Steganography & Steganalysis

INSTRUCTOR: JOHN ORTIZ
STEGO@SATX.RR.COM

TRANSFORM DOMAIN

Transform Domain

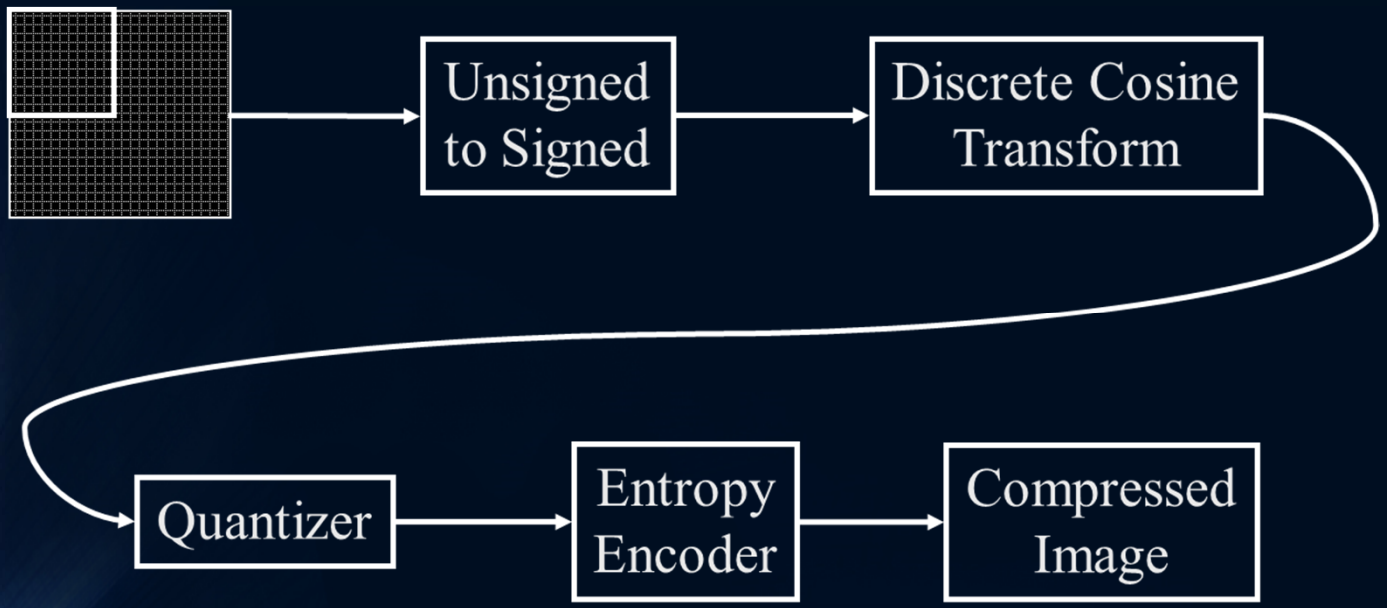
- Apply a mathematical transform and hide data in that process
- Hide data in more significant portions of the cover
- Generally, more robust than substitution techniques
- Can better survive common image/audio manipulations
  - Affine transforms
    - scaling, rotating, shearing, translating, flipping
  - Lossy compression
  - Analog-to-Digital and Digital-to-Analog Conversions
    - Scan/print, fax, compact disc, DVD, etc.

Transform Domain

- Many mathematical transforms exist
  - Discrete Cosine Transform (DCT)
  - Discrete Fourier Transform (DFT)
  - Laplace Transform
  - Wavelet Transforms
  - Modulated Complex Lapped Transform
  - Mellin-Fourier Transform
  - Is this enough?
- All of these can be applied to images and/or audio
- We'll be focused on the DCT which is used in JPEG

JPEG Algorithm

- The JPEG algorithm is a complex series of steps which makes use of lossy and lossless compression techniques



JPEG Algorithm

- The JPEG algorithm first converts RGB to YC_rC_b
  - Y is the luminance component
  - C_r & C_b are the color components (hue, saturation)
  - Grayscale images only have the Y component

JPEG Algorithm

- The human eye is less sensitive to chrominance than luminance
  - Compression algorithms take advantage of this and sub-sample the values of C_b & C_r without significant visual degradation
    - can average 4 chrominance pixels and treat as one
    - i.e. result is better compression of C_r & C_b
- JPEG uses different quantization tables for chrominance components
- The Discrete Cosine Transform (DCT) is applied to an 8x8 image block

JPEG Algorithm

- The results are quantized to the desired quality
  - "The purpose of this is to modulate the influence of different spectral components on the image"
    - Follow that?
  - The higher frequencies contribute fewer details to the image and can therefore be reduced or eliminated
- A combination of Run-Length Encoding (RLE) and Huffman coding is applied

JPEG Algorithm

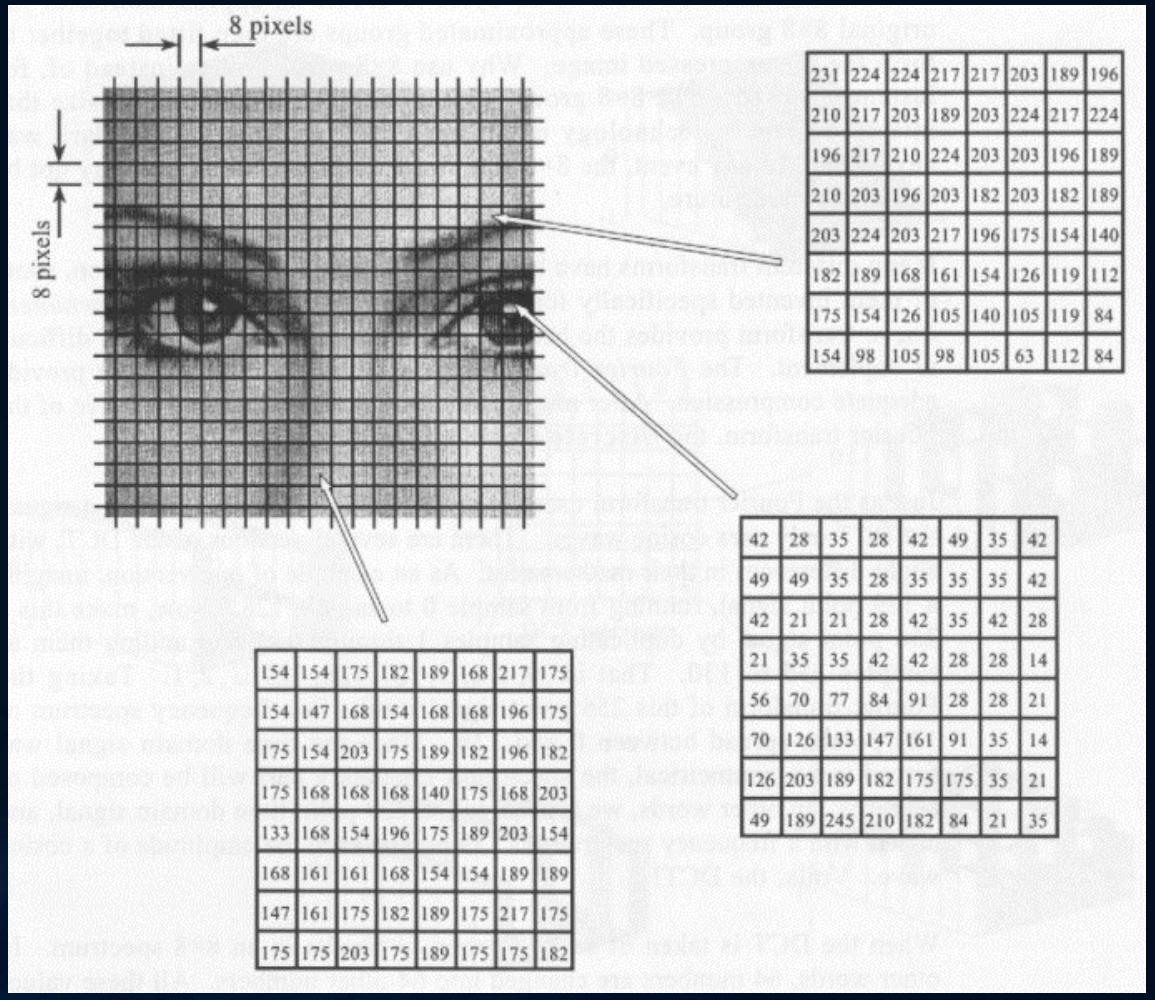
- To get the image back, the process is reversed
- The restored image *looks* very similar but is mathematically completely different than the original
- If high quality was used, there should be little, if any, *perceptible* difference

JPEG Algorithm

- An image is grouped into 8 X 8 blocks
  - There is less change between spatially adjacent pixels
- The image pixels are changed from unsigned to signed
  - For grayscale images with pixel values ranging from 0 to 255, 128 is subtracted to result in signed values –128 to +127
  - For RGB color images, JPEG converts RGB to YC_rC_b and treats each as it's own "grayscale" image

$$Y = 0.2989R + 0.5866G + 0.1145B$$
$$C_b = -0.1687R - 0.3313G + 0.5B + 2^4$$
$$C_r = 0.5R - 0.4187G - 0.0813B + 2^4$$

JPEG Algorithm



JPEG Algorithm

- The DCT is applied to each pixel in the 8 x 8 matrix
  - The output of the DCT algorithm is itself an 8 X 8 matrix
  - These are called the "DCT Coefficients"
  - There is a small loss due to the cosine approximation
- An 8 x 8 quantization table is used to scale each corresponding DCT coefficient
  - The quantization step is where the greatest loss occurs
  - The Q tables were developed via experimentation based on human visual perception
  - Different Q tables are used for different quality levels

JPEG Algorithm

- The resulting 8 x 8 matrix generally has a large proportion of zero values
  - Lower quality jpegs will have higher numbers of zeros
- --- The remaining compression is lossless ---
- It is run-length encoded using a simple count
- The few values left are treated as raw data and entropy encoded using either Huffman or Arithmetic techniques
  - Based on my own experience, Huffman seems to be the predominant choice
- The process is repeated for the entire image

JPEG Algorithm

8 X 8
Image
Block

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

Color Plane Conversion → DCT → Quantizer → Run-Length Encoding → Entropy Encoder (Huffman)

Discrete Cosine Transform

- 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?

Discrete Cosine Transform

- Programmatically, this calculation can be implemented using four nested "for" loops
- 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) }
- More on basis[u,v,x,y] shortly
  - Finally! A practical use for a 4-dimensional array!

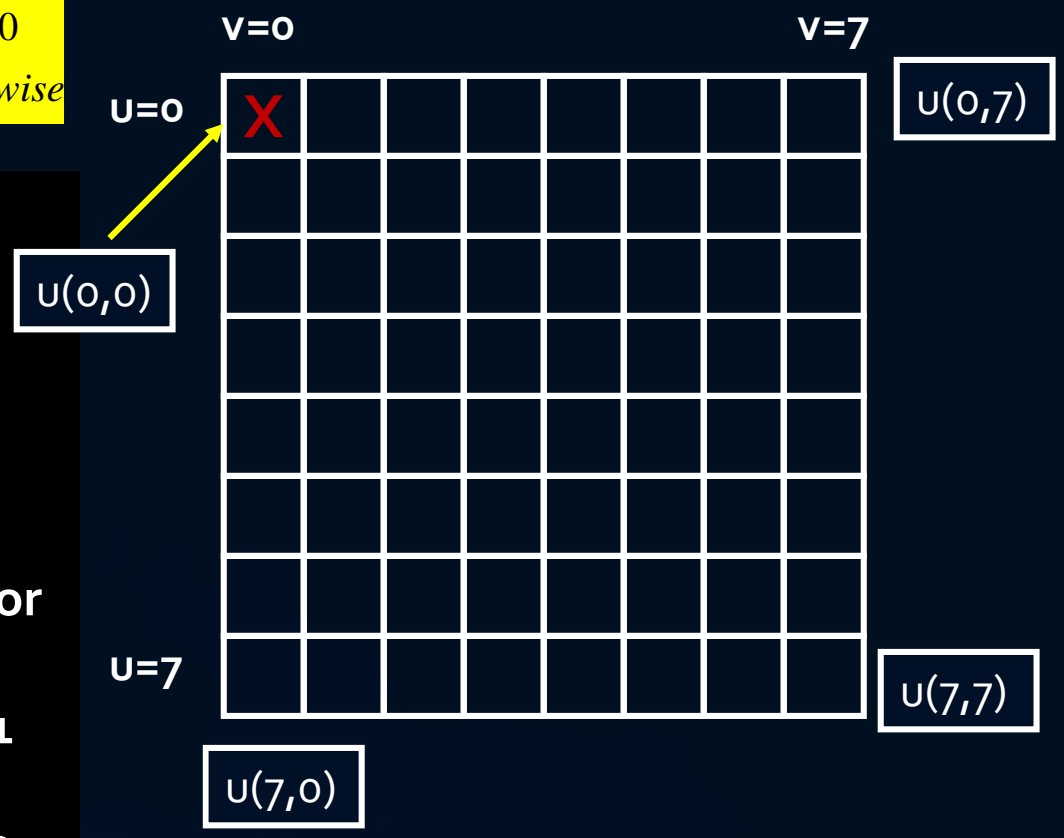
Discrete Cosine Transform

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

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

- X is called the DC coefficient
  - DC is 0 frequency
- the rest are AC coefficients
- there are 64 summations to determine each square
- there are 64 squares in a block
- that's 4096 total calculations for a single 8 x 8 block
- for u=0, v=0, COS terms go to 1

