An analysis of Digital Steganography for Image Watermarking

Colceru Cosmin
Nadu Toma
Cristina Andreea Antonescu
Răzvan Adrian Filip

University of Bucharest

Faculty of Mathematics and Computer Science

Signal Processing Course
January 2024

Contents

Subject description

While many of the watermarks we are familiar with are simply layers of pixels which are overlaid onto an image, steganographic techniques make this rather annoying practice invisible to the naked eye.

Introduction to steganography

Generally speaking, steganography is the practice of concealing information within other information, in the form of digital data or physical objects, with the ultimate purpose of hiding that information's presence in its entirety. For thousands of years, steganography has been practiced in various forms for keeping communications private. For example, in ancient Greece, people would carve messages on wood and then use wax to conceal them (Provos et al., n.d.). The infamous invisible ink that we are all familiar with has been in use since at least the 4th century BC (Arnold & Macrakis, n.d.). Of course, modern approaches are centered around digital mediums instead, the most popular of which are documents, images, videos and audio, because of their large file size.

The two main categories of steganographic techniques used on digital images can be separated based on the domain in which information is hidden: spatial domain techniques and frequency domain techniques.

Given that steganography stems to make the hidden information as imperceivable as possible, the embedded messages can naturally be used to tag the covert media. As such, image watermarking becomes a valid use case for digital steganographic techniques.

Steganography, cryptography, compression, obfuscation

While steganography strives to make information hidden through obscurity, cryptography aims to hide the content or meaning of said information, not its presence altogether, most commonly by turning plaintext messages into unintelligible nonsense. Even so, their common goal remains secure communication.

When it comes to compression, the main goal is encoding the original information in such a way that fewer bits are used in its representation. The decoding process can either yield an identical reconstruction of the original data, in the case of lossless compression, or a 'close enough' representation, with acceptable amounts of information loss, in the case of lossy compression. Either way, the purpose of compression is not security, but storage reduction.

Obfuscation seeks to obscure the meaning of information, not by hiding the information itself, but by deliberately making the message confusing and hard to interpret or decode.

Though related and sometimes complementary, it's important to point out that these practices serve different purposes.

Justification for the chosen subject

Steganography, as a force of good, can be indispensable in certain situations, such as the tracking of illegal use of copyrighted material. Let's say you're an online magazine which employs dozens of photographers. Before publishing an article, there is some risk for original photos to get leaked. To prevent this, you can create digital signatures which certify the ownership of each photo, as well as the identity of people authorized to access said photo. Using a steganographic technique, these signatures are embedded into every photo taken by every photographer. If a photograph gets leaked, the magazine can not only prove ownership, but it can also pinpoint the precise source of the leakage (the employee who is guilty). A real world example of copyright enforcement measures using steganography is the use of undefined and undocumented instructions in the first models of x86 Intel CPUs (Pires, 2023).

Another valid use case is avoiding censorship. A lot of territories around the world have strict laws against free speech, and some even ban encryption altogether (*Cryptography Law*, n.d.). Using steganography, citizens are able to communicate without the restrains enforced by authoritarian regimes.

In certain scenarios, steganography can also be used in the storage of classified information by law enforcement. When national security is at stake, hiding certain information can be of utmost importance against a common threat.

At the same time, steganalysis is extremely important in detecting unlawful use of steganographic techniques, of which there are many. Steganography has been used as a vector of attack, through which arbitrary information hidden within files becomes a malicious payload, which is then extracted on the target host (*What Is Steganography? Definition and Explanation*, n.d.). Multiple CVEs are based on this simple principle.

Naturally, distribution of illicit content is also of great concern. Anything from illegal pornographic imagery to detailed instructions on drug manufacturing can be distributed using these methods. Espionage, through the extraction of valuable confidential information about a company or state, is another form of such distribution.

As such, studying both disciplines is important for a lot of applications, not only for image watermarking. Also, they're simply fascinating, so they can be intrinsically worthwhile to explore.

Technical details

Principles behind spatial domain techniques

Altering the spatial domain of images is suitable for raster-graphic image formats, such as BMP and PNG, which store the pixel values themselves, with varying amounts of compression, instead of their frequency domain representation, which is used in formats such as JPEG.

The method we will describe is LSBR (Least Significant Bit Replacement). It involves modifying the least significant bits of the value(s) in each pixel. Although quite simple, this technique can be very effective, thanks to the relatively poor visual perception of humans. Essentially, when modifying least significant bits, the values of the pixels change very slightly. Depending on the number of pixels b which are modified, the value of each pixel will change by at most $+/-(2^b - 1)$. In practice, the maximum value for which the covert data remains imperceptible will differ based on the structure of the image. Based on the tests we performed, for high quality photographs, this value should never be greater than 2, while for low dynamic range, low-definition images, we may go as high as 4. Of course, the storage capacity of the cover image is directly proportional to b, so it is often the case that a tradeoff must be made between storage capacity and perceptibility.

Choosing the pixels which get modified, and their order, is another detail worth discussing. The most basic strategy is modifying pixel values sequentially, but this approach makes visual attacks extremely effective. Custom ordering schemes can be used to make both detection and extraction more difficult. For example, a random sequence of three-dimensional coordinates of the length of the covert data can be generated and distributed to the intended receiver through some secure means. The idea is to use only the pixel values with those specific coordinates, in sequential order. This was just one example, and many other creative schemes can be developed.

Another technique, which improves on the statistical shortcomings of the LSBR method, is the LSBM (Least Significant Bit Matching) method. The main difference between the two is that each pixel value which doesn't match its associated covert bit in the least significant bit, is either incremented or decremented randomly. This approach has proved to be more robust against visual and statistical attacks (Ker, 2016).

Principles behind frequency domain techniques

Images can be interpreted as bidimensional signals, which means we can use techniques such as the Discrete Fourier Transform, in order to obtain their representation in the frequency domain. The resulting signal transform can then be used to apply many of the same strategies suitable for spatial domain embeddings, as previously described. As we will show in the later sections, small alterations in the frequency domain can be much less noticeable, when using certain metrics.

JPEG is the best candidate for a cover image format type, as it is widely used for storing arbitrary types of images, and it is the dominant format for photos produced by smartphones and other consumer-grade digital cameras, therefore it is less likely to attract attention.

The JPEG specification describes taking 8x8 blocks from the original image, and applying the Discrete Cosine Transform on each of them.

The first problem which arises is whether or not to use all of the coefficients of a block. The main assumptions made in the case of JPEG are that in the top-left corner of the resulting 8x8 DCT blocks, which contains information about the lower frequency pairs, the coefficients have a much higher visual impact on the overall image, because lower frequencies are more likely to occur in real, natural images. This is why any change in the coefficients located in that particular spot are more likely to be caught by the human eye, which means that altering values in that region should be avoided. JPEG takes advantage of this by quantizing the other regions of the blocks, so that a lot of coefficients are reduced to 0, which make the blocks more easily compressible.

