

## **Introduction to More Advanced Steganography**

**John Ortiz**

**Senior Computer Engineer  
San Antonio, TX**

**[John.Ortiz@Harris.com](mailto:John.Ortiz@Harris.com)**

# Public Service Announcement

## Please complete the Speaker Feedback Surveys

**This will help speakers to improve and for Black Hat to make better decisions regarding content and presenters for future events**

# Can YOU See the Difference?

- Which one of these pictures has a secret message?



- One of them contains 205,700 bytes of secret data.
- The file is 1,027,303 bytes
- That's 21.00 % data hiding capacity



# A Closer Look – Picture #1





## A Closer Look – Picture #2





# What's Hiding in There?



# Agenda

- **Attention Getter**
- **A little bit about me**
- **Overview – Introduction to Steganography**
- **Hiding in the Least Significant Bit**
- **Advanced Techniques for Geeks**
  - **Bit Plane Complexity Segmentation (BPCS)**
  - **Hiding in Compressed Jpeg Images**
  - **Hiding in the Silence of Sound**
  - **Hiding in uncompressed AVI**
- **Questions/Comments/Complaints**



# About Me

- Graduated from Rose-Hulman Institute of Technology in 1988 (BSEE)
- Joined the U.S. Air Force as a Communications Officer
  - Varied tasks from database automation, leading a comm unit, teaching computer networks course, planning unit picnics
- Graduated from the Air Force Institute of Technology in 1997 with 2 master's degrees (MSEE, MSCE)
- Joined Trident/Veridian/General Dynamics in 2000 to work on computer security programs for DoD clients
  - C source code analysis
  - Reverse engineering tasks
  - Computer security software
- Joined RABA/SRA in 2005
  - Analyzed DoD security programs
  - Researched steganography
  - Make steganography presentations to college students



# More About Me

- **Joined Crucial Security Inc. in 2010**
  - PDF Signatures
  - Flash decompilation
  - Packet stream analysis and file extraction
- **Started teaching at University of Texas at San Antonio (UTSA) in 2003**
  - Various computer science and electrical engineering classes
  - Developed and taught steganography course since 2004

# Enough About Me

## Overview



# What is Information Hiding?

- **Information Hiding is a branch of computer science that deals with concealing the existence of a message**
  - It is related to cryptography whose intent is to render messages unreadable, except by the intended recipients
- **It employs technologies from numerous science disciplines:**
  - Digital Signal Processing (Images, Audio, Video)
  - Cryptography
  - Information Theory\Coding Theory
  - Data Compression
  - Discrete Math
  - Data Networks
  - Human Visual/Auditory perception

# What is Steganography

- **There are four primary sub-disciplines of Information Hiding**
  - **Steganography**
  - **Watermarking**
  - **Covert Channels**
  - **Anonymity**

# Goals of Steganography

- Steganography's primary goal is to hide data within some other data such that the hidden data cannot be detected even if it is being sought



# Goals of Steganography

- **Security**
- **Capacity**
- **Robustness**

# Goals - Security

- It is secure if it cannot be detected, even if the attacker has full knowledge of the embedding algorithm
  - Only the knowledge of the secret key should allow the detection
- Can it be seen or heard?
  - Imperceptible to humans
- Can it be detected by statistical analysis?
  - Imperceptible to computers
- Does it leave easily detectable signatures?

# Goals - Security

- **Levels of Failure:**
  - Detection - Proof of existence of message
  - Extraction – removing without destroying the cover
  - Destruction – destroying the message without destroying the cover
- **Rating Perceptibility**
  - Indistinguishable from original
  - Sense distortion when looking/listening for it
  - Blatantly obvious to the most casual observer



# Goals - Capacity

- How much data can a cover image hold?
  - Must consider if a change in file size is noticeable
- Maximize amount of hidden data for a given cover file size
- As more data is hidden, the effects become more noticeable
  - Direct tradeoff between capacity and security

# Goals - Robustness

- How well does the data maintain integrity in the face of modifications?
- Common modifications include the following:
  - Images: blurring, sharpening, scaling, cropping, contrast, gamma, brightness, rotation, skewing, printing/copying/scanning, etc.
  - Audio: filtering - bass/treble, volume adjustment, stereo to mono, etc.
  - Video: any image/audio hiding, add/delete frames, temporal adjustments, frame swapping, frame averaging
  - Also: lossy compression, A/D and D/A conversion (scan, print, fax), as well as sophisticated attacks

# Goals - Robustness

- Robustness is achieved through redundant encoding of the message
  - Direct tradeoff between capacity and robustness
  - Redundant bits used for robustness could be used for capacity instead
  - Robustness is primary goal of watermarking

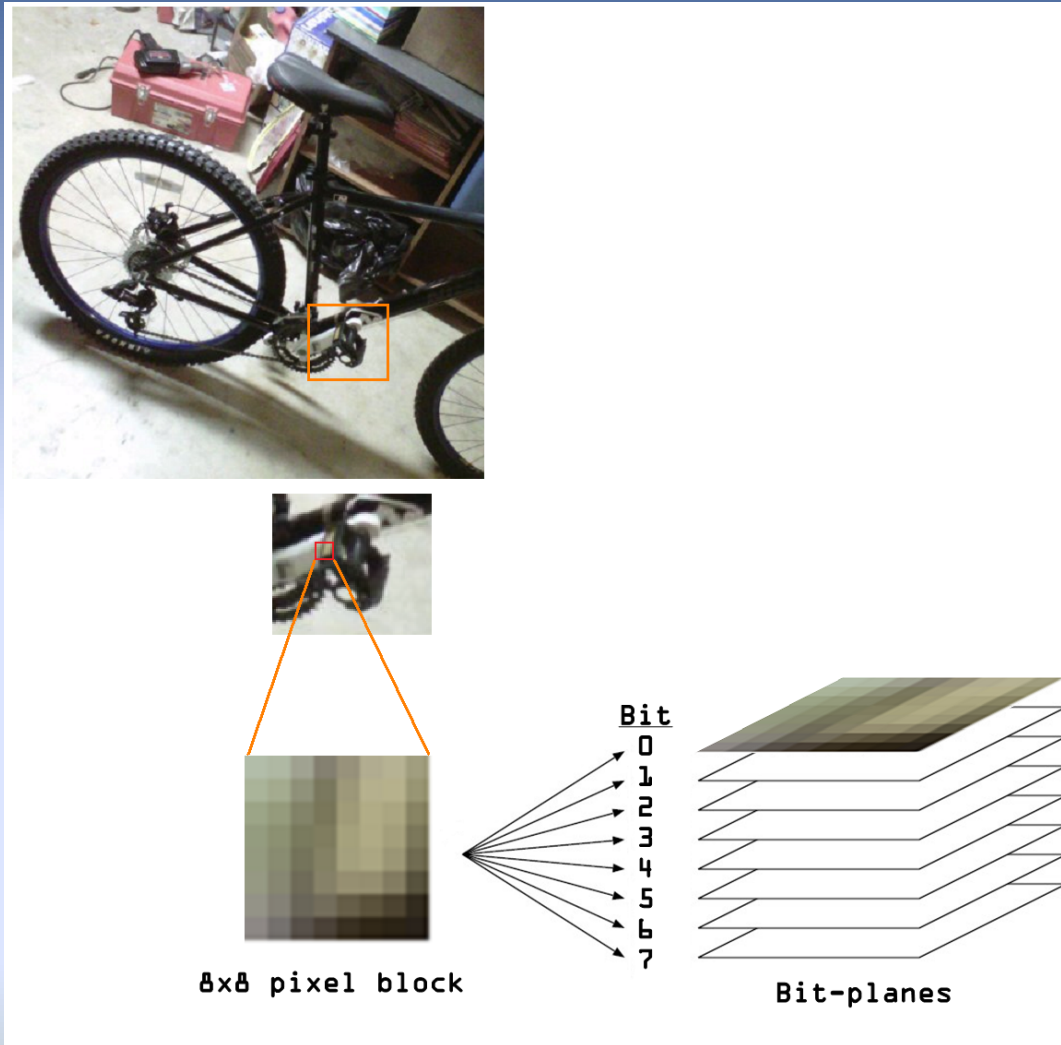
# Less Advanced Steganography

## Basic Hiding

# Least Significant Bits

- **Substitution**: Replace information in the cover with the stego-message
- Most common: replace the Least Significant Bit (LSB) with your message data
- Each pixel in the next image is composed of 24 bits
  - 8 bits for RED, 8 for GREEN, and 8 for BLUE (RGB)
  - Lower four bits of each color, hold the upper 4 bits of the hidden picture's colors in each corresponding pixel

# Bit Planes





# We're Gonna Hide This:



# In This:



# Then That In This:





# Can YOU See a Difference?

- The Dalmatian is hiding in 4 bits of the Mandrill



# Least Significant Bits

- Other images with more solid backgrounds **DO NOT** provide the same level of imperceptibility
  - To maximize capacity while maintaining imperceptibility, the cover image is a consideration



# You CAN See a Difference!

- More uniform colors in the cover is NOT as effective





# Limitations

4



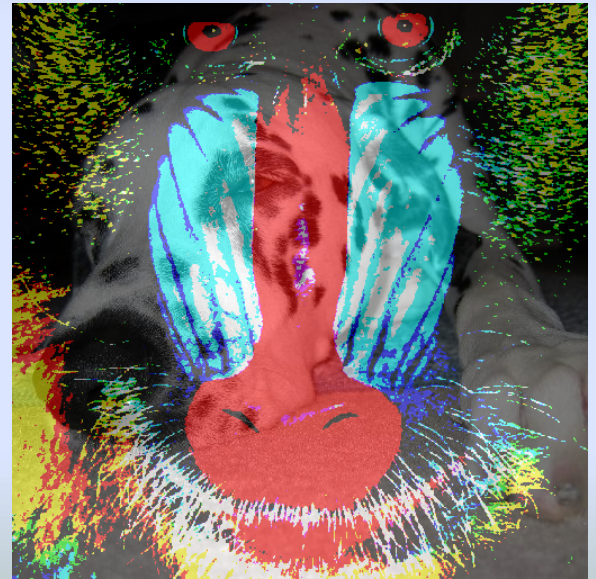
5



6



7





# Limitations

4



5



6



7



# Least Significant Bits

- This particular technique substitutes image bits of one picture into another
- Both pictures must be the same size
- More typical is to substitute bits from the message one by one
  - The message can be anything



# When Message is Encrypted

4



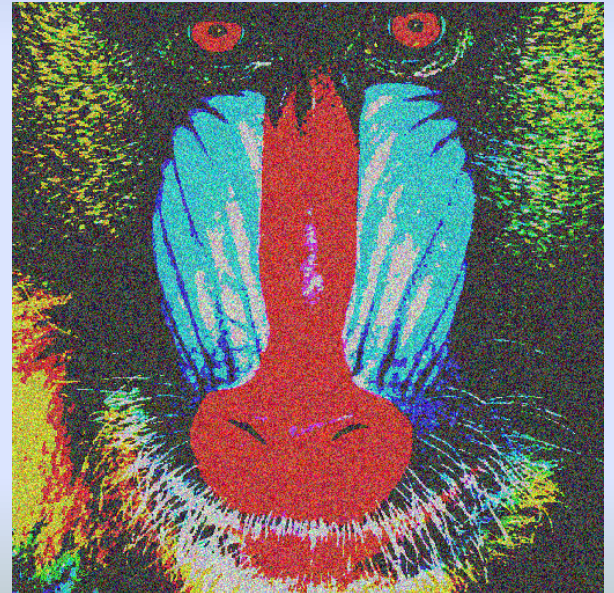
5



6



7

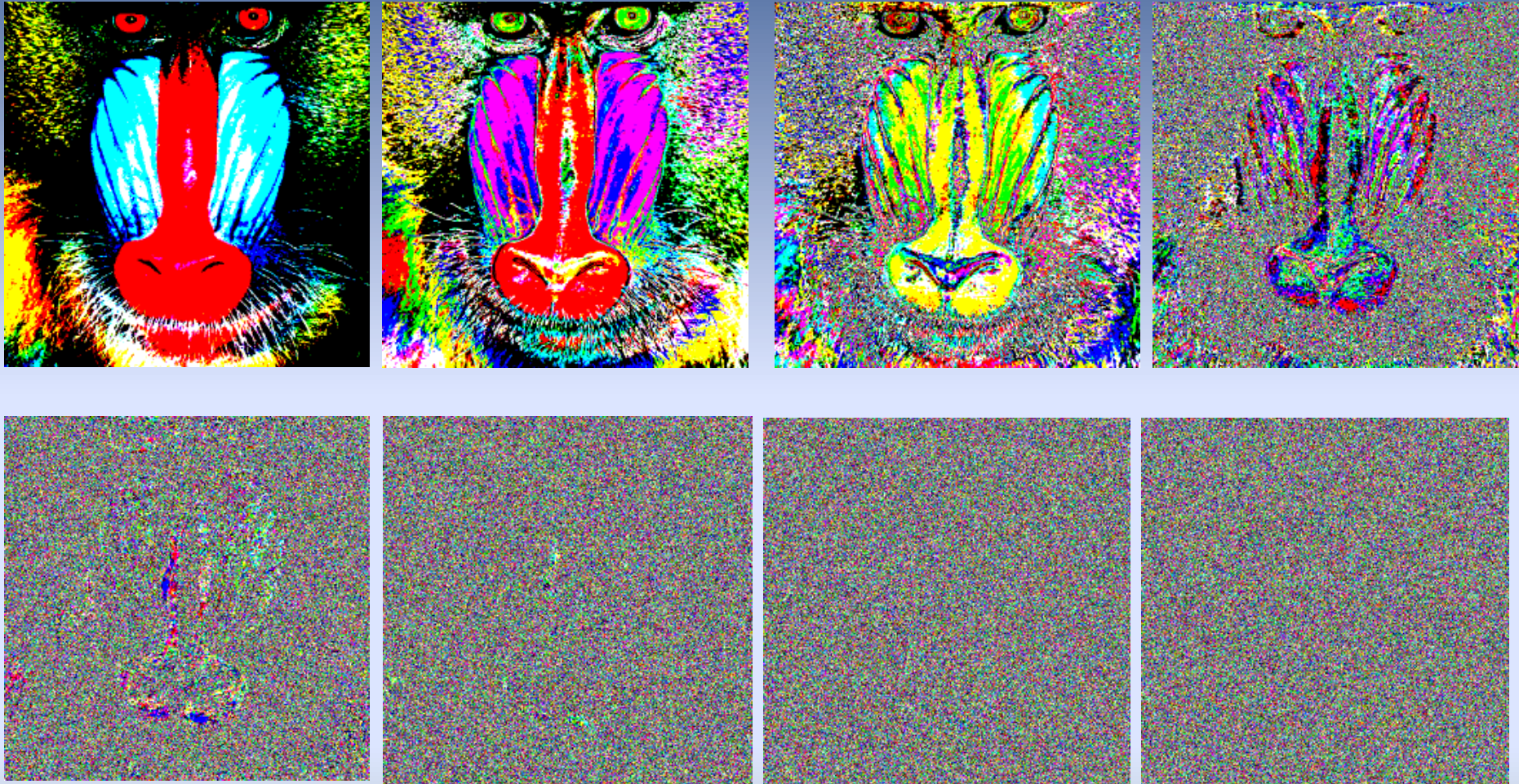


# Least Significant Bits

- **Easily detectable**
  - Slice the image into bit planes
  - Examine the histograms for anomalies

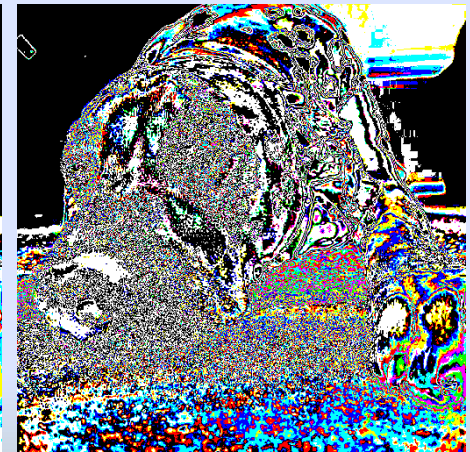
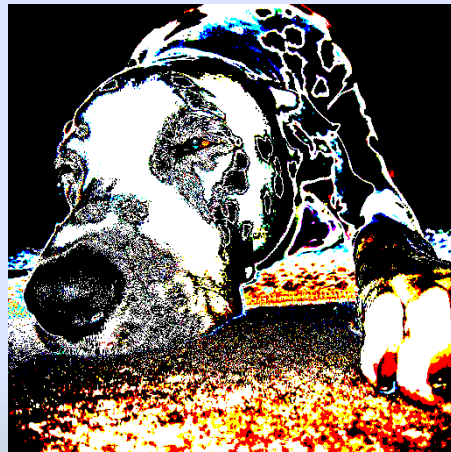


# Sliced into Bit Planes



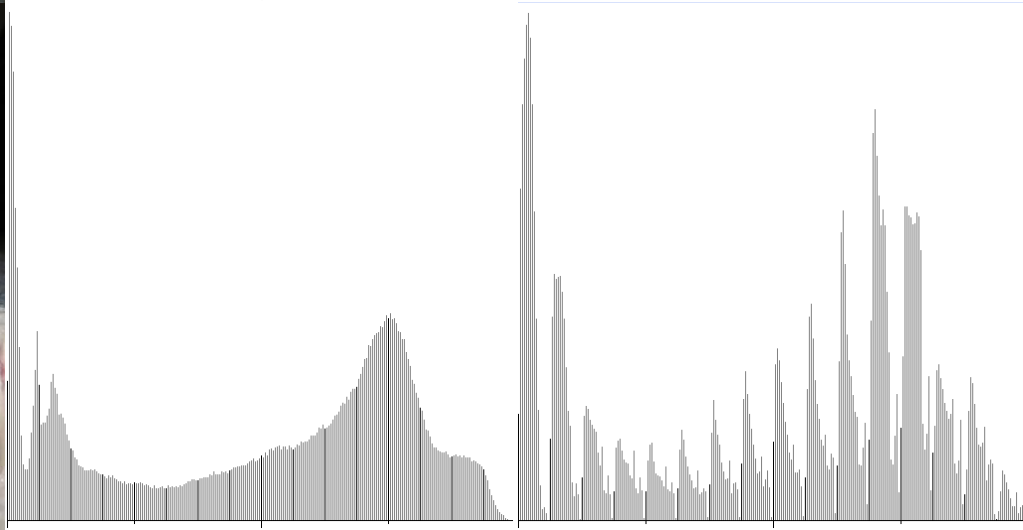
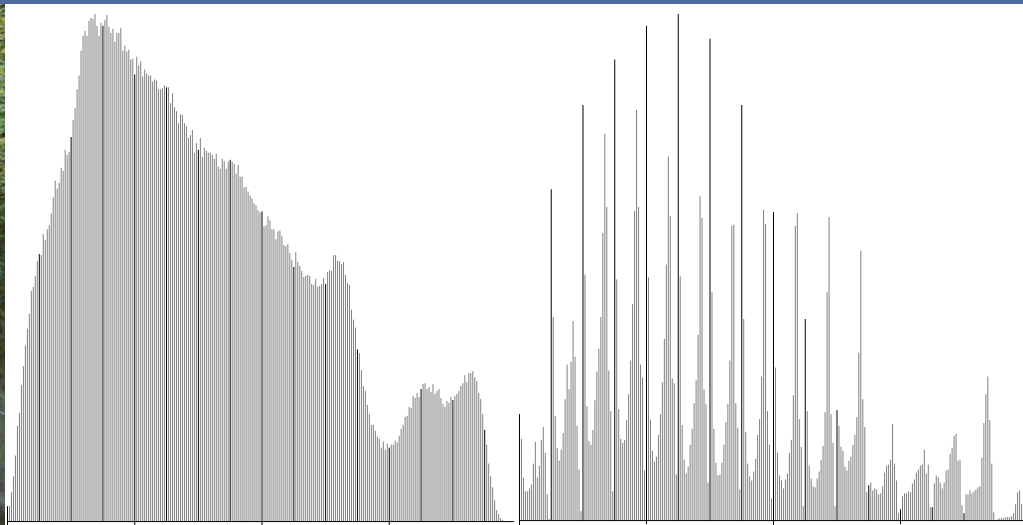
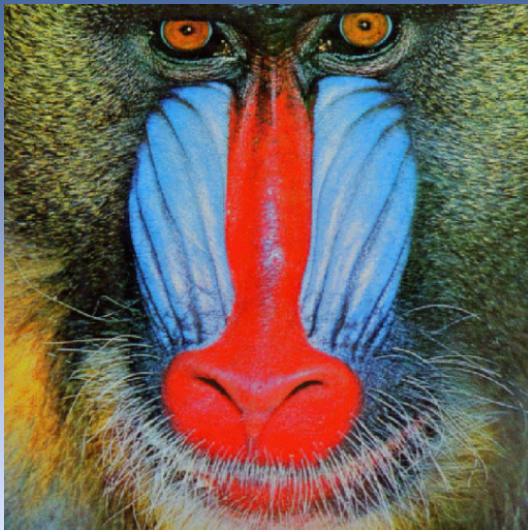


# Sliced into Bit Planes





# Least Significant Bits - Histograms



# More Advanced Steganography

## **Bit-Plane Complexity Segmentation (BPCS)**

# Bit Plane Complexity Segmentation

- More advanced **Substitution** Technique
- Little less capacity
  - Harder to detect
  - Harder to extract
- Message is spread across several bit planes
  - Possibly even the Most Significant Bit (MSB) plane

# Bit Plane Complexity Segmentation

- Hides in areas of image that are “complex”
  - The mandrill has a large number of complex areas
  - The dalmatian has much fewer complex areas
- The Least Significant bit plane is complex for both

MSB



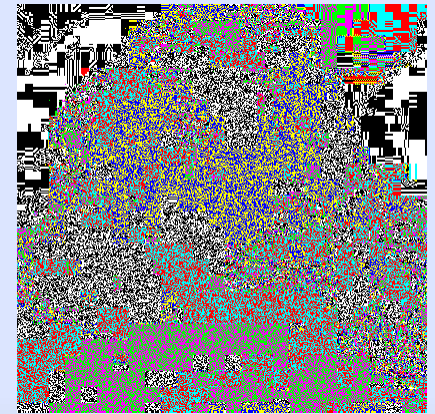
LSB




MSB



LSB



# Bit Plane Complexity Segmentation

- BPCS requires Canonical Gray Coding (CGC) for pixel values in order to accurately measure complexity
  - A gray code is one in which there is only a 1-bit difference between successive values
- 3-bit Gray Code example:
  - 000, 001, 011, 010, 110, 111, 101, 100
- Easy to construct in software



# Bit Plane Complexity Segmentation

- The numbers 127 and 128 are just 1 value apart
  - To the human looking at a region with alternating values of 127 and 128, it looks smooth
  - To the computer checking the MSB bit plane, it is complex
- Look at their bit representations:
  - 127: 0111 1111
  - 128: 1000 0000
- Using CGC, these two values only differ by one bit

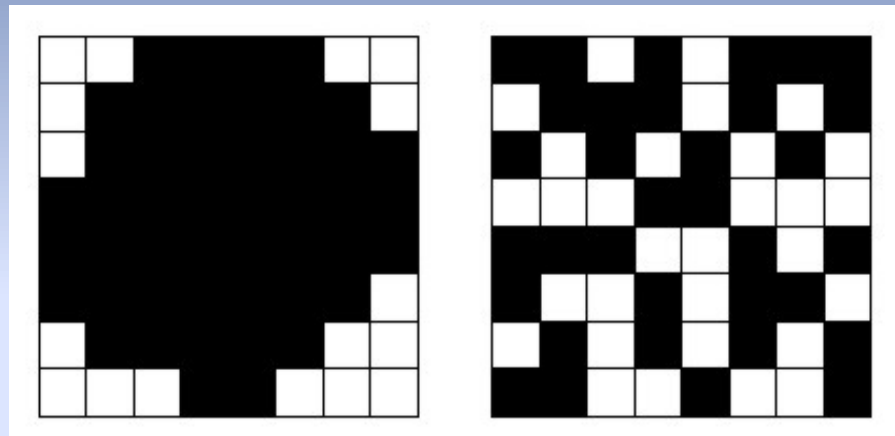
# Bit Plane Complexity Segmentation

- A complexity measure is taken for each 8 x 8 matrix
  - No standard complexity measure
- The one used in the initial paper is a black and white border length complexity measurement
  - If the border length is long, the image is complex
- The total length of the border is the sum of the number of black/white changes along the rows and columns
  - Remember, we are using the measure on bit planes, so every pixel is either a one or zero
  - Max = 112 for a checkerboard pattern
  - Ex. A black pixel, surrounded by all white, has a border length of 4

# Bit Plane Complexity Segmentation

- Left: A simple block with low complexity
- Right: A complex block

$$\alpha = \frac{k}{M}$$



- $\alpha_{th}$  is the threshold
  - Border length over total
  - Experimentally determined to be  $\sim 0.3$
  - Must be less than 0.5 (we'll see why shortly)

# Bit Plane Complexity Segmentation

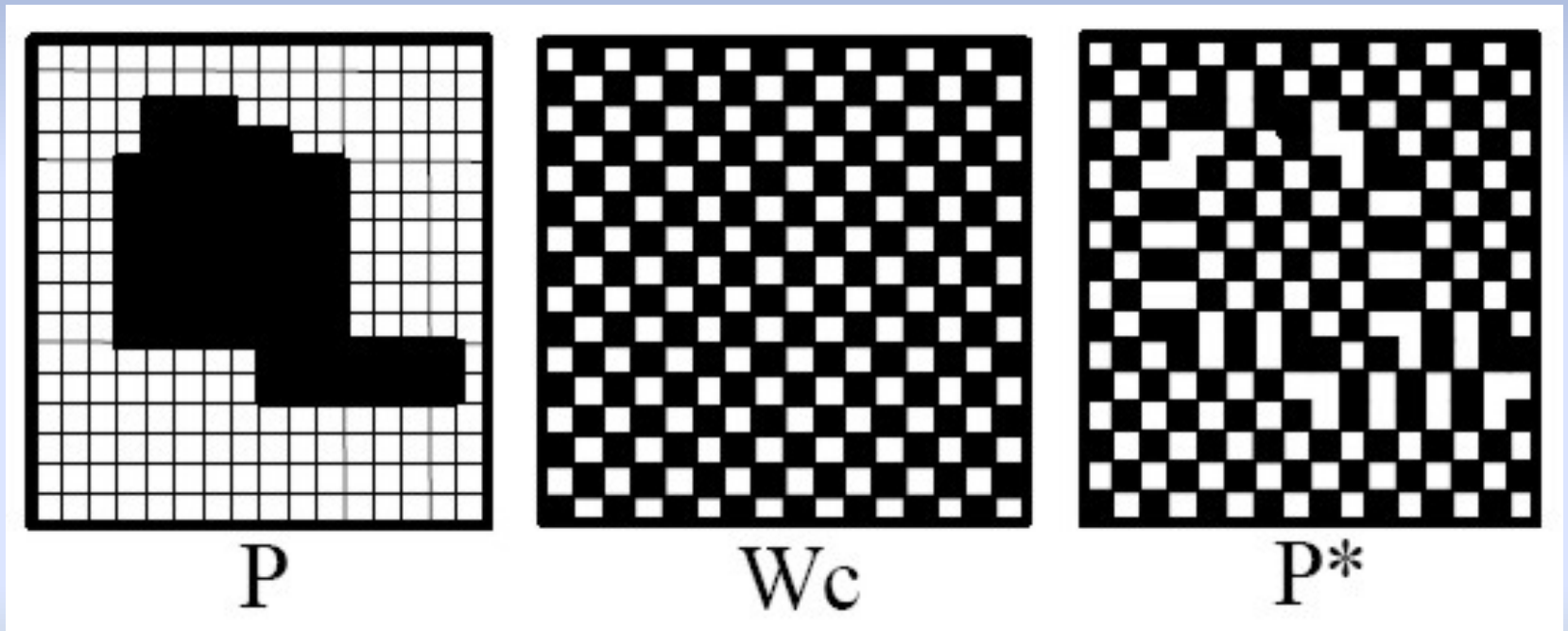
- If a region is complex enough, the image data is replaced by the message data
- The message data is first transformed into an 8x8 bit array, and that array is stored in place of the original data
- Does anyone see a problem during extraction?

# Bit Plane Complexity Segmentation

- What if the *message* data is not complex?
  - During extraction, the region will no longer exceed the threshold
  - No data will be extracted
- Solution: Conjugate the message data
- The 8x8 matrix is exclusive-or'd with a checkerboard pattern

# Bit Plane Complexity Segmentation

- Conjugation shown graphically
- $P \text{ xor } W_c = P^*$





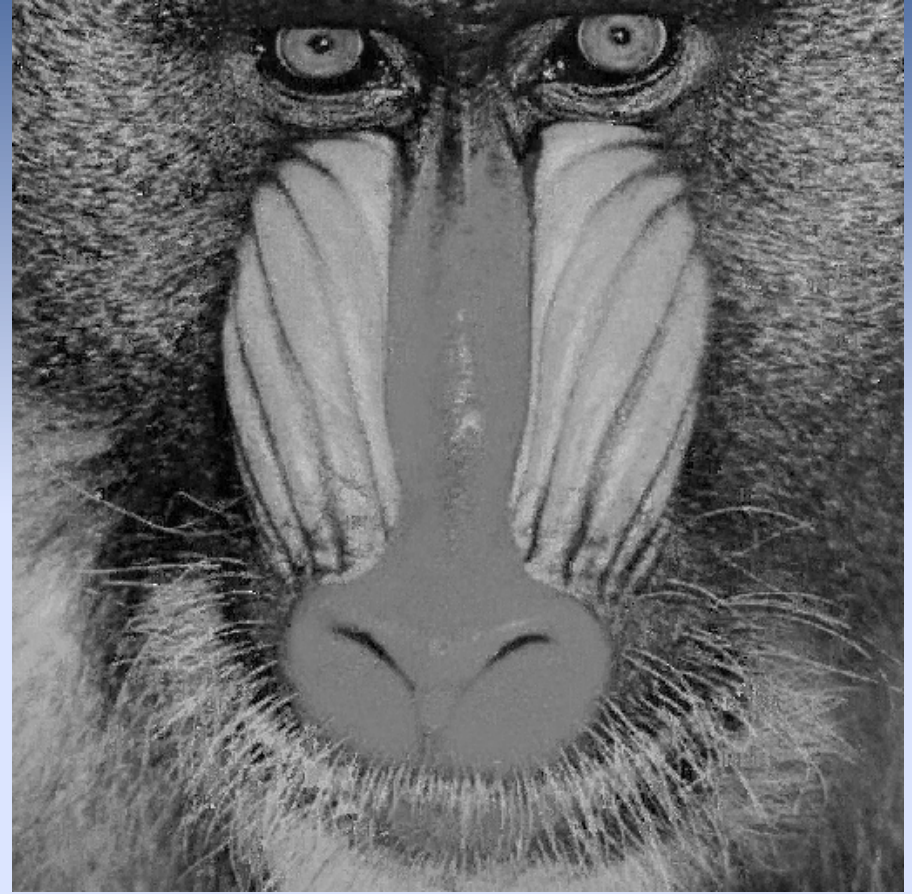
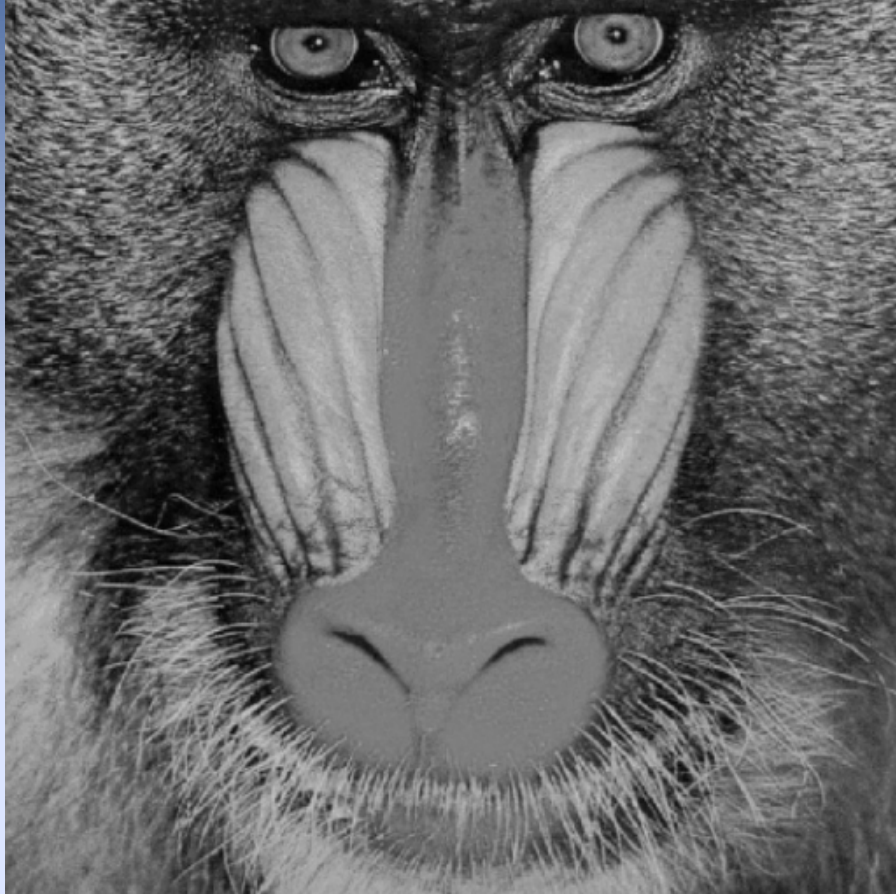
# Bit Plane Complexity Segmentation

- The complexity of  $P^*$  is  $(1 - \alpha_p)$
- Given:  $\alpha_{th} < 0.5$ ,
  - if  $P$  is not complex enough,  $P^*$  will be
- Note:  $(P^*)^* = P \rightarrow (a \text{ xor } b) \text{ xor } a == b$
- This ensures that whenever information is embedded, the complexity will be greater than the threshold
- Now the problem is determining which regions are original data and which ones are conjugate data

# Bit Plane Complexity Segmentation

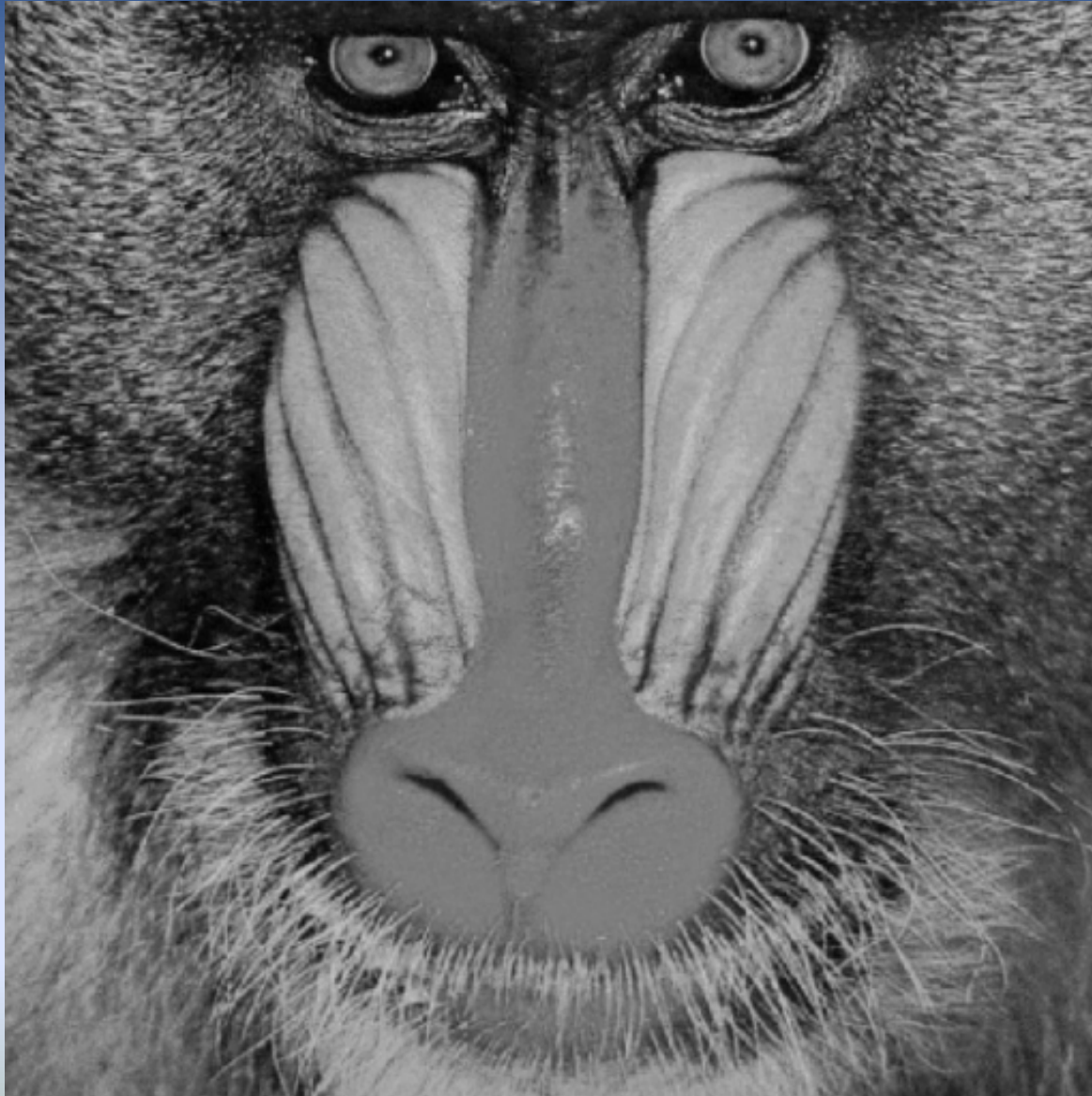
- **Solution!**
- **Reserve one bit of each region to indicate conjugation**
  - Make the lower left bit of the 8x8 matrix a zero
  - If conjugation occurs, it will become a one
- **This does use 1/64 of your embedding capacity**
- **Other solutions proposed, this is the simplest**

