Análisis Comparativo de Algoritmos de Esteganografía Basados en LSB

Girod Joaquín[1], **Ijjas Christian**[1] **y Ferruti Francisco**[1]

[1]ITBA - Instituto Tecnológico de Buenos Aires, Criptografía y Seguridad 72.44

Este informe presenta el análisis de un proyecto de implementación de esteganografía basado en varios algoritmos de inserción en el bit menos significativo (LSB), incluyendo LSB1, LSB4 y LSBI (Inversión de Bit en el LSB). El objetivo principal del proyecto fue ocultar y extraer archivos de imágenes BMP utilizando estos métodos, comparando su eficiencia en términos de capacidad, seguridad y complejidad. A través de experimentos prácticos, evaluamos la efectividad de cada método y realizamos una comparación detallada. El análisis revela que, mientras LSB1 y LSB4 ofrecen simplicidad y mayor capacidad de almacenamiento, LSBI, al introducir la inversión de bits y la modificación selectiva de pixeles, mejora significativamente la seguridad, aunque a costa de una mayor complejidad. Concluimos con propuestas para futuras mejoras del programa desarrollado, como la compatibilidad con más formatos y la extracción automatizada por fuerza bruta.

Contacto: *jgirod@itba.edu.ar cijjas@itba.edu.ar fferruti@itba.edu.ar*

Introducción

En este trabajo se presenta un análisis comparativo de diferentes algoritmos de esteganografía basados en la técnica de LSB (Least Significant Bit). Los algoritmos evaluados incluyen LSB1, LSB4 y LSBI (Least Significant Bit Inversion), los cuales fueron implementados con el objetivo de ocultar información en imágenes BMP. El estudio se centra en comparar la eficiencia de cada método en términos de capacidad de almacenamiento, seguridad y robustez, utilizando métricas como PSNR (Peak Signal-to-Noise Ratio) y el análisis de canales RGB. A través de experimentos prácticos, se evidencia que LSB1 y LSB4 ofrecen simplicidad y mayor capacidad de inserción de datos, mientras que LSBI mejora significativamente la seguridad mediante la inversión selectiva de bits, aunque a costa de una mayor complejidad. Este trabajo también explora posibles mejoras en las técnicas implementadas, proponiendo nuevas funcionalidades como la compatibilidad con más formatos de archivo y la automatización de procesos de extracción de información.

1. Análisis del Documento de Inversión de Bits (1)

La organización formal del documento nos pareció estándar y correcta, aunque creemos que es beneficioso el uso de una única columna para el texto principal cuando se muestra pseudo código, ya que las indentaciones forzadas por el poco espacio pueden ser confusas para el lector.
Por otro lado la descripción del algoritmo nos pareció suficientemente clara, y el ejemplo que proporcionan es muy valioso, pero no se hace suficiente énfasis en el manejo del patrón de inversión. En principio dan a entender que los 4 bits del patrón de inversión a los que hace referencia el enunciado no son necesarios para la extracción y que con tan solo la imagen esteganografiada se puede procesar,

> So in de-steganography, the information that leads us to the pixels patterns that has been inversed is to firstly read from the stego-image to know which pattern was inversed and which pattern was not. Then classify the stego-image according to the four patterns and next is to reinverse the bits in order to extract the massage bits.

Luego, en el pseudo código parece que suponen que el receptor tiene acceso a tanto la imagen original como la imagen con el mensaje oculto.

> Compare between cover value and stego value to count how many pixel are changed and how many are not.

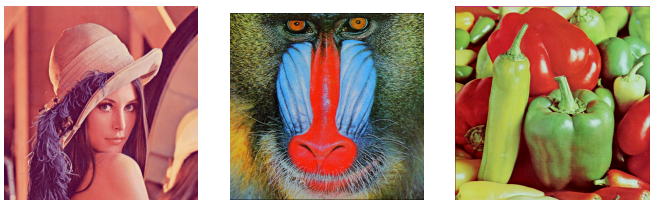
Pero la suposición de que el receptor cuenta con ambos elementos es muy optimista bajo nuestra mirada. Por ende, consideramos que cuanto menos esta sección esta débilmente explicada y que el uso del prefijo de 4 bits que describe el enunciado es correcto.
Además, gran parte del documento se centra en conceptos que, aunque útiles para comprender el algoritmo, no deberían formar parte central de la publicación. Consideramos que las secciones verdaderamente relevantes y que representan una contribución significativa al conocimiento son la 5, 6, 7 y 8. Por lo tanto, las secciones de la introducción, la explicación de la técnica LSB1 y la descripción del modelo RGB podrían eliminarse sin afectar el valor del trabajo.
Sugerimos que estas secciones sean sustituidas por citas o enlaces directos a sus entradas correspondientes en fuentes accesibles, como Wikipedia o sus respectivos artículos, para simplificar y optimizar el contenido.

También se han detectado errores ortográficos y gramaticales, además de que los gráficos carecen de etiquetas y títulos adecuados. Sin embargo, estos detalles, aunque afectan la presentación, no impiden la comprensión del documento.

2. Comparación de Algoritmos de Esteganografía: LSB1, LSB4 y LSBI

La diferencia en la cantidad de bytes que se pueden almacenar con cada uno de los algoritmos de esteganografía tiene un efecto directo sobre la **calidad final del esteganograma** y la imagen original.

Esta característica la analizamos mediante el **PSNR** (Peak Signal-to-Noise Ratio). Para introducirnos en el análisis comparativos de los distintos LSBs hicimos uso de los archivos de la figura 1 que son clásicos archivos de pruebas.



(a) Lenna.bmp 512×512 píxeles. **(b)** Babooon.bmp 512×512 píxeles. **(c)** Pepper.bmp 512×512 píxeles.

Fig. 1. Imágenes usadas para hacer análisis PSNR.[1]

Además para poder replicar los experimentos del documento (1) usamos el mismo texto de 226 caracteres

> In poverty and other misfortunes of life, true friends are a sure refuge. The young they keep out of mischief; to the old they are a comfort and aid in their weakness, and those in the prime of life they incite to noble deeds.

Dadas estas condiciones, decidimos probar los tres métodos de esteganografía sobre los archivos y luego calculamos los PSNRs correspondientes obteniendo el siguiente gráfico de barras:

