838	2.63	6.95
Baboon	50	3	0.023888	148.16	276982	2074	2.31	6.29
Peppers	100	6	0.01866	150.63	276982	4145	0.59	1.80
Peppers	50	6	0.002068	172.63	276982	269	0.57	1.26
Bird	100	9	0.003734	166.72	276982	631	0.11	0.41
Bird	50	9	6.6e-05	207.01	276982	11	0.07	0.12

Listed in the next table is all of the data which we collected in relation to the images which are outlined above - excluding the data above. As reflected in the data listed F 4.9 has implemented the F5 algorithm (outside of a more advanced PRNG). The capacity measurements and MSE/PSNR values directly correlate with each other outlining the trade off between storage and perceptibility. The following tables contain the related to the image sets outlined in section 5.1 when manipulate at differing settings. Data sizes in these tables are left in bits.

5.2.1 Lena

Image	Quality	K	MSE	PSNR	Embed File Size	Written	Total DCT%	Total File %
lenna	100	2	0.24379	124.94	2215856	200952	3.72	11.86
		3	0.138808	130.57	2215856	135712	2.51	8.01
		4	0.075041	136.72	2215856	87024	1.61	5.13
		5	0.038723	143.34	2215856	53544	0.99	3.16
		6	0.020213	149.84	2215856	31856	0.59	1.88
		7	0.010303	156.58	2215856	18504	0.34	1.09
		8	0.005046	163.72	2215856	10544	0.20	0.62
		9	0.002518	170.67	2215856	5912	0.11	0.35
	50	2	0.015504	152.49	2215856	10312	3.32	7.12
		3	0.010239	156.64	2215856	7144	2.30	4.93
		4	0.005856	162.23	2215856	4736	1.52	3.27
		5	0.003441	167.55	2215856	2952	0.95	2.03
		6	0.001797	174.04	2215856	1768	0.57	1.22
		7	0.000829	181.77	2215856	1032	0.33	0.71
		8	0.000475	187.35	2215856	568	0.18	0.39
		9	0.00019	196.53	2215856	304	0.10	0.20

5.2.2 Baboon

Image	Quality	K	MSE	PSNR	Embed File Size	Written	Total DCT%	Total File %
baboon	100	1	0.464548	118.49	2215856	352176	5.83	15.43
		2	0.244642	124.90	2215856	242672	4.02	10.63
		4	0.06536	138.10	2215856	99616	1.65	4.36
		5	0.033012	144.93	2215856	60544	1.00	2.65
		6	0.016829	151.67	2215856	35808	0.59	1.57
		7	0.008364	158.66	2215856	20744	0.34	0.91
		8	0.004192	165.57	2215856	11792	0.20	0.52
		9	0.00207	172.63	2215856	6616	0.11	0.29
	50	1	0.055116	139.81	2215856	30848	4.30	11.69
		2	0.037224	143.73	2215856	23592	3.29	8.94
		4	0.014023	153.50	2215856	10968	1.53	4.16
		5	0.00811	158.97	2215856	6864	0.96	2.60
		6	0.004261	165.41	2215856	4136	0.58	1.57
		7	0.002161	172.20	2215856	2408	0.33	0.91
		8	0.001019	179.71	2215856	1360	0.19	0.52
		9	0.000571	185.51	2215856	752	0.10	0.28

5.2.3 Peppers

Image	Quality	K	MSE	PSNR	Embed File Size	Written	Total DCT%	Total File %
peppers	100	1	0.431413	119.23	2215856	299992	5.35	16.30
		2	0.244553	124.91	2215856	214328	3.82	11.65
		3	0.134966	130.85	2215856	143280	2.55	7.79
		4	0.071841	137.16	2215856	91144	1.62	4.95
		5	0.03652	143.92	2215856	55840	0.99	3.03
		7	0.009398	157.50	2215856	19232	0.34	1.05
		8	0.00481	164.20	2215856	10952	0.20	0.60
		9	0.002415	171.09	2215856	6136	0.11	0.33
	50	1	0.028922	146.26	2215856	16336	4.36	9.57
		2	0.018803	150.56	2215856	12528	3.35	7.34
		3	0.012186	154.90	2215856	8712	2.33	5.10
		4	0.006902	160.59	2215856	5760	1.54	3.37
		5	0.003867	166.38	2215856	3592	0.96	2.10
		7	0.00113	178.68	2215856	1240	0.33	0.72
		8	0.000534	186.17	2215856	696	0.19	0.41
		9	0.000281	192.61	2215856	376	0.10	0.22

5.2.4 Bird

Image	Quality	K	MSE	PSNR	Embed File Size	Written	Total DCT%	Total File %
bird	100	1	0.355963	121.15	2215856	200008	4.32	16.09
		2	0.2411	125.05	2215856	151576	3.27	12.19
		3	0.15609	129.40	2215856	106640	2.30	8.58
		4	0.09135	134.76	2215856	70848	1.53	5.70
		5	0.05086	140.61	2215856	44536	0.96	3.58
		6	0.027627	146.71	2215856	26840	0.58	2.16
		7	0.013752	153.69	2215856	15720	0.34	1.26
		8	0.007355	159.95	2215856	8984	0.19	0.72
	50	1	0.008932	158.01	2215856	5216	4.61	7.52
		2	0.005211	163.40	2215856	3960	3.50	5.71
		3	0.003313	167.92	2215856	2696	2.38	3.88
		4	0.001868	173.65	2215856	1752	1.54	2.52
		5	0.001137	178.62	2215856	1064	0.93	1.52
		6	0.000517	186.50	2215856	632	0.55	0.90
		7	0.000236	194.33	2215856	360	0.31	0.51
		8	8.9e-05	204.14	2215856	184	0.16	0.27

6. Bibliography

[1] A., Westfeld: High capacity despite better steganalysis (F5 – a steganographic algorithm). In I. S. Moskowitz, editor, Information Hiding, 4th International Workshop, volume 2137 of Lecture Notes in Computer Science, Pittsburgh, PA, April 25–27, 2001. Springer-Verlag, New York.
URL: http://www.htw-dresden.de/~westfeld/publikationen/21370289.pdf

[2] Sathya, V.; Balasubramaniyam, K.; Murali, N.; Rajakumaran, M.; Vigneswari, "Data hiding in audio signal, video signal text and JPEG images," *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on* , vol., no., pp.741,746, 30-31 March 2012
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6215937&isnumber=6215562

[3] University of Southern California, . N.p.. Web. 3 Aug 2013.
URL: http://sipi.usc.edu/database/database.php?volume=misc

[4] Nicola, Asuni. "TESTIMAGES." *Open Source Software Solutions*. Tutti i diritti riservati. Web. 3 Aug 2013.
URL: http://testimages.sf.net

[5] Ron Crandall: Some Notes on Steganography. Posted on Steganography Mailing List, 1998.
URL: http://os.inf.tu-dresden.de/~westfeld/crandall.pdf

[6] Fridrich, Jessica, Tomáš Pevný, and Jan Kodovský. "Statistically undetectable jpeg steganography: dead ends challenges, and opportunities." *Proceedings of the 9th workshop on Multimedia & security*. ACM, 2007.

[7] Meena, Omprakash, Sathisha Basavaraju, and Arijit Sur. "DCT block location based data hiding." *Information and Communication Technologies (WICT), 2011 World Congress on*. IEEE, 2011.

[8] Pandian, Nithyanandam, and Ravichandran Thangavel. "A hybrid embedded steganography technique: optimum pixel method and matrix embedding." *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*. ACM, 2012.