Advanced JPEG Steganography & Detection

JOHN ORTIZ
HARRIS IT SERVICES
JOHN.ORTIZ@HARRIS.COM

Which Image has 411,080 bytes (~20%) of Hidden Data

A Closer Look – Picture #1 on Left

A Closer Look – Picture #2 on Right

Here is the Hidden Data

Overview

- The JPEG Algorithm
- Exploiting the JPEG Algorithm
- Detecting Exploited JPEGs
- DEMO-nstration of StegJpeg

A Little About Me

ENGINEER / PROFESSOR / COMEDIAN

An Engineer from Birth

- Wondered why I had 2 extra fingers and 2 extra toes
- Counted powers of 2 instead of sheep
- Sold my first computer game at age 2^4
  - My 1.0 MHz, 8-bit 6502, 48 K ram ATARI computer
- Published 2 more games eventually
  - None that you've ever heard of! ☺
- Graduated from Rose-Hulman Institute of Technology in 1988 (BSEE)
- Joined the U.S. Air Force

Even More About Me

- Graduated Air Force Institute of Technology (AFIT) in 1997 (MSEE, MSCE)
- Worked for Trident/Veridian/General Dynamics (2000)
- Worked for Raba/SRA International (2005)
- Worked for Crucial/Harris (2010 – present)
  - My companies keep getting sold off!!!
- I'm a reverse engineer/malware analyst and …
- I teach classes at the University of Texas in San Antonio
  - Steganography being one of them
- ENOUGH about me!

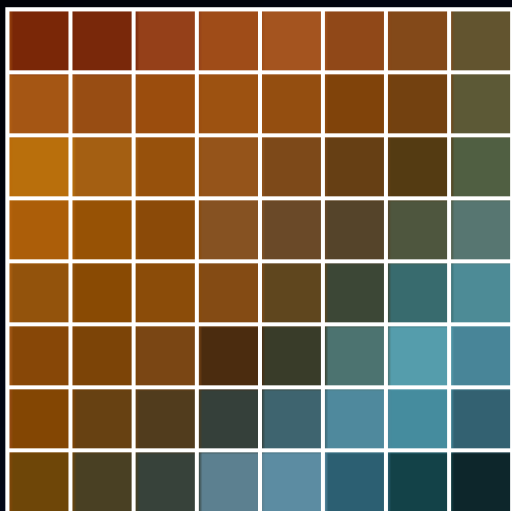
The JPEG Algorithm

IT'S SUPER EASY!

Why JPEG?

- The Joint Photographic Experts Group (JPEG) algorithm combines both lossless and lossy data compression techniques
- Achieves a small file size with little perceptible difference
- Particularly suited to natural images
  - Less efficient for cartoons with sharply defined lines
- One of the most common image formats if not *the* most common

JPEG Algorithm Overview

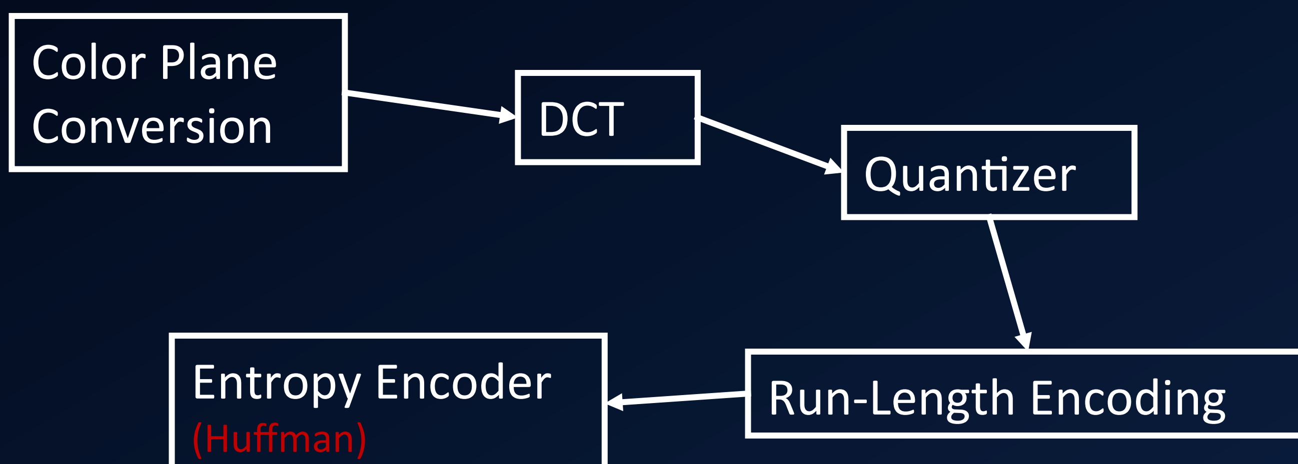


8 X 8
Image
Block

Std Quantization Table

(0,0)

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



JPEG Algorithm – Color Plane Conversion

- The JPEG algorithm first converts RGB to YC_rC_b
  - Y is the luminance component
  - C_r & C_b are the red difference and blue difference chroma components
  - Grayscale images only have the Y component
- The human eye is less sensitive to chrominance than luminance
  - Compression algorithms take advantage of this and sub-sample the values C_b & C_r without significant visual degradation
    - Can average 4 chrominance pixels resulting in better compression of C_r & C_b
- JPEG uses different quantization tables for chrominance components

Detailed JPEG Algorithm – Color Planes

- The image is grouped into 8x8 blocks
- Pixel values are converted from unsigned to signed
- JPEG converts RGB to YC_rC_b and treats each as it's own grayscale image
  - Grayscale has only the Y component

$$Y = 0.2989R + 0.5866G + 0.1145B$$

$$C_b = -0.1687R - 0.3313G + 0.5B + 2^4$$

$$C_r = 0.5R - 0.4187G - 0.0813B + 2^4$$

JPEG Algorithm – Discrete Cosine Transform

- The 2-dimensional Discrete Cosine Transform (DCT) is applied to a 8x8 image block
  - This process breaks down the frequency components of an image
    - Low Frequencies are smooth transitions in an image
    - High frequencies are sudden changes (like in a cartoon)
  - The purpose of this is to modulate the influence of different spectral components on the image
  - I.E. Higher frequencies contribute less information to the image and therefore can be reduced or eliminated

Detailed JPEG Algorithm – DCT Coefficients

- **Forward equation for the Discrete Cosine Transform**

$$b(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} a(x,y) \cos\left(\frac{\pi u(2x+1)}{2N}\right) \cos\left(\frac{\pi v(2y+1)}{2N}\right)$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & if\ u = 0 \\ 1 & otherwise \end{cases}$$

$$C(v) = \begin{cases} \frac{1}{\sqrt{2}} & if\ v = 0 \\ 1 & otherwise \end{cases}$$

- For JPEG, N = 8
- b(u, v) is the transform of the matrix
- a(x, y) is the pixel value at x, y
- when computing the cosine, make sure function is in radians
- How many times will this loop when implemented in a nested for loop?