# Suggested Threshold of 0.3



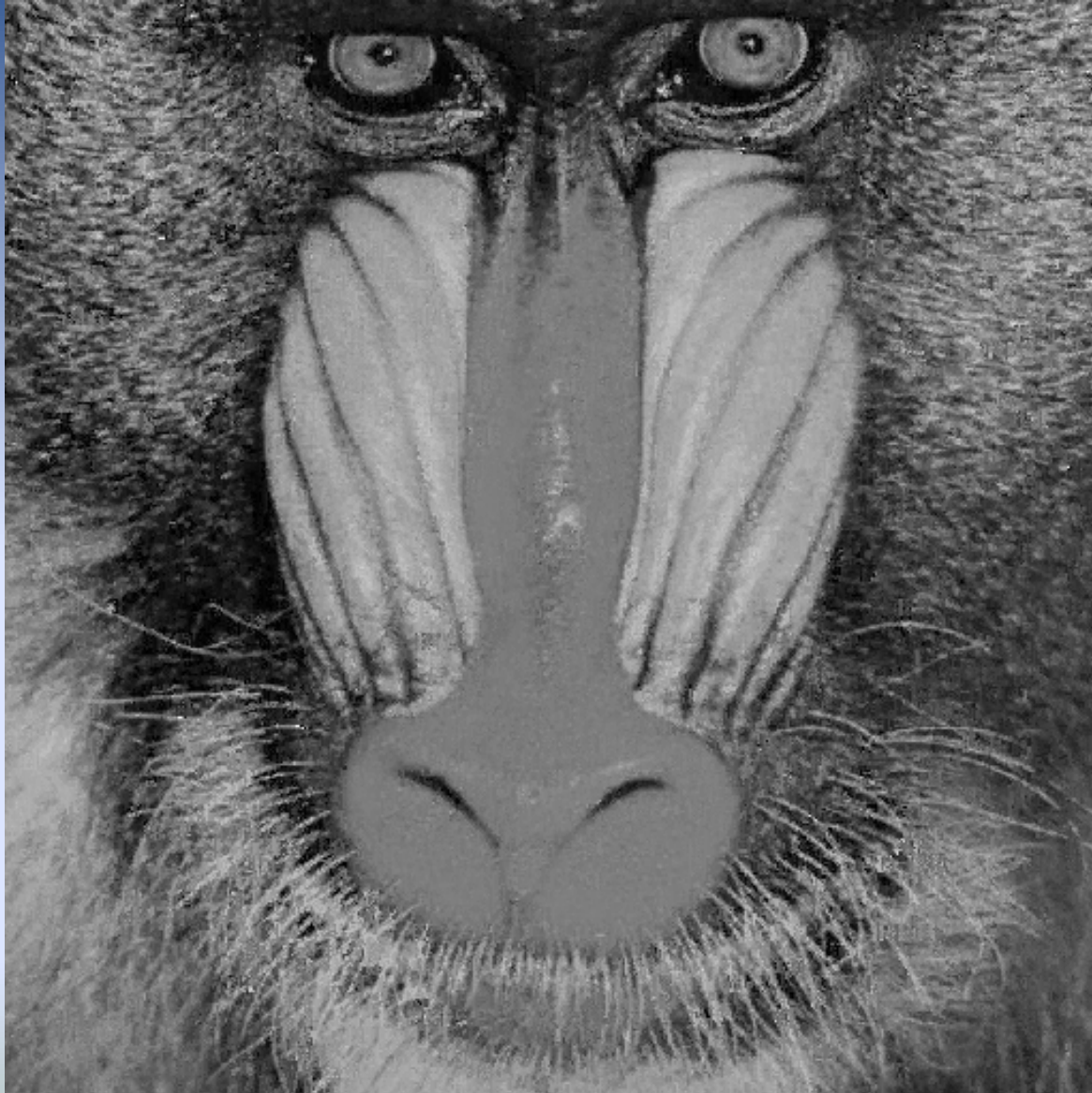
**Thresh=0.3, cap = 134KB/258KB**

# Unmodified

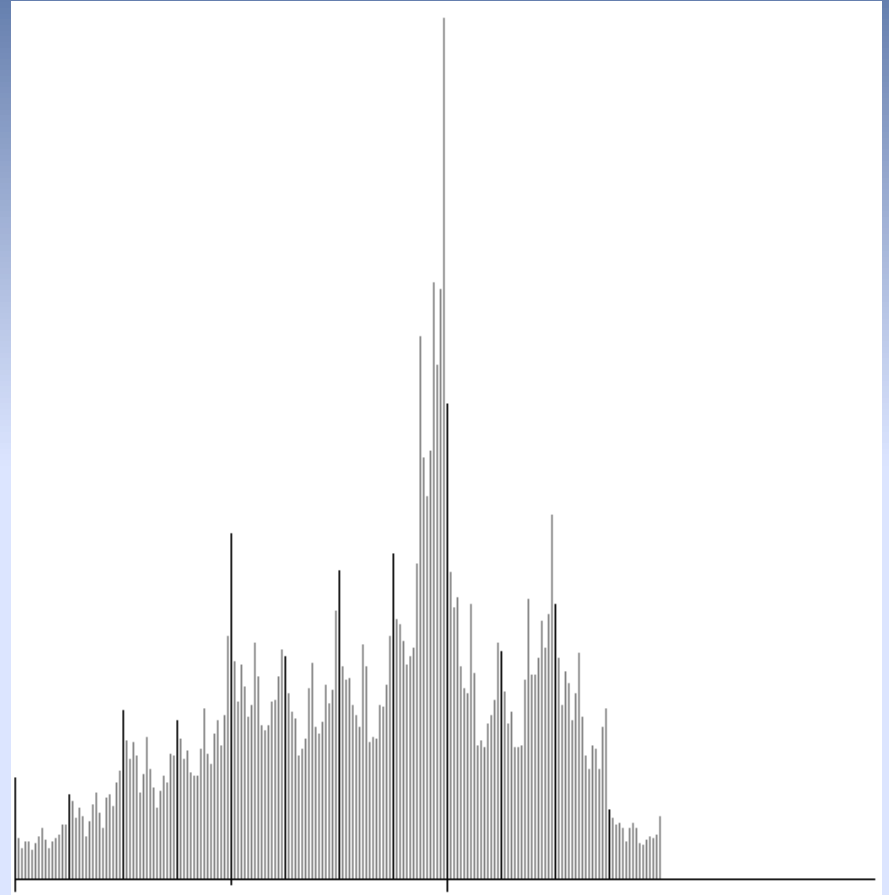
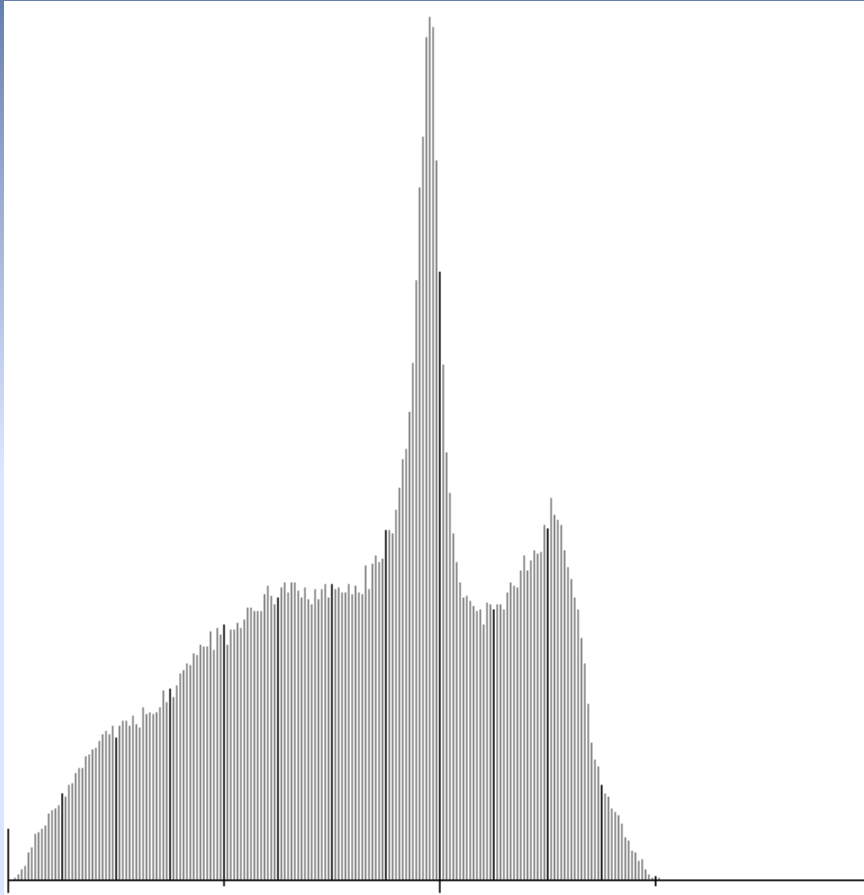




**Thresh=0.3, cap = 134KB/258KB**

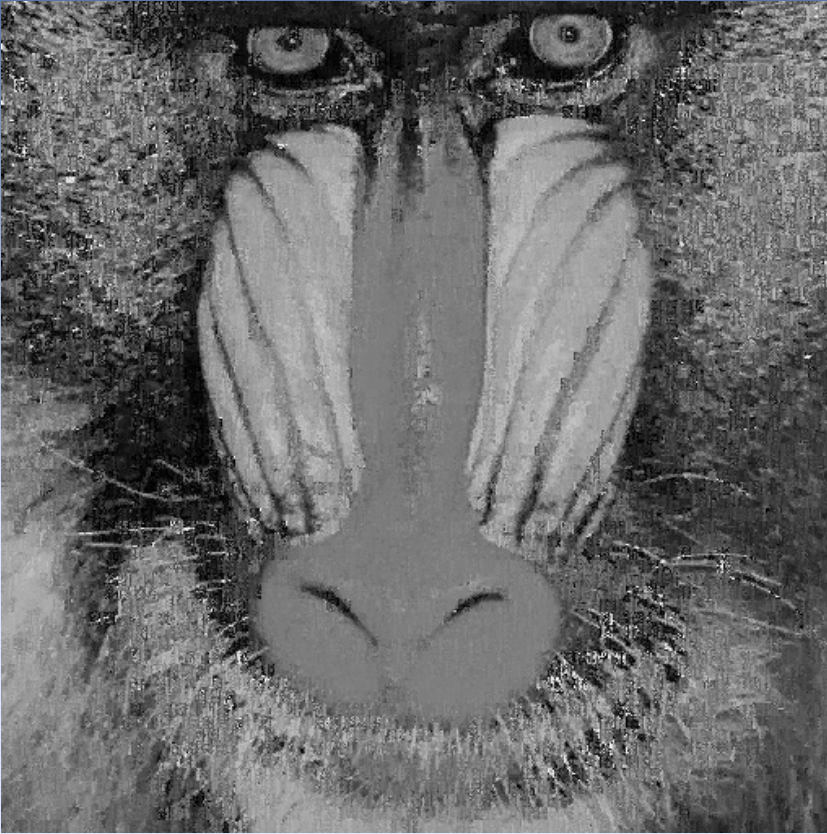


# Bit Plane Complexity Segmentation



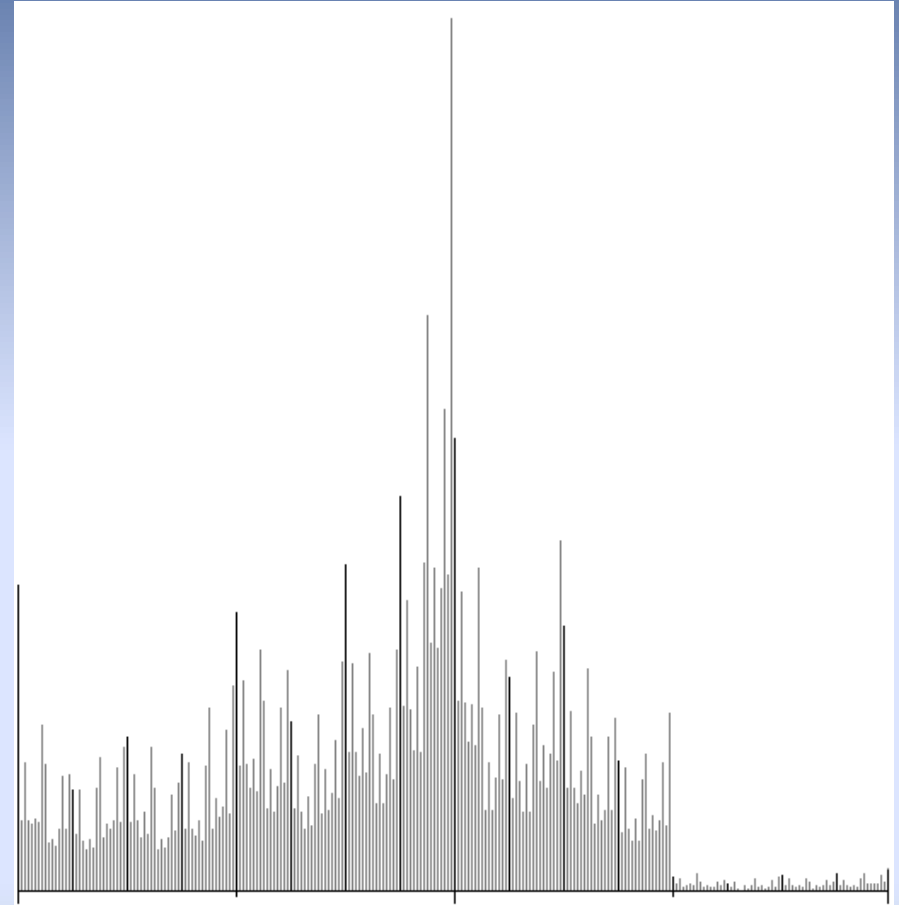
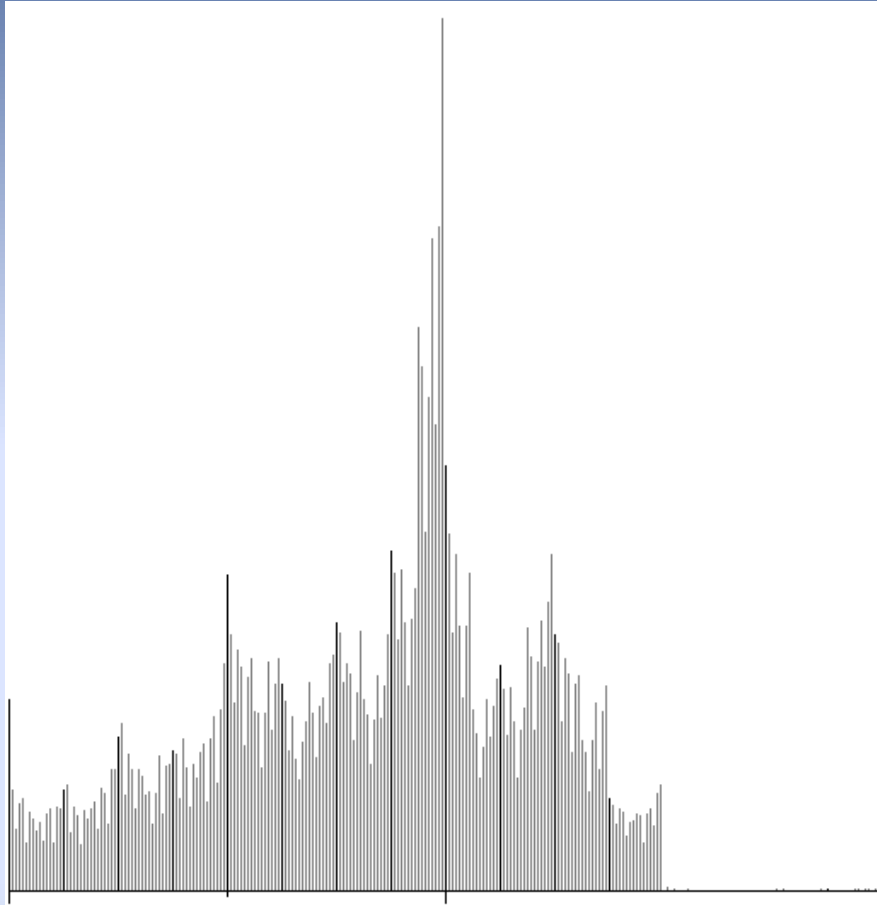


# Lower Complexity Threshold



**Thresh=0.2, cap = 163KB/258KB    Thresh=0.1, cap = 193KB/258KB**

# Lower Complexity Threshold



# Less Complex Image



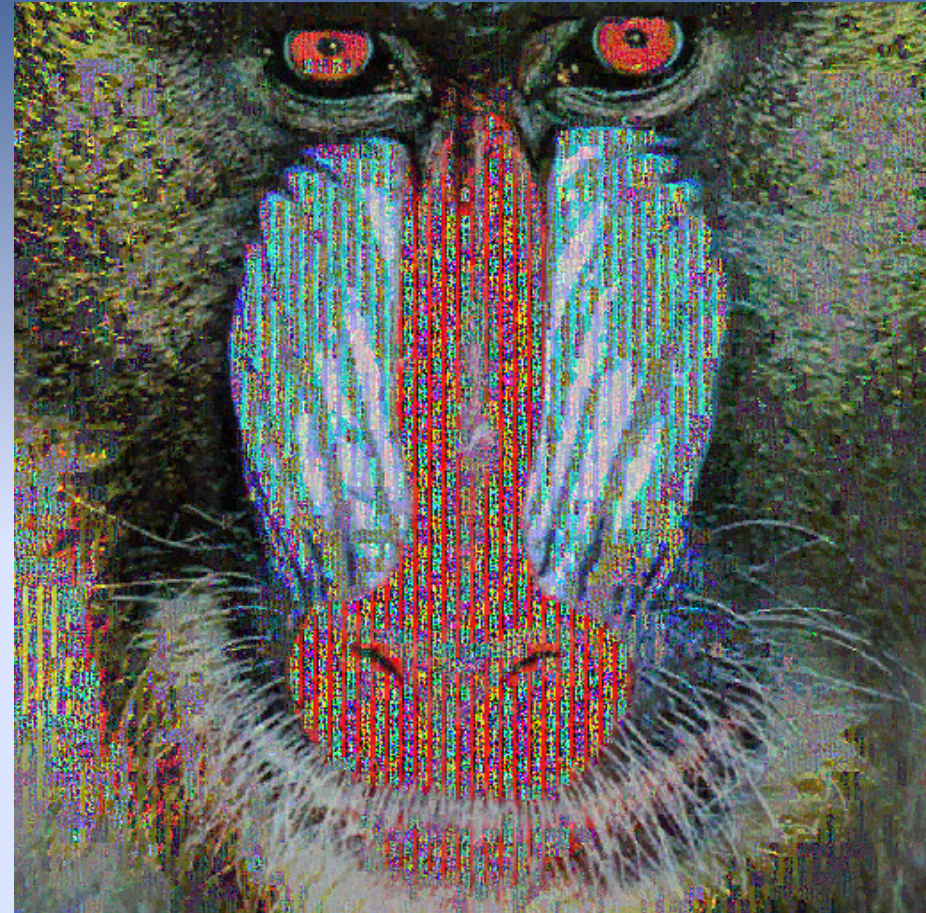
**Thresh=0.3, cap = 106KB/258KB**



# Color BPCS



**Thresh=0.4, cap = 405KB/769KB**



**Thresh=0.3, cap = 557KB/769KB**

# Bit Plane Complexity Segmentation

- The cover image matters!!!
- Other authors proposed better complexity measures
  - Less perceptible
  - BUT, less capacity