X = (a(0,0) + a(0,1) + a(0,2) …) / 8



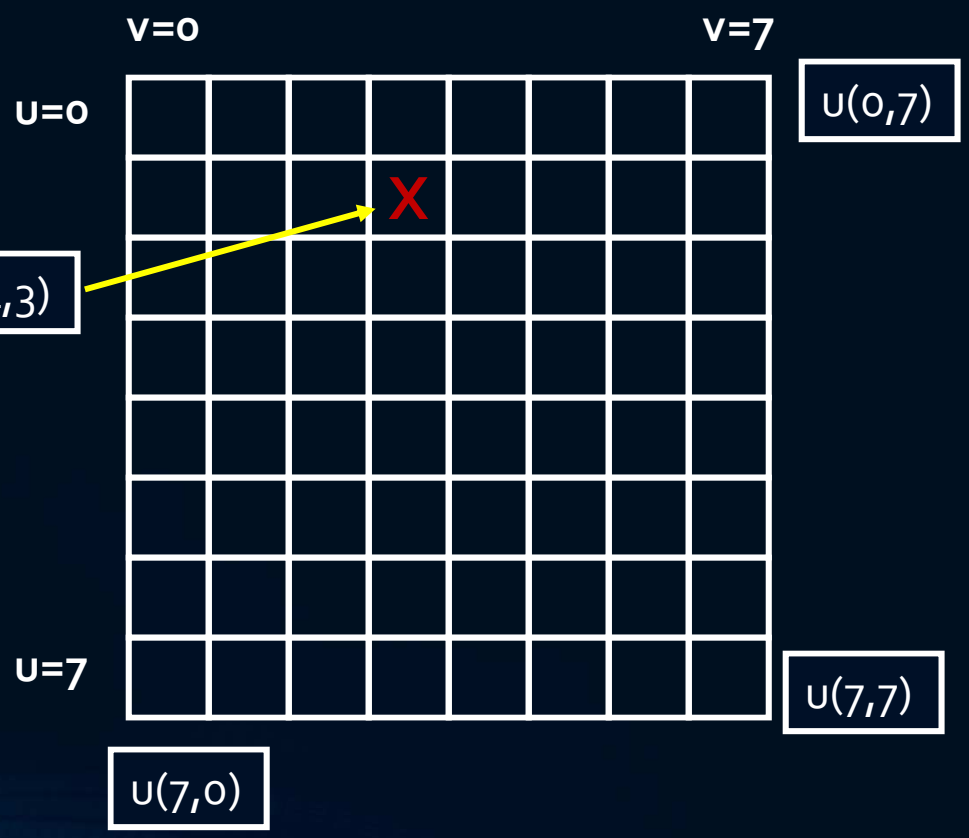
Discrete Cosine Transform

$$b(1,3) = \frac{2}{N} C(1)C(3) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} a(x,y) \cos\left(\frac{\pi 1(2x+1)}{2N}\right) \cos\left(\frac{\pi 3(2y+1)}{2N}\right)$$

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

- arbitrarily take the case u=1, v=3
- do another 64 summations
- this time the cosines will have values other than one
- repeat the calculation for all combinations of u and v

That's a LOT of slow cosine calculations!



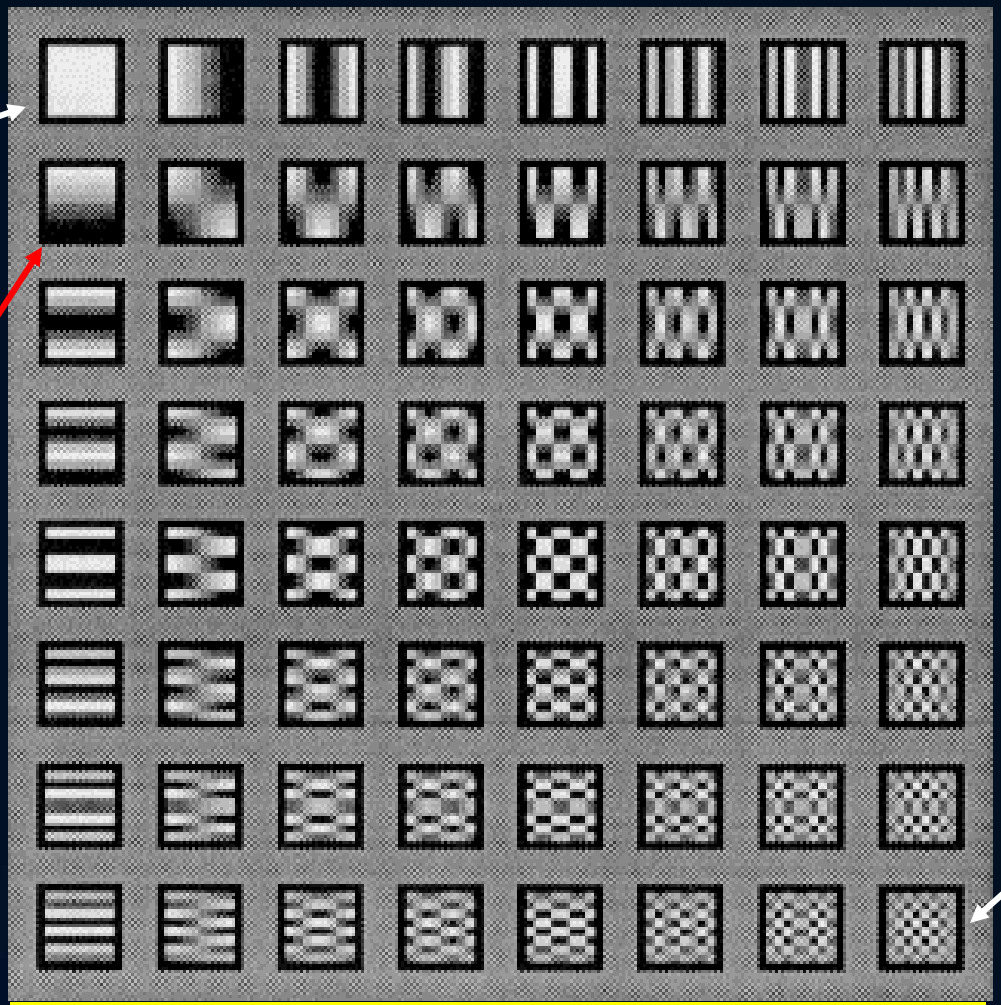
Discrete Cosine Transform

- Improve performance by pre-calculating the *basis functions*
- The *basis* is the cosine portion of the formula
  - It is unaffected by pixel values, only position in the matrix
- Calculate the 8 x 8 basis matrix once for all valid u, v pairs (0, 0; 0, 1; … 0, 7 --- 1, 0; 1,1; … 7, 7)
- For u=0, v=0, the result is one
- For v=0, the value of y is irrelevant
  - These are the horizontal frequencies
- For u=0, the value of x is irrelevant
  - These are the vertical frequencies

