Hiding information in images: Steganography and Watermarking

Presented by Gaudiano Antonio

Why hiding data in images?

Covert communications: spies / whistleblowers hidden informations.

Forensics: embedding metadata inside evidence images.

Copyright protection: embed creator’s «signature» to prove authorship.

Content tracking: monitor where and how a media is shared.

Authentication: verify that a file has not been tampered.

And more…

Steganography vs. Watermarking

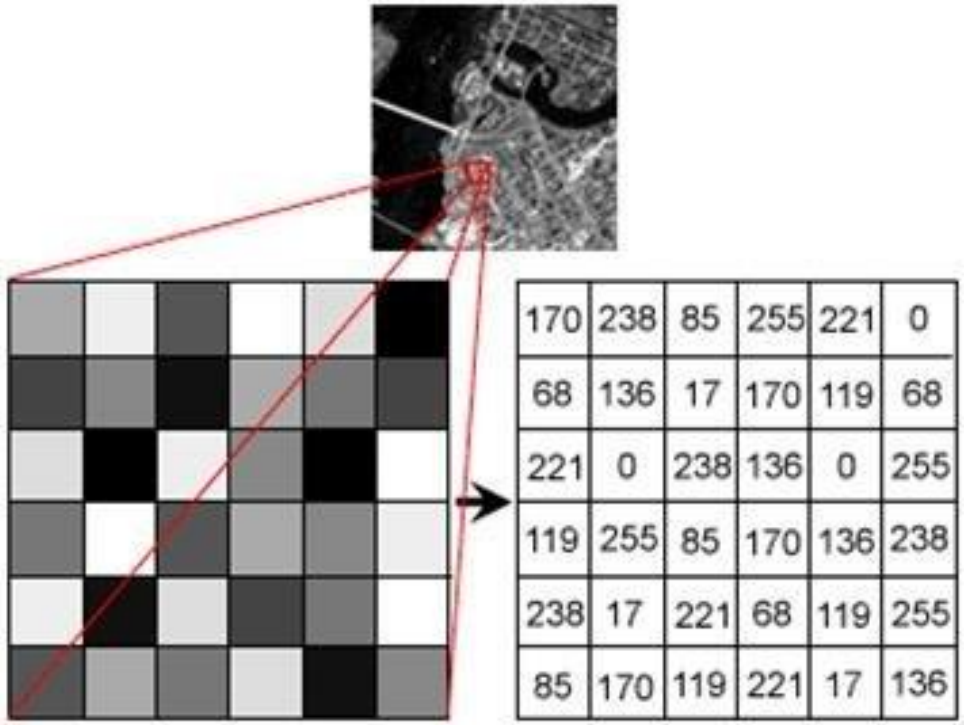
Feature	Steganography	Watermarking
Main purpose	Secret communication	Content protection
Visibility	Completely invisible	Invisible or visible
Payload	Medium to large	Small
Robustness	Sensitive to distortion / compression	Survives compression, resizing
Use cases	Espionage, censorship evasion, covert channels	Copyright, forensic tracking, authentication

Some theory: Color Spaces

- A color space defines how color values are represented in digital images.
- RGB: additive color model, each pixel is represented as a triple (R,G,B)
- CMY: subtractive color model, each pixel is represented as a triple (C,M,Y)
- YCbCr: more perceptual color space, modeled with Brightness and Blue/Red-difference chroma
- HSB: perceptual color spaces, modeled with Hue, Saturation and Brightness of the image

Some theory: Images as signals

- Images can be represented and processed as 2D (or 3D) discrete signals in space or frequency domains, as matrices of pixel intensities.
- But why? This representation makes easier manipulating them with well-known techniques such as applying filters (blur, sharpening…), transformations (Fourier, cosine), Sampling, Quantization…



Some theory: Discrete Cosine Transform

- DCT decomposes a signal into a series of harmonic cosine functions.