Even though most of the coefficients in regions of the block other than the top left corner are null, greater values do occur. That is where the desired information can be hidden. The threshold which separates the top-left corner from the rest of the block may be determined on a case by case basis. We propose a simple configurable sliding diagonal. Given a value of the parameter *p*, the coordinates of the threshold can be determined as such:

$$j = \max(0, 8 - i - p), \forall i \in \{1 \ldots 8\}$$

There are a myriad of strategies to modify the coefficients. Our focus will be on changing their least significant bit, an approach used by the JSTEG algorithm. We should do so, carefully. When the value of a coefficient is either 0 or 1, and the bit which will be inserted doesn't match the least significant bit of the coefficient, the 0s become 1s and vice-versa. Many of the coefficients in the DCT block are naturally 0, and most data will contain a comparable number of 1s and 0s, which means that the naturally occurring 0s will be flipped into 1s. This would be a disaster, because even though the decoded image might look fairly similar, any actor performing steganalysis will check the DCT blocks and immediately tell that the image has been modified. To avoid this situation, both 1s and 0s will remain unchanged. Altering the blocks this way is bound to create statistical imbalances (Ortiz, 2014), which make this technique fairly detectable.

Another famous algorithm is F3. F3 improves on the statistical shortcomings of JSTEG, by decreasing the magnitude of the coefficients, whenever the last significant bit doesn't match the bit in the covert data, instead of simply flipping it. The effect is that negative coefficients are incremented, while positive coefficients are decremented, instead of all values being decremented, as it were the case with JSTEG.

Steganalysis techniques

The goal of steganalysis is to identify suspicious media, to determine whether or not it has data encoded into it, and, if possible, recover that data. The simplest method to detect steganography in files is to compare them to known originals, but this is not always possible.

Steganalysis often relies on statistical analysis to detect patterns or deviations from expected statistical properties. For example, changes in the distribution of pixel values, frequency domain analysis, or other statistical measures can be indicative of hidden information. Other approaches include visual attacks, structural attacks, file format analysis and metadata examination.

The visual attack is the most basic steganalysis technique there is. In unmodified images, the last bits of the values of the color channels in each pixel usually preserve the overall structure of the image. We can isolate the last bit or two in each color channel (by filling the rest of the bits with 0s) and visually inspect the result. If it seems to be completely random noise, it is most likely a sign that an LSB has been used, and we are viewing the embedded message. If only the first portion of the image appears to contain random noise, while the rest preserves the structure of the image, it is certain that there is steganographic content within the image.

The chi-squared (χ^2) test is a statistical test that is used to determine if there is a significant association between two categorical variables. It is commonly employed to assess whether the observed distribution of categorical data differs from the expected distribution. The test is based on the difference between the expected and observed frequencies in a contingency table. This test can be used to detect both LSB and DCT steganography in images. The first step in this test is to formulate the Null Hypothesis, usually that there is no significant difference between the observed and expected frequencies, and the Alternative Hypothesis, that there is a significant difference between the observed and expected frequencies. Then we organize the data into a contingency table, which displays the frequencies of the different categories for each variable. The next step is to calculate the expected frequencies for each cell in the contingency table based on the assumption that there is no association between the variables. After this, we can calculate the chi-squared statistic using the formula:

$$\chi^2 = \sum \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

O_{ij} is the observed frequency in cell (i, j) and E_{ij} is the expected frequency in cell (i, j). After calculating the statistic we can determine the p-value, and, if it is less than the chosen significance level (commonly 0.05), reject the Null hypothesis. Otherwise, fail to reject the Null hypothesis.

In the context of LSB steganography, the chi-squared test can be applied to analyze the distribution of pixel values in an image. We begin by formulating the hypotheses, where the Null hypothesis says that the distribution of least significant bits

in the pixels of an image aligns with expectations in a non-steganographic scenario, with an equal likelihood of 0s and 1s. The Alternative hypothesis suggests a significant discrepancy between the expected and observed LSB distributions, indicating potential LSB steganography.

In the case of color images, we perform the chi-squared test in the same manner, separately for each color channel. If any of the p-values is smaller than our threshold there is a possibility that the image has steganographic content.

Having established the functionality of the chi-squared test, it is essential to examine its limitations. Firstly, the effectiveness of the chi-squared test can be influenced by the characteristics of the cover image. Images with high entropy or complex patterns may exhibit altered LSB distributions even without steganography, leading to potential false positives. Also, the chi-squared test may be less reliable when detecting small-sized hidden payloads. If the embedded information is minimal, the alterations to the LSB values might not be significant enough to trigger the test, leading to false negatives. The test assumes that the LSB values of pixels are independent of each other. In some steganographic methods, the alteration of one pixel's LSB may depend on the values of neighboring pixels, violating this assumption.

The chi-squared test can also be used in the frequency domain to detect JSTEG, but it is not as effective. We perform the test in a similar manner as for LSB, but this time we look at the frequencies of the least significant bits of the DCT coefficients, excluding the ones that have the value 0 or 1.

Several studies have shown that machine learning can be very effective for steganalysis. The specified machine learning model is trained on a dataset containing both non-steganographic images and images with information embedded inside them (using one or multiple steganography methods). This way, the problem of steganalysis can be addressed as a classification problem, where the images are classified as steganographic or non-steganographic, based on the acquired training patterns. The machine learning model can be trained to detect anomalies in the distribution of pixels or of the DCT coefficients, or to identify other patterns or structures in the image.

A number of classification methods have been proposed for JPEG steganalysis. These include the use of algorithms such as support vector machines or logistic regression. In recent years, neural networks have resulted in much better detection than the traditional classifiers. Variants of convolutional networks such as XuNet, ResNet, DenseNet or AleksNet are most often used for this purpose.

Moreover, researchers have delved into experimenting with various image pre-processing techniques, such as compression and resizing, before passing them through the detection models. The results have shown that these pre-processing steps can significantly impact how well the model identifies hidden messages. Some techniques enhance detection accuracy, while others may reduce it.

In a recent study (Płachta et al., 2022), their ensemble classifiers achieved 99.9% accuracy in detecting the F5 algorithm and 56.3% for the J-Uniward algorithm. They used a variety of shallow and deep learning algorithms for analyzing the images and concluded that the performance depended heavily on the steganographic method used and on the density of the hidden data.

Comparison between spatial and frequency domain techniques

Through careful experimentation, we have determined in which of the main areas of interest — storage capacity, detectability, and suitability for a given image format — the two technique types differ.

It becomes quite apparent that, when storage capacity is a major concern, spatial domain techniques should be considered in the detriment of their frequency counterparts. As we will conclude later in the report, LSBR can achieve a storage capacity even 100 times larger than JSTEG on the same image. The large difference mainly stems from the nature of the DCT blocks in the JPEG versions. Depending on the image, quantization may reduce most coefficients to either 0 or 1 in the majority of the DCT blocks, for pictures with very few high frequency components, which shrinks the available pool of coefficients that can be modified. This becomes even more problematic for low p perceptibility values, which set the boundary for coefficients more towards high-frequency components, and which disqualifies the few low-frequency components that might be present. That is the case with the photograph of a chess table which we used for experimentation, for which changing p from 1 to 8 increased the storage capacity from 0.1KB to 24KB. On the other hand, when using LSBR in sequential order, all pixel values can be altered, and the difference can very rapidly add up. For the same chess photo, the storage capacity shot up to 2MB, when 2 least significant bits were modified.

What spatial domain techniques win in terms of capacity, they give away in detectability. When an adversary is actively analyzing photos, frequency domain techniques prove to be much harder to spot. For instance, LSBR is very susceptible to visual attacks, even to the most basic ones, while JSTEG renders them largely ineffective. In our tests, JSTEG also proved to be more resistant to the chi-squared test.