# More Advanced Steganography

## Transform Domain



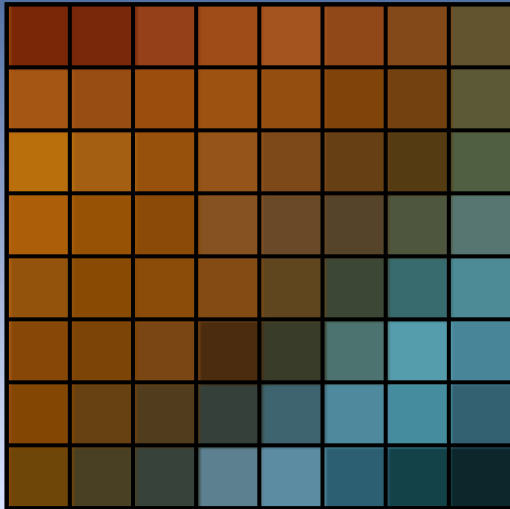
# Transform Domain

- **Transform domain** methods hide data in significant portions of the cover
  - as opposed to least significant
- **Generally more robust to manipulation**
  - affine transforms
    - scaling, rotating, shearing, translating, flipping
  - lossy compression
  - analog to digital and digital to analog conversions

# Transform Domain

- Number of transforms exist
  - Discrete Cosine Transform (DCT)
  - Discrete Fourier Transform (DFT)
  - Laplace Transform
  - Discrete Wavelet Transforms (DWT)
  - Modulated Complex Lapped Transform
  - Mellin-Fourier Transform
- All of these can be applied to images and/or audio
- The Discrete Cosine Transform (DCT) is used in JPEG

# Jpeg Process

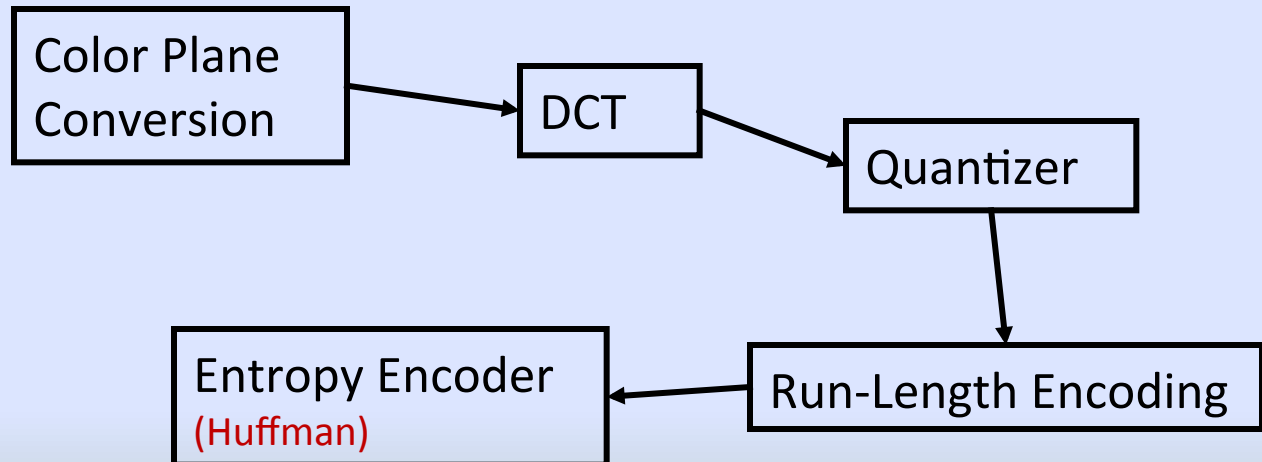


8 X 8  
Image  
Block

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



# JPEG Process

- **Converts color RGB to  $YC_rC_b$** 
  - Y is the luminance component
  - $C_r$  &  $C_b$  are the chrominance components
  - Grayscale images only have the Y component
- **The image is divided into 8 x 8 blocks**
- **A 2-dimensional Discrete Cosine Transform (DCT) is performed on each**

# JPEG Process

- Results is quantized according to desired quality
  - The quantization is the primary “lossy” part
- A combination of Run-Length Encoding (RLE) and Huffman coding is applied to finish the compression
  - This process is lossless
- To get the image back, the process is reversed
- The restored image is similar in appearance, but mathematically different from the original
- If high quality is used, there is little, if any, perceptible difference



# DCT Hiding Technique #1

- 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
    - Insignificant values are eliminated
  - the pair (2,0) & (1,2) work [#14]
  - Let  $C_1$  = the 1<sup>st</sup> coefficient,  $C_2$  = the 2<sup>nd</sup>
  - NOTE: these are the calculated coefficients, not the values in the quantization table
- Select a cover block
- Get the DCT transform of the block
- If the message bit is 0,  $C_1 < C_2$
- If the message bit is 1,  $C_1 > C_2$
- If either of the above conditions is not true, swap the coefficients

**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

# DCT Hiding Technique #2

- **“High Capacity Data Hiding in JPEG Compressed Images”**
  - Chang, C.C. and Tseng, Hsien-Wen
- **An adaptive Discrete Cosine Transform, Least Significant Bit technique**
  - Hides in lower and middle frequency components
  - Adapts to different characteristics of each block
  - Performs capacity estimation
- **>> Greater than 1 bit per 8x8 block**

# DCT Hiding Technique #2

- **Capacity Estimation**
  - Determine max number of bits that can be modified while remaining imperceptible
- **Uses a capacity table based upon the quantization table**
  - user sets an  $\alpha$  (alpha) factor
    - higher  $\alpha$ , higher bit rate, but increased distortion
  - lower frequency components hold fewer bits
  - higher frequency can hold more bits, but there are fewer

# DCT Hiding Technique #2

- Each table is 8x8, we'll use  $x,y$  to denote a specific element
- $C_Q(x,y) = \lg(\alpha * Q(x,y))$ 
  - Capacity based on Quantization table
- $M(x,y) = \lg(|D(x,y)|)$ 
  - Capacity based on DCT coefficients
- Use the lower of these two

## DCT Hiding Technique #2

- **Block Classification** – determine which blocks are better candidates for hiding
- If a background has a strong texture, the Human Visual System (HVS) is less sensitive to distortions
- **Blocks divided into two classes:**
  - **uniform blocks**
  - **non-uniform blocks**
- **Non-uniform blocks use a larger  $\alpha$  value ( $1.2 * \alpha$ )**
- **$D_x$  is the  $x^{\text{th}}$  AC coefficient**
- **If  $G$  is below a threshold, the block is uniform**

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



# Embedding Algorithm:

- Set the  $\alpha$  value
- Choose the block to be embedded
- Determine classification of block:
  - uniform, non-uniform
- Determine number of bits to hide in each *quantized* DCT coefficient
- Embed the data
- Apply the normal JPEG entropy coding

# High Capacity Example #1 (quality=95%)



Mandrill512.bmp\_q95\_a8\_u8.jpg ---> 71745/ 322564 22.24% of stego

## High Capacity Example #2 (quality=95%)



Domino512.bmp\_q95\_a8\_u8.jpg ---> 38827 / 175915 22.07% of stego

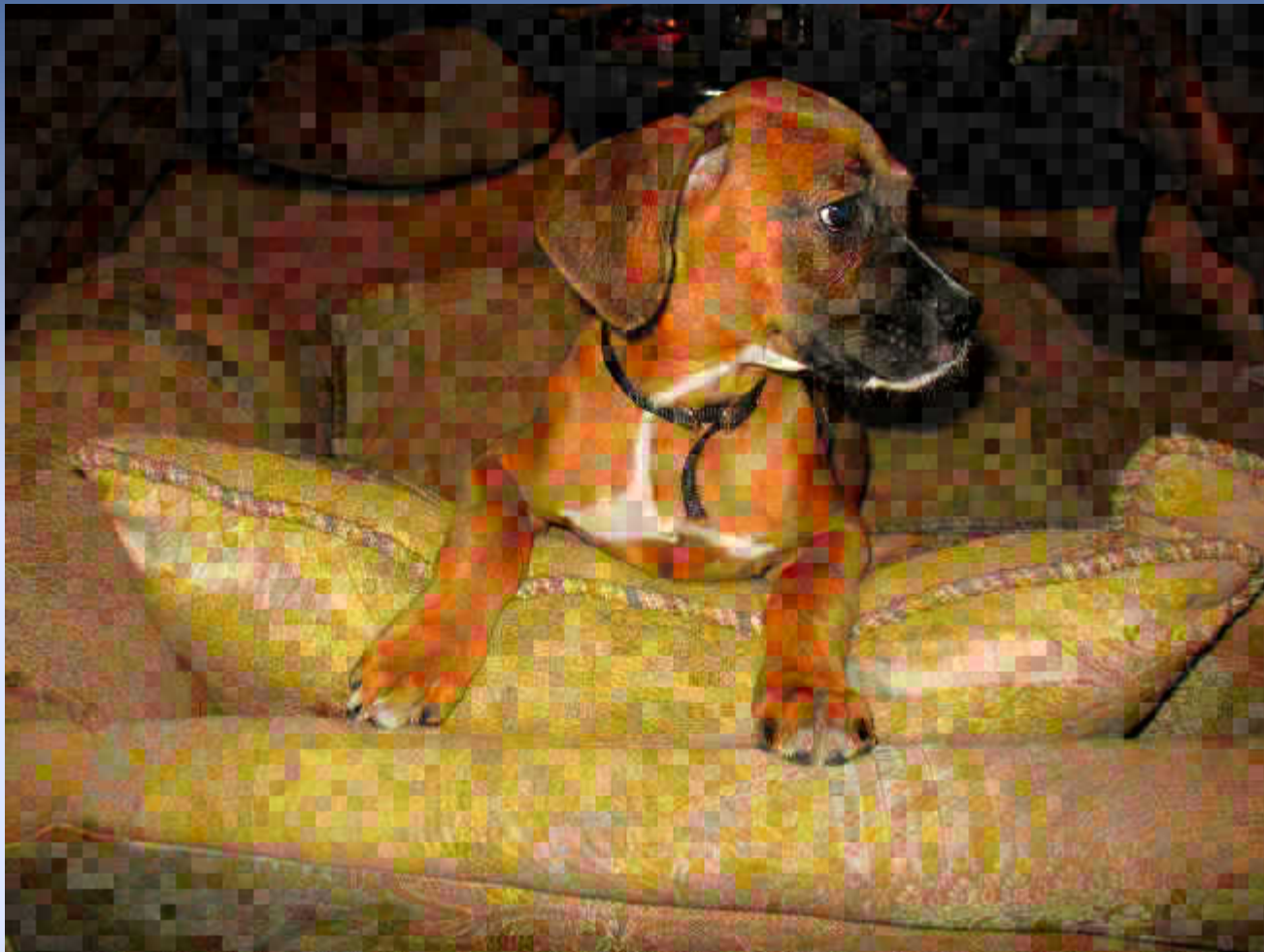


# High Capacity Example #3 (quality=99%)



S2\_Rocky.jpg\_q99\_a8\_u8.jpg ---> 107643 / 511089 21.06% of stego

# High Capacity Example #4 (quality=50%)



S2\_Rocky.jpg\_q50\_a8\_u8.jpg ---> 6612/ 37437 17.66% of stego

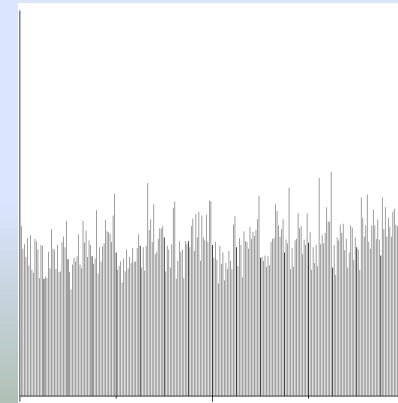
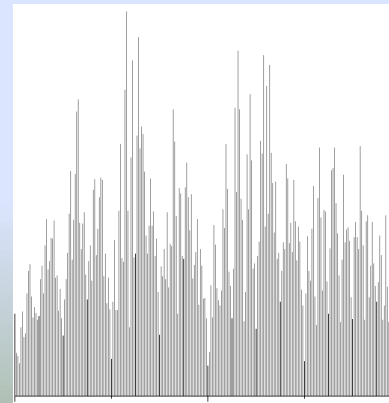
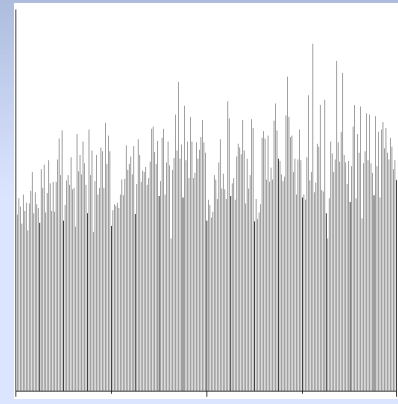
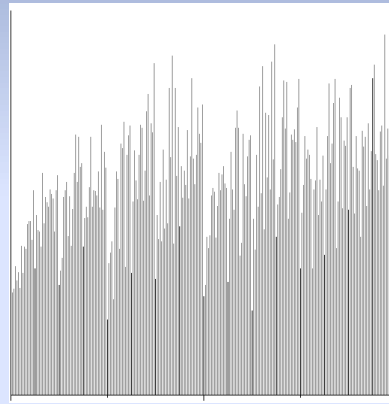
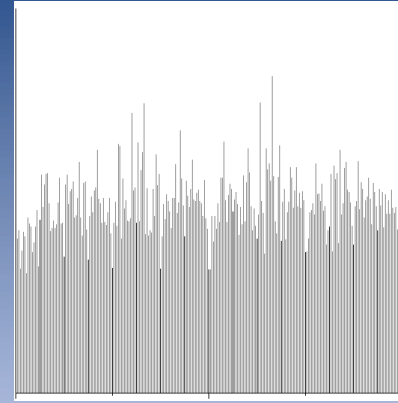
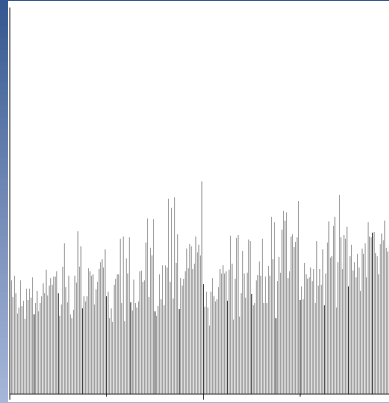