$$T(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)g(x,y,u,v)$$

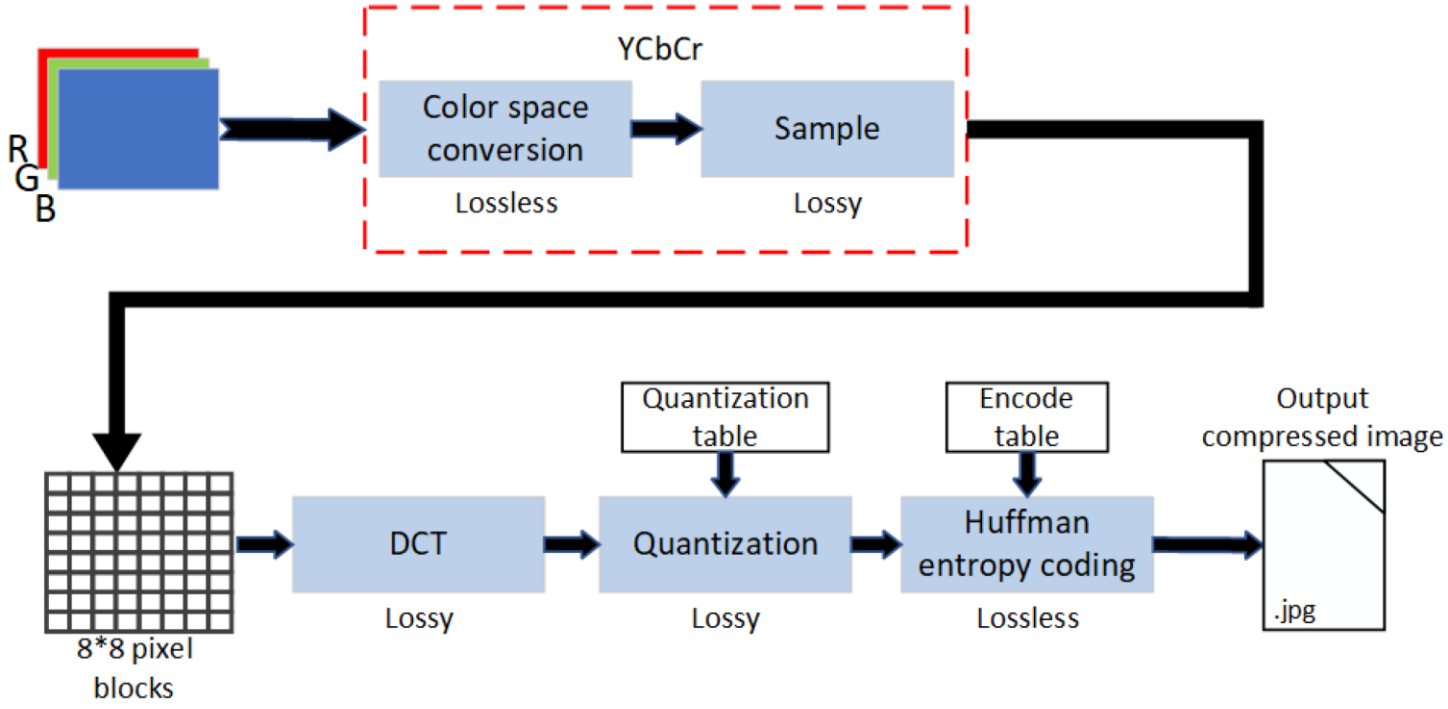
$$g(x,y,u,v) = \alpha(u)\alpha(v)\cos\left[\frac{(2x+1)\pi u}{2N}\right]\cos\left[\frac{(2y+1)\pi v}{2N}\right]$$

- The DCT transforms image data into frequency components, allowing us to embed data in parts of the image that the human eye is less sensitive to.

Some theory: JPEG compression

Lossy image compression method, removes details in high frequencies

1. Color space conversion
2. Block splitting
3. DCT
4. Quantization
5. Entropy encoding



Hiding data: the algorithm

- Steganography and digital watermarking are very different but shares the same operation pipeline with a slightly difference in embedding data.
- Both of them use DCT to select embedding locations but differ in the frequencies to manipulate.

- In the following slides the detailed algorithm explained.