Discrete Cosine Transform

u = 0, v = 0

u = 0, v = 7



- lowest frequencies
- DC is lowest
- shown next

- +1 is white
- -1 is black
- 0 is gray

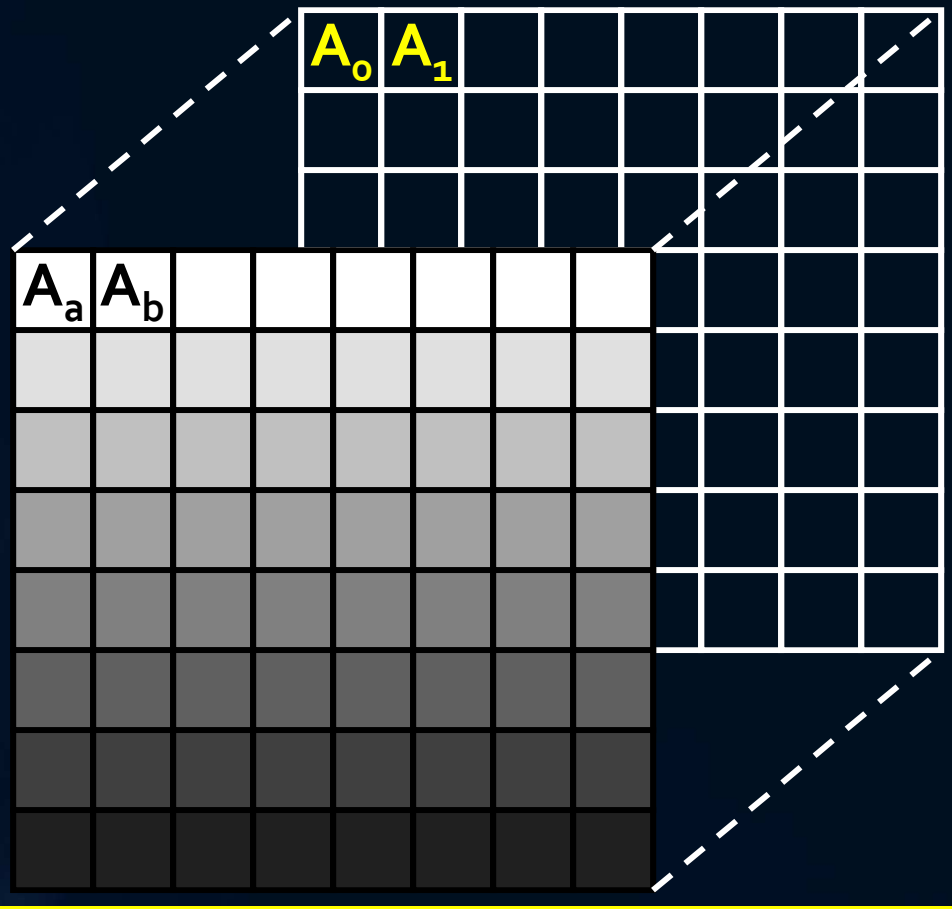
- highest frequencies

u = 7, v = 0

u = 7, v = 7

$$basis = \cos\left(\frac{\pi u(2x+1)}{2N}\right)\cos\left(\frac{\pi v(2y+1)}{2N}\right)$$

Discrete Cosine Transform



- 8 X 8 block from image
- A_0 = a(0,0)
- A_0 is multiplied by A_a
- A_1 is multiplied by A_b
- etc.

- basis matrix for u=1, v=0

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

Discrete Cosine Transform

- The top row of basis images
  - vertical frequencies from low (0,1) to high (0,7)
- The left column of basis images
  - horizontal frequencies from low to high
- The middle basis images
  - combinations of both vertical and horizontal frequencies
- The basis images in the top left corner
  - lowest frequencies (including the DC frequency)
- The basis image in the bottom right corner
  - the highest frequencies

Discrete Cosine Transform

- The DCT coefficients in the upper left corner representing the lower frequencies are typically higher in magnitude
  - The lower frequencies contain most of the image information
- The lower right DCT coefficients often become zero
  - This is OK since these frequencies contain less information

Discrete Cosine Transform

- The inverse DCT equation is used to reverse the process

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

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

- It is basically the same except:
  - b(u,v) is inside the summation
  - C(u), C(v) must be inside the summation because they change as u and v change

JPEG Algorithm - Quantization

- The quantization tables determine the "lossiness"
- They are altered for different levels of quality

a. Low compression

1	1	1	1	1	2	2	4
1	1	1	1	1	2	2	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

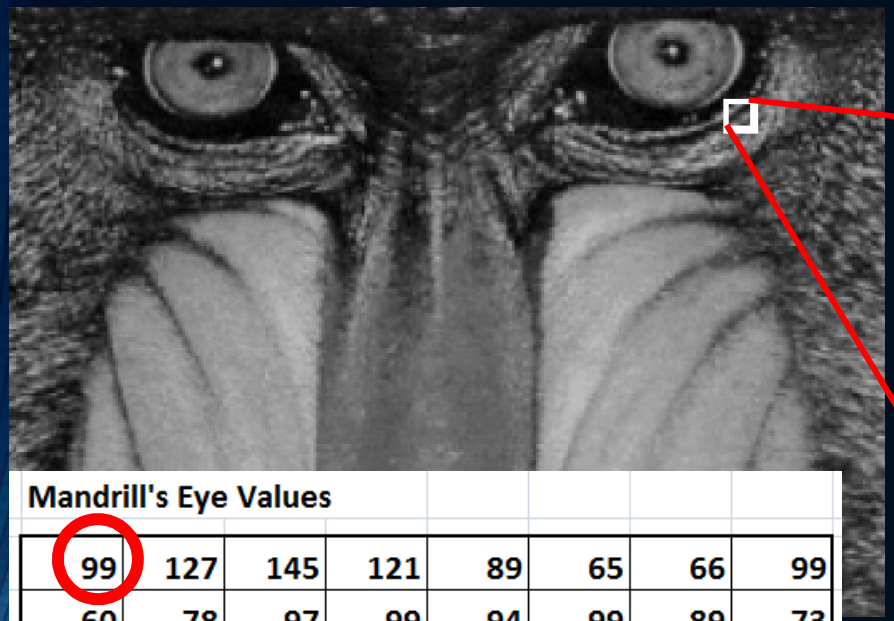
b. High compression

1	2	4	8	16	32	64	128
2	4	4	8	16	32	64	128
4	4	8	16	32	64	128	128
8	8	16	32	64	128	128	256
16	16	32	64	128	128	256	256
32	32	64	128	128	256	256	256
64	64	128	128	256	256	256	256
128	128	128	256	256	256	256	256

- See how high compression will result in a lot of zeros?