# High Capacity Example #5 (quality=50%)



S2\_Rocky\_A.jpg\_q50\_a8\_u8.jpg ---> **1575/ 6612** 18.81% of stego

# Histograms Are Not Effective



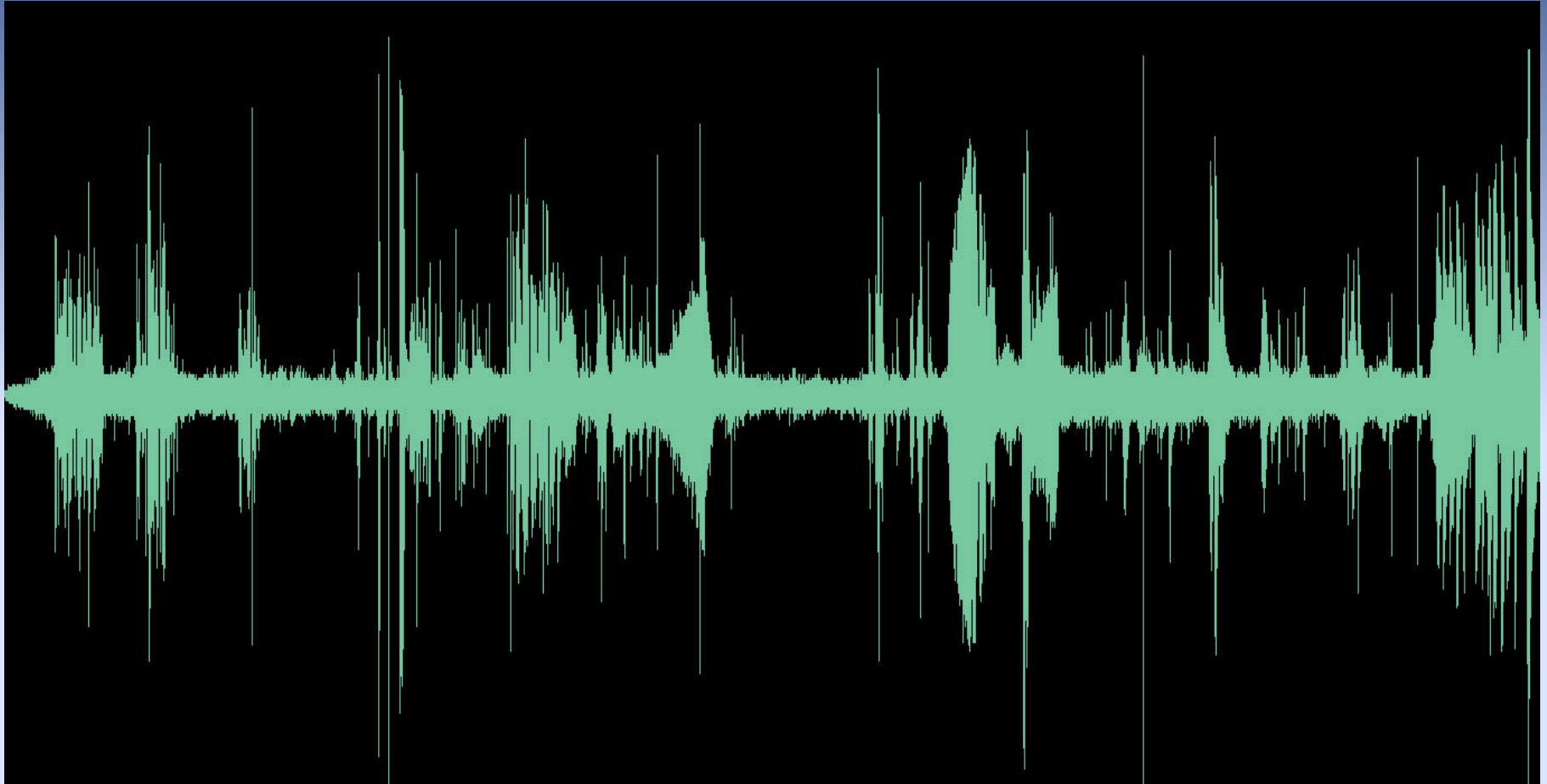
# More Advanced Steganography

## Audio

# Audio

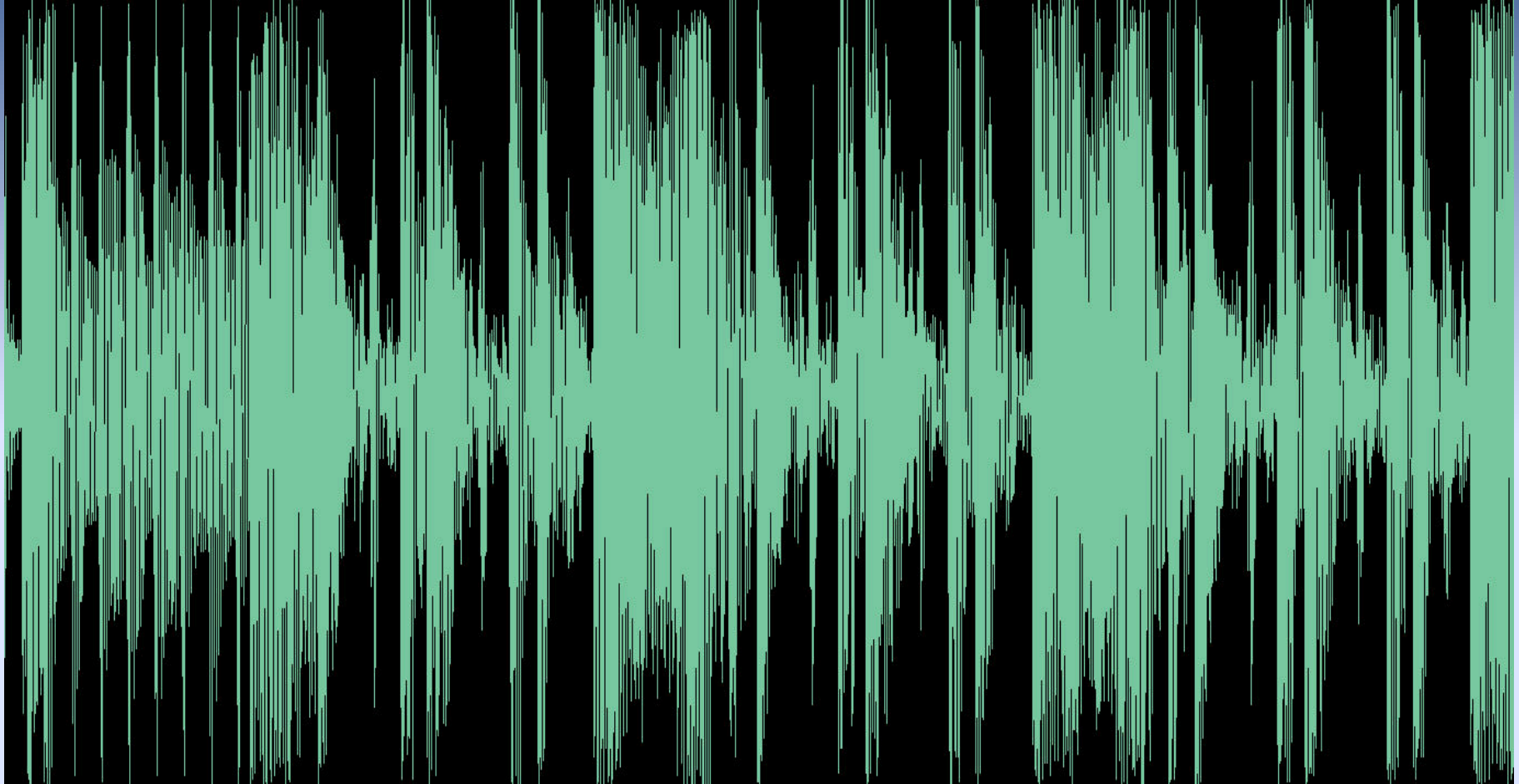
- **The Human Auditory System (HAS) is more sensitive than the Human Visual System (HVS)**
- **Hiding discretely in audio is tougher**
- **LSB techniques apply to audio and are effective but area also easily detected**
- **For compressed audio, such as mp3, you could apply similar jpeg techniques**

# Speech Waveform



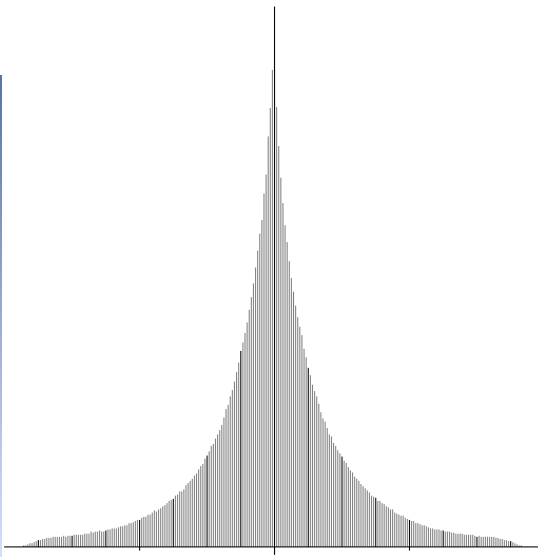


# Music Waveform (Pop)

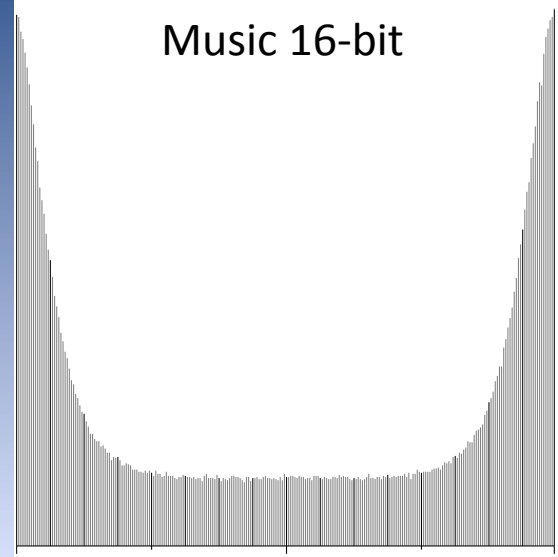


# Music Histograms

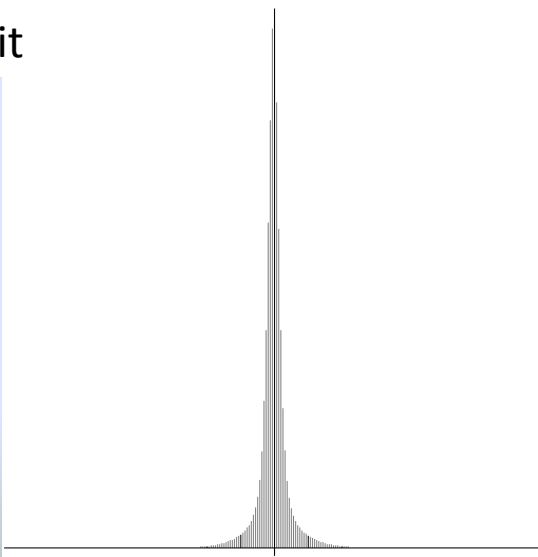
Music 8-bit



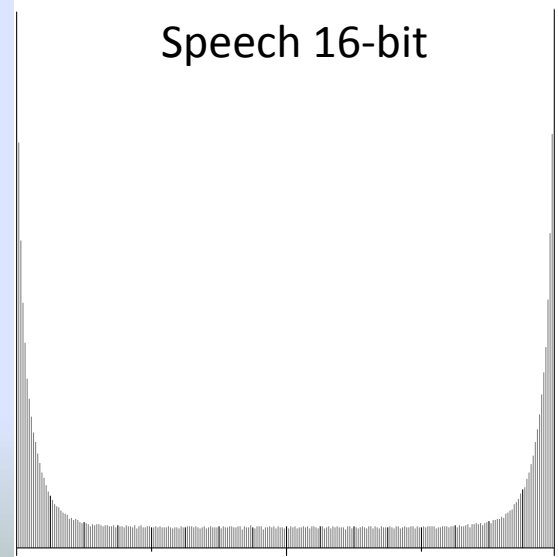
Music 16-bit



Speech 8-bit



Speech 16-bit



# Uncompressed Audio

- **Since mp3s are so prevalent, why would we care about uncompressed audio?**
- **Who shares those on the Internet?**

# Uncompressed Audio

- Since mp3s are so prevalent, why would we care about uncompressed audio?
- Who shares those on the Internet?
- **NO ONE**
  - A song is 40MB to 60MB
  - An mp3 is 3MB to 6MB

# Uncompressed Audio

- Since mp3s are so prevalent, why would we care about uncompressed audio?
- Who shares those on the Internet?
- NO ONE
  - A song is 40MB to 60MB
  - An mp3 is 3MB to 6MB
- BUT Compact Discs are STILL popular ...
  - and innocuous
- Want to be *VERY* discrete about your data?
  - Put it on an audio CD



# Uncompressed Audio

- All techniques for hiding in an uncompressed wave file can be used to
  - Modify that .wav file
  - Write it to an audio CD that will play in a regular CD player
  - Extract the hidden data from that CD