Block-wise DCT Transformation

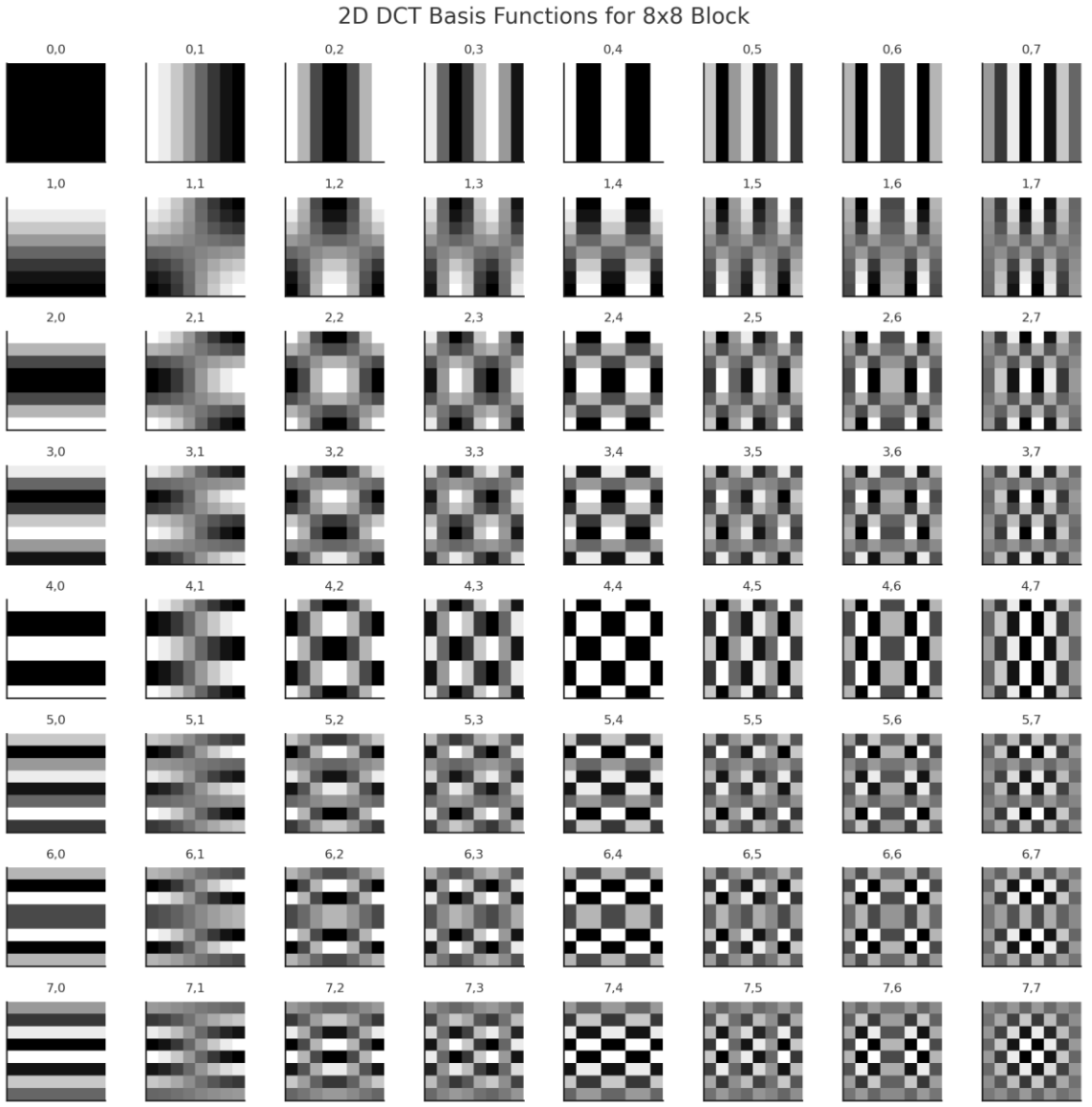
- The image is divided into 8×8 blocks.
- Each block is transformed using 2D DCT into a frequency coefficient matrix:

$$F(u,v) = DCT(f(x,y)) \text{ for each block}$$

- Now we have 64 coefficients per block:
  - (0,0) is the DC component (average intensity)
  - Others are AC components (varying frequencies)

Select Embedding Location

- Choose mid-frequency coefficients, such as (2,3), (3,2), (3,3), (4,1), (4,2), etc.
- Avoid:
  - Low frequency (e.g. DC): affects image too much.
  - High frequency: likely removed by JPEG compression or noise.



