A New Method of Image Steganography Using 7th Bit of a Pixel as Indicator by Introducing the Successive Temporary Pixel in the Gray Scale Image

Rodrigo Céspedes *UTEC*, Benjamín Díaz *UTEC*, Gabriel Spranger *UTEC*

***Resumen*—Hoy en día la transferencia de data es sumamente importante e interiorizada en las dinámicas digitales de todos los días. A lo largo de la historia de la computación, se han usado diferentes métodos y técnicas para ocultar datos durante una transmisión, para hacerlos más seguros. Una de estas técnicas es la llamada Esteganografía, que abarca el estudio y la aplicación de métodos para poder ocultar mensajes dentro de objetos u otros medios. En el contexto digital. Este concepto es de gran relevancia porque permite ocultar datos dentro de medios, volviéndo seguro el dato de manera intrínseca para no depender de otros protocolos. Esta experiencia trata de recrear un método nuevo de esteganografía utilizando bits de los pixeles de alguna imagen para esconder un mensaje dentro del archivo.**

I. Introducción

La esteganografía ha sido uno de los métodos de protección de datos más usado dentro del mundo de la computación, especialmente con la llegada del internet y la necesidad de transferir datos de manera segura a través de la red. Conforme la tecnología avanza, se vuelven mucho más complejas las interacciones y por lo tanto, tienen que ser más seguras. Hace algunos años, quizás la seguridad a la hora de transmitir mensajes no era tan importante como lo es ahora, pero por la masificación que existe con la tecnología hoy en día, es de alta importancia tener sistemas seguros de ocultación y transmisión de datos, pues son cada vez más las personas dispuestas a piratear conexiones y robar datos.

Muchos se preguntarán: ¿qué diferencia tiene la criptografía de la esteganografía? La diferencia yace en el hecho de que la criptografía ofusca, a través de encriptación, la visibilidad de un mensaje determinado y, la esteganografía, esconde el hecho de que el mensaje si quiera existe, pues está embebido de manera oculta dentro de otro mensaje cuya relevancia parece ser mínima. No se puede robar, lo que no se sabe que existe, por ello es importante encontrar buenos métodos de esteganografía para poder llevar a cabo transmisiones de datos más seguras. En este documento, se explicará a detalle cómo se recreó el método creado por el autor, mediante el cual se pueden ocultar datos dentro de imágenes en escala de grises, alterando ciertos bits de los pixeles.

II. Marco teórico

Antes de seguir con este documento, es importante dar contexto de lo que se está haciendo, asi como definir términos importantes que serán usados a lo largo del trabajo.

II-A. Diccionario de Términos

1. **Cover Image**:
   Imagen en donde se ocultará el mensaje.
2. **Stego Image**:
   Imagen con el mensaje ya oculto.
3. **PSNR (Peak Signal to Noise Ratio)**:
   Es la relación entre la señal original y el ruido generado por la introducción del mensaje [1]. Entiéndase señal como *Cover Image* y señal con ruido como *Stego image*. La fórmula usada para obtener el PSNR es la siguiente:
   $$\text{PSNR} = 10\ \log_{10}(\frac{I^2}{\text{MSE}})$$
   Donde I es el máximo valor posible de un pixel de la imagen. Como la imagen está en escala de grises, este valor es 255. Mientras que MSE es el *Mean Squared Error*.
4. **MSE (Mean Squared Error)**:
   Es una función de error. Sirve para calcular el valor medio de los cuadrados de la diferencia entre los valores obtenidos y los valores reales [2]. Nótese que es otra medida para contrastar la *Cover Image* de la *Stego Image*, es decir, su diferencia cuadrática. Mientras menor sea este valor, menor diferencia habrá entre la *Cover Image* y la *Stego Image*. La fórmula usada para obtener el MSE es la siguiente:
   $$\text{PSNR} = \frac{1}{RC}\sum_{i=1}^{R}\sum_{j=1}^{C}(x_{ij} - x'_{ij})^2$$
   Donde R es el número de filas de la imagen, C el de columnas, x_{ij} el valor del pixel en la posición i,j de la *cover image* y x'_{ij} el valor del pixel en la posición i,j de la *stego image*.
5. **Histograma (de una imagen)**:
   Entiéndase como un gráfico que muestra el número de pixeles que tienen cada valor en la escala de grises (0-255) [3]. Es por ello que el histograma tiene valores en el eje x de 0 a 255, mientras que el eje y representa en número de pixeles que tienen cierto valor de gris. Es una métrica gráfica y estadística para comparar la similitud entre la *Cover Image* y la *Stego Image*.

II-B. ¿Qué es exactamente la Esteganografía?

Es una técnica mediante la cual, un mensaje ïnofensivo"puede ocultar información secreta dentro de él. En del mundo digital, se utilizan mucho las imágenes, videos y audios, por su gran capacidad de almacenaje y múltiples sitios donde poder ocultar información, como por ejemplo, los metadatos de algún archivo [4]. Como ya se mencionó anteriormente, para esta experiencia, se utilizarán imágenes en escala de grises.

II-C. Trabajos Previos

Como en cualquier ámbito, ya existen trabajos previos y métodos comúnmente utilizados. Se hablará a continuación de los métodos en el contexto de las imágenes. A continuación, se presentarán 3 de los métodos más interesantes encontrados en el paper de referencia y son los siguientes:

1. **LSB (Least Significant Bit)** Consta de ocultar información en el *LBS* de cada pixel de alguna imagen y funciona especialmente bien cuando se quiere ocultar una imagen de 2 (o pocos) colores, pues el *Stego Image* idealmente, tendrá cambios poco perceptibles o hasta imperceptibles al *Cover Image* (cuando se encubra una imagen de pocos colores) [5].
   ¿Qué desventajas tiene?
   Tiene una observación y es que al ser tan sencilla, es evidente en qué pixeles buscar y por ende, fácilmente detectable.
2. **Método de Batra y Rishi** [6] Batra y Rishi propusieron un método usando el 6to, 7mo y 8vo bit de un pixel en una imagen en escala de grises. Nótese como en este método, el problema del **LSB** deja de tener relevancia, pues se vuelve mucho más complejo detectar el mensaje.
3. **FMM (Five Modulus Method)**: Este método divide la *Cover Image* en N bloques con tamaño k*k pixeles y modifica los pixeles en este bloque de manera tal, que su valor sea divisible por 5. Este método es más complejo y necesita de llaves para poder ser descifrado.
   ¿Qué desventajas tiene?
   La capacidad para esconder mensajes puede ser bastante baja, hablando en términos promedio, menos de 1 bit por pixel.

III. Propuesta del Autor

En esta sección se explicará a detalle la propuesta del autor de manera que se pueda entender la posterior recreación de la experimentación original.

Antes de explicar el algoritmo propuesto, se tienen que tomar en cuenta los siguientes puntos:

1. El emisor y el receptor saben el largo exacto del mensaje.
2. Entiéndase I como la *Cover Image* de $R * C$ pixeles
3. Entiéndase S como el mensaje de N bits
4. Entiéndase x como el valor del pixel
5. Entiéndase s como el bit del mensaje

Para cada pixel x, mientras sigamos teniendo bits que ocultar de S, extraemos el séptimo bit del valor de dicho pixel (x_{ij} & $0x02$, llamémosle b_1) y el séptimo bit del valor dicho pixel + 1 ($(x_{ij}+1)$ & $0x02$, llamémosle b_2), lo cual nos da dos bits en total. Comparamos estos bits con los dos bits s_a y s_b actuales del mensaje S que queremos ocultar en la imagen I. Si son iguales, entonces no hay nada más que hacer y pasamos a ocultar los siguientes dos bits de S. Si no son iguales, entonces hay subcasos que se detallan en la Sección 4, pero la idea básica es que se tiene que modifcar b_1 y b_2 de tal manera que sea iguales a los dos bits actuales del mensaje (s_a y s_b) que se quiere ocultar en la imagen I.

IV. Ejemplo del algoritmo propuesto

Tenemos el mensaje $m = \{10001011\}$ el cual entraría en los primeros 4 píxeles de la imagen que tienen los siguientes valores $p = \{47, 22, 2, 48\}$, para encriptar el mensaje se realiza lo siguiente:

Obtenemos el primer pixel de la imagen y pixel + 1.

$$\begin{aligned} P_1 &= 47\ (1011\ \mathbf{1}\ 1) \\ P_1 + 1 &= 48\ (1100\ \mathbf{0}\ 0) \end{aligned} \tag{1}$$

Podemos ver que el 7mo bit de P_1 y $P_1 + 1$ forman "10" y los 2 primeros bits del mensaje son "10", en este caso no se realiza nada y copia el pixel.

Seguimos con el segundo pixel:

$$\begin{aligned} P_2 &= 22\ (101\ \mathbf{1}\ 0) \\ P_2 + 1 &= 23\ (101\ \mathbf{1}\ 1) \end{aligned} \tag{2}$$

En este caso P_2 y $P_2 + 1$ forman "11", los siguientes bits del mensaje son "00", entonces a P_2 le tenemos que sumar 2, ahora P_2' sería igual a 24. P_2' es el *stego pixel*.

El tercer pixel tiene valor 2, realizamos lo siguiente.

$$\begin{aligned} P_3 &= 2\ (\mathbf{1}\ 0) \\ P_3 + 1 &= 3\ (\mathbf{1}\ 1) \end{aligned} \tag{3}$$

P_3 y $P_3 + 1$ forman el par "11" y se busca almacenar "10", para realizar esto le sumamos 1 a P_3'.

Finalmente, el cuarto pixel tiene valor 48, entonces

$$\begin{aligned} P_4 &= 48\ (1100\ \mathbf{0}\ 0) \\ P_4 + 1 &= 49\ (1100\ \mathbf{0}\ 1) \end{aligned} \tag{4}$$

P_4 y $P_4 + 1$ forman el par "00" y el mensaje es "11", en este caso le sumamos 2 a P_4'

Los píxeles que contienen el mensaje secreto son $p' = \{47, 24, 3, 50\}$, para descifrar el mensaje realizamos lo siguiente.

El primer pixel es

$$\begin{aligned} P_1' &= 47\ (1011\ \mathbf{1}\ 1) \\ P_1' + 1 &= 48\ (1100\ \mathbf{0}\ 0) \end{aligned} \tag{5}$$

Los bits combinados forman "10", el segundo pixel es

$$\begin{aligned} P_2' &= 24\ (110\ \mathbf{0}\ 0) \\ P_2' + 1 &= 25\ (110\ \mathbf{0}\ 1) \end{aligned} \tag{6}$$

Los bits forman “00”, el tercer pixel es

$$\begin{aligned} P'_3 &= 3\ (0\ \mathbf{1}\ 1) \\ P'_3 + 1 &= 4\ (1\ \mathbf{0}\ 0) \end{aligned} \tag{7}$$

Los bits forman “10”, y finalmente el cuarto pixel es

$$\begin{aligned} P'_4 &= 50\ (1100\ \mathbf{1}\ 0) \\ P'_4 + 1 &= 51\ (1100\ \mathbf{1}\ 1) \end{aligned} \tag{8}$$

Los bits de este pixel forman “11”, al juntar todas estos mensajes obtenidos de los píxeles obtenemos el mensaje oculto “10001011”.

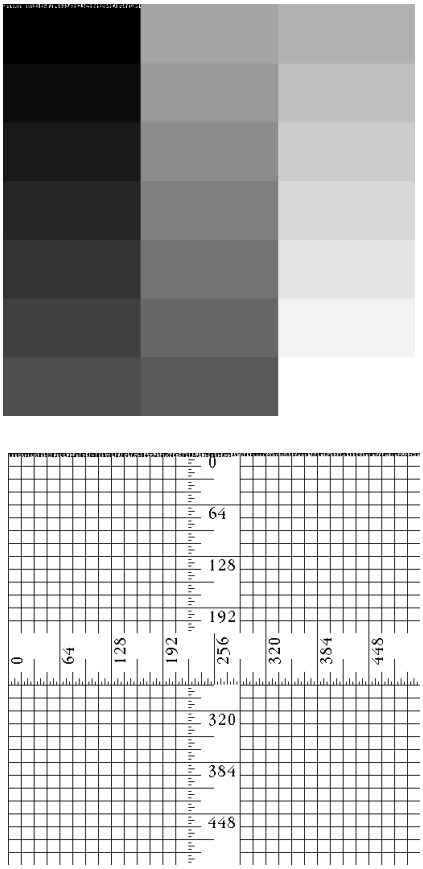

Figura 1: Dos ejemplos de imágenes que tienen colores blancos (255) y negros (0). El algoritmo del autor no toma en cuenta mantener la consistencia en estos casos y por lo tanto se ve claramente una distorsión en la parte superior izquierda en ambas imágenes.

V. Nuestra Propuesta

El algoritmo propuesto extiende el algoritmo del autor. A partir de una análisis de la implementación replicada, notamos dos puntos fuertes que podían ser mejorados:

- **Casos de borde:** El algoritmo propuesto por el autor le resta dos, resta uno, suma uno o suma dos al valor del pixel dependiendo del valor del mensaje que se quiere ocultar en ese pixel y del valor actual del pixel. Sin embargo, el autor no toma en cuenta que si el pixel tiene valor 244, entonces sumarle dos causaría un *overflow* y el pixel pasaría de ser blanco a ser negro, lo cual viola completamente el objetivo de la esteganografía que es esconder mensajes a simple vista, ya que un cambio de blanco a negro es muy visible como se puede ver en la figura 1. Además, este caso también ocurre cuando el valor del pixel es cero y se le resta uno o cuando es uno y se le resta dos, en este caso, el pixel cambiaría de negro a blanco.

  Nuestra propuesta se basa en corregir estos casos de borde para que se garantice un cambio mínimo entre la *cover image* y la *stego image*.
  El algoritmo que presenta el autor no considera el caso en el cual tenga que sumarle o restarle valores al pixel y este sea 0 (negro) o 255 (blanco), en caso la imagen sea completamente negra, el algoritmo podría restarle valores al pixel de la imagen que almacenaría el mensaje y el pixel terminaría siendo blanco, en este caso se podría ver el cambio en los píxeles y no se estaría ocultando el mensaje. Los cambios que realizamos fueron los siguientes:
  - Si el pixel es 0 y se le debe restar 1, sumarle 3.
  - Si el pixel es 255 y se le debe sumar 1, restarle 3.
  - Si el pixel es 254 o 255 y se le debe sumar 2, restarle 2.
- **Suposición de conocer el $length$ del mensaje:** El algoritmo del autor asume que el destinatario del mensaje conoce el tamaño del mensaje, pero esto no es práctico porque hace que la implementación del algoritmo sea dependiente de una manera de compartir el tamaño del mensaje, pero la idea es que sea *stand-alone*, es decir, que baste solo con tener la imagen para extraer el mensaje exitosamente.

Esto lo logramos reservando los primeros 64 bits de la imagen (de acuerdo al algoritmo ojo, no literalmente los 64 primeros bits de la imagen) para poner el número de bits que el mensaje oculto tiene. De esta manera, el algoritmo de extracción primero lee los 64 bits para obtener el $length$ del mensaje en bits y luego lee $\frac{length}{2}$ pixeles para obtener de cada uno 2 bits del mensaje y al final poder obtener el mensaje completo.

- **Color:** Se aplica el mismo algoritmo propuesto a los tres canales de color *R,G,B* de cada pixel para así poder trabajar con imágenes a color en vez de únicamente con imágenes en escala de grises. Nuevamente, se aplica el trato especial de nuestra propuesta para los casos de borde para evitar cambios muy bruscos de color en los pixeles.
- **Confidencialidad e Integridad:** Si bien es externo algoritmo, es relevante mencionar qué se decidió hacer con la llave k compartida por tanto el remitente como el destinatario k, ya que el paper menciona el uso de dicha llave k mas no especifica su uso específico. En esta situación, se decidió usarla para cifrar el mensaje a mandar con AES-256-CBC. Como llave, no usamos 256 bits aleatorios criptográficamente seguros, sino que usamos una frase (por ejemplo, "inca kola"), y la llave de 256 bits es el *hash* de la frase con el algoritmo SHA-256 ($k = h_{\text{SHA-256}}(\text{frase})$). Además, añadimos un HMAC para asegurar la integridad del mensaje. De esta manera, solo se requiere que el remitente y el destinatario compartan una frase en común que permitirá que se comuniquen mediante la estenografía de manera segura y confiable.

VI. Experimentación

A continuación mostramos los resultados obtenidos de nuestro algoritmo propuesto y los resultados del algoritmo del autor. Primero mostramos las *cover image* junto con su *stego image* correspondiente y luego el histograma de la *cover image* junto con el de la *stego image*. Todas las imágenes a continuación contienen el siguiente mensaje oculto: *Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.*.

Como resultado, podemos observar que el algoritmo propuesto genera diferencias mínimas entre la *cover* y la *stego image*, lo mismo ocurre con los histogramas, los cuales tienen diferencias casi perceptives. Esto se comprobará con los resultados del MSE y PSNR a continuación.

El autor realizó pruebas con mensajes de 2kB, 4kB, 6kB, 8kB y 10kB. Nosotros también, pero en las tablas se muestra el número de *kilobytes* menos ocho, dado que los primeros ocho bytes de las *stego images* los reservamos para guardar el tamaño del mensaje oculto. Es por ello que se verá en las tablas 2040 en vez de 2048, 4088 en vez de 4096, etc, excepto en las tablas del autor, donde sí se verá el número de *bytes* correspondiente.

Figura 2: *Cover image*.

Figura 3: *Stego image*.

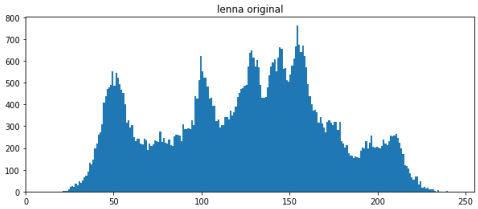

Figura 4: Histograma del *cover image*.

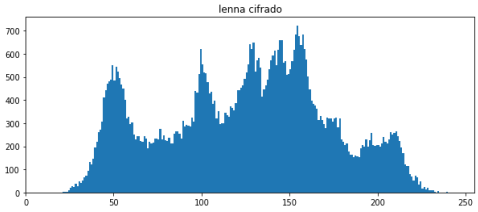

Figura 5: Histograma del *stego image*.

Figura 6: *Cover image*.

Figura 7: *Stego image*.

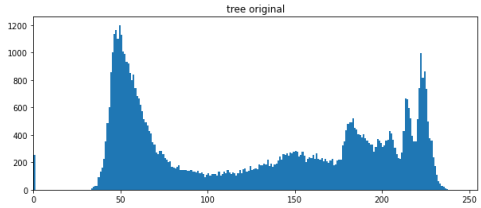

Figura 8: Histograma del *cover image*.

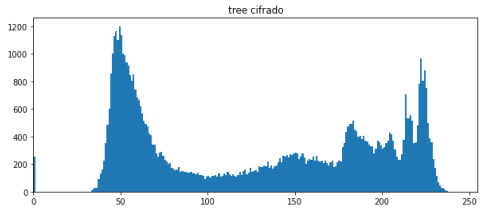

Figura 9: Histograma del *stego image*.

Figura 10: *Cover image*.

Figura 11: *Stego image*.

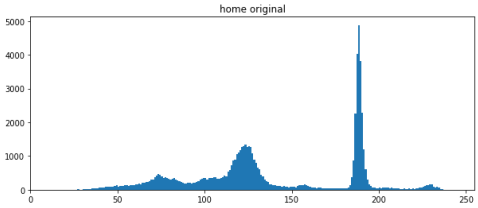

Figura 12: Histograma del *cover image*.

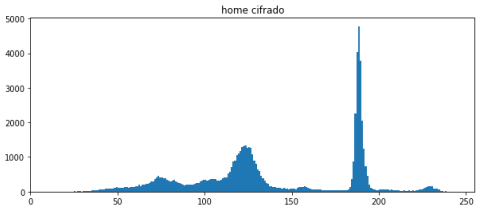

Figura 13: Histograma del *stego image*.

Image name	Size	PSNR	MSE
baboon.tiff	2040	61.3858	0.0473
clock.tiff	2040	55.3976	0.1876
girl.tiff	2040	55.4118	0.187
home.tiff	2040	55.4984	0.1833
lenna.jpeg	2040	55.5343	0.1818
ruler.tiff	2040	57.8022	0.1079
gray.tiff	2040	60.3647	0.0598

Cuadro I: Resultados del algoritmo **propuesto** en distintas imágenes cuando se oculta un mensaje de 2kB.

Image name	Size	PSNR	MSE
baboon.tiff	4088	58.3901	0.0942
clock.tiff	4088	52.3741	0.3764
girl.tiff	4088	52.4031	0.3739
home.tiff	4088	52.399	0.3743
lenna.jpeg	4088	52.4771	0.3676
ruler.tiff	4088	54.7316	0.2187
gray.tiff	4088	57.4234	0.1177

Cuadro II: Resultados del algoritmo **propuesto** en distintas imágenes cuando se oculta un mensaje de 4kB.

Figura 14: *Cover image*.

Figura 15: *Stego image*.

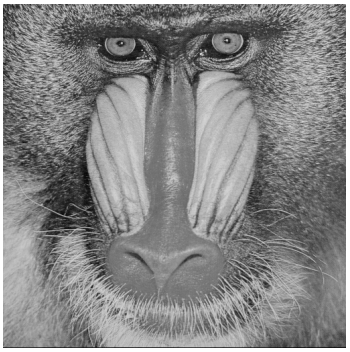

Figura 18: *Cover image*.

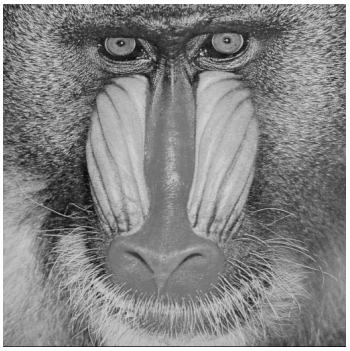

Figura 19: *Stego image*.

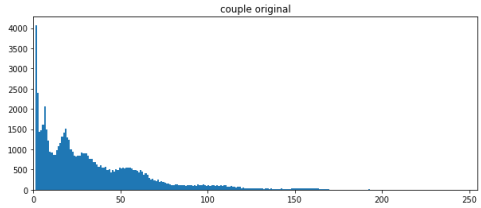

Figura 16: Histograma del *cover image*.

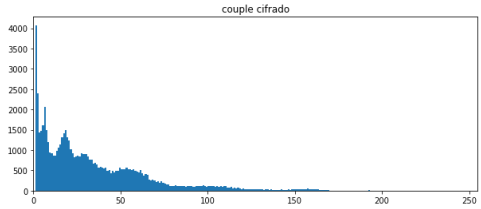

Figura 17: Histograma del *stego image*.

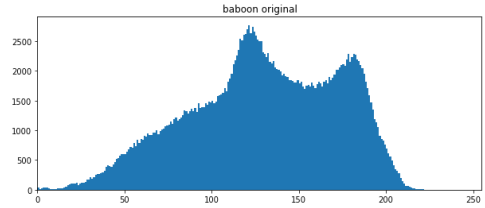

Figura 20: Histograma del *cover image*.

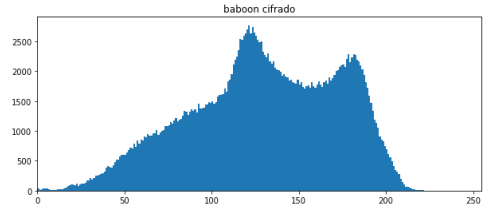

Figura 21: Histograma del *stego image*.

Image name	Size	PSNR	MSE
baboon.tiff	6136	56.6288	0.1413
clock.tiff	6136	50.6442	0.5606
girl.tiff	6136	50.6243	0.5632
home.tiff	6136	50.6499	0.5599
lenna.jpeg	6136	50.6699	0.5573
ruler.tiff	6136	53.005	0.3255
gray.tiff	6136	55.6062	0.1788

Cuadro III: Resultados del algoritmo **propuesto** en distintas imágenes cuando se oculta un mensaje de 6kB.

Image name	Size	PSNR	MSE
baboon.tiff	8184	55.3964	0.1877
clock.tiff	8184	49.3458	0.756
girl.tiff	8184	49.3813	0.7498
home.tiff	8184	49.3959	0.7473
lenna.jpeg	8184	49.4054	0.7457
ruler.tiff	8184	51.7204	0.4376
gray.tiff	8184	54.3927	0.2365

Cuadro IV: Resultados del algoritmo **propuesto** en distintas imágenes cuando se oculta un mensaje de 8kB.

Image name	Size	PSNR	MSE
baboon.tiff	10232	54.4069	0.2357
clock.tiff	10232	48.3886	0.9424
girl.tiff	10232	48.3821	0.9438
home.tiff	10232	48.4397	0.9314
lenna.jpeg	10232	48.4591	0.9272
ruler.tiff	10232	50.7655	0.5452
gray.tiff	10232	53.5141	0.2895

Cuadro V: Resultados del algoritmo **propuesto** en distintas imágenes cuando se oculta un mensaje de 10kB.

A continuación se muestran los resultados que al autor obtuvo. Además de dichos resultados, hemos agregado los resultados obtenidos con dos imágenes extra: *ruler.tiff* y *gray.tiff*. Estas imágenes se pueden apreciar en la figura 1. Dichas imágenes tienen varios pixeles que tiene como valor blanco (255) y negro (0), por lo que se espera que el algoritmo del autor tenga un error alto dado que no se asegura que tomar en cuenta estos casos. Los resultados a continuación confirman que nuestro algoritmo obtiene menor error.

Image name	Size	PSNR	MSE
baboon.tiff	2048	55.418	0.1867
clock.tiff	2048	55.333	0.1904
girl.tiff	2048	55.307	0.1916
home.tiff	2048	55.477	0.1842
lenna.jpeg	2048	55.401	0.1875
ruler.tiff	2048	17.881	1059.04
gray.tiff	2048	26.015	162.75

Cuadro VI: Resultados del algoritmo **del autor** en distintas imágenes cuando se oculta un mensaje de 2kB.

Image name	Size	PSNR	MSE
baboon.tiff	4096	52.396	0.3745
clock.tiff	4096	52.388	0.3752
girl.tiff	4096	52.321	0.3810
home.tiff	4096	52.409	0.3734
lenna.jpeg	4096	52.40	0.3741
ruler.tiff	4096	14.81	2144.34
gray.tiff	4096	23.257	307.16

Cuadro VII: Resultados del algoritmo **del autor** en distintas imágenes cuando se oculta un mensaje de 4kB.

Image name	Size	PSNR	MSE
baboon.tiff	6144	50.6195	0.5638
clock.tiff	6144	50.6094	0.5651
girl.tiff	6144	50.5525	0.5726
home.tiff	6144	50.6268	0.5629
lenna.jpeg	6144	50.6416	0.5609
ruler.tiff	6144	13.0861	3194.98
gray.tiff	6144	21.3041	481.57

Cuadro VIII: Resultados del algoritmo **del autor** en distintas imágenes cuando se oculta un mensaje de 6kB.

Image name	Size	PSNR	MSE
baboon.tiff	8192	49.3804	0.7500
clock.tiff	8192	49.3483	0.7555
girl.tiff	8192	49.2966	0.7646
home.tiff	8192	49.3446	0.7562
lenna.jpeg	8192	49.3701	0.7517
ruler.tiff	8192	11.8018	4294.3
gray.tiff	8192	20.1893	622.50

Cuadro IX: Resultados del algoritmo **del autor** en distintas imágenes cuando se oculta un mensaje de 8kB.

Image name	Size	PSNR	MSE
baboon.tiff	10240	48.411	0.9374
clock.tiff	10240	48.366	0.9529
home.tiff	10240	48.377	0.9447
lenna.jpeg	10240	48.406	0.9385
ruler.tiff	10240	10.837	5362.1
gray.tiff	10240	19.551	721.02

Cuadro X: Resultados del algoritmo **del autor** en distintas imágenes cuando se oculta un mensaje de 10kB.

Como se puede observar, el algortimo del autor obtiene un PSNR mayor y un MSE menor al algoritmo del autor el 91 % de veces. Este número sale porque hemos probado 7 imágenes con 5 distintos tamaños de mensaje, lo cual da 35 experimentos en total. Comparamos el resultado del autor y el del algoritmo propuesto de cada uno de esos 35 experimentos y resultó que en 32 experimentos el algoritmo propuesto obtuvo un PSNR mayor y un MSE menor. Por lo tanto, podemos concluir que nuestro algoritmo mejoró el del autor.

En cuanto a los resultados de la misma propuesta pero tomando en cuenta imágenes a color, obtuvimos resultados muy similares. Dado que el autor no implementó el algoritmo para imágenes a color, basaremos la discusión de los resultados solamente en el PSNR y MSE entre la *cover* y *stego image*.

Image name	Size	PSNR	MSE
baboon.tiff	2040	64.823	0.0214
clock.tiff	2040	58.7743	0.0862
girl.tiff	2040	58.9378	0.083
home.tiff	2040	58.5172	0.0915
lenna.jpeg	2040	58.8189	0.0853
ruler.tiff	2040	60.9799	0.0519
gray.tiff	2040	63.9433	0.0262

Cuadro XI: Resultados del algoritmo **propuesto a color** con un mensaje de 2kB.

Image name	Size	PSNR	MSE
baboon.tiff	4088	61.8891	0.0421
clock.tiff	4088	55.9115	0.1667
girl.tiff	4088	55.929	0.166
home.tiff	4088	55.6558	0.1768
lenna.jpeg	4088	55.8064	0.1708
ruler.tiff	4088	57.9382	0.1045
gray.tiff	4088	61.1083	0.0504

Cuadro XII: Resultados del algoritmo **propuesto a color** con un mensaje de 4kB.

Image name	Size	PSNR	MSE
baboon.tiff	6136	60.1435	0.0629
clock.tiff	6136	54.1706	0.2489
girl.tiff	6136	54.2108	0.2466
home.tiff	6136	54.0065	0.2585
lenna.jpeg	6136	54.0595	0.2553
ruler.tiff	6136	56.2098	0.1556
gray.tiff	6136	59.3726	0.0751

Cuadro XIII: Resultados del algoritmo **propuesto a color** con un mensaje de 6kB.

Image name	Size	PSNR	MSE
baboon.tiff	8184	58.9047	0.0837
clock.tiff	8184	52.8867	0.3345
girl.tiff	8184	52.9312	0.3311
home.tiff	8184	52.8129	0.3402
lenna.jpeg	8184	52.878	0.3352
ruler.tiff	8184	54.9709	0.207
gray.tiff	8184	58.1974	0.0985

Cuadro XIV: Resultados del algoritmo **propuesto a color** con un mensaje de 8kB.

Image name	Size	PSNR	MSE
baboon.tiff	10232	57.9203	0.105
clock.tiff	10232	51.9195	0.418
girl.tiff	10232	51.9399	0.416
home.tiff	10232	51.8804	0.4217
lenna.jpeg	10232	51.8843	0.4214
ruler.tiff	10232	54.0038	0.2586
gray.tiff	10232	57.2304	0.123

Cuadro XV: Resultados del algoritmo **propuesto a color** con un mensaje de 10kB.

Figura 22: *Cover image*.

Figura 23: *Stego image* escondiendo un mensaje de 10kB.

Como se puede observar, todos los resultados del algoritmo propuesto con soporte de imágenes a color obtuvo un mayor PSNR y un menor MSE. Esto es esperado, dado que ahora cada pixel puede guardar ahora 6 bits de data en vez de 2 bits comparado con la versión en blanco y negro. Esto es dado que cada pixel tiene tres canales (cada uno de 0-255) en el caso de las imágenes a color, mientras que las imágenes en blanco y negro solo tienen un canal. Por lo tanto, el mensaje ocupará menos pixeles y por lo tanto habrán más pixeles libres que reducen el MSE con la imagen original dado que el MSE en esos pixeles libres sería cero. Además, se sigue manteniendo la mejora que toma en cuenta con casos de borde para evitar cambios bruscos de color.

VII. Limitaciones

- **Eficiencia:** El algoritmo propuesto no es paralelo, aunque su naturaleza permite que sea paralelizado sin problema alguno. Mientras más grande sea la imagen, se podrá

guardar un mensaje más largo dentro de ella, por lo que una imagen 4K podría guardar un mensaje muy grande, el presente algoritmo se demoraría en procesar los millones de pixeles dado que es secuencial. Se recomienda que un trabajo futuro paralelice el algoritmo. Además, el algoritmo fue implementado en Python, por lo que tiene las penalidades en la eficiencia que vienen con los lenguajes interpretados. Dado todo esto, recomendaríamos usar la combinación de C++ con OpenMP para desarrollar una versión más rápida del algoritmo.

- **Espacio:** Dado que los primeros 64 bits de la imagen (de acuerdo al algoritmo) las usamos para guardar el tamaño del mensaje, tenemos 8 *bytes* menos para guardar información en la imagen. Sin embargo, esta penalidad es mínima y vale la pena dado que ahora el destinatario no necesita saber el tamaño del mensaje oculto *a priori*.
- **Character Set:** El algoritmo propuesto solo funciona cuando se mandan mensajes cuyos caracteres estén contenidos en la tabla ASCII. Para permitir el uso de emojis, cambiamos el emoji a su representación en ASCII (el de un pulgar se convierte en :thumbs_up:) para que pueda ser guardado en la imágen sin problema. Si se tratan de guardar caracteres que no estén en la tabla ASCII, como caracteres chinos, no se podrán guardar en la imagen. Sin embargo, este proceso es externo al algoritmo, es decir, no forma parte de él.

VIII. Conclusiones

El algoritmo propuesto obtuvo un PSNR más alto y un MSE más bajo en 91 % de las pruebas realizadas. Esto se debe a que se toma en cuenta cuando el cambio del bit va a hacer un cambio brusco en el color del pixel. Además, se removió la necesidad de que el destinatario sepa el tamaño del mensaje, lo cual en la práctica es poco conveniente, ya que uno sabría que **hay** un mensaje, pero no exactamente de qué tamaño. Finalmente, se extendió el algoritmo propuesto para que también funcione con imágenes de color y se propuso un uso específico de la llave k para garantizar que la comunicación entre el remitente y el destinatario sea confidencial e íntegra.

El algoritmo propuesto con soporte de imágenes a color obtuvo mejores resultados que el algoritmo del autor, aunque estos resultados no son tan comprabales entre sí dado que los experimentos que el autor hizo fueron con imágenes en escala de grises. Sin embargo, aun así se ve una mejora con casos de borde y al comparar imágenes a color como se ve en esta figura las diferencias no son percentibles para el ojo humano.

Recomendamos que un trabajo futuro se encargue de mejorar la eficiencia del algoritmo usando un lenguaje compilado como C++ y paralelizando el algoritmo (con OpenMP, por ejemplo). Además, se podría modificar el algortimo para que distribuya el mensaje en toda la imagen en vez de ponerlo en un bloque contiguo; es probable que dicha modificación haga aun más imperceptible el mensaje escondido.

IX. Bibliografia

[1] Compute peak signal-to-noise ratio (PSNR) between images - Simulink. (s/f). Mathworks.Com. https://www.mathworks.com/help/vision/ref/psnr.html

[2] (S/f). Numxl.com. Recuperado el 16 de mayo de 2022, de https://support.numxl.com/hc/es/articles/115001223423-MSE-Error-Cuadr

[3] Chen, J. (2021, agosto 18). Histogram. Investopedia. https://www.investopedia.com/terms/h/histogram.asp

[4] Kuksov, I. (2019, julio 4). ¿Qué es la esteganografía digital? Kaspersky. https://www.kaspersky.es/blog/digital-steganography/18791/

[5]Páez, R. (2013, marzo 7). Esteganografía: Ocultando el uso de LSB. Security Art Work. https://www.securityartwork.es/2013/03/07/estenografia-ocultando-el-uso-de-lsb/

[6] S. Batra, R. Rishi, "Insertion of message in 6th, 7th and 8th bit of pixel values and its retrieval in case intruder changes the least significant bit of image pixels," International Journal of Security and Its Applications, vol. 4, no. 3, pp. 1–10, 2010.

[7] K. Bailey , K. Curran, "An evaluation of image based steganography methods," Multimedia Tools and Applications, vol. 30, no. 1, pp. 55–88, 2006.

X. Anexos

- La implementación realizada se encuentra en los siguientes links:
  - Implementación siguiendo las pautas del autor
  - Implementacion propuesta en blanco y negro
  - Implementación propuesta incluyendo soporte para imágenes a color, emojis y confidencialidad e integridad con AES-256-HMAC-SHA256
- Las imágenes usadas se encuentran aquí.