JPEG Algorithm - Quantization

- Removed 8x8 block from Mandrill's eye



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 - Quantization

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 - Quantization

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 - Quantization

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

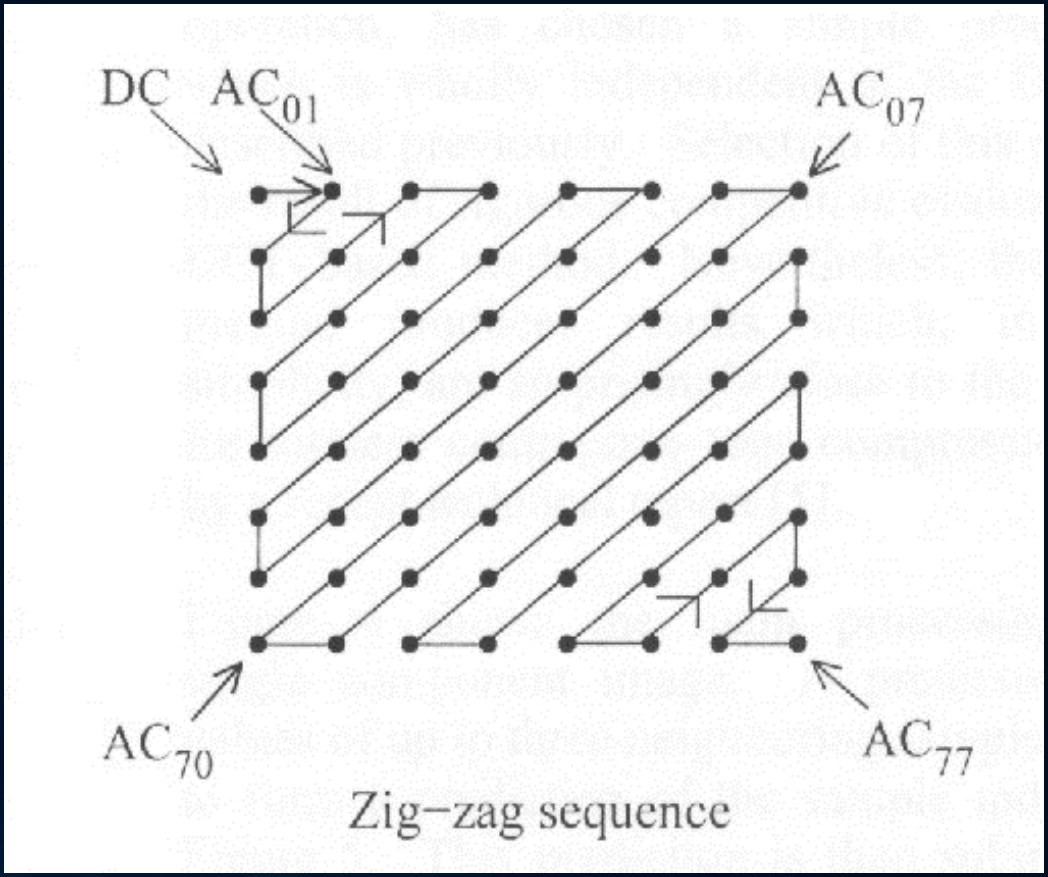
95	129	147	122	84	70	78	88
55	79	98	97	90	90	87	82
32	38	49	69	98	117	111	95
67	50	38	52	88	119	126	119
125	94	59	46	57	79	98	109
139	116	86	60	41	35	49	66
92	96	103	97	69	37	26	33
37	66	110	133	112	65	33	26

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 – Run Length Coding

- Uses a zig-zag pattern to maximize runs of zeros
- Lower frequencies are grouped together



JPEG Algorithm – Entropy Coding

- Entropy coding is the final, lossless stage of coding
  - Includes run-length coding and Huffman/Arithmetic algorithms
- The final matrix has a lot of zeros
- The Run-Length coding is then compressed using a Huffman (or Arithmetic) code
- The DC term is coded differently because of its size
  - The difference from the last DC term is encoded
- Repeat for each 8x8 block
- For a 512x512 image, that is 64x64 blocks!

JPEG Algorithm

- Seems like a LOT of calculations!
- 4096 per block! Plus the color conversion
- And then that is done for each of the 3 color planes!
- I always get asked, "How long does this take?"
- When you open a jpeg file, how long does it take to be seen on your screen?
- 3.6 GHz is fast!

JPEG Hiding Technique #1 – Swap DCT

- Choose two DCT coefficients which have the same value in the quantization table
  - Select middle frequencies so hidden bits are in significant 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 calculated coefficients, NOT the Q-Table values

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

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 zero, then $C_1 < C_2$ must be true
  - If the bit is a one, then $C_2 < C_1$ must be true
- If this condition is already true, then continue to the next block and message bit
- 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!

Swap DCT

- Our Q-Table value is 14, so a coefficients with the values 7 to 20 will all quantize to 1
  - Modify the coefficient values slightly to make sure the difference is large enough
  - You can add to one and/or subtract from the other

Swap DCT

- Weaknesses in this approach
  - No attempt is made to determine if a particular cover block is a good/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
  - What do you think would be the downside of that?