Embed the Message

Least Significant Bit: Steganography

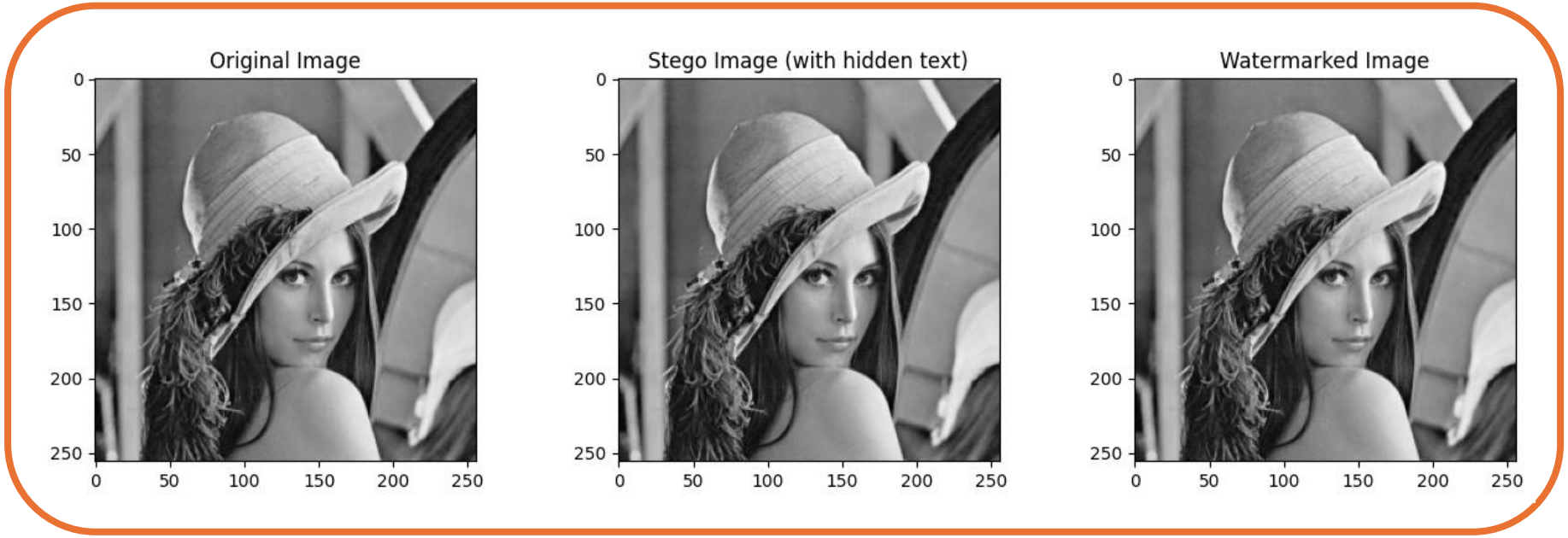
- Take the DCT coefficient, for example 113.44 and convert it to int: 113
- Modify the least significant bit:
  - 113 → 112 if embedding 0
  - 113 → 113 if embedding 1

Quantization-Based: Watermarking

- Embed watermark by modifying quantized coefficients:
  - If watermark bit is 1, round the coefficient to even
  - If 0, round to odd (or vice versa)

Inverse DCT and Reconstruction

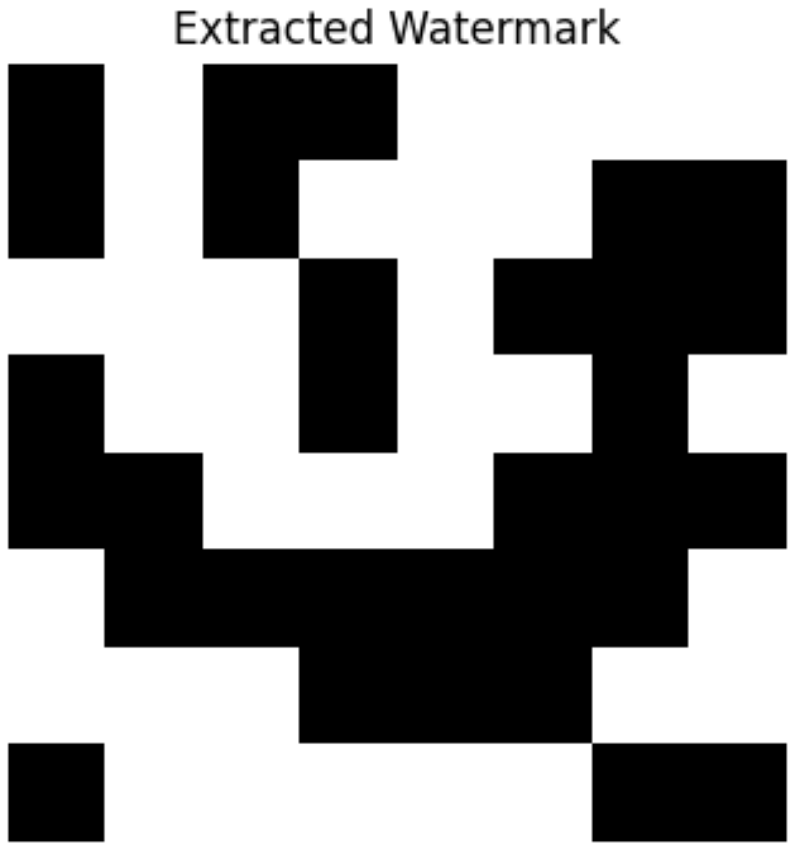
- Apply inverse DCT on each block.
- Combine all blocks → reconstruct the full image.
- The image looks nearly identical to the original, but secret data is embedded in its frequency domain.