Lastly, whether or not a type of technique can be applied on an image depends on the format of that image. If we were to apply the LSBR method on an image stored as a JPEG, a JPEG 2000, or a lossy WebP, all of which store information, either partly or wholly, in the frequency domain, we would first have to decode the image, by applying various inverse transforms, such as the IDCT or IDWT, in the case of JPEG and JPEG 2000, respectively. Next, we would modify the pixel values we obtained, and save the resulting image. During the encoding process though, there are a few factors which would render our modifications useless. For one, DCT is applied on each block, which introduces precision errors — floating point, system architecture and

implementation related —, and for another, when the coefficients are quantized, our changes would be totally lost, because they wouldn't account for any change in the final value of the coefficients, due to the large values in the quantization table by which they are divided. As such, spatial domain techniques aren't suitable at all for these formats. Frequency techniques aren't great for other types of formats, which store the image data in the spatial domain, either. The precision errors encountered previously would still occur when applying transforms and inverse transforms. In both cases, the only way in which we can guarantee the integrity of the covert data, no matter the type of compression being used (lossy/lossless), is by ensuring that no major precision error prone operation occurs between the embedding and the extraction processes.

Methods for assessing quality and similarity

In order to assess the amount of dissimilarity between the original and the modified images, we will use a number of both visual and numerical metrics.

Histograms of relevant images will be displayed, in order to determine the impact of both LSBR and JSTEG on the resulting stego images.

The Peak Signal-to-Noise Ratio (PSNR), often used to measure the reconstruction fidelity for images subject to lossy compression, is another metric with which we can quantify the impact of our steganographic techniques, by considering the original image as the signal, and the error introduced by the steganographic process as the noise. Generally speaking, the higher the PSNR, the higher the similarity between the images, which would translate to reduced perceptibility.

$$PSNR = 10 \times \log_{10}\left(\frac{MAX^2}{MSE}\right), where\ MAX = 2^B - 1$$

In this case, *B* is the number of bits used in the binary representation of a pixel value.

Due to the fact the PSNR can sometimes not be indicative of the overall perceived similarity between two images, even though it accurately measures absolute dissimilarity, we also used the Structural Similarity Index Measure (SSIM), which also takes into account the structural and the textural characteristics of images.

$$SSIM(x,y) = \frac{\left(2\mu(x)\mu(y) + c_1\right)\left(2\sigma(x,y) + c_2\right)}{\left(\mu^2(x) + \mu^2(y) + c_1\right)\left(\sigma^2(x) + \sigma^2(y) + c_2\right)}$$

where μ(x) = mean of x, σ(x) = variance of x, σ(x, y) = covariance of x and y, $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$, $L = 2^b - 1$, $k_1 = 0.01$ and $k_2 = 0.03$.

Preserving watermarks after cropping

Cropping is a common practice when trying to evade traditional watermarks, but using a sufficiently small digital watermark, enough copies of that watermark will remain embedded to confidently assess the provenance of a particular image, even when cropped.

Given that the image might be cropped in such a way that the first bit of the embedded data is from within an original byte, and not the actual start of a byte, we can count the number of occurrences of a given watermark by leaving out a number of bits from the beginning of the extracted sequence at a time, from 0 to 7, so we can make sure that the sequence of bytes begins from the correct bit.

Using encryption in tandem

Even though encryption is a separate practice and field of study, we can use it to protect the contents of the covert data, even when detected or extracted. Symmetric cryptography schemes can be used when the goal of the embedding is communication between two parties who can exchange keys in advance, but asymmetric cryptography may be more suitable for copyrighting purposes, as it is broadly used to determine the authenticity and provenance of a message or of an authority, through Public Key Infrastructure (PKI). Organizations with valid certificates may use their private key to encrypt watermarks. When decrypting the watermark using the public key associated with that certificate, its authenticity is also proven by default.

Libraries and Packages

1. **bitarray**

   Allows manipulation of data at the bit level, not the byte level (which python supports), which is useful when embedding and when extracting data, because up to 4 bits have to be used separately.

2. **jpeglib**

   Lots of libraries allow reading JPEG images. The problem is that almost all of them do so by decoding the file back to pixel values. However, we only needed a package which would decode the Huffman code, and place the DCT blocks into a numpy array.

3. **matplotlib**

   We mainly used it to generate histograms of the original and the stego images.

4. **openCV**

The main functionalities we used are reading and writing both PNG and JPEG images from and to the disk.

5. **pyexiv2**

This package fulfilled the need to hide various parameters — size of embedded data, number of least significant bits/perceptibility value — into EXIF metadata, for subsequent extraction.

6. **scikit-image**

We used the built-in SSIM implementation to assess structural similarity.

7. **scipy**

The chi-squared implementation was used for watermark detection.

8. **stego-toolkit**

This is an open-source project distributed as a docker image, which brings together popular steganalysis projects, such as stegoVeritas, zsteg, stegdetect and stegbreak.

Results

In order to conduct our tests, we chose a portrait, a picture of the sky and a picture of a chess board, to represent common and relevant photo categories. The watermark we used as the covert data is *'This is the property of John Smith. This specific copy was given only to Alice and Bob'.*

Storage capacity based on different parameters

Portrait, 3000x4500 pixels

Number of significant bits	1	2	3	4
Storage capacity (bytes)	5062500	10125000	15187500	20250000

Perceptibility	1	2	3	4	5	6	7	8
Storage capacity (bytes)	676973	826670	962497	1080263	1181186	1263524	1326918	1365099

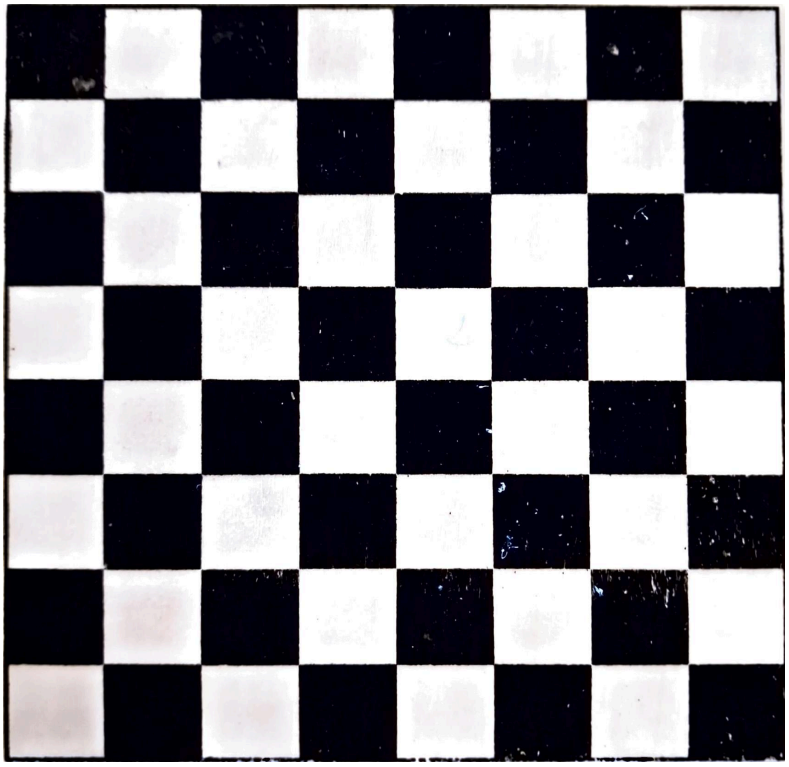
Given that the portrait contains high-frequency components, the storage capacity in the case of LSBR is only roughly 20 times larger than in the case of DCT.