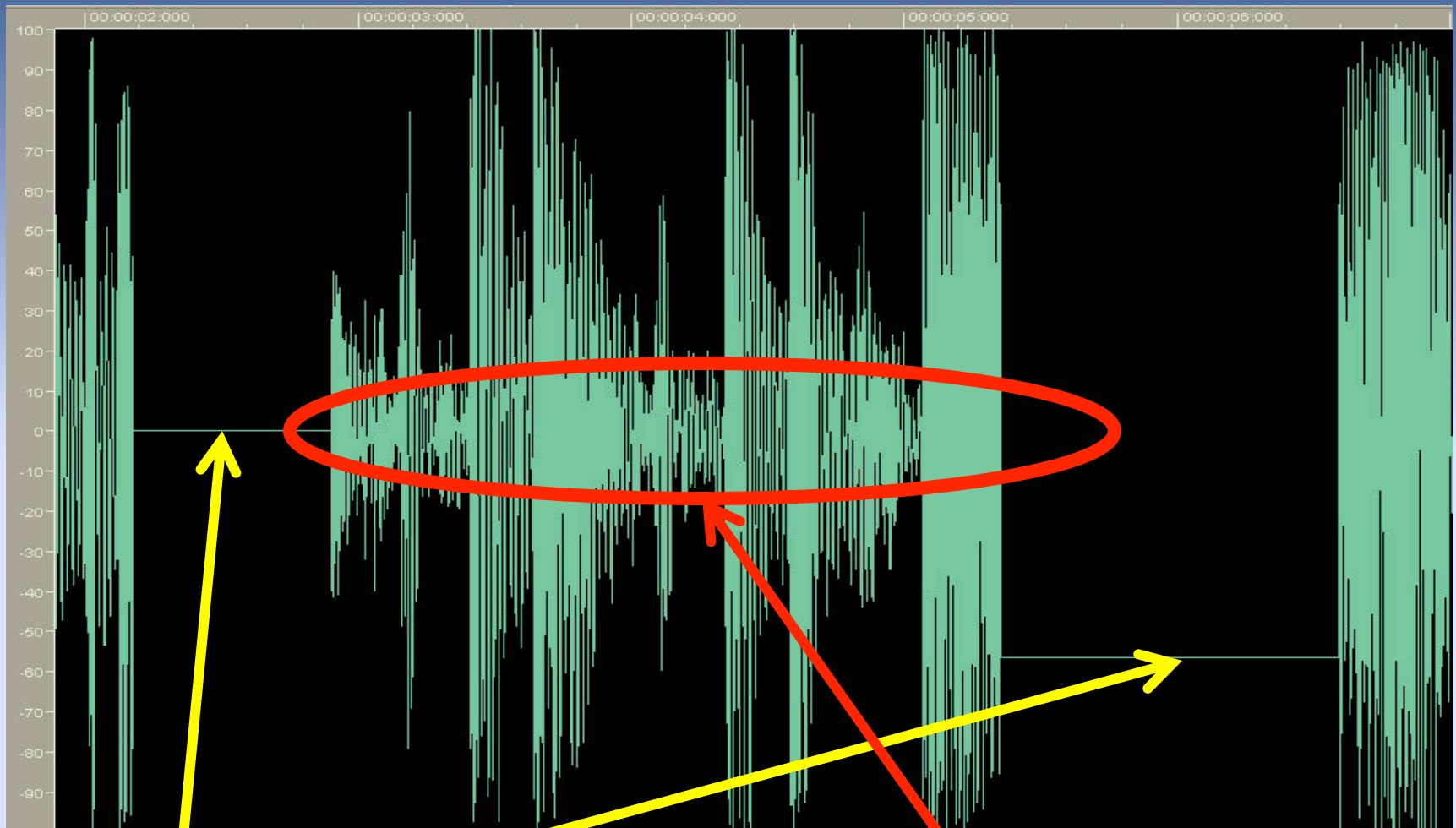
# Uncompressed Audio Hiding

- **Least Significant Bit**
  - High capacity
  - Easily detectable
- **Hiding in the Silence of Sound (HISS)**
  - Low capacity
  - Nearly undetectable

# Hiding in the Silence of Sound (HISS)

- **Modulate temporal properties of the low amplitude portions of an audio signal to carry the secret message**
  - **I.E. modify the length of low amplitude periods of an audio signal**
- **Originally designed for hiding in voice**
- **Works well with hard rock too**
- **Not so well with soft music, particularly jazz and especially classical**

# Low Amplitude vs. Silence



**SILENCE**

**Low Amplitude**

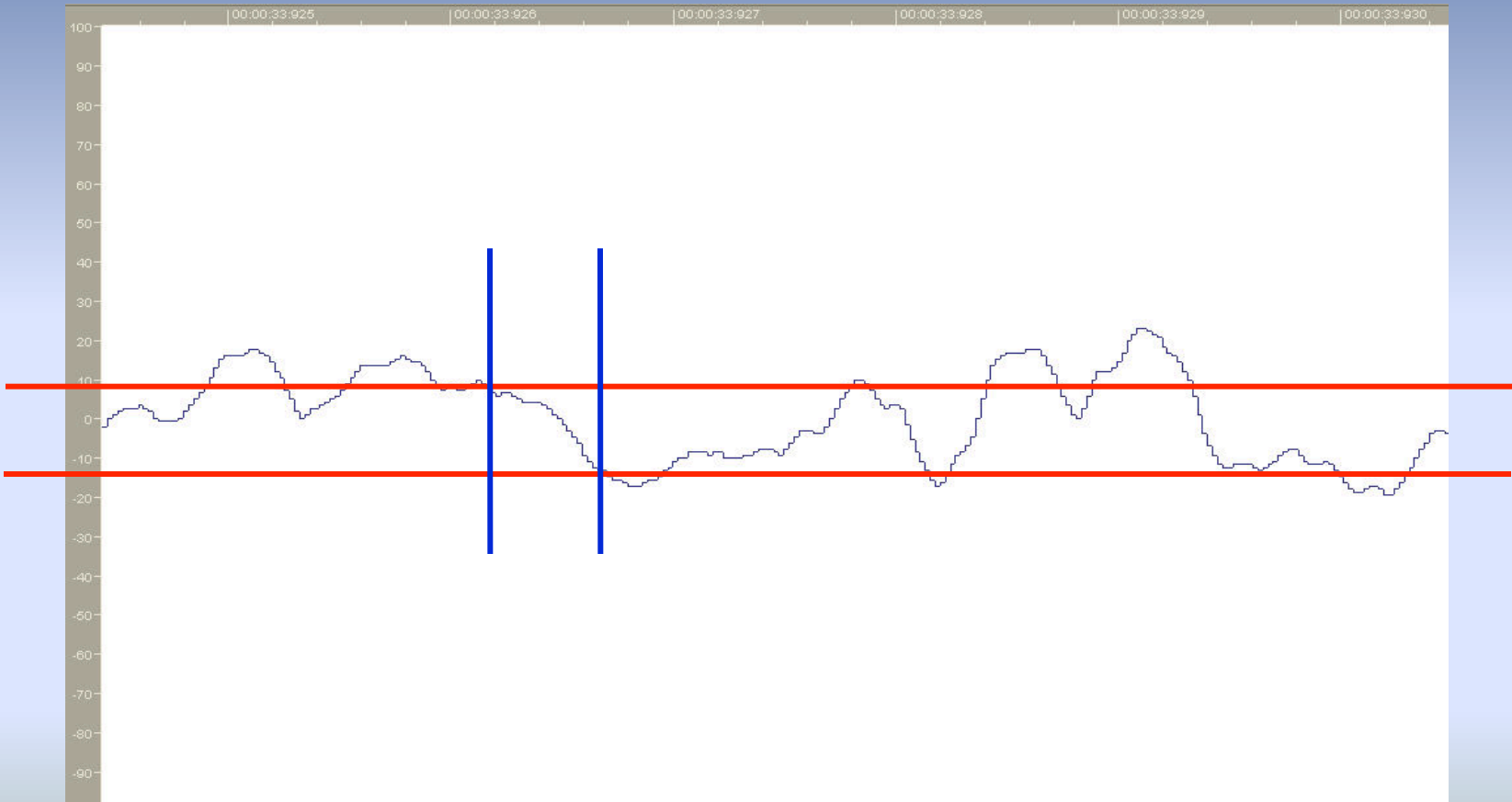
# HISS

- HISS finds periods of low amplitude based on some threshold
- 10% is a good starting point
  - For 8-bit audio, 10% of the maximum is about +/- 12
  - For 16-bit audio, 10% of the maximum is about +/- 3277
  - Max +value may not equal the max -value
  - Each wave file is sampled for it's peak values, so the actual thresholds vary between samples
    - ff



# Audio

- Speech up close – blue lines show a snippet of audio within the 10% threshold, ~ 30 samples



# HISS Algorithm

- Determine the number of samples in the period
  - Example: 30 samples
- Determine the maximum number of message bits that can be hidden
  - $\log_2 30 = 4.91$ , truncated to 4
  - Can hide a maximum of 4 bits of data
- Since  $2^4 = 16$ , could alter the length by 15
  - +/- 15 out of 30 is too much change
- A “capacity” factor reduces this to 2 or 3 bits
  - the same capacity factor must be used for extraction

# HISS Algorithm

- A capacity factor of 1 yields 3 bits
- $2^3 = 8$
- The number of samples, mod 8, needs to equal the value of the 3 message bits
  - $30 \bmod 8 = 6$ 
    - $24 \bmod 8 = 0$
    - $25 \bmod 8 = 1$
    - $26 \bmod 8 = 2 \dots$
    - $31 \bmod 8 = 7$

# HISS Algorithm

- **HISS retrieves 3 bits of the message**
  - Say the 3 bits of the message equals 6 (  $110_2$  )
    - **Nothing changes**
  - Say the 3 bits of message equal 7 (  $111_2$  )
    - **Then the 30 samples must be changed into 31 samples**
    - **→  $31 \bmod 8 = 7$**
    - **A single sample is inserted in the block**
  - Say the 3 bits of message equal 5 (  $101_2$  )
    - **Then 30 samples must be changed into 29 samples**
    - **→  $29 \bmod 8 = 5$**
    - **A single sample is removed**

# HISS Algorithm

- Insert/remove samples to match the message
- The new range will be  
24 (message == 0) to 31 (message == 7)
- Removing samples is straightforward, delete it
- If there are multiple samples to remove, HISS deletes samples throughout
  - If HISS needs to remove half the samples, it removes every other one



# HISS Algorithm

- Inserting samples works the same, except there is the additional consideration of amplitude
- I have researched the values of the amplitude of a sample, in between two other samples
  - Given three samples, the one in the middle is most likely to be the average of the other two

# HISS Algorithm

- When HISS inserts a sample, the amplitude is the average of the other two
- Does this present an easy way to detect that an audio clip has been modified by HISS?
  - I.E. Too many samples such that the middle sample is the exact average of the other two?
- No! Because the capacity is low enough that this does not significantly alter the overall statistics

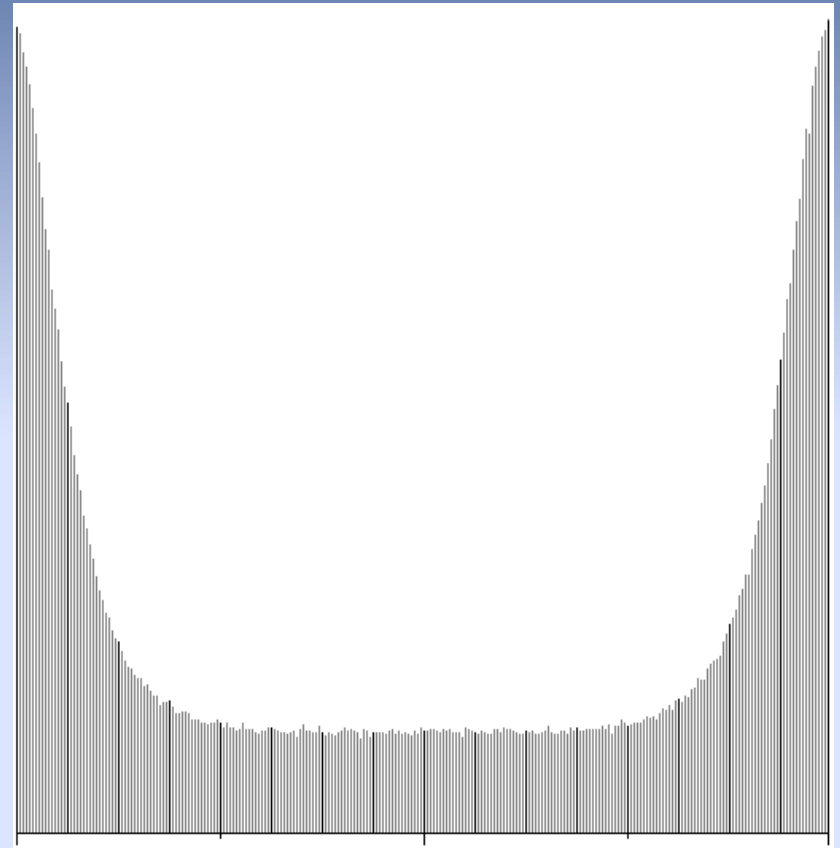
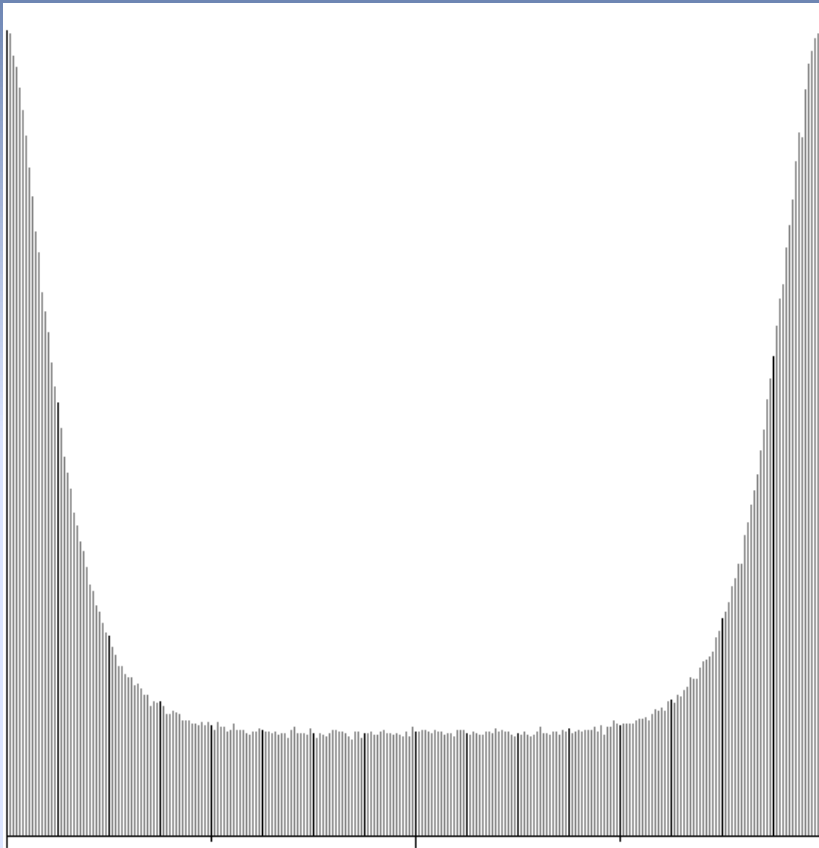
# HISS Algorithm