Detailed JPEG Algorithm – DCT Coefficients

- There IS a use for a 4-dimensional array!!!
- This calculation can be done with a nested "for" loop

```
for( u = 0; u < 8; u++ )
    for( v = 0; v < 8; v++ )
        for( x = 0; x < 8; x++ )
            for( y = 0; y < 8; y++ )
            {   b(u, v) = b(u,v) + basis[u,v,x,y] * a(x, y)   }
```

- basis[u,v,x,y] holds all possible cosine results so they are not recalculated for each 8x8 block

JPEG Algorithm – Quantization & Entropy Codi

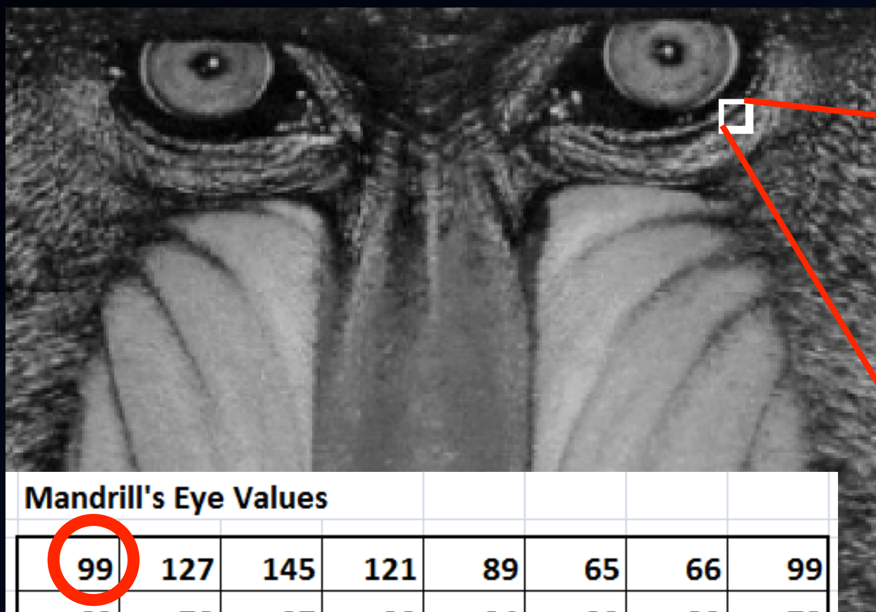
- The DCT results are quantized at a desired quality level
- Entropy coding is applied
  - A combination of Run-Length Encoding (RLE) and (Huffman or Arithmetic coding
- To view the image, the process is reversed
- The restored image looks similar or almost exact to the human, bu mathematically it is completely different
- For a high quality JPEG, there is almost no perceptible difference

Detailed JPEG Algorithm – Quantization & Cod

- An 8x8 **quantization** table is used to scale these coefficients
  - This is where the greatest loss occurs
  - The result is the "quantized DCT coefficients"
  - The Q-Tables control the quality
- The quantized DCT coefficient matrix generally has a lot of ZERO values – which are Run-Length coded away
  - The remaining compression is lossless
- With the small remaining numbers, Huffman or Arithmetic compression is applied
- Loop to next 8x8 block and repeat until the image is complete

Steganogr
applied he

JPEG Algorithm – Example Calculation



Mandrill's Eye Values

99	127	145	121	89	65	66	99
60	78	97	99	94	99	89	73
38	39	51	72	91	120	122	89
69	47	46	60	83	116	134	126
116	85	56	48	58	82	101	112
148	133	88	49	29	35	47	66
90	94	111	93	60	34	28	36
35	65	117	135	112	63	28	23

Note: BMP files start from bottom left pixel

JPEG Algorithm – Example Calculation

Mandrill's Eye Values

99	127	145	121	89	65	66	99
60	78	97	99	94	99	89	73
38	39	51	72	91	120	122	89
69	47	46	60	83	116	134	126
116	85	56	48	58	82	101	112
148	133	88	49	29	35	47	66
90	94	111	93	60	34	28	36
35	65	117	135	112	63	28	23

DCT Values

-377	24.8	-5.13	-20.1	-4.5	6.17	5.91	-0.45
64.86	-86.5	10.52	9.32	-15.5	1.34	-1.27	-1.64
9.38	97.41	-115	-70.7	7.12	-9.63	5.25	2.56
22.38	119	96.93	-29.4	4.57	-14.7	1.94	-1.87
34.5	-31.5	-8.7	-9.13	15	-3.4	2.52	2.15
2.4	0.09	9.72	-13.2	2.57	2.78	1.97	2.69
0.44	10.36	9.25	0.95	-2.41	-4.82	-3.21	-2.3
-6.03	-10.2	-5.95	0.46	1.93	2.59	0.32	1.12

Standard 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

Final Results After Quantization

-24	2	-1	-1	0	0	0	0
5	-7	1	0	-1	0	0	0
1	7	-7	-3	0	0	0	0
2	7	4	-1	0	0	0	0
2	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

JPEG Algorithm – Example Calculation

Final Results After Quantization

-24	2	-1	-1	0	0	0	0
5	-7	1	0	-1	0	0	0
1	7	-7	-3	0	0	0	0
2	7	4	-1	0	0	0	0
2	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Standard 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

De-Quantized DCT Values

-384	22	-10	-16	0	0	0	0
60	-84	14	0	-26	0	0	0
14	91	-112	-72	0	0	0	0
28	119	88	-29	0	0	0	0
36	-22	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Original DCT Values

-377	24.8	-5.13	-20.1	-4.5	6.17	5.91	-0.45
64.86	-86.5	10.52	9.32	-15.5	1.34	-1.27	-1.64
9.38	97.41	-115	-70.7	7.12	-9.63	5.25	2.56
22.38	119	96.93	-29.4	4.57	-14.7	1.94	-1.87
34.5	-31.5	-8.7	-9.13	15	-3.4	2.52	2.15
2.4	0.09	9.72	-13.2	2.57	2.78	1.97	2.69
0.44	10.36	9.25	0.95	-2.41	-4.82	-3.21	-2.3
-6.03	-10.2	-5.95	0.46	1.93	2.59	0.32	1.12

JPEG Algorithm – Example Calculation

De-Quantized DCT Values

-384	22	-10	-16	0	0	0	0
60	-84	14	0	-26	0	0	0
14	91	-112	-72	0	0	0	0
28	119	88	-29	0	0	0	0
36	-22	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

New Mandrill's Eye Values

95	129	147	122	84	70	78	88
55	79	98	97	90	90	87	82
32	38	49	69	98	117	111	95
67	50	38	52	88	119	126	119
125	94	59	46	57	79	98	109
139	116	86	60	41	35	49	66
92	96	103	97	69	37	26	33
37	66	110	133	112	65	33	26

Original Mandrill's Eye Values

99	127	145	121	89	65	66	99
60	78	97	99	94	99	89	73
38	39	51	72	91	120	122	89
69	47	46	60	83	116	134	126
116	85	56	48	58	82	101	112
148	133	88	49	29	35	47	66
90	94	111	93	60	34	28	36
35	65	117	135	112	63	28	23

JPEG Algorithm

- A LOT of calculations – 4096 per 8x8 block, plus color conversion
- * 3 color planes
- For a 512x512 image, that's 64x64 blocks * 4096/block
- $2^6 * 2^6 * 2^{12} = 2^{24}$ = 16+ million!
- Gosh, how long does this take?

JPEG Algorithm

- A LOT of calculations – 4096 per 8x8 block, plus color conversion
- * 3 color planes
- For a 512x512 image, that's 64x64 blocks * 4096/block
- $2^6 * 2^6 * 2^{12} = 2^{24}$ = 16+ million!
- Gosh, how long does this take?

- Well, How long does it take to open and view a JPEG file?
  - 3.6 GHz is fast!
- There is more detail, but we need to get to the steganography!

Exploiting the JPEG Algorithm and Hide Lots of Stuff

IT'S SUPER FUN!