Sky, 2560x1922 pixels

Number of significant bits	1	2	3	4
Storage capacity (bytes)	1845120	3690240	5535360	7380480

Perceptibility	1	2	3	4	5	6	7	8
Storage capacity (bytes)	80407	107178	134841	161206	185160	205209	222554	236955

As the high frequency components diminish, so does the ratio of the storage capacity. The LSBR capacity is now 31 times larger than the DCT capacity.



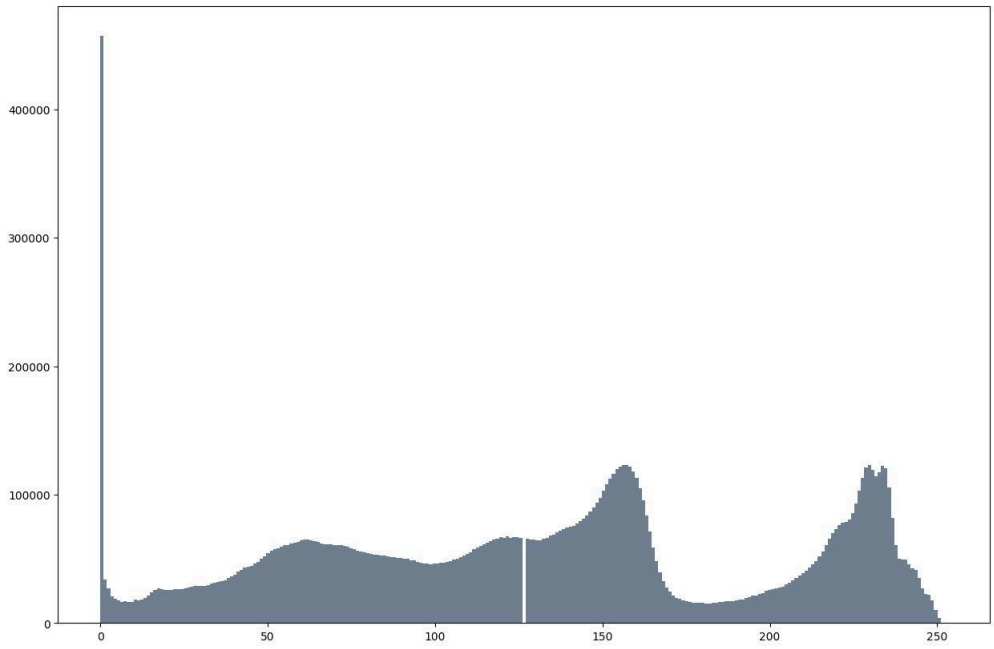
Chess board, 1700x1647 pixels

Number of significant bits	1	2	3	4
Storage capacity (bytes)	1049962	2099925	3149887	4199850

Perceptibility	1	2	3	4	5	6	7	8
Storage capacity (bytes)	185	918	2520	5057	8473	12693	17173	24159

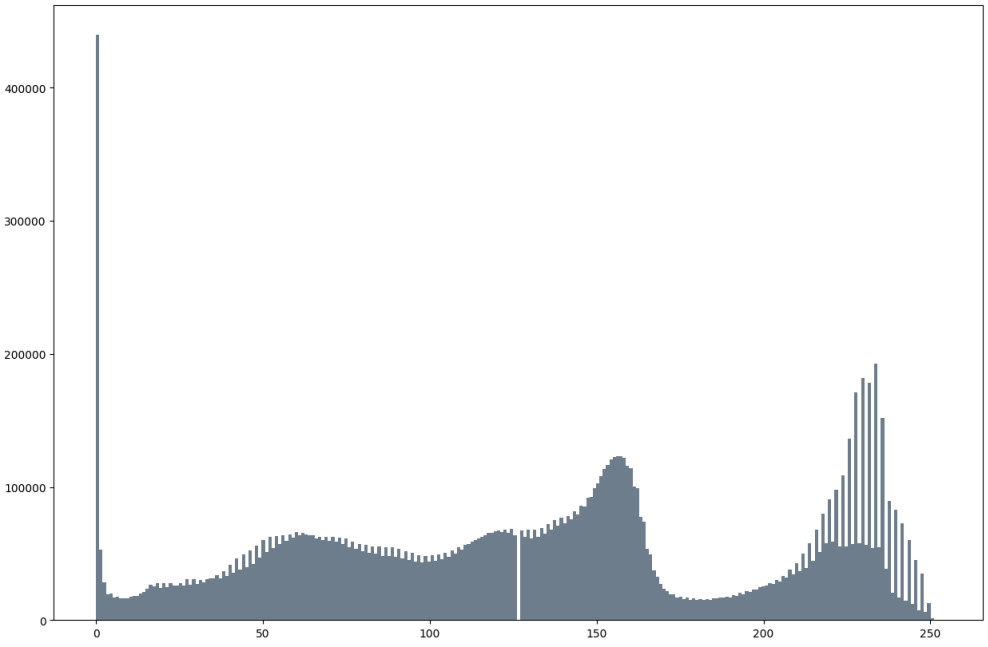
We intentionally photographed the chess board in low quality, both in terms of the quantization table applied and in terms of the overall definition and sharpness, in order to show that the acceptable number of significant bits which can be modified using LSBR is image dependent. At the same time, in this example, the storage capacity ratio is stretched to the extreme, to a whopping 173x, also due to the absence of high frequency components.