- HISS has another limitation ... time
- Suppose a long period of silence, ~ 10 seconds
- HISS would determine is could hide 18 bits
  - The capacity factor reduces that to 17 bits
- A 17 bit change, worst case, is 25% (~2.5 seconds)
- Changing by 2.5 seconds would likely be noticeable
- HISS limits itself based on a time maximum
  - How much time would be altered by the insertion/deletion
  - The default threshold ~12 milliseconds

# HISS Algorithm

- **HISS works with 8 or 16-bit files, any sampling rate, and 1 or 2 channels**
  - **With multiple channels have to find when both channels are within the threshold**
    - **Can't lengthen one channel while shortening another!**
  - **A better approach is to average the two channels and check that against the threshold**
- **HISS works – let's try it out!**

# HISS Histograms are NOT Effective



**The clean histogram is on the left**

# Uncompressed Video

## Audio Video Interleave (AVI)



# Audio/Video

- **Audio/Video offers an enormous capacity**
- **Hiding in audio/video is heavily researched**
- **With popular sites like YouTube you can imagine the potential for covert data exchange**
- **We'll wrap up this presentation with a statistical averaging technique used on uncompressed Audio Video Interleaved (AVI) files**

# AVI

- AVI files contain multiple streams of various data
- A common format is a sequence of images interleaved with audio track(s)
  - Can be in a compressed format
  - We'll focus on uncompressed media
- A technique that hides in a bitmap can apply to the sequence of images in AVI
- A technique that hides in a wave can be applied to the audio track(s) in AVI

# Hiding in AVI (Video Stream)

- The current implementation is limited to 24-bit RGB video and 16-bit audio
- For the video stream, this technique calculates the average of an array of pixels
- The LSBs of the *average* are the message data
  - 1x1 to 8x8
    - For the 1x1 array size, it reduces to simple LSB hiding
  - Bits to hide ranges from 1 to 8
    - 8 being that the entire pixel IS the message

# Hiding in AVI (Video Stream)

- Example: hiding 2 bits ( $10_2$ ) in a 4x4 array
- 4 samples have the following values
  - 252, 178, 199, 210
  - Sum = 839
  - $839/4 = 209.75 \rightarrow 209$ 
    - **Truncate result**
  - $209 == 0xD1 == 1101\ 0001_2$
  - Since message is  $10_2$ , but data is  $01_2$ , need to alter pixel values

# Hiding in AVI (Video Stream)

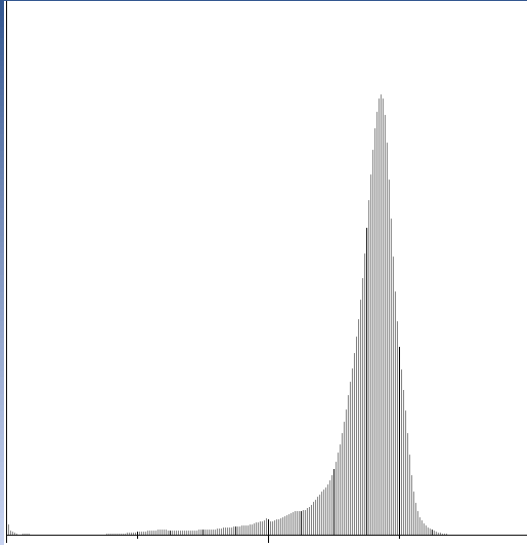
- Average must come up to 210
- In this case, we can add 1 to any of the pixel values
- We could also subtract 12, 3 from each
- $249 + 175 + 196 + 207 = 827$
- $827 / 4 = 206.75 \rightarrow 206$
- $206 == 0xCE == 1100\ 1110_2$ 
  - Last two binary digits are  $10_2$ , to match the message

# Hiding in AVI (Audio Stream)

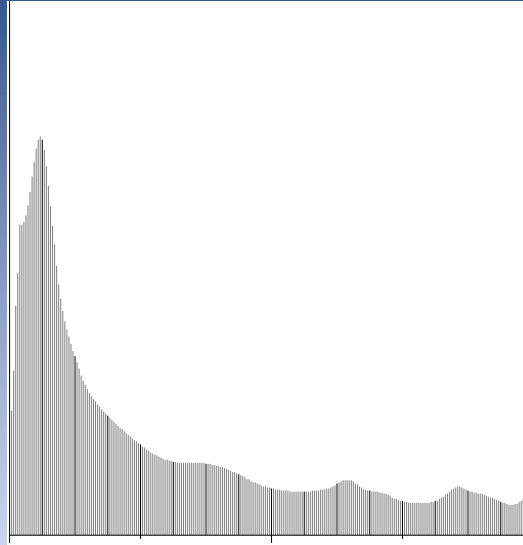
- The same approach is applied to the audio stream
- The number of samples to average is chosen
  - 1 to 8
- The number of bits (N) to hide per block is chosen
  - 1 to 8
- N message bits are read and the sample values are adjusted up/down to make the N LSBs of their average equal the message value



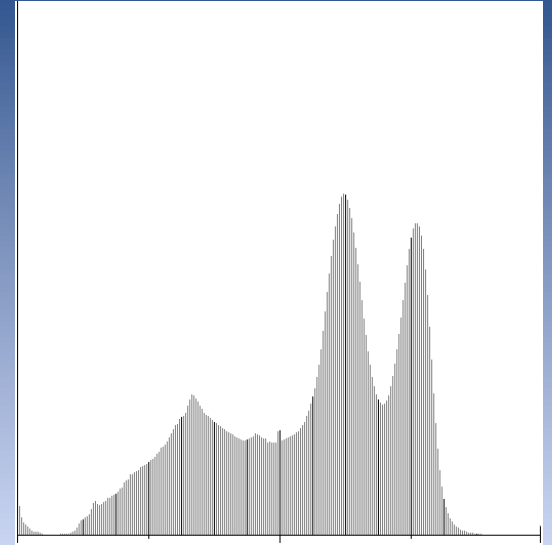
# Hiding in AVI (Typical Histograms)



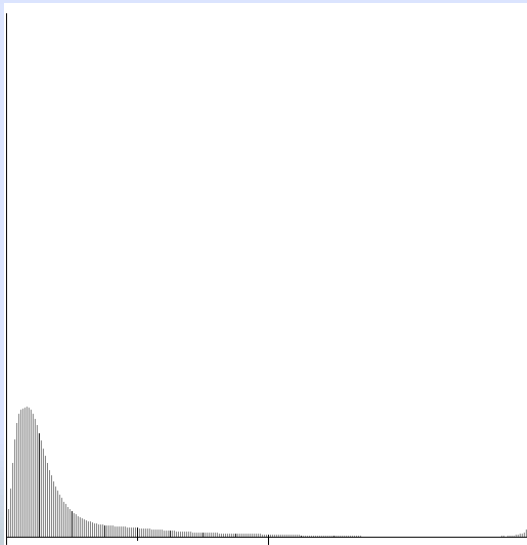
Snowboard



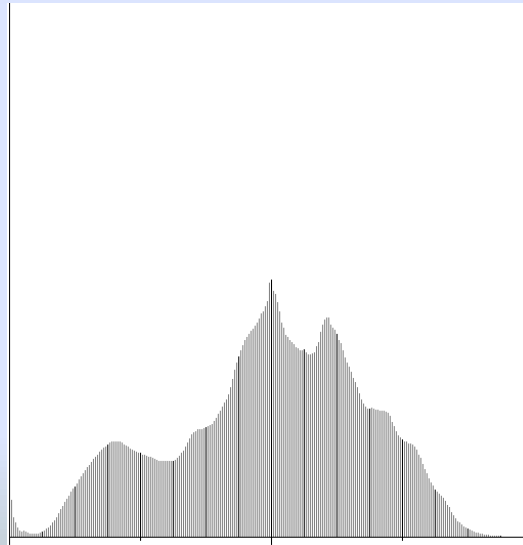
Vegas



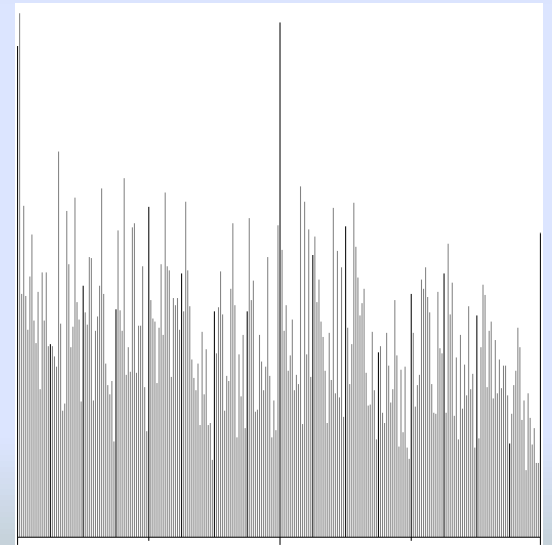
Boogie Bahn



Acrobats



Waterslide



Boogie Bahn - Compressed

# Hiding in AVI (Max Capacity)

- **Steg\_AVI.exe -e -vb 4 -vd 2 -ab 8 -as 1 Acrobats\_B.AVI \_DowntownVegas.avi rnd\_01.pad Out.avi**
  - **Input File: 469,523,396 bytes**
  - **57,915,133 bytes embedded within the video stream (12.3%)**
  - **1,441,370 bytes embedded within the audio stream (0.307%)**
- **The two values above represent max embedding capacity for these parameters**

# Hiding in AVI (Max Capacity)

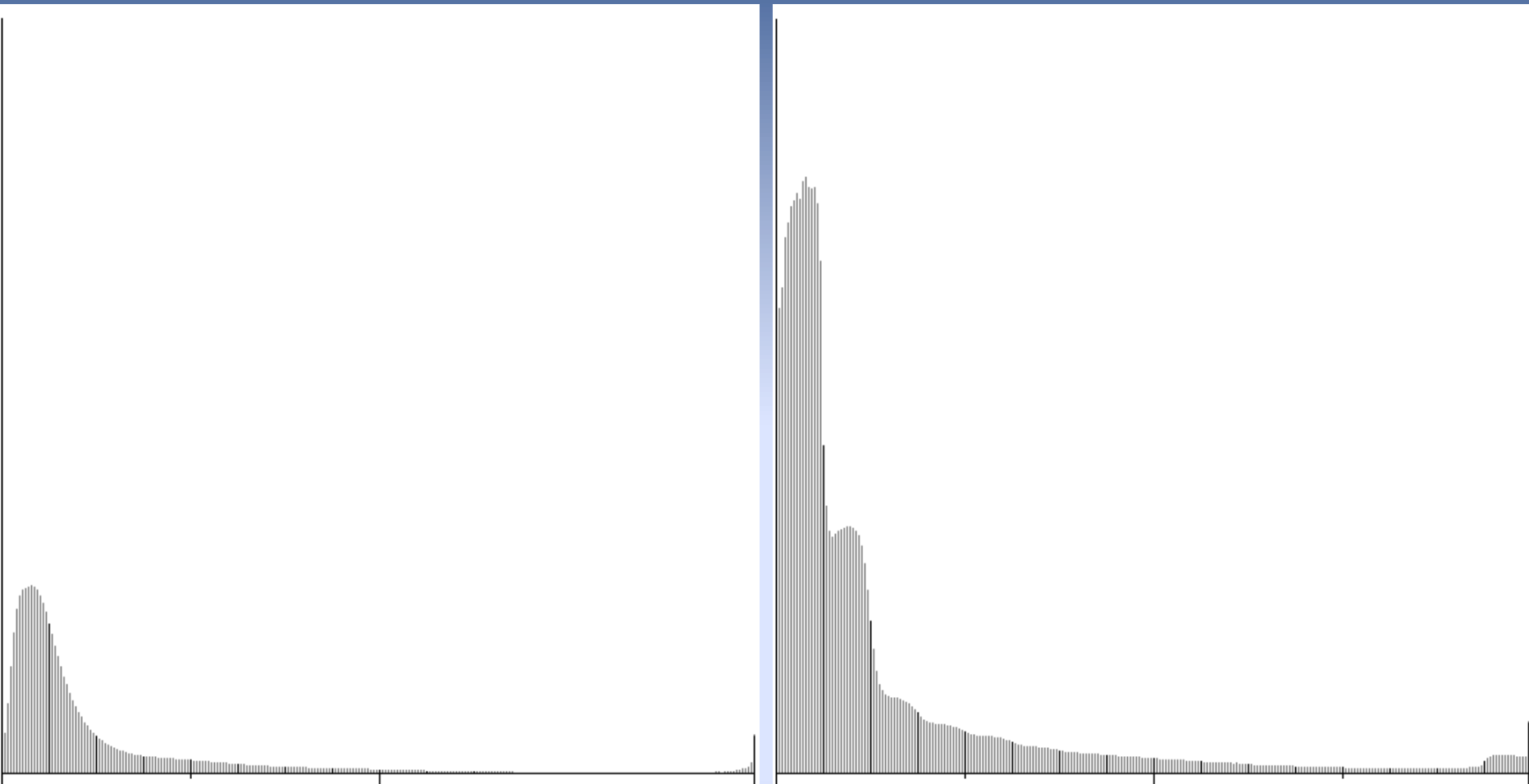


**clean**



**-vb 4 -vd 2 -ab 8 -as 1**

# Hiding in AVI (Max Capacity Histogram)



clean

**Note: Histograms not scaled**

**-vb 4 -vd 2 -ab 8 -as 1**

# Hiding in AVI (Max Capacity)

- **Detection from histogram NOT obvious**
- **Visual detection NOT obvious**
- **57 MB in 450MB**
- **That's a lot of data, while not undetectable, it's not obvious either**

# Whew!

## What Questions Do You Have?

- For more information or the actual software contact me @ [John.Ortiz@Harris.com](mailto:John.Ortiz@Harris.com)

## Please complete the Speaker Feedback Surveys

- This will help speakers to improve and for Black Hat to make better decisions regarding content and presenters for future events