JPEG Hiding – Swap DCT

- Choose two DCT coefficients which have the same value in the quantization table
  - Select middle frequencies so hidden bits are in <u>significant</u> portions of the image

  - The pair (2,0) & (1,2) works [#14]
    - Other pairs are highlighted

  - C_1 = coefficient for 2, 0
  - C_2 = coefficient for 1, 2
    - C_1 and C_2 are the coefficients, NOT the Q-Table values

Quantization Tab

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

JPEG Hiding – Swap DCT

- Select a cover block
- Get DCT transform of the block
- Read a message bit from the file to be hidden
  - If the bit is a 0, then $C_1 < C_2$ must be true
  - If the bit is a 1, then $C_2 < C_1$ must be true
- If the condition is already true, continue
  - By chance, our message bit is already there
- If the condition is not true, SWAP the coefficients
  - Note: this is done prior to quantization, so the difference must be large enough to hold true after quantization!
    - I can't figure out why the authors did it before quantization!

JPEG Hiding – Swap DCT

- Weaknesses in this approach
  - A particular cover block may be a poor candidate for hiding
  - Capacity is 1 bit per 8x8 block
    - For a 256 x 256 image, that's 32 x 32 = 1024 blocks (i.e. message bits) max
- You could increase capacity by using all 3 pairs …

JPEG Hiding – Swap DCT Improved

- Zhao & Koch improved on this technique
  - "Embedding Robust Labels into Images for Copyright Protection"
  - Operate on coefficients ***after*** quantization
  - Use 3 coefficients to store the message
- if message bit = 1
  - $C_1 > C_3 + D$ and $C_2 > C_3 + D$
  - D is a minimum distance between coefficients, normally D = 1
  - Greater D, greater robustness, but also greater perceptibility
- if message bit =0
  - $C_1 + D < C_3$ and $C_2 + D < C_3$
- Middle frequencies are selected

JPEG Hiding – Swap DCT Improved

- If modifications to coefficients exceed a threshold, block is marked invalid
- To increase security they use a triple of coefficients randomly chosen from the shaded values
- Need same random key for extraction

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	10
72	92	95	98	112	100	103	99

JPEG Hiding – High Capacity Swap DCT

- This technique unpublished to my knowledge
- Rather than picking a few matching pairs of coefficients, use multiple pairs
- For each pair, compare the *de-quantized* coefficients
  - The q-table values do not have to be equal
- If message bit is 0, make $C_1 < C_2$
- If message bit is 1, make $C_2 < C_1$
- If the two de-quantized values are equal, skip
  - Could modify them, but that increases detectability

JPEG Hiding – High Capacity Swap DCT

- Pairs chosen to more or less balance
- Start with outer pairs
  - Exclude the inner most pair, image affected substantially
    - Unless, capacity trumps perceptibility
  - Exclude DC component and last AC component as well
- Matching pairs are color coded

Exclude

	0	1	2	3	4	5	6	7
0	16	11	12	14	12	10	16	14
1	13	14	18	17	16	19	24	40
2	26	24	22	22	24	49	35	37
3	29	40	58	51	61	60	57	51
4	56	55	64	72	92	78	64	68
5	87	69	55	56	80	109	81	87
6	95	98	103	104	103	62	77	113
7	121	112	100	120	92	101	103	99

Least visual ir
but often zerc
less capacity

JPEG Hiding – High Capacity Swap DCT

- In implementation, the number of matching pairs used is an optio
  - 1 - 31 pairs
  - Quality=100, pairs = 24, 14587 (6.52%) random bytes hidden
  - 10,764 blocks, 10.84 bits/block – much better than 1

JPEG Hiding – High Capacity Swap DCT

- Using max of 31 coefficients to swap
  - Quality=100, pairs = 31, 22,291 (9.95%) random bytes embedded
  - 10,764 blocks, 16.57 bits/block, but …
  - Top 3 must be excluded IF visual distortion is an issue

JPEG Hiding – High Capacity Swap DCT Cryptographic

- You can always encrypt your data before you embed it …
  - That's no fun!!! ☹
- Cryptography is permutation and substitution
- Let's set limit at 24 pairs
  - Experimentally shows low visible distortion
- Save each pair of each block in a list
- 1920x1080 has 240 x 135 blocks x 24 pairs = 777,600 pairs (bits)
  - Max capacity is 97,200 bytes (less due to some equal coefficients)
- Permute the list, encrypt each bit
- Message spread cryptographically over entire image

JPEG Hiding – DCT Least Significant Bit (LSB)

- JSteg uses this approach
- Alter the LSB of each quantized DCT coefficient to hold our messa
- More than one bit/block capacity
  - Depends on number of non-zero coefficients
- Can use any coefficients that are not zero or one
  - If we used zero, that would increase capacity, distortion, and image size
  - A clear indication of data hiding would be a low number of DCT coefficient with a zero value
- Can't use '1' because … 0001_2 --> change LSB and 0000_2!
- '-1' is OK, change LSB of 1111_2 to 1110_2 … it becomes '-2'

JPEG Hiding – DCT LSB

- Quality: 65
- File Size: 351,151 bytes
- Storage Capacity: 52032 bytes
  - 14.82% of file size
- Used: 42779 bytes
  - 82.22% of available
- @ Q=100
  - Storage Capacity: 97639 bytes
  - Better looking image, BUT
  - File Size: 962,763 bytes
    - Is that bad???

JPEG Hiding – Outguess

- Outguess is available for free download
- Hides in LSBs of DCT coefficients
- Pseudo-randomly permutes selection DCT coefficients that are no ZERO or ONE
- After embedding a second pass is made to make corrections to unused coefficients such that DCT histogram is preserved
  - Reduces capacity as some coefficients are used for correction
  - Makes detection more difficult

JPEG Hiding – Average DCT

- Have not seen a paper with this specific idea
  - It may exist, there are 100's of papers on JPEG manipulation
- Choose some number of non-zero DCT coefficients to average
- Store the message in the LSB (s) of the average
  - $(4 + 5 + -2 + 1) / 4 = 8 / 2 = 4 = 0100_2$
    - If the LSB of the message is zero, we're done
    - If the LSB of the message is one, subtract 1 from 4
    - $(3 + 5 + -2 + 1) / 4 = 7 / 2 = 3.75 = 3 = 0011_2$
- Predetermine number of coefficients to average
- Predetermine number of bits to store in each average

JPEG Hiding – Average DCT

- As number of coefficients for each average goes up, number of bit to hide goes down
- Can't re-use coefficients since you can't change them twice
  - Could track which ones change and not reuse those …
- As number of bits per average goes up, more perceptible since more change required

JPEG Hiding – F5

- F5 takes a different approach to hiding in the DCT coefficients
- F5 has a fairly high capacity, but very low detectability
- F5 decrements the magnitude of the coefficient values when the LSB does not match the message
  - As opposed to overwriting them with the message bits
  - Note that 1 and -1 become zero – called shrinkage
    - Must be decremented to zero since a 2 will become a 1
  - The DCT average technique may decrement or increment
- Skips zero for embedding and extraction

JPEG Hiding – F5

- Inverts the *meaning* for negative DCT coefficients
  - An LSB of 1 in a negative coefficient represents a zero
  - Prevents uneven distribution of odd vs. even coefficients
- Uses permutative straddling
  - Spreads the message over the entire image
  - Like the cryptographic spreading of the other techniques
- Uses matrix encoding to reduce the amount of change required
  - Embed 2 bits using 3 modifiable coefficients – 1 change hides 2 bits
    - $x_1 = a_1$ xor a_2; $x_2 = a_2$ xor a_3 --- change nothing
    - $x_1 \neq a_1$ xor a_2; $x_2 = a_2$ xor a_3 --- change a_1
    - $x_1 = a_1$ xor a_2; $x_2 \neq a_2$ xor a_3 --- change a_2
    - $x_1 \neq a_1$ xor a_2; $x_2 \neq a_2$ xor a_3 --- change a_3

JPEG Hiding – Statistically Invisible Steganogra

- SIS performs a complexity analysis of each 8x8 DCT block
- Number of non-zero coefficients must exceed a threshold or the entire block is skipped
  - thr = 0.3 to 0.6
    - 20 to 39 coefficients out of a block must be non-zero
- Adds up different sets of |coefficients| to produce a sum
- If the LSB of the sum equals the message, next block
- If not, add/subtract 1 from the largest magnitude

JPEG Hiding – YASS

- Yet Another Steganographic Scheme that resists blind steganalysis
- What YASS does a little differently is to select blocks larger than 8x
  - Example: 10 x 10
  - Has 9 possible sub-blocks
- Out of the larger block, YASS selects an 8x8 block, performs the D conversion and quantization
- Hides in those coefficients
- Must use an error correcting code since there will be some errors when converted to JPEG

JPEG Hiding – High Capacity DCT

- High Capacity Data Hiding in JPEG Compressed Images"
  - Chang, C.C. and Tseng, Hsien-Wen
  - Much greater capacity than 1 bit per 8x8 block
- It is an adaptive DCT LSB technique
  - Hides mostly in lower and middle frequency components
  - Performs a capacity estimation
  - Adapts to different characteristics of each block
- Experimentally, the quality must be >= ~75 to remain imperceptib
  - At quality = 50 (the standard) visual distortion is obvious
  - At quality = 60, it is noticeable if you are looking for it

JPEG Hiding – High Capacity DCT - Algorithm

- Choose the block to be embedded
- Determine classification of the block:
  - Uniform
  - Non-uniform
- Set the α value (to be discussed shortly)
- Determine the number of bits to hide in each *quantized* DCT coefficient
- Replace these bits with bits from the message data
- Repeat

JPEG Hiding – High Capacity DCT

- If a background has a strong texture, the Human Visual System (HVS) is less sensitive to distortions (non-uniform)
- Non-uniform blocks can use a larger α value
  - X * α where X is between 1.0 and 9.9
- Sum of squares of AC, DCT coefficients
  - If G < threshold, block is uniform

$$G = \sqrt{\sum_{x=1}^{63} (D_x)^2}$$

- Calculate capacity based upon the quantization table
  - User sets an α (alpha) factor
    - Higher α, higher bit rate, but increased distortion
  - User also sets a uniformity factor
    - How much to increase α for non-uniform block

JPEG Hiding – High Capacity DCT

- Lower frequency components hold fewer bits and cause more visual distortion
- Higher frequency components have more bits, but there are fewe overall (many are zero)
- Bits can be hidden in the DC component
  - Since it is generally large, more bits can be hidden
  - However, it is more perceptible sooner, especially if the quality factor is hi
- Can use any coefficient except zero and one

JPEG Hiding – High Capacity DCT

- Do two calculations to determine capacity, take the lower value
  - @ x = 3, y = 1
  - C_Q = floor(α * lg(17)) >= 4
  - M = floor (lg(7)) = 2
  - Can hide 2 bits
    - Msg = 10_2 and 7 = 111_2. 7 is changed to 6 … 110_2

* Note: lg = $\log_2$, lg (8) = 3

Q-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	1

DCT Coeffic

Final Results After Quantization

-24	2	-1	-1	0	0	0	0
5	-7	1	0	-1	0	0	0
1	7	-7	-3	0	0	0	0
2	7	4	-1	0	0	0	0
2	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

JPEG Hiding – High Capacity DCT

- The extractor must be able to determine which blocks are uniform and non-uniform too
- Can't use the same calculation because the modified values will result in a different G value
  - This may change whether it crosses the threshold
- Chang et al. chose to use the last AC coefficient as flag
  - Zero if uniform
    - most blocks are uniform
    - most AC coefficients are zero
  - One if non-uniform
  - Requires modified Q table with 64^{th} Q value = 1

Look Here!

JPEG Hiding – High Capacity DCT

- Modified Q-Table is a BIG RED FLAG that something is unusual
- Used only upper bits (max limited to 5 or less) to calculate uniformity
- Never affected by data changes
- No modified Quantization Table!!!
- Extractor performs same operations except grabs the bits

- This technique works best with higher quality JPEGs

JPEG Hiding – High Capacity DCT

hid_S_OrangeCatailPatch_a8.00_u8.0_max_5_q100.jpg

- Image: 700 x 474
- Size: 500,921
- Quality: 100
- $\alpha = 8, u = 8$
- Cap: 128,932 bytes
  - 25.74 %
- Used: 122,583 bytes
  - 24.47 % total
  - 95.08 % of available

JPEG Hiding – High Capacity DCT

hid_S_OrangeCatailPatch_a8.00_u8.0_max_5_q80.jpg

- Image: 700 x 474
- Size: 282,818
- Quality: 80
- α = 8, u = 8
- Cap: 74,994 bytes
  - 26.52 %
- Used: 74,994 bytes
  - 26.52 % total
  - 100 % of available

JPEG Hiding – High Capacity DCT

hid_S_OrangeCatailPatch_a8.00_u8.0_max_5_q60.jpg

- Image: 700 x 474
- Size: 113,863 bytes
- Quality: 60
- $\alpha = 8$, $u = 8$
- Cap: 27,624 bytes
  - 24.26%
- Used: 27,624 bytes
  - 24.26 % total
  - 100 % of available

JPEG Hiding – High Capacity DCT

hid_S_OrangeCatailPatch_a2.00_u2.0_max_5_q60.jpg

- Image: 700 x 474
- Size: 113,332 bytes
- Quality: 60
- $\alpha = 2$, $u = 2$
- Cap: 24,887 bytes
  - 21.96 %
- Used: 24,887 bytes
  - 21.96 % total
  - 100 % of available

JPEG Hiding – High Capacity DCT

hid_S_OrangeCatailPatch_a4.00_u4.0_max_5_q50.jpg

- Image: 700 x 474
- Size: 66,517 bytes
- Quality: 50
- α = 4, u = 4
- Cap: 15,091 bytes
  - 22.69%
- Used: 4128 bytes
  - 6.21% total
  - 27.35% of available

JPEG Hiding – High Capacity DCT Cryptographic

- Create a list of all available DCT coefficients, regardless of block
  - Many techniques only permute the order of the blocks
- Permute the list, encrypt the bits
- Q=50, cap=15,091
- Used: 4128

Detecting Exploited JPEGs

IT'S SUPER COOL!

Steganalysis – Defeating Steganography

- Three levels of defeat for steganography:
  - Detection
  - Extraction
  - Destruction
- Beware!!! Steganalysis papers are *highly* mathematical!
  - For Me: 2 + 2 = 4 Them: $sum = \sum_{x=1}^{2} (2)$
  -
- NOTE: Detection ability based on embedded data size!
  - If you embed a single bit into a single DCT coefficient, it is not detectable!

Steganalysis – Defeating Steganography

- General Approach
  - Get as much information as possible
    - Adversary's goal
    - Tool(s) likely used
    - Types of cover files
    - Type of message(s)
  - Check for tool signatures
    - Ex: modified Quantization table

Steganalysis – Defeating Steganography

- Get as much information as possible
  - Adversary's goal
  - Your goal
    - Detection
    - Extraction
    - Destruction
  - Tool(s) likely/possibly used
  - Types of cover files
  - Type of message(s)
    - Text
    - Encrypted/compressed
    - Other images

Steganalysis – Defeating Steganography

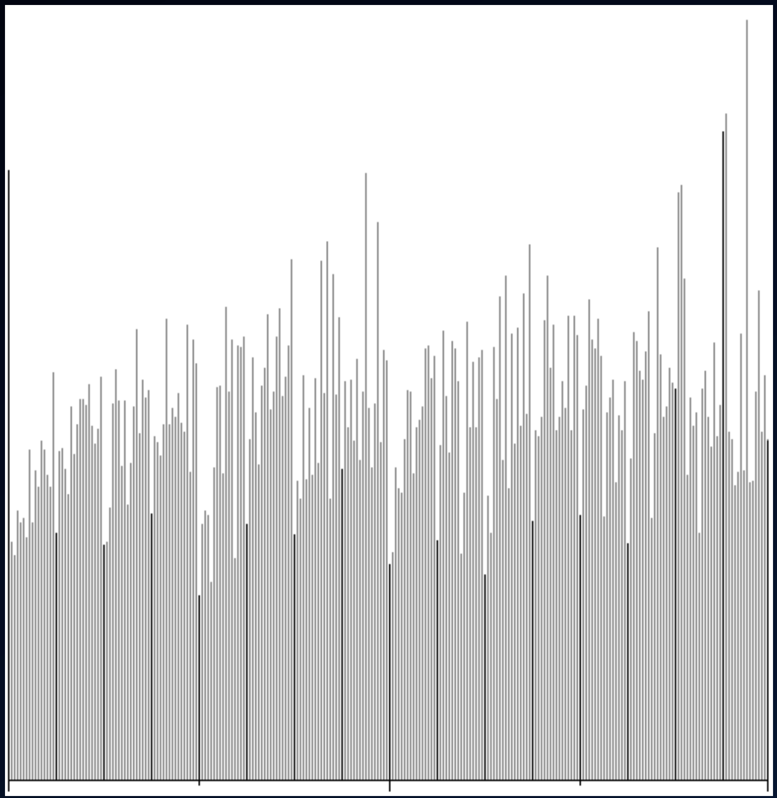
- Check for tool signatures
  - Ex: modified Quantization table
  - Specific files existence
    - if you are forensically examining a disk
  - Specific types of distortion
    - For JPEG, the typical artifact is BLOCKINESS
- Apply analysis specific to the tool or
  - … to the target cover files

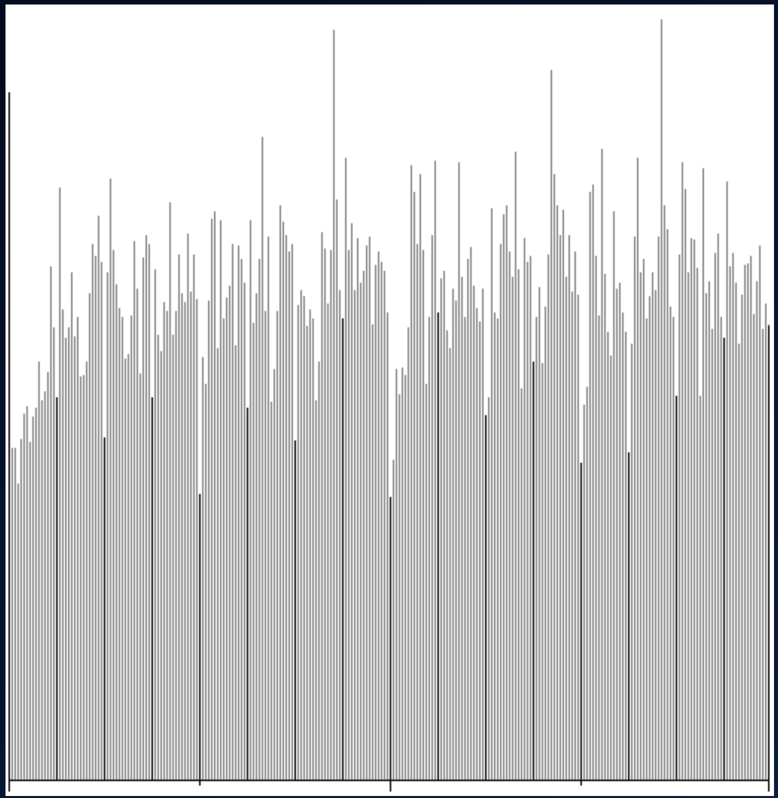
Steganalysis – Defeating JPEG Steganography

- Two general approaches to JPEG detection
  - Analysis of DCT Coefficients
  - Block edge detection
    - Blockiness
- Some techniques rely on training on clean images
- Apply Statistics
  - Histograms & Entropy
  - Chi Squared test
  - More … lots more!

Steganalysis – Defeating JPEG Steganography Analyzing DCT Coefficients

- Histograms and Entropy of a JPEG file are not very useful (Q=100)
  - Entropy is 7.98 for hidden data vs. 7.96, BUT hid a text file! --- **WHICH ONE**





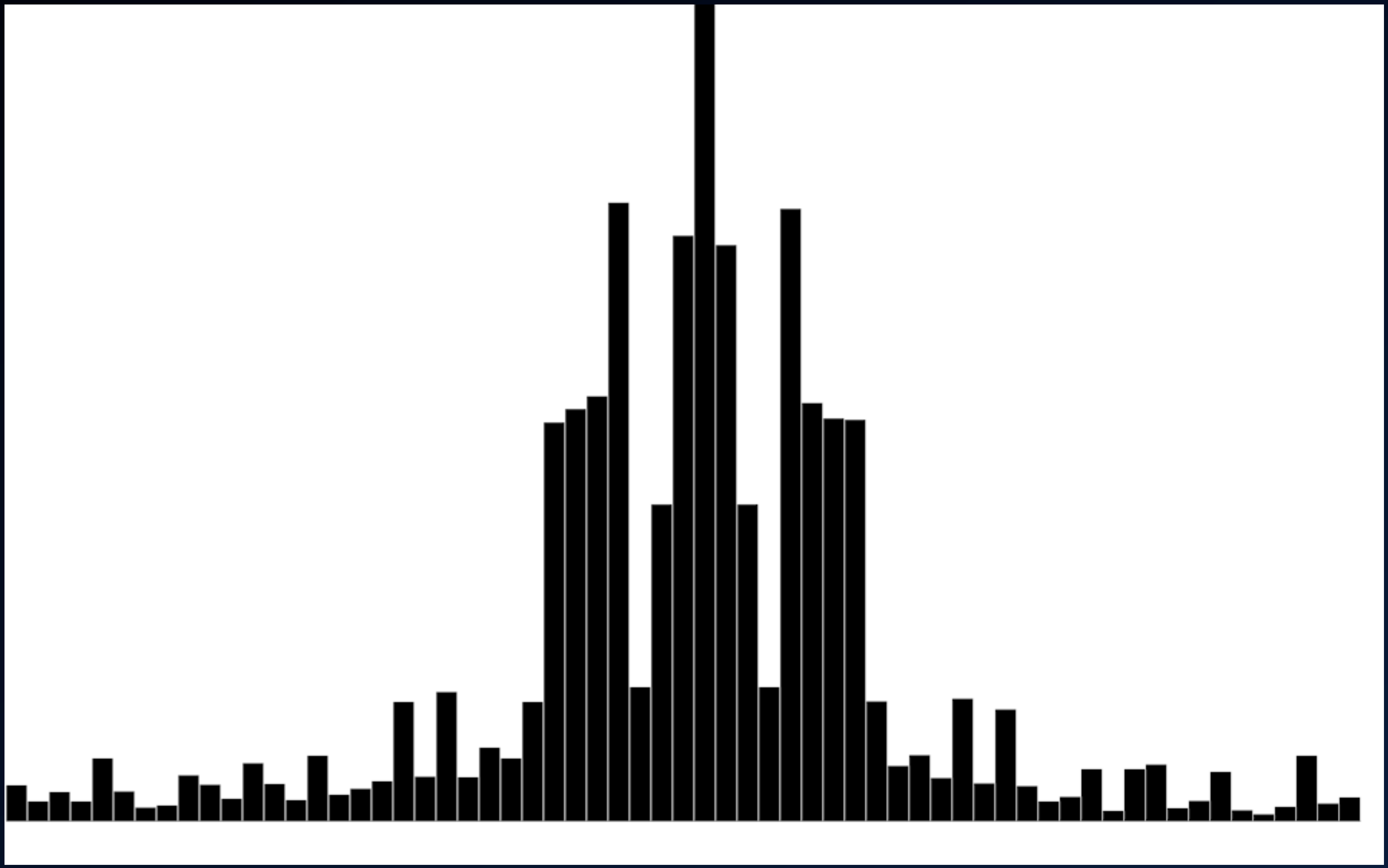
Steganalysis – Defeating JPEG Steganography

- The DCT coefficient distribution is naturally balanced
  - The number of coefficients that equal “1” is roughly equal to the number o coefficients that equal “-1”
  - The number that equal “2” ~= number that equal “-2”
- When substitution a bit into the coefficients
  - A “2” becomes a “3”, but a “-2” becomes a “-1”
  - A “3” becomes a “2”, but a “-3” becomes a “-4”

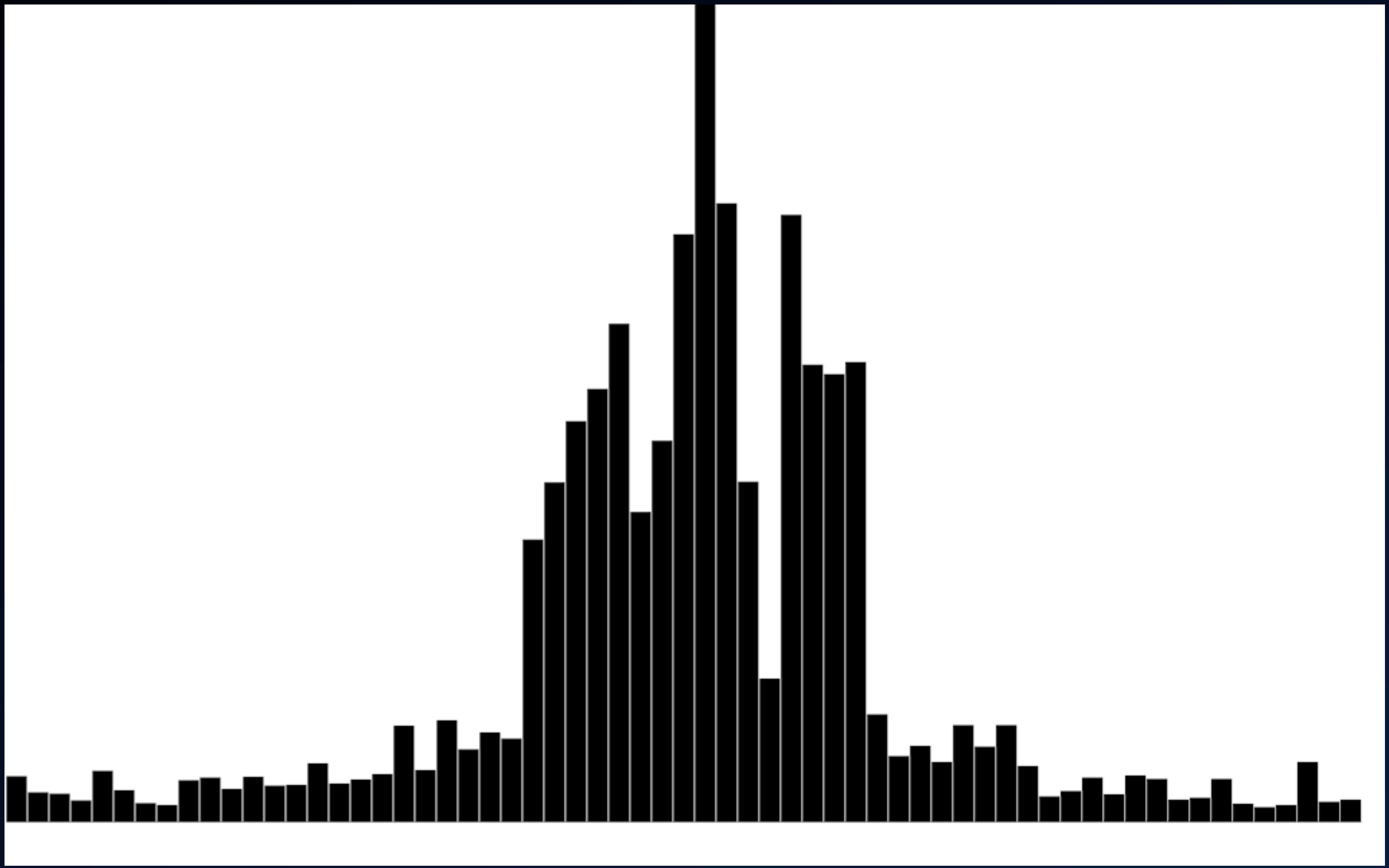
Steganalysis – Defeating JPEG Steganography
Analyzing DCT Coefficients

- High Capacity DCT hiding
- Image: 1751 x 1173
- Size: 460,758
- Quality: 75
- $\alpha = 2$, $u = 2$
- Cap: 86,455 bytes
  - 18.76 %
- Used: 42,779 bytes
  - 9.28 % total
  - 49.48 % of available

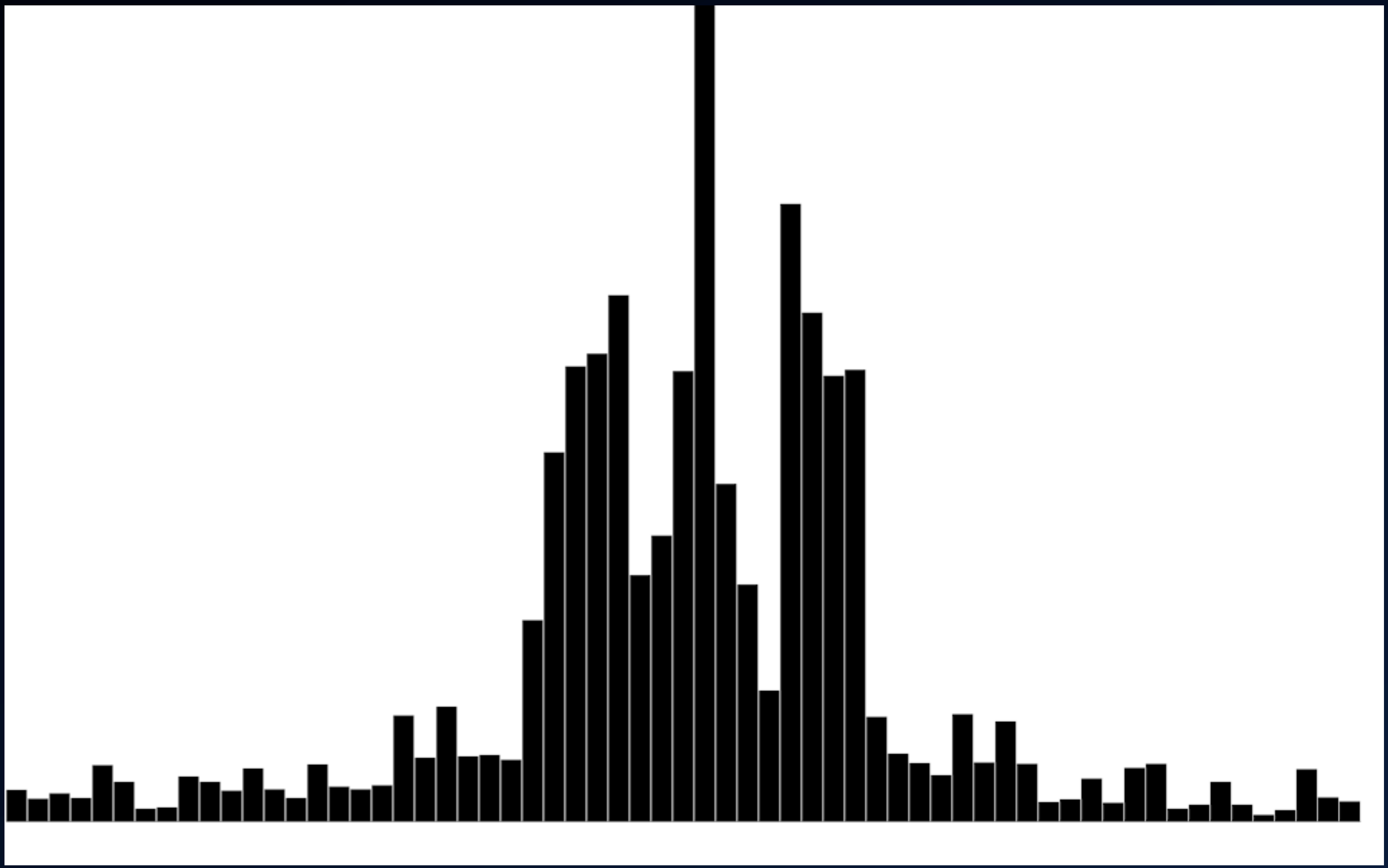
Steganalysis – Defeating JPEG Steganography Analyzing DCT Coefficients - Before



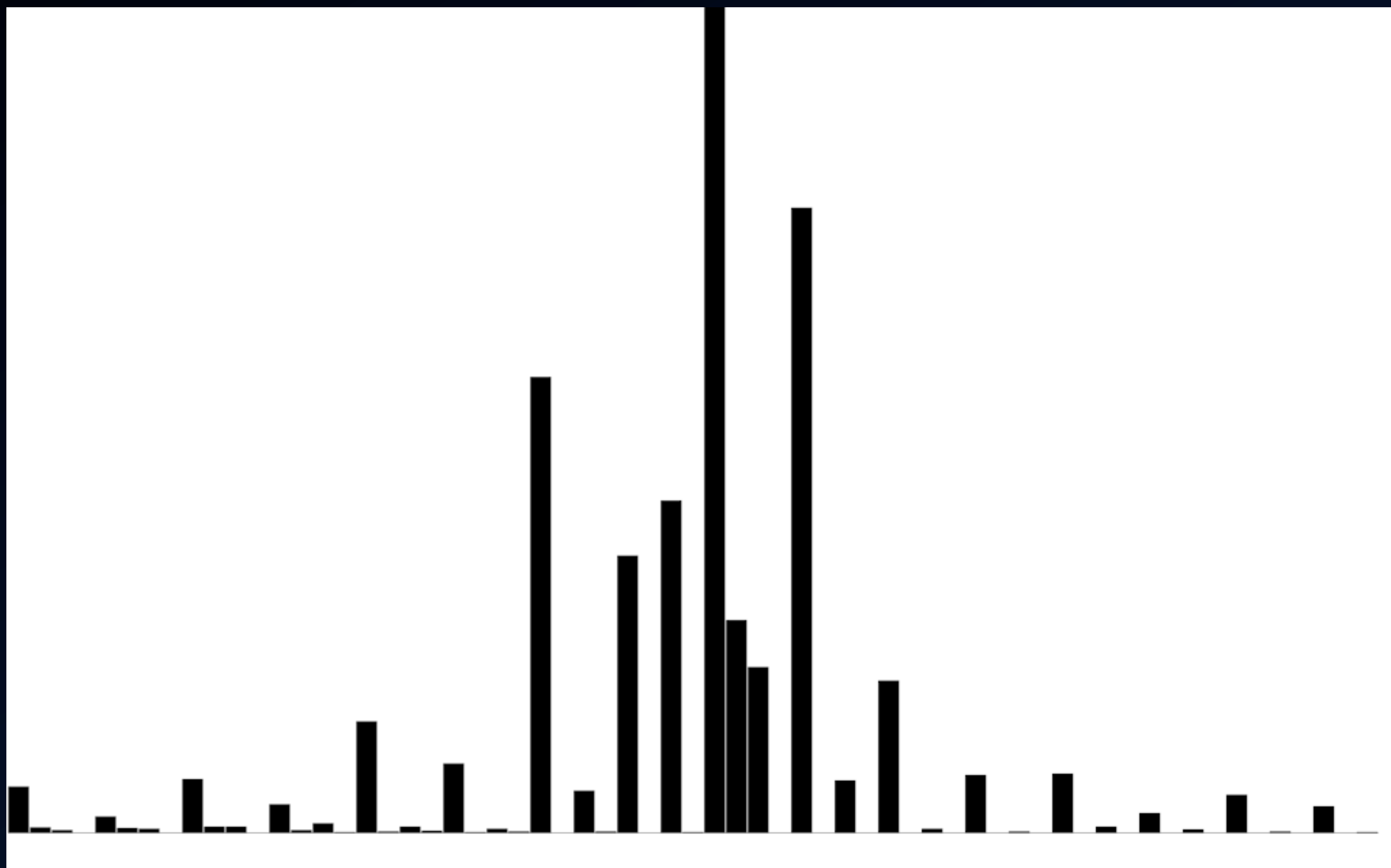
Steganalysis – Defeating JPEG Steganography
Analyzing DCT Coefficients – After Hiding



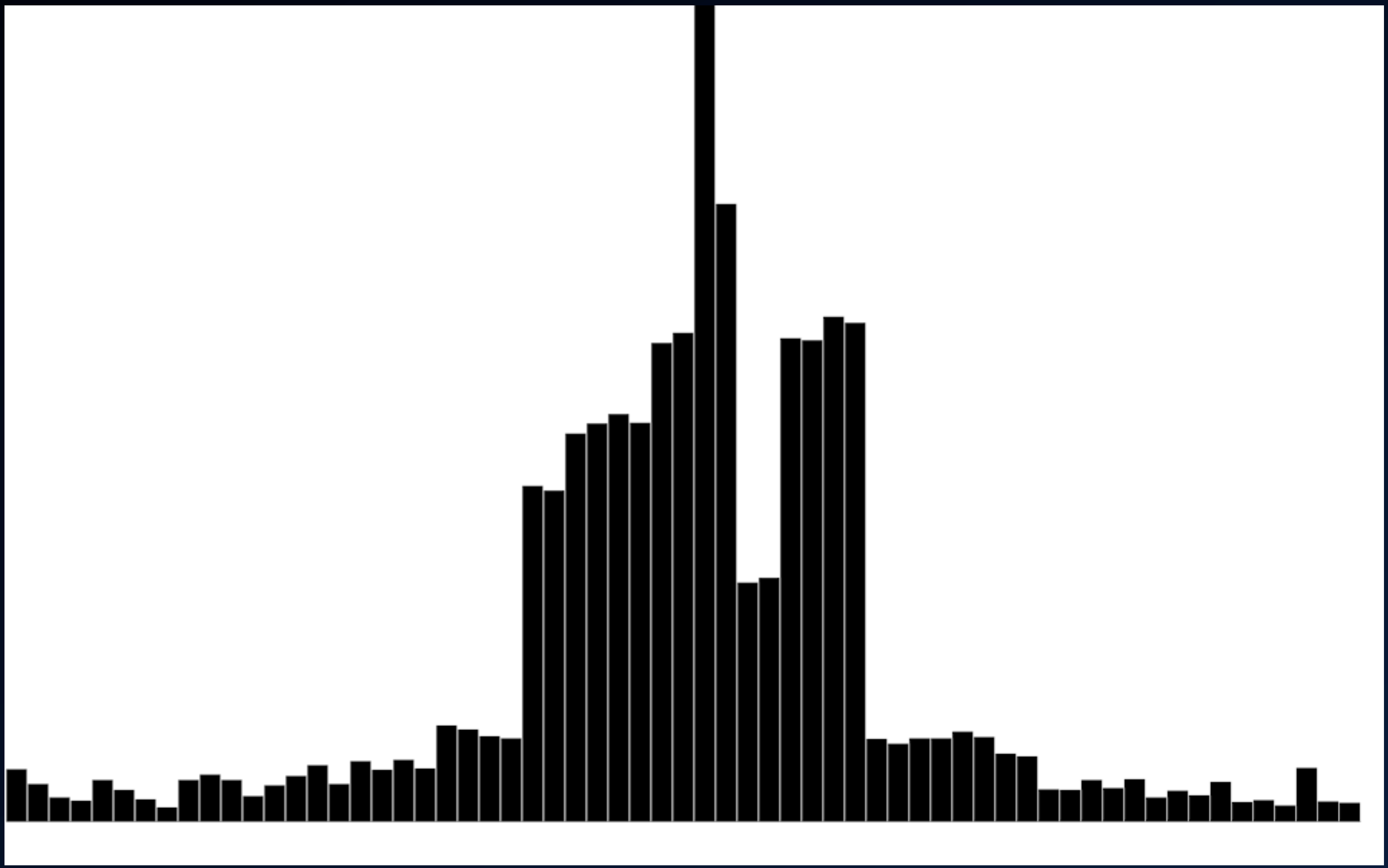
Steganalysis – Defeating JPEG Steganography
Analyzing DCT Coefficients – After Hiding LSB



Steganalysis – Defeating JPEG Steganography Analyzing DCT Coefficients – After Wiping



Steganalysis – Defeating JPEG Steganography
Analyzing DCT Coefficients – After Randomizin



Steganalysis – Defeating JPEG Steganography

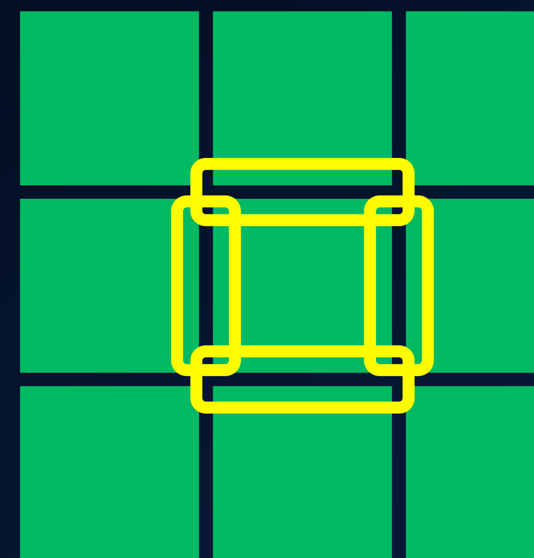
- Outguess uses excess capacity to make adjust DCT coefficients
  - Keeps the balance
- SwapDCT does not change the value of any coefficients
  - This analysis reveals nothing for SwapDCT
- F5 mitigates changes in coefficients
  - Uses matrix encoding to reduce actual number of changes
  - Does not substitute bits, decrements existing values, maintaining balance
- For these and other techniques, different detection methods needed

Steganalysis – Defeating JPEG Steganography

- An approach to detect F5 is to predict the histogram of the original cover image
  - F5 *does* increase the number of ZERO coefficients
- Decompress the stego image, crop it by 4 columns, recompress using same quantization table
  - Spatially, an image cropped by just 4 vertical columns is nearly identical
- Apply a blurring algorithm to reduce blockiness introduced by the cropping
- Compare predicted histogram with stego-image histogram
- Able to calculate approximate message length as well

Steganalysis – Defeating JPEG Steganography

- When modifying DCT coefficients, spatial discontinuities increase the 8x8 boundary
  - i.e. "blockiness"
- Measure the discontinuity at the 8x8 edges
  - Most 8x8 blocks will have 4 boundary edges
- Again, use the cropped image as estimate
- Measure blockiness of both and compare



$$B = \sum_{i=1}^{\lfloor (M-1)/8 \rfloor} \sum_{j=1}^{N} \left| g_{8i,j} - g_{8i+1,j} \right| + \sum_{j=1}^{\lfloor (N-1)/8 \rfloor} \sum_{i=1}^{M} \left| g_{i,8j} - g_{i,8j+1} \right|$$

Steganalysis – Extracting JPEG Steganography

- Extraction is much more difficult than detection
- Cryptography complicates extraction
  - Doesn't prevent detection
- Knowing the method is critical
  - If you extract LSBs from a JPEG that used Swap DCT, you gain no informati about the message

Steganalysis – Destroying JPEG Steganography

- Sterilization of data hidden in a jpeg is easy
- Could ZERO or RANDOMIZE the LSBs of the DCT coefficients
  - But that's too hard
- Could hide another message on top of prior message
  - Similar to randomization
  - Use the same tool if known
- Resize the image – EASY!
  - NOT in multiples of 8!
  - Resize by a single (or 2) horizontal columns and vertical rows
  - Completely changes DCT coefficients

DEMO - nstrations

WILL THEY ACTUALLY WORK?
THE FIRST TIME?

>:-)

Demonstrations

- DCT Swap
- DCT LSB
- High Capacity DCT

Questions

COMMENTS OR COMPLAINTS

References

THANKS TO ALL THE FOLLOWING FOR THEIR HARD WORK

References

- http://en.wikipedia.org/wiki/YCbCr
- "Embedding Robust Labels into Images for Copyright Protection", Jian Zhao, Eckhard Koch
- "A Method of Embedding Binary Data into JPEG Bitstreams", Hiroyuki Kobayashi, Yoshihiro Noguchi, Hitoshi Kiya
- "High Capacity Data Hiding in JPEG Compressed Images", Chang, C.C. and Tseng, Hsien-Wen
- Compressed Image File Formats, JPEG, PNG, GIF, XBM, BMP, John Miano, Addison Wesley
- "Defending Against Statistical Steganalysis", Niels Provos

References

- **“**A JPEG-Based Statistically Invisible Steganography**”,** Qingzhong Li Andrew H. Sung, Zhongxue Chen, Xudong Huang
- “F5 - A Steganographic Algorithm - High Capacity Despite Better Steganalysis”, Andreas Westfeld, Technische Universit¨at Dresden
- “Detection of Hiding in the LSB of DCT Coefficients”, Mingqiao Wu Zhongliang Zhu, and Shiyao Jin
- “STEGANALYSIS OF BLOCK-DCT IMAGE STEGANOGRAPHY”, *Ying Wang and Pierre Moulin,* University of Illinois at Urbana-Champaig
- “Attacking the OutGuess”, Jessica Fridrich, Miroslav Goljan, Dorin Hogea

References

- "An Effective Algorithm for Breaking F5", Hong Cai, Sos Agaian, University of Texas at San Antonio
- "Pairs of Values and the Chi-squared Attack", Christy Stanley, Iowa State University
- "Steganalysis of JPEG Images: Breaking the F5 Algorithm", Jessica Fridrich, Miroslav Goljan, Dorin Hogea, SUNY Binghamton, NY
- "New Methodology for Breaking Steganographic Techniques for JPEGs", Jessica Fridrich, Miroslav Goljan, Dorin Hogea, SUNY Binghamton, NY