The effect on perceptibility of various parameter values



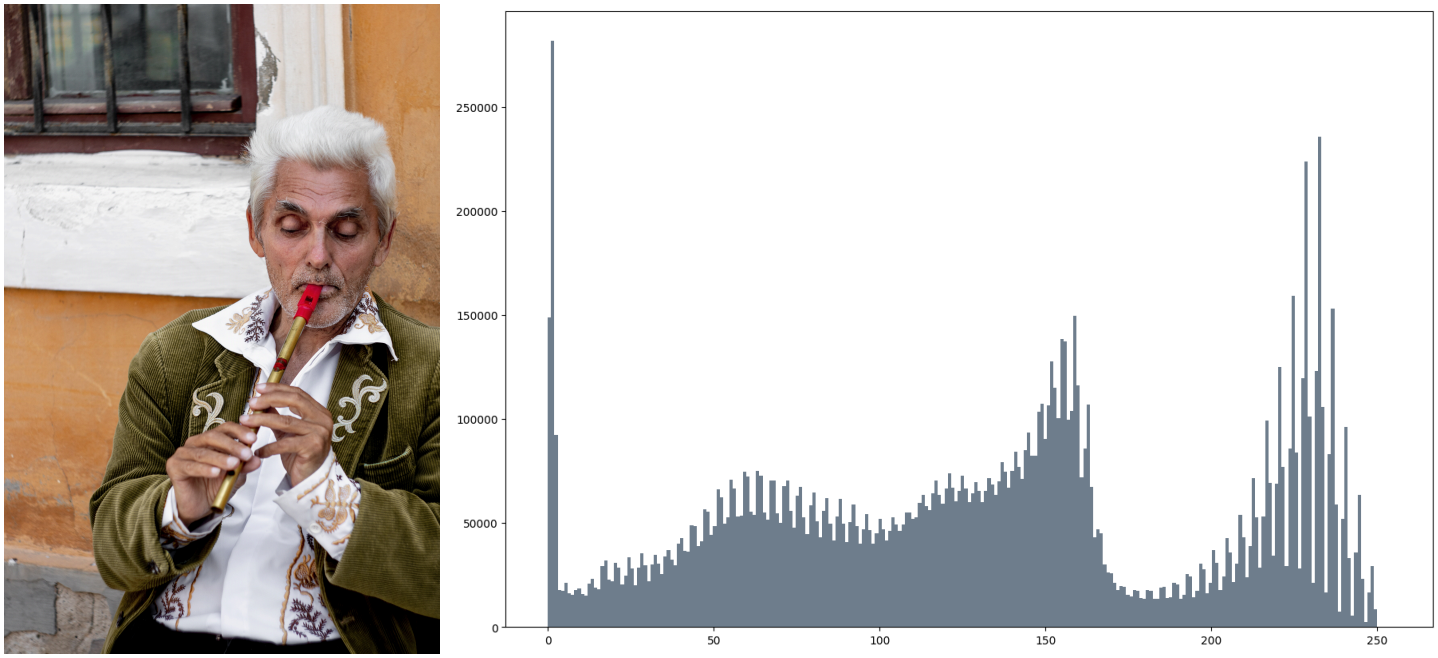
Original image

- Embedding using LSBR



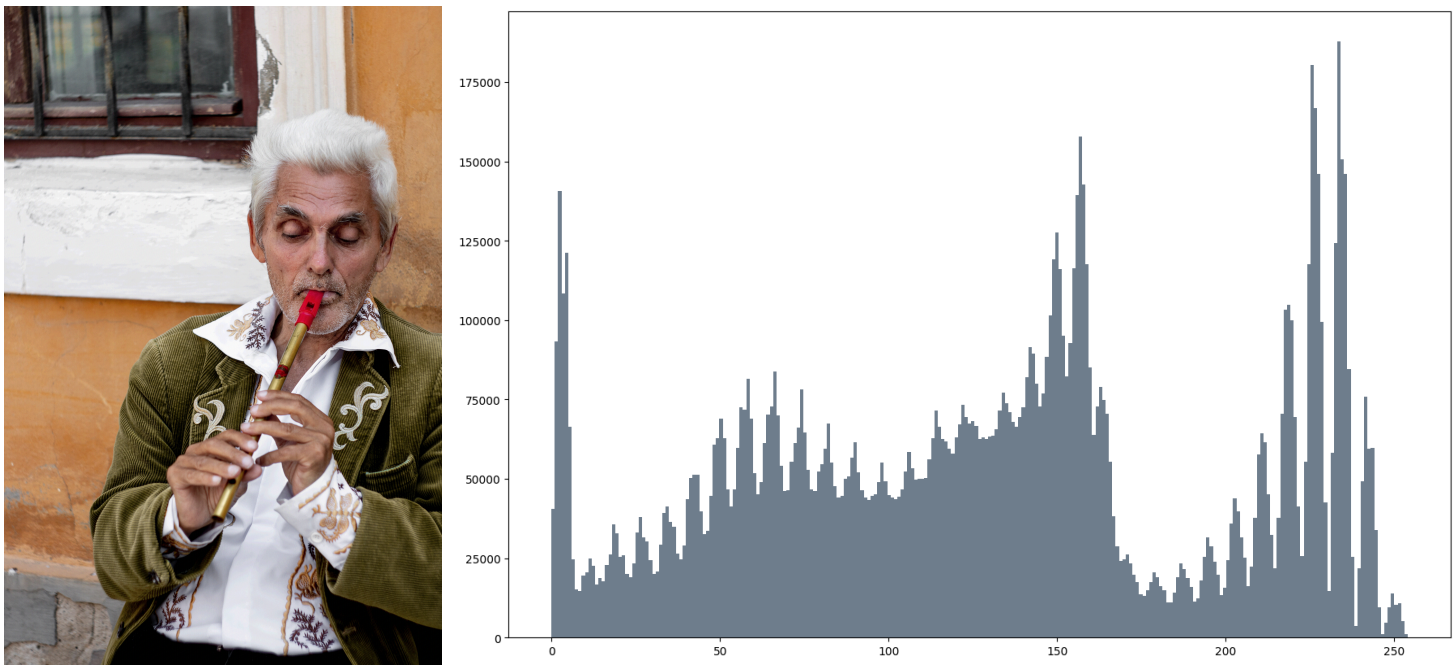
At 1 LSB. PSNR = 51.16dB. SSIM = 1

The overall shape of the histogram is maintained, and the structural similarity is maximum.



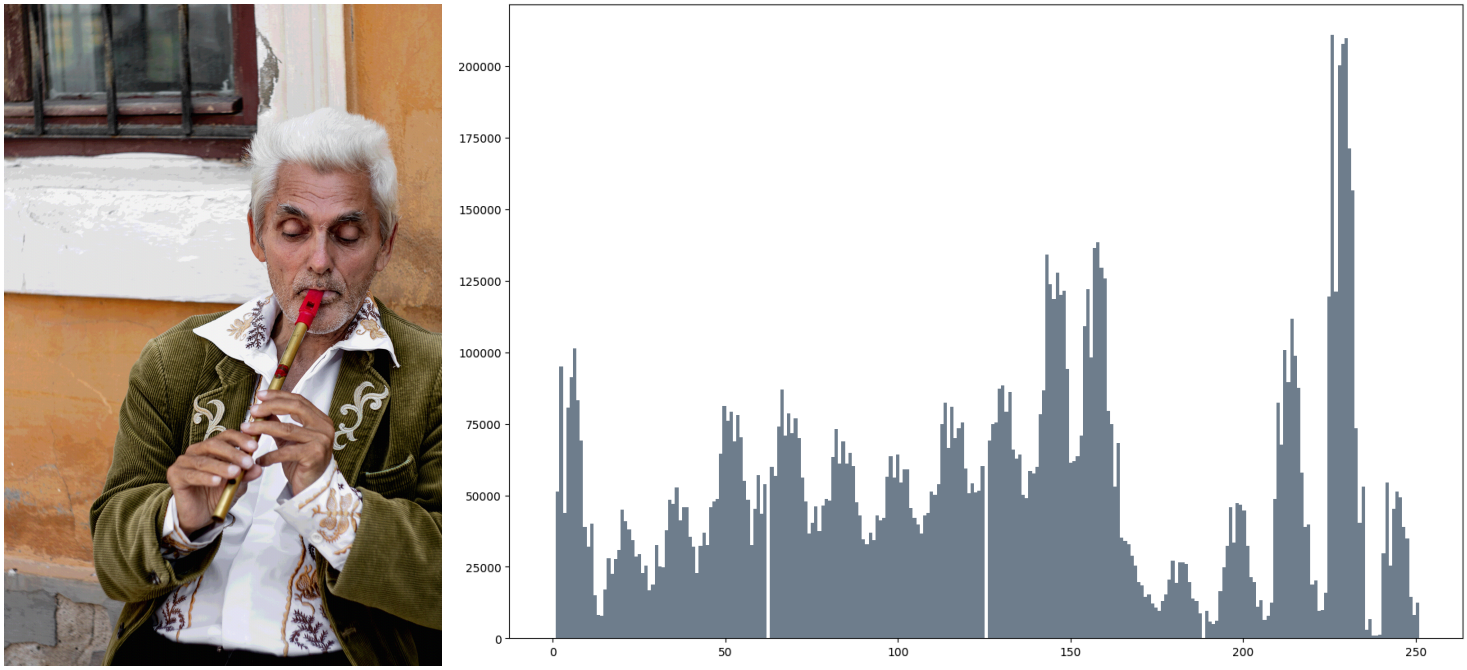
At 2 LSBs. PSNR = 44.41dB. SSIM = 0.98

We begin to notice a more intense alteration of the histogram, the PSNR drops, and the two images begin to be slightly dissimilar, but still not noticeably so.