Swap DCT

- Weaknesses in this approach
  - No attempt is made to determine if a particular cover block is a good/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
  - What do you think would be the downside of that?
  - Increased perceptibility and detectability

JPEG Hiding Technique #2 Improved Swap DCT

- 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 m=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 m=0
  - $C_1 + D < C_3$ and $C_2 + D < C_3$
- Middle frequencies are selected

Improved Swap DCT

- 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	101
72	92	95	98	112	100	103	99

JPEG Hiding Technique #3 - 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 1, make C1 > C2
- If message bit is 0, make C2 > C1
- If the two de-quantized values are equal, skip
  - Could modify them, but that increases detectability

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 impact, but often zero, so less capacity

High Capacity - Swap DCT

- In implementation, the number of matching pairs used is an option
  - 1 - 31 pairs
  - Quality=100, pairs = 24, 14587 (6.52%) random bytes hidden
  - 10764 blocks, 10.84 bits/block – much better than 1

High Capacity - Swap DCT

- Using max of 31 coefficients to swap
  - Quality=100, pairs = 31, 22291 (9.95%) random bytes embedded
  - 10764 blocks, 16.57 bits/block

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 Technique #4

- "A Method of Embedding Binary Data into JPEG Bitstreams"
  - Kobayashi et al.
- 1 bit per 8x8 block
- Uses high-frequency components for cover
  - since these coefficients often become zero, if m=0, no need to change anything
  - high frequency coefficients are more resistant to noise – lower perceptibility
- Modifies quantization table such
- that highest frequency is not lost
- Capacity is 1 bit/block
- Always uses same coefficient
- Compression ratio slightly affected

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

JPEG Hiding Technique #5 – DCT LSB

- JSteg uses this approach
- Alter the LSB of each quantized DCT coefficient to hold our message
- 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 coefficients 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 Technique #5 – 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 Technique #6 - Outguess

- Outguess is available for free download
- Hides in LSBs of DCT coefficients
- Pseudo-randomly permutes selection DCT coefficients that are not 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 Technique #7 – 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 Technique #7 – Average DCT

- As number of coefficients for each average goes up, number of bits 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 Technique #8 – 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 Technique #8 – 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 permutatative 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 encodes 2 bits
    - $x_1 = a_1 \text{ xor } a_2$; $x_2 = a_2 \text{ xor } a_3$ --- change nothing
    - $x_1 != a_1 \text{ xor } a_2$; $x_2 = a_2 \text{ xor } a_3$ --- change a_1
    - $x_1 = a_1 \text{ xor } a_2$; $x_2 != a_2 \text{ xor } a_3$ --- change a_3
    - $x_1 != a_1 \text{ xor } a_2$; $x_2 != a_2 \text{ xor } a_3$ --- change a_2

JPEG Hiding Technique #9 Statistically Invisible Steganography

- 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 Technique #10 - YASS

- Yet Another Steganographic Scheme that resists blind steganalysis
- What YASS does a little differently is to select blocks larger than 8x8
  - Example: 10 x 10
  - Has 9 possible sub-blocks
- Out of the larger block, YASS selects an 8x8 block, performs the DCT 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 Technique #11 – High Capacity

- "High Capacity Data Hiding in JPEG Compressed Images"
  - Chang, C.C. and Tseng, Hsien-Wen
  - Greater capacity than 1 bit per 8x8 block
- It is an adaptive DCT LSB technique
  - Hides mostly in lower and middle frequency components
  - Able to perform a capacity estimation
  - Adapts to different characteristics of each block

High Capacity Hiding in Jpeg

- 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
- Apply the normal JPEG entropy coding
- Repeat

High Capacity Hiding in Jpeg

- Determine which blocks can contain more hidden information while remaining imperceptible
- If a background has a strong texture, the Human Visual System (HVS) is less sensitive to distortions
- Blocks are divided into two classes:
  - uniform blocks (smaller α)
  - non-uniform blocks
- Non-uniform blocks can use a larger α value
  - X * α where X is between 1.0 and 9.9
    - There is a mathematical limit as to when a larger α will matter
    - Experimentally I have found diminishing returns around 4
    - The DEMO program allows you to select an α value

High Capacity Hiding in Jpeg

- The equation below calculates the energy intensity of the AC DCT coefficients
  - D_x is the x^{th} AC coefficient
  - The DC coefficient is not considered
  - If most of the coefficients are zero, this result will be low
  - This is the common case and the block is uniform
- If G is below a threshold, the block is uniform
  - I used a threshold of 100

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

High Capacity Hiding in Jpeg

- Determine max number of bits in the DCT coefficient that can be modified while remaining imperceptible
- Uses a capacity table 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
  - Lower frequency components hold fewer bits
  - Higher frequency can hold more bits, but there are fewer

High Capacity Hiding in Jpeg

- Q = Quantization table
- D = Quantized DCT coefficient value
- C_Q = capacity table based upon quantization table
- M = max bits based upon DCT coefficient value
- E = number of bits that can be embedded
- For each Q table element
  - $C_Q(x, y) = \lg(\alpha * Q(x, y))$
  - [lg is the log, base 2 → $\lg X = \log_2 X = \log X / \log 2$]
- For each DCT coefficient where D < -1 *OR* D > 1
  - $M(x,y) = \lg(|D(x,y)|)$
- E is the minimum of C_Q and M

High Capacity Hiding in Jpeg

- Given the standard quantization table and the quantized DCT coefficients from the Mandrill's eye
  - @ 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

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

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

High Capacity Hiding in Jpeg

- 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 high
- Uses a slightly different capacity estimate
  - $C_Q(0,0) = \lg(\alpha * Q(0,0)*2/\lg(\text{quality factor}))$
  - Was determined experimentally
  - Will reduce the number of bits hidden in DC coefficient

High Capacity Hiding in Jpeg

- Note that negative coefficients must be adjusted
  - A -2 (1110_2) would allow a bit to be hidden
  - Changing the LSB to 1 results in a value of -1!!!
  - The extractor can't tell the difference between an ignored, "-1" and a modified "-2"
  - So -3 is the minimum negative value that can be used for hiding
- Note 2: I decided to use -1's (and -2's) in my implementation – works just fine
  - Just can't ignore -1 and use -2

High Capacity Hiding in Jpeg

- 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
  - 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

High Capacity Hiding in Jpeg

- A modified Q table is a HUGE flag indicating some type of manipulation
- I don't like it
- During implementation, my solution was to limit the number of bits to be hidden to 4
  - During experimentation, 5 would rarely occur anyway, so minimal impact on capacity
  - Never saw a 6 …
    - Q table value would have to be at least 64
    - DCT coefficient would also have to be 64 after dividing by 64 (4096)
- Only used the upper 4 bits of the coefficients to calculate G

High Capacity Hiding in Jpeg

- Chang et al. did some statistical analysis
- Used a Signal to Noise Ratio to determine stego-image quality
  - Signal in this case is the original image
  - Noise is the stego image
- Mean Squared Error (MSE)
- PSNR = Peak Signal to Noise Ratio (higher is better)
- N*M image dimensions, p is original pixel value, q is decoded pixel value

$$MSE = \left(\frac{1}{N*M}\right) * \sum_{x=1}^{N}\sum_{y=1}^{M}(p[x,y]-q[x,y])^2$$

$$PSNR = 10*\log_{10} 255^2 / MSE$$

High Capacity Hiding in Jpeg

Mandrill512.bmp_q95_a8_u8.jpg ---> 71745/ 322564 22.24% of stego

High Capacity Hiding in Jpeg

Domino512.bmp_q95_a8_u8.jpg ---> 38827 / 175915 22.07% of stego

High Capacity Hiding in Jpeg

S2_Rocky.jpg_q99_a8_u8.jpg ---> 107643 / 511089 21.06% of stego

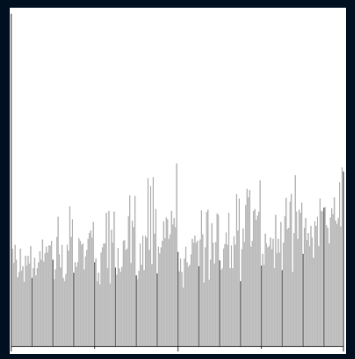
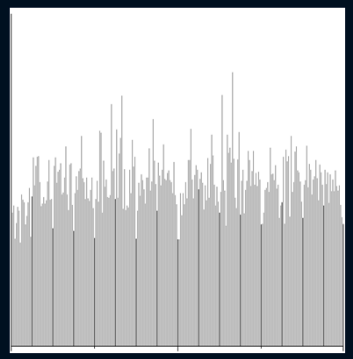
High Capacity Hiding in Jpeg

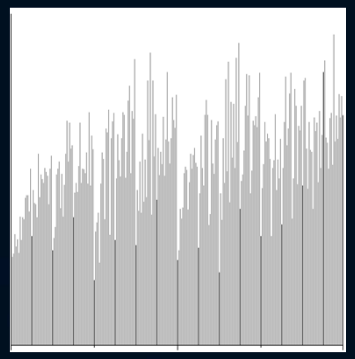
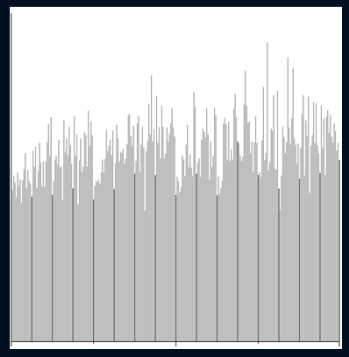
S2_Rocky.jpg_q50_a8_u8.jpg ---> 6612/ 37437 17.66% of stego

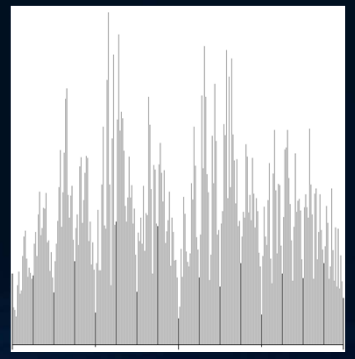
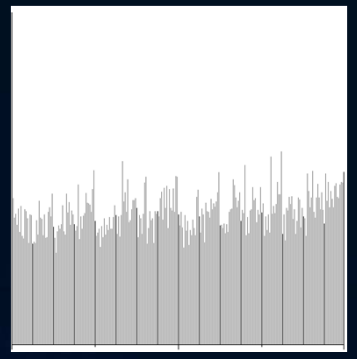
High Capacity Hiding in Jpeg

S2_Rocky_A.jpg_q50_a8_u8.jpg ---> 1575/ 6612 18.81% of stego

Histograms are NOT Effective

Detection

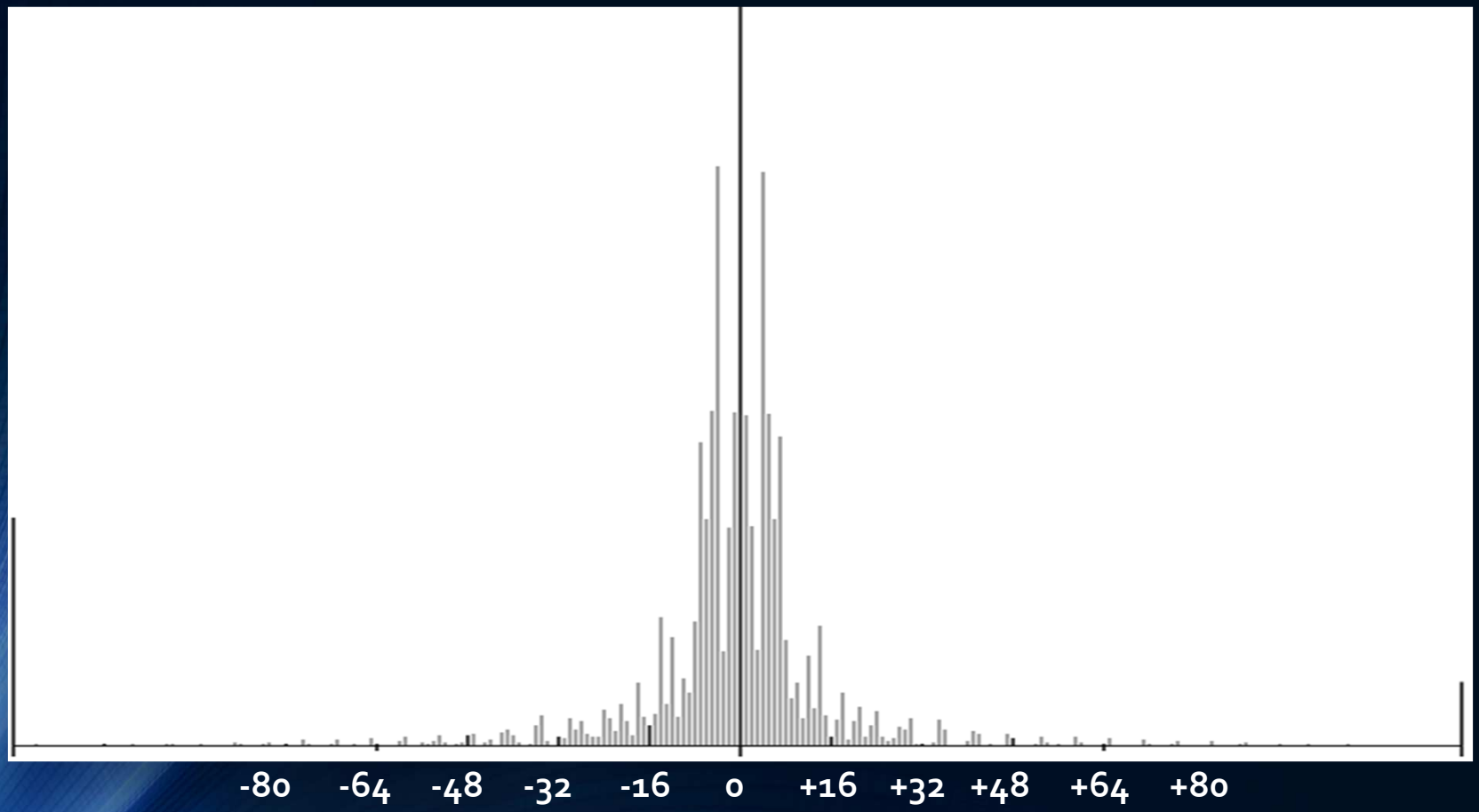
- The following example illustrates how modification of the DCT coefficients can be detected
- In their unmodified state, the count of coefficients tend to be symmetrical about zero
  - The number of +1 values is roughly equal to the number of -1 values
  - The number of +2 values is approximately the same as the number of -2 values
  - The number of +3 values …

Detection

Storage Capacity: 140500 bytes --> 26.01% Total File Size

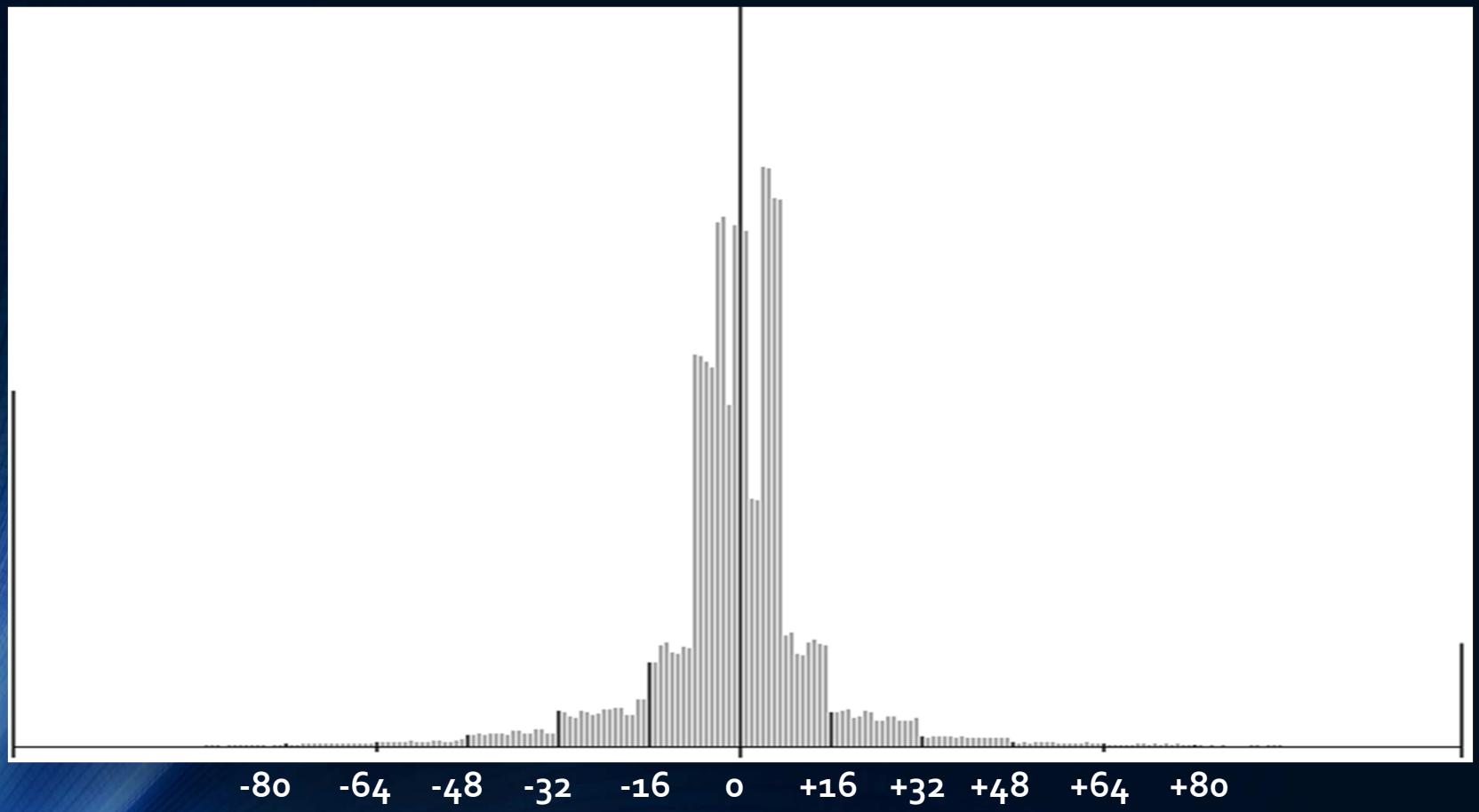
Detection

- Histogram of DCT Coefficients – note the symmetry



Detection

- DCT Coefficients after embedding 26% random data



Detection

- WHY do the DCT coefficient values become non-symmetrical?
- When we change a +2 it becomes a +3
  - 00000010_2 → alter LSB → 00000011_2
- And when we change a +3, it becomes a +2
- BUT, when we change a -2, it becomes -1
  - 11111110_2 → alter LSB → 11111111_2
- And when we change a -1, it becomes -2
- So if the number of changes to these coefficients is balanced (i.e. randomized or encrypted data), the +/- balance is destroyed

Detection

- We generally do not use zero for hiding because it would negate a large component of the compression
- PLUS, an unusually small number of ZEROs would be an indication!
- So we cannot use a +1 either, because a change in the LSB results in ZERO
  - The decoder can not tell the difference between an actual zero and a one that was altered to a zero

Detection

- Since +1's are not changing, but -1's are, they become unbalanced too
- For positive numbers, 2's and 3's swap
  - 4's/5's, 6's/7's, etc.
- For negative numbers, it's -2's and -1's
  - -4's/-3's, -6's/-5's, etc.

Questions & Comments

References

- "Embedding Robust Labels into Images for Copyright Protection", Zhao, Koch
- "A Method of Embedding Binary Data into JPEG Bitstreams", Kobayashi et al.
- "High Capacity Data Hiding in JPEG Compressed Images", Chang, C.C. and Tseng, Hsien-Wen
- **"**A JPEG-Based Statistically Invisible Steganography**",** Qingzhong Liu, Andrew H. Sung, Zhongxue Chen, Xudong Huang
- "F5 - A Steganographic Algorithm - High Capacity Despite Better Steganalysis", Andreas Westfeld, Technische Universit¨at Dresden
- Compressed Image File Formats, JPEG, PNG, GIF, XBM, BMP, John Miano, Addison Wesley
- "Defending Against Statistical Steganalysis", Niels Provos