Extraction

- Receiver divides image into same blocks.
- Applies DCT on each of them.
- Reads LSB or quantized bit from coefficients.
- Finally reconstructs original message or checks watermark (for ownership, tampering, …)

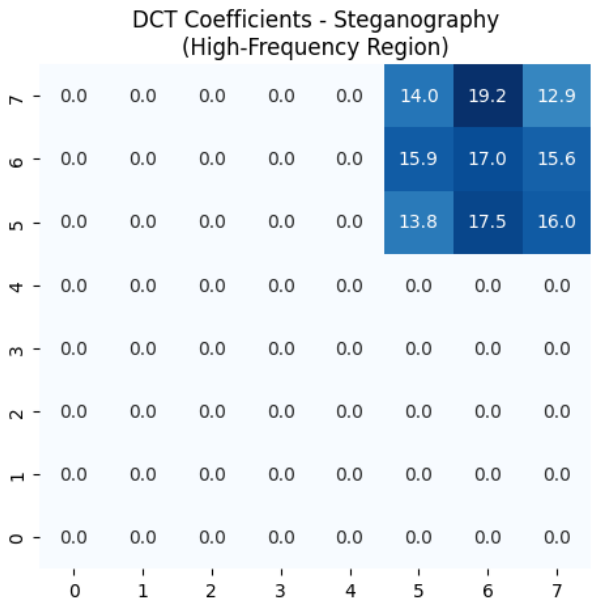
```
Extracted Message: 'Super secret meeting tomorrow at midnight!'
```

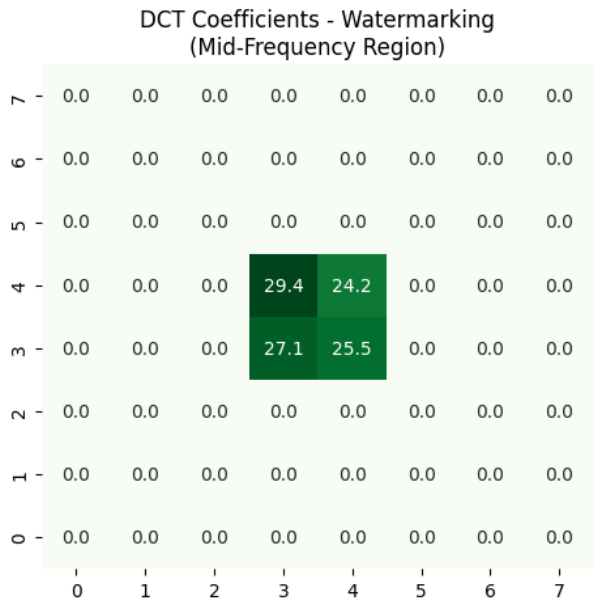


How about compression?

Both the techniques presented exploits DCT, which is used by JPEG, but the effect of compression on them are very different:

- Watermarks are embedded into mid-frequencies → message survives
- Steganography modifies high frequencies → destroyed!

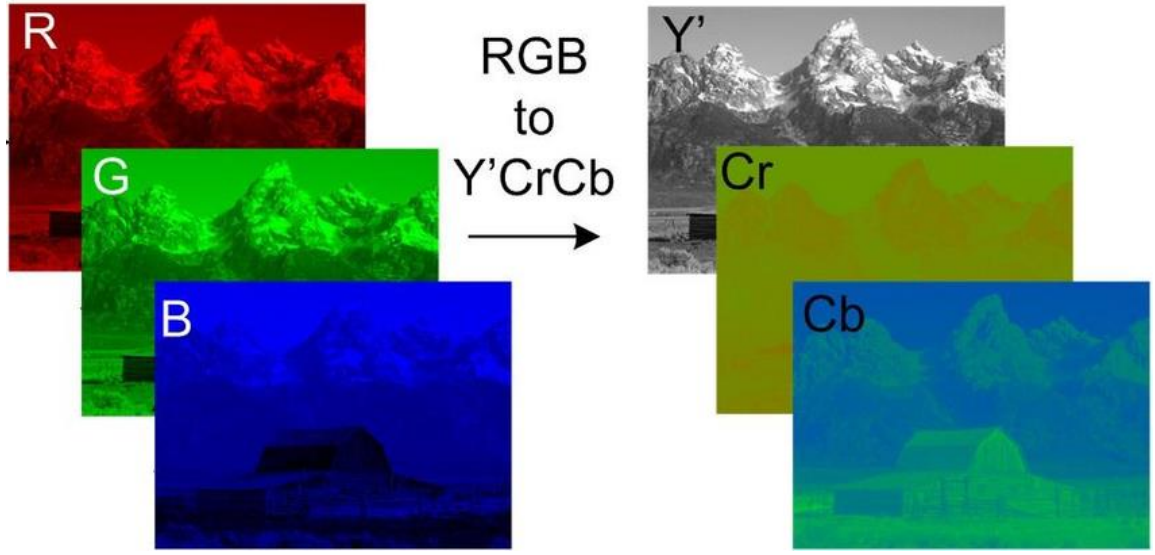





```
Message Before Compression:
Compression test: not so good!
Extracted Message After Compression:
$$f½$¦_¶6$¥]lütV
```

What about non-grayscale images?

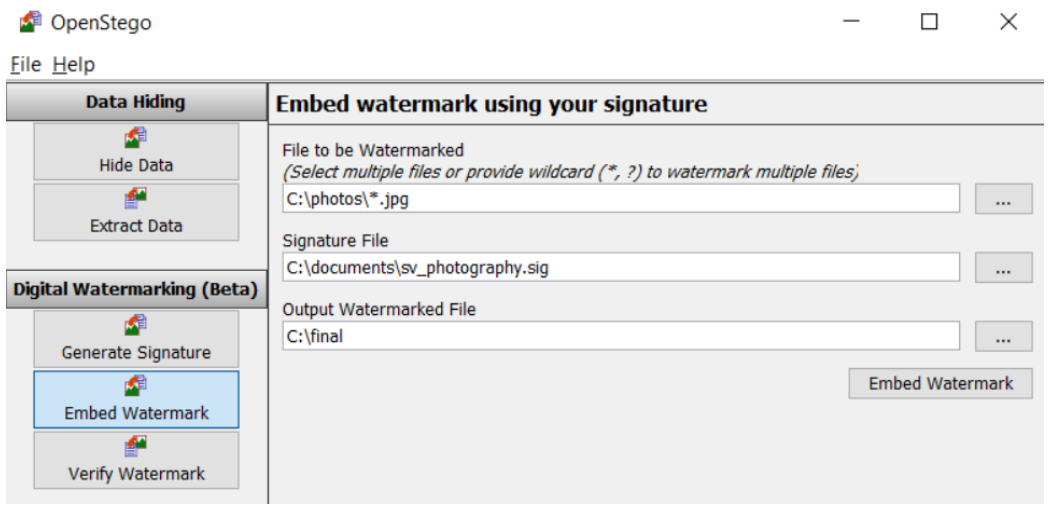
- We can manipulate each channel (RGB) separately, embedding data in them (3x more payload space theoretically) but…
- RGB isn't perceptually uniform! Changing Red and Green is more noticeable.
- A better option is to convert into YCbCr color space and embed data in chrominance components (Cb, Cr) to make it less noticeable.



Some tools…

- Steghide: specific for steganography, supports encryption, CLI-only
- OpenStego: supports watermarking, simple GUI
- Custom implementation with Matlab, Python…

```
$ steghide embed -cf picture.jpg -ef secret.txt
Enter passphrase:
Re-Enter passphrase:
embedding "secret.txt" in "picture.jpg"... done
```



Hands on with some code!

Thanks for the attention!