At 3 LSBs. PSNR = 37.93dB. SSIM = 0.94

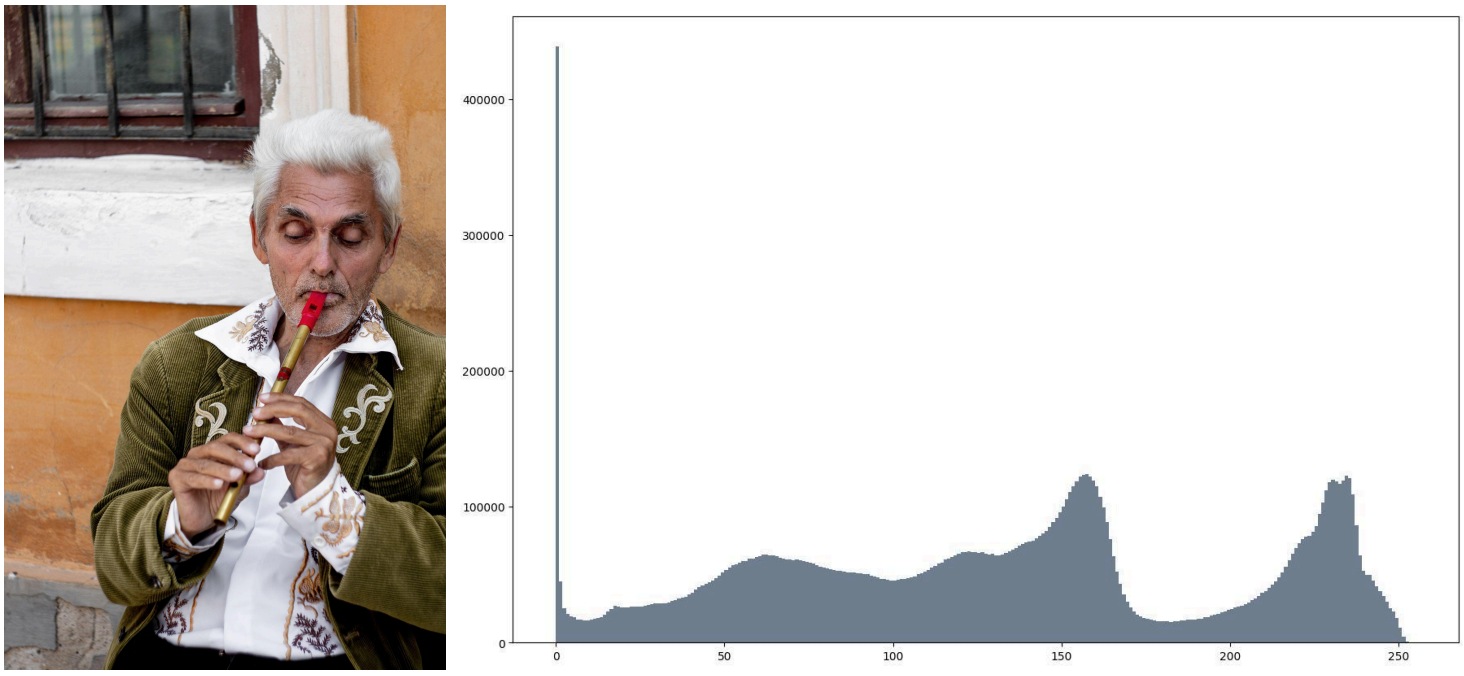
When displayed on a large enough screen, it becomes quite obvious that the colors blend in unnaturally and abruptly, even without access to the original image.



At 4 LSBs. PSNR = 32.20dB. SSIM = 0.85

The histogram hardly resembles the original, and colors in the walls are noticeably distorted. Structural similarity is significantly affected.

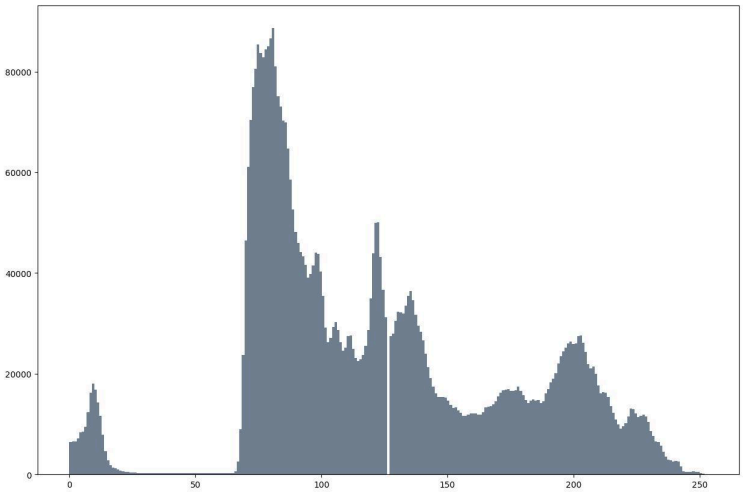
- Embedding using DCT



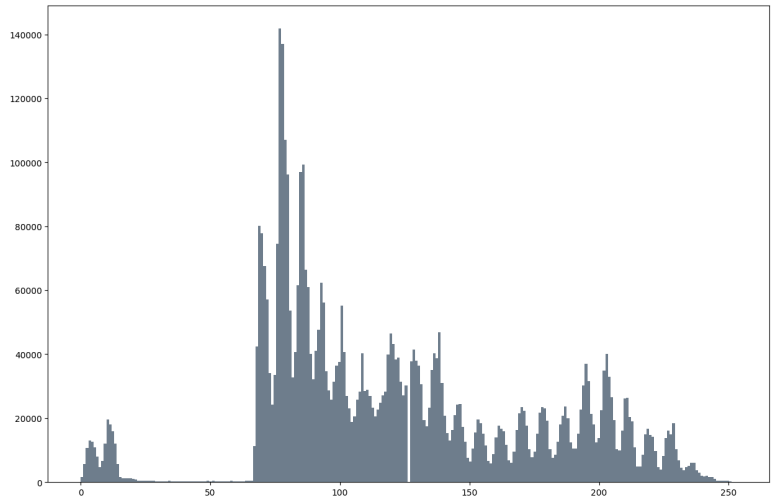
At perceptibility 8. PSNR = 48.53dB. SSIM = 1

Even when modifying all the values in the DCT blocks, the histogram remains more or less identical, and the structural similarity is maximum. Only the PSNR betrays the fact that the images aren't one and the same. In fact, all the images on which we used the DCT technique are unrecognizably similar to the originals.

