Comparing Encode Times and Size Difference of Lossless Image Compression Formats for LSB Image Steganography

Geef mij maar Nasi Goreng

Reza Hadi Fairuztama, 18218046 (*Author*)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): rreza510@gmail.com

***Abstract*—This document will attempt to benchmark the effects of LSB Image Steganography on various image compression methods. After testing, it resulted that there is no significant change in time encoding and file size before and after LSB steganography.**

Keywords—image, steganography, lossless, compression

TABLE OF CONTENTS

TABLE OF FIGURES

I. INTRODUCTION

Steganography is a method of hiding a plaintext in another object so that it is undetectable by third party observers. This is useful for when you want to hide the fact that you sent a message in the first place, and not just to hide the true meaning of your message.

However, as data compression, and lossless image compression in particular, exploits statistical redundancies within an image, the insertion of a non-random steganography data might impact the algorithm in a way that might tip off third party listeners that something is off. It may also give overhead budget for when we want to use steganography for things such as metadata storage or 3d texture shader shenanigans.

In this paper, I will use the LSB image steganography method and look at the impact it does in the various image compression methods available today, by encoding time and size difference. Whether it is statistically significant enough to be exploited in further cryptoanalysis or not.

II. DEFINITIONS

In this chapter, i will give a short explanation that will be useful for understanding this paper, but sadly not the author.

A. *Steganography*

Steganography refers to the act of hiding or concealing an information within an object in such a way that any third party looking at the object is unaware of the existence of the information within. This differs from cryptography in that, while steganography ensures that the information is hidden from third party "listening in", they do not obfuscate the meaning of the message by itself. Instead, it makes sure that the message does not seem to exist in the first place. Thus they can, and usually are used together, though not always, as in the form of a watermark.

Steganography has been used since long ago in many form, from writing a message on a slave's shaved head then waiting for it to grow back, to hiding the message in an ink only visible after appliances of a specific chemical, to obfuscating the message by mixing within non-important and innocent-looking informations (eg. Taking the first letter of every word, taking every fifth words, etc.), to digitally hide them within a digital file.

Digital steganography covers the usage and methods of hiding information within digital format. It might uses a text file, an image file, audio/video file, or any other file format available, though each format would of course warrant different method in hiding the message. In this paper we will use one such method, for hiding information within an image: the LSB Image Steganography.

B. *Least Significant Bit (LSB) Image Steganography*

In a digital raster image, images are represented by a matrix of pixels. A pixel can be represented by a number of color channel, usually 1, 3 or 4, within each channel is a number representing how "bright" each channel is. The most common image format are 8-bit RGB, where there are 3 channels of Red, Green, and Blue, each with bitlength 8. Thus, each channel have a range of 0 to $2^8 - 1 = 255$.

Least significant bit, thus, refer to the 2^0 bit of each of these channels in the pixels of an image. The LSB Image Steganography uses these bits to insert the plaintext. This will change the image, but as it was the least significant bit, it is perceptually almost indistinguishable from the original image to our eyes.

Note that, while this paper will only use the 2^0-st bit, there is nothing stopping us to use the next, 2^1-nd bit, or the 2^2-rd bit, and so on. It is, however, a trade-off between more space, and distruption on the image represented.

C. *Digital Image Compression*

A common format for a raster image is composed of a matrix of pixels, composed of 3(RGB) or 4(RGBA) 8-bit channels. These are enough to acceptably represent images in the digital realm without leaving too much color detail. It is, however, not very economical to store these naively in disk. A common resolution used in camera output, 1600x1200 pixels, is already 1.920.000 pixels, which have 3 channels of 8 bits, or 2 bytes.

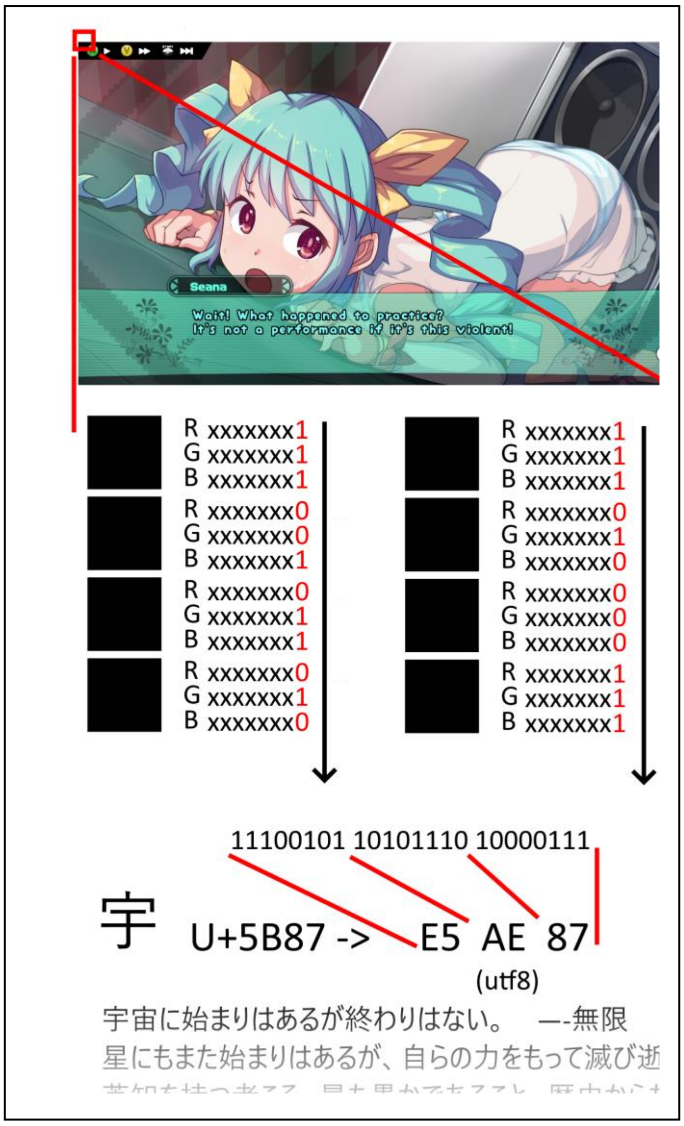

Fig. 1. LSB Steganography infographic

That means that a picture taken by a common camera would consume around 11 megabytes each. Not much by today's standard, but it's still a waste of space.

Instead, images are usually stored by first running it through a data compression algorithm. Data compression is the practice of encoding information so that it is smaller than the original representation.

There are two type of compression: lossy, and lossless. Lossy compression exploit the deficiencies of human perception by eliminating details that won't be noticeable to us, such as the exact color value of a pixel within its surrounding pixels, or the overlapping nearby frequencies in a sound bite.

Lossless compression, meanwhile, tries to keep the exact data intact while reducing its size. These usually exploit patterns in the data instead, by reducing these found patterns by a single, shorter representation, such as its redundancy or "entropy", or information content.

D. Image Encoding

At the base level, computers only work with 1s and 0s. We might group them by bytes and thus work in hexadecimal, but it still stands that computer does not innately have any way to store an image. Therefore, an encoding method to transform a 2D image into a string of binary data is required for the computer to be able to store our precious and less-precious memories in visual form. Hence, image encodings.

In this paper, we will be working with raster images, which store images by a matrix of pixel values. There are other kind of image types like vector images, which store images by the basic geometries that, when the math is done, form that image. Unlike raster images, scaling up or down a vector image won't lose any detail. But unlike raster images, it is hard to make a vector image with a camera. As such, they're mostly used for logos, icons, and websites, places where resizing images and keeping their detail is rather important. There might be some way to make LSB work in vector images, but this paper is nearing its deadline already as is.

And so, in this paper, we will be using these four image encoding formats, for they are the easiest to code for within the constraint of all these final papers asked by *every class all at once*--

1) PNG

Portable Network Graphics (PNG) is a file format for raster images. It supports either grayscale image, palleted images (image with a set number of colors), and full-color non-pallete-based images. As the PNG format is designed for sending images over internet, it only supports RGB color space. [1]

Compression in PNG consist of two steps; a pre-processing step where with each image line, the algorithm tries to predict, from a set of five filter, which one is best for efficient compression. Then the second step uses DEFLATE, an algorithm that combines LZ77 and Huffman Coding, to compress each image line.

2) BMP

BMP fileformat, also known as Bitmap image file or Device-Independent Bitmap (DIB) file is a image file format for raster images. It supports images in grayscale and color, in various color depths, and optionally with data compression and alpha channels. [1]

Due to its simplicity as a file format, its widespread usage thanks mostly to Windows, and that it's an open format, BMP is a very common file format for image processing programs to write and read in.

3) TIFF

Tag Image File Format (TIFF) is an image file format for raster images. It was created as a standard for scanning documents, but has also gotten popular for faxing, OCR, image manipulation, desktop publishing, etc. It supports image in grayscale, RGB(A), and even CYMK. It is created by Aldus Corporation in 1986, and continued updating up to 1992[1].

Unlike other formats, a .tiff file might contain more than one image within it, which might contain one or more tag-value pair within it, true to its name. Baseline tiff images are compressed by *strips*, which contains one or more horizontal lines.

4) QOI

Quite OK Image (QOI) format is an image format and algorithm specification created by Dominic Szablewski recently at 24th November 2021. It supports RGB and RGBA 8-bit channels image. The goal of this format was to make an open-source, lossless compression method that's faster and easier to implement than PNG. The result were that, while PNG has smaller file size, QOI has 20-50x faster encoding and 3-4x faster decoding. [2]

The format encodes the image from left to right, up to down. It uses a combination of simple run-length encoding, pixel mapping, and difference marking. This allows it to be versatile and relatively quick in encoding an image, without making it a hodgepodge of special cases, prediction algorithms, and a thick specificaiton standard a la PNG.

III. Methodology

In this chapter, i will explain the experiment i've done to find out the thesis of this paper; whether LSB steganography significantly changes the encoding time or size. Should this then prove insufficient, the author expresses condolences and ask for the reader's forgiveness for the flaws not apparent in this fifth paper in a row.

A. Analysis

The goal of this paper, and thus this experiment, is to see whether LSB steganography have a noticeable impact to encoding time and size. Therefore, i will be using a statistical, two-sample T test to analyze the data, with significance level 0,01. Equation (1) is the main equation used to find the T-value from the data, where n_i is the data count for sample i (each at 12 images x 10 repetition = 120), $\Delta\mu$ is the difference of averages between the two value, and $\Delta\sigma$ is the standard error of difference, gathered from (2).

$$t = \frac{\Delta\mu}{\Delta\sigma} \tag{1}$$

$$\Delta\sigma = \sqrt{\frac{\left((n_1-1)\sigma_1^2\right)+\left((n_2-1)\sigma_2^2\right)}{n_1+n_2-2} \times \left(\frac{1}{n_1}+\frac{1}{n_2}\right)} \tag{2}$$

As for the analysis, the two hypothesis are as follow:

1. Hypothesis 0: LSB steganography does not change the encoding time and compression size of an image.
2. Hypothesis 1: LSB steganography significantly change the encoding time and compression size of an image.

Therefore, the samples will be the time and size for a plain image, and the time and size for a steganogrpahed image.

B. Data gathering

All of the codes for data gathering are in Rust. Everything in this step (except for encoding into QOI, which uses the qoi 0.4.0 crate) is using Rust's standard library crates, with default parameters, all running non-consecutively as i will be running this in windows and ~~i am not sure and is too busy to find out if~~ Windows are unable to have concurrent disk writing, which might skew the time figure by orders of magnitude.

For the experiment I have prepared a dataset of 12 images, and 5 plaintext of various length and language. The code will open eacah image and save them again, both to have uniform image channels between images, and to give a baseline of an unedited image. Each save is repeated 10 times, with each time logged.

TABLE I. PLAINTEXTS

n	*Plaintext*	*Bytes*
1	宇宙に始まりはあるが終わりはない。 ——無限 星にもまた始まりはあるが、自らの力をもって滅び逝く。 ——有限 英知を持つ者こそ、最も愚かであること。歴史からも読み取れる。 海に生ける魚は、陸の世界を知らない。彼らが英知を持てば、それもまた滅び逝く。 人間が光の速さを超えるのは、魚たちが陸で生活を始めるよりも滑稽。 これは抗える者たちに対する、神からの最後通告とも言えよう。	552
2	UNDANG-UNDANG DASAR NEGARA REPUBLIK INDONESIA TAHUN 1945 PEMBUKAAN Bahwa sesungguhnya kemerdekaan itu ialah hak segala bangsa dan oleh sebab itu, maka penjajahan di atas dunia harus dihapuskan, karena tidak sesuai dengan peri-kemanusiaan dan peri-keadilan.	260
3	[illegible]	432
4	Ми не чужі любити Ви знаєте правила, і я теж Я думаю про повну відданість Ви не отримаєте цього від жодного іншого хлопця Я просто хочу розповісти тобі, як я почуваюся Треба змусити вас зрозуміти Ніколи не здам тебе Ніколи не підведу Ніколи не буду бігати і покидати вас Ніколи не змусить вас плакати Ніколи не попрощаюсь Ніколи не скажу неправду і не зашкодю тобі	673
5	Kentang	7

Fig. 2. Plaintexts used in the experiment

Afterwards, with each image, the code will steganography each plaintext from the start of the image via LSB, and then save it 10 times in each image format as well, making sure that it too contains the same image channels all across the board, and like the non-steganographed ones.

With the data at hand, i will put it in an Excel spreadsheet and ~~fiddle around it until i~~ find the values needed for the statistical analysis to take place.

IV. Results

And thus, after all that preamble, which is 50% there so that this paper have the neccesary page count, in this chapter i will give the results i've gotten.

A. Time

After a night of leaving the computer on running the code, and a morning of lugging the laptop turned on and churning the algorithms through, the following tables are the processed results of the data logged by the code. The immediately following table is for the total average of saving time samples per plaintext:

TABLE II. AVERAGES FOR SAVING TIME BY PLAINTEXT

	None	*1*	*2*	*3*	*4*	*5*
Bmp	757.84	750.63	755.98	770.22	826.35	795.71
Png	20930.92	20846.17	20883.67	21502.50	21837.92	21719.67
Qoi	23260.43	23201.96	23367.24	23881.02	24720.57	24363.59
Tiff	10.81	10.38	10.53	10.90	11.00	10.55

Fig. 3. The average values for saving time by plaintext

And the following are for the standard deviation for saving time sample per plaintext:

TABLE III. THE STANDARD DEVIATION FOR SAVING TIME BY PLAINTEXT

	None	*1*	*2*	*3*	*4*	*5*
Bmp	605.50	596.57	603.75	625.48	783.67	663.30
Png	16994.30	16910.94	16955.23	17872.05	18420.52	18203.20
Qoi	19528.89	19471.64	19666.54	20541.45	21972.52	21146.60
Tiff	7.99	7.50	7.70	8.15	8.57	7.67

Fig. 4. The standard deviation for saving time by plaintext

Using these numbers, and using (2) and (1) to get the two-sample T value for no-plaintext and each plaintext, these are the resulting T-values:

TABLE IV. T-VALUES FOUND FOR SAVING TIMES BY PLAINTEXT

	1	2	3	4	5
bmp	9.29.E-02	2.38.E-02	1.56.E-01	7.58.E-01	4.62.E-01
png	3.87.E-02	2.16.E-02	2.54.E-01	3.96.E-01	3.47.E-01
qoi	2.32.E-02	4.22.E-02	2.40.E-01	5.44.E-01	4.20.E-01
tiff	4.35.E-01	2.81.E-01	8.89.E-02	1.76.E-01	2.61.E-01

Fig. 5. The T-values for saving time by plaintext

From the T-values alone, the differences seem to be very small as to be insignificant. And indeed, as our T-critical value for α=0.01 and with 120 + 120 – 2 = 238 degrees of freedom just about equals 2.60, it seems that none of our results are above the critical value by more than an order of magnitude, and we can't reject our null hypothesis.

B. Size

And in this subsection i have provided further processed results in the following tables. The immediately following table is for the total average of resulting filesize samples per plaintext:

TABLE V. THE AVERAGES FOR RESULTING FILESIZES BY PLAINTEXT

	(blank)	1	2	3	4	5
bm p	220202 14.00	220202 14.00	220202 14.00	220202 14.00	220202 14.00	220202 14.00
png	853600 6.33	853653 6.58	853627 5.92	853640 1.17	853664 6.42	853601 8.50
qoi	976498 6.00	976524 7.00	976510 0.25	976517 9.00	976531 3.92	976498 9.17
tiff	220202 80.00	220202 80.00	220202 80.00	220202 80.00	220202 80.00	220202 80.00

Fig. 6. The average values for resulting filesize by plaintext

And, just like before, the following table are for the standard deviation for resulting filesize sample per plaintext:

TABLE VI. THE STANDARD DEVIATION FOR RESULTING FILESIZES BY PLAINTEXT

	(blank)	1	2	3	4	5
bm p	175873 00.29	175873 00.29	175873 00.29	175873 00.29	175873 00.29	175873 00.29
png	735965 7.17	735967 5.34	735960 0.64	735964 8.76	735969 6.10	735965 5.32
qoi	825391 6.37	825383 9.86	825386 3.82	825385 3.24	825382 6.04	825391 4.85
tiff	175877 15.53	175877 15.53	175877 15.53	175877 15.53	175877 15.53	175877 15.53

Fig. 7. The standard deviation for resulting filesize by plaintext

...Before we continue, of significant note is that the .bmp and .tiff format does not seem to change sizes between the plaintext inserted. It seems that the default for Rust is to *not* encode them. Good to know instead of the fact that a DynamicImage implements From.

Anyway, just like before, we use the two table values to get our T-value via (1) and (2), in which the results are as follows

TABLE VII. T-VALUES FOUND FOR RESULTING FILESIZE BY PLAINTEXT

	1	2	3	4	5
Bmp	0.00.E+00	0.00.E+00	0.00.E+00	0.00.E+00	0.00.E+00
Png	5.58.E-04	2.84.E-04	4.16.E-04	6.74.E-04	1.28.E-05
Qoi	2.45.E-04	1.07.E-04	1.81.E-04	3.08.E-04	2.97.E-06
Tiff	0.00.E+00	0.00.E+00	0.00.E+00	0.00.E+00	0.00.E+00

Fig. 8. The T-values for resulting filesize by plaintext

Even more than saving time, the resulting differences seem to be very small. And indeed, with the T-critical value for α=0.01 and with 238 degrees of freedom just about equals 2.60, it seems that none of our results are above the critical value by more than an order of magnitude, and we can't reject our null hypothesis on this factor either.

V. CONCLUSION

From this short experiment, it was found that the LSB steganography method does not seem to impact the encoding time and compression size of an image significantly enough. Though all of them tend to increase in size, this does not seem to be statistically significant enough. On the contrary, some image formats, namely .bmp and .tiff, does *not* have any difference in size at all.

This could mean that detecting a steganographied image by its (re-)encoding time or file size would be statistically infeasible and would cause either false positives or false negatives. Or at least, if it's via a machine learning algorithm, then it should not be used with it alone.

It could also mean that storing metadata for things that need to run in real time, like 3d texture shader rendering in video games or simulators, will not impact significantly in size and storetime. Which means it might be possible for them to store more data within its assets.

Or it could mean that this paper is not thorough enough. Either or. It's not like this is rushed through a pipeline of a handful other, just as lengthy and researchful papers and projects all announced and deadlined in the span of just about one month, in the middle of trying to study for finals as well.

ah well, i tried.

REFERENCES

[1] Murray, James D.; vanRyper, William. Encyclopedia of Graphics File Formats (Second ed.). O'Reilly, 1996.

[2] Lnu, Reducible. How PNG Works: Compromising Speed for Quality. Youtube, 2022. URL: https://www.youtube.com/watch?v=EFUYNoFRHQI Accessed by 25th May 2022.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 25 Mei 2022

[illegible]

Reza Hadi Fairuztama, 18218046