- More examples of LSBR steganography

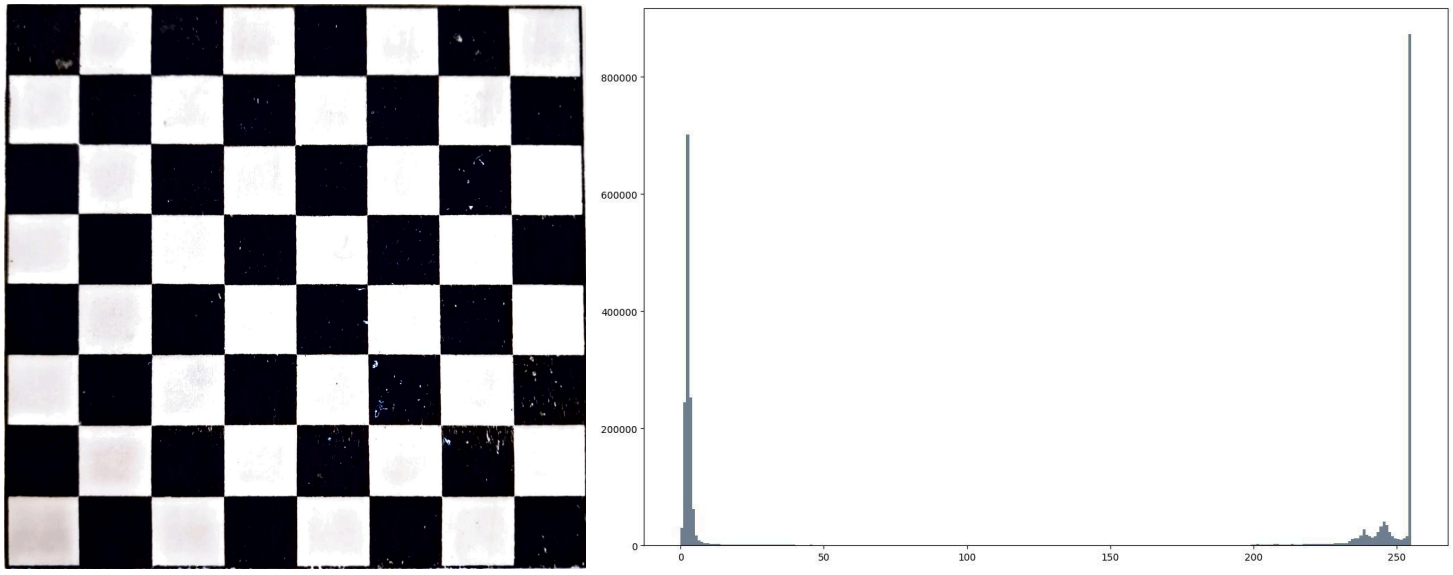


Original image

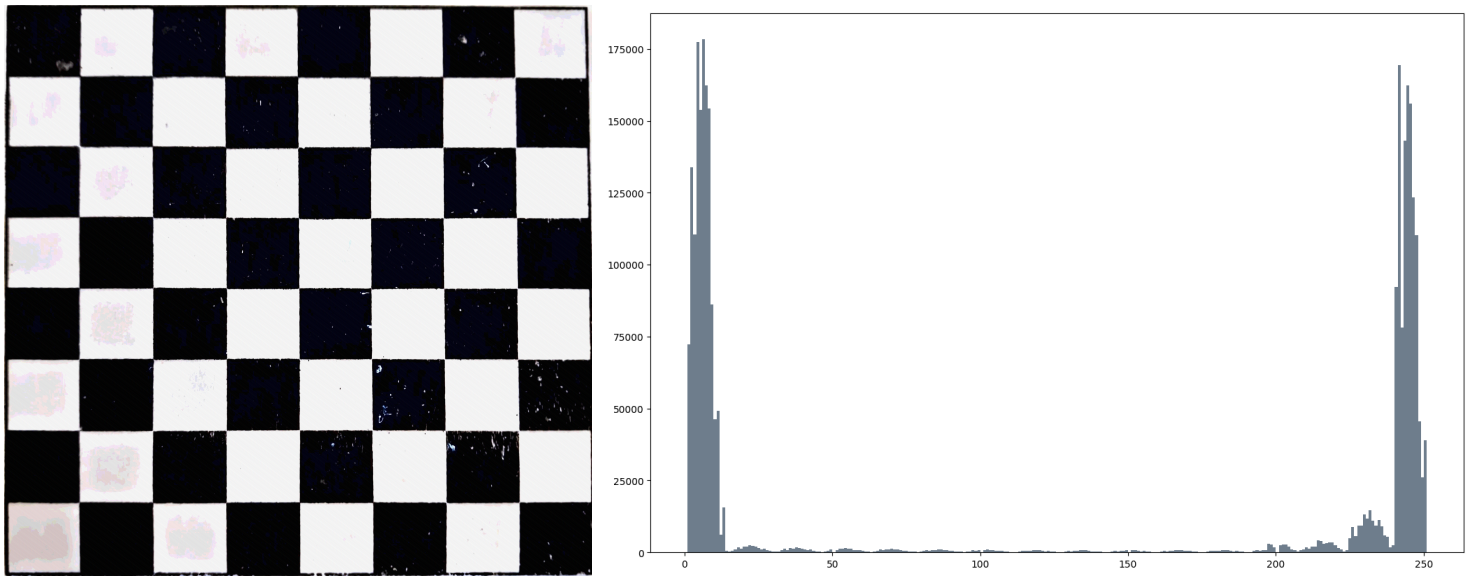


At 3 LSBs. PSNR = 38.01dB. SSIM = 0.94

Even though the SSIM is high, significant and noticeable bending occurs in the sky. It should be noted, then, that surfaces even in texture and color exacerbate the artifacts.



Original image

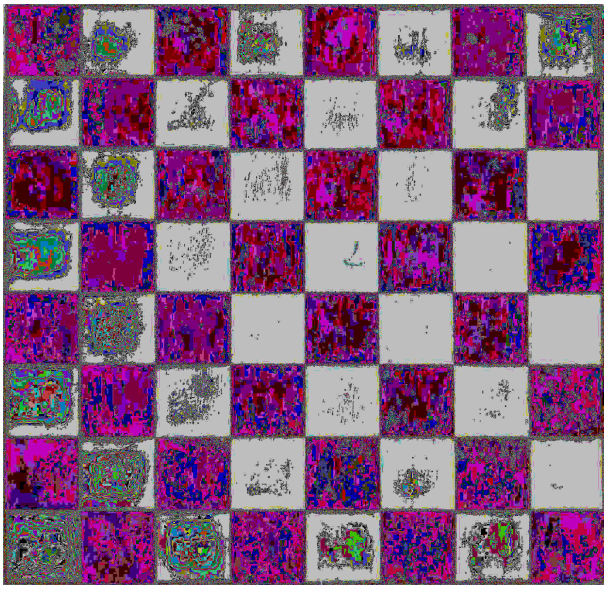


At 4 LSBs. PSNR = 30.46dB. SSIM = 0.77

Besides some color aberrations, I would consider the modified image passable, especially if the original was not available, as the artifacts could be considered a result of scanning, not of steganography.

Visual attacks and the chi-squared test

- Visual attack efficacy on the LSBR method

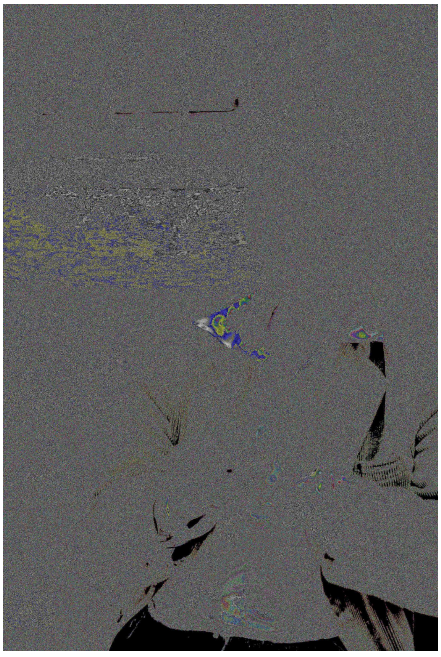


Last 2 LSBs of the original image

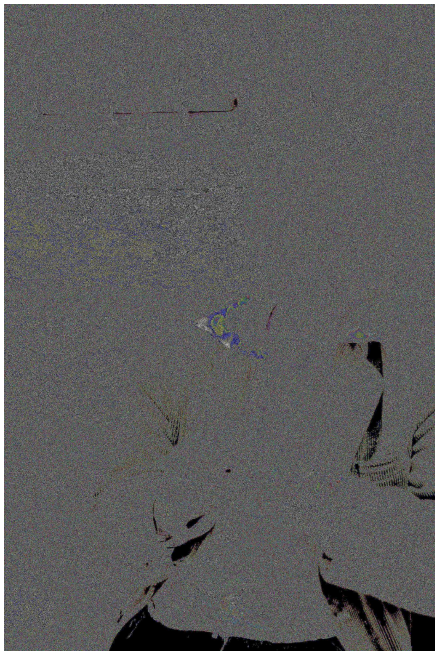
Last 2 LSBs of the stego image

The visual attack images had their pixel values boosted (left shifted by 6 bits), in order to make the difference more noticeable. Given that the covert that is a watermark, the repeating pattern clearly emerges, but with other arbitrary pieces of data, it would look much more similar to white noise.

- Visual attack efficacy on the DCT method



Last 2 LSBs of the original image



Last 2 LSBs of the stego image

Unfortunately, this detection method is not effective on frequency domain techniques, as expected.

- Chi-squared efficacy on the LSBR method

Applying the chi-squared method on the image of the sky, we obtain the p-value 0.18, which is higher than our threshold of 0.05, so the test indicates that there is no evidence of LSB steganography.

Next we embed the watermark described earlier all over the original image, using only the least significant bit. We perform the chi-squared test on the new image and the p-value resulted is 9.6645×10^{-7}, which rejects the Null hypothesis and indicates that there is hidden information in the image.

Recovery of watermarks from cropped images

Using the method described in the *Technical details* section, we embedded the watermark within the portrait (perceptibility p=8) and an encrypted watermark within the image of the sky (number of least significant bits b = 1), cropped both images to their middle third in height and width, extracted the contents and counted the number of occurrences of each type of watermark.

In the portrait, we found 205 occurrences, and in the image of the sky, there were 214 occurrences. This could certainly be considered proof, if the provenance of the images were to be verified.

Open-source steganalysis software

Using the docker image *stego-toolkit*, we ran both the PNG and JPEG steganalysis tools, and also performed brute force using various extraction programs, with varying results.

sky.png, b=1

Tool	zsteg	openstego	lsb-steg	stegoVeritas
Result	*positive*	*negative*	*negative*	*negative*

Zsteg was able to detect and also extract the watermark we embedded.

chess.png, b=3

Tool	zsteg	openstego	lsb-steg	stegoVeritas
Result	*positive*	*negative*	*negative*	*negative*

portrait.jpg, $p = 1$

Tool	stegdetect	steghide	outguess	stegoVeritas
Result	*negative*	*negative*	*negative*	*negative*

sky.jpg, $p = 8$

Tool	stegdetect	steghide	outguess	stegoVeritas
Result	*positive*	*negative*	*negative*	*negative*

Stegdetect also attempts to determine the tool used to produce the stego image. In this case, it concluded that the tool was outgess, with a 3 out of 5 certainty.

Conclusions

Our LSBR and JSTEG implementations have proved that visually imperceptible images can be obtained, with minimal chance of detection of the embedded information. It has been observed that choosing between the two categories of steganographic methods requires a tradeoff between storage capacity and detectability. In regards to storage capacity, spatial domain techniques have the clear advantage, while frequency domain techniques excel in detectability, with the ability to evade visual attacks, the chi-squared test, and even open-source popular steganalysis tools. With LSBR, we were able to use as much as ½ of the size of an image for maximum storage capacity, and with JSTEG, as much as 0.01% for maximum protection against steganalysis techniques.

We can confidently state that steganography is a valuable practice for lots of use cases, when used as a force for good. Particularly, the utilization of steganography for image watermarking can aid in the protection of intellectual property by embedding imperceptible information within images. This process enables content owners to assert ownership without significantly degrading the visual quality of the image. By encrypting the watermarks, we can ensure their integrity, even though their removal is still possible.

Without specific information about images, the detection of steganographic content is a complex task, due to the sheer variability in the statistical properties of natural, unaltered images, making it challenging to assess with any degree of certainty whether content has been embedded within an image, and even harder to differentiate between legitimate applications like image watermarking and potentially malicious use cases.

References

Al-Iedane, H. A., & Mahameed, A. I. (2023, April 24). Applying and Evaluating Machine Learning Models for the Detection of Digital Image Steganography. *SSRN*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4427951

Arnold, P. S., & Macrakis, K. (n.d.). *Invisible ink*. Wikipedia. https://en.wikipedia.org/wiki/Invisible_ink

Beneš, M. (n.d.). *jpeglib*. https://pypi.org/project/jpeglib/

Breuker, D. (n.d.). *stego-toolkit*. https://github.com/DominicBreuker/stego-toolkit

Cryptography law. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Cryptography_law

Data compression. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Data_compression

Gibson, R. (n.d.). *Steganography: Hiding Data In Plain Sight*. https://www.cs.unc.edu/~lin/COMP089H/LEC/steganography.pdf

Hsiao, L. (n.d.). *pyexiv2*. https://pypi.org/project/pyexiv2/

Ker, A. D. (2016). *Information Hiding*. Department of Computer Science, Oxford University. http://www.cs.ox.ac.uk/andrew.ker/docs/informationhiding-lecture-notes-ht2016.pdf

Ortiz, J. (2014, August 3). *Advanced JPEG Steganography and Detection*. YouTube. https://www.youtube.com/watch?v=BQPkRlbVFEs

Peak signal-to-noise ratio. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

Pires, F. (2023, July 18). *Intel's First x86 CPU Had Secret Instructions Meant to Catch IP Thievery*. Tom's Hardware. https://www.tomshardware.com/news/intels-first-x86-cpu-had-secret-instructions-meant-to-catch-ip-thievery

Płachta, M., Krzemień, M., Szczypiorski, K., & Janicki, A. (2022, May 13). Detection of Image Steganography Using Deep Learning and Ensemble Classifiers. *MDPI*. https://www.mdpi.com/2079-9292/11/10/1565

Provos, N., Honeyman, P., & Raggo, M. T. (n.d.). *Steganography*. Wikipedia. https://en.wikipedia.org/wiki/Steganography

Schnell, I. (n.d.). *bitarray*. https://pypi.org/project/bitarray/

Secrets Hidden in Images (Steganography) - Computerphile. (2015). https://www.youtube.com/watch?v=TWEXCYQKyDc

Steganalysis. (n.d.). Wikipedia. Retrieved January 30, 2024, from https://en.wikipedia.org/wiki/Steganalysis

Steganalysis. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Steganalysis

Structural similarity. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Structural_similarity

Westfeld, A. (n.d.). *F5 - A Steganographic Algorithm*. Technische Universität Dresden. https://www2.htw-dresden.de/~westfeld/publikationen/f5.pdf

What Is Steganography? Definition and explanation. (n.d.). Kaspersky. https://www.kaspersky.com/resource-center/definitions/what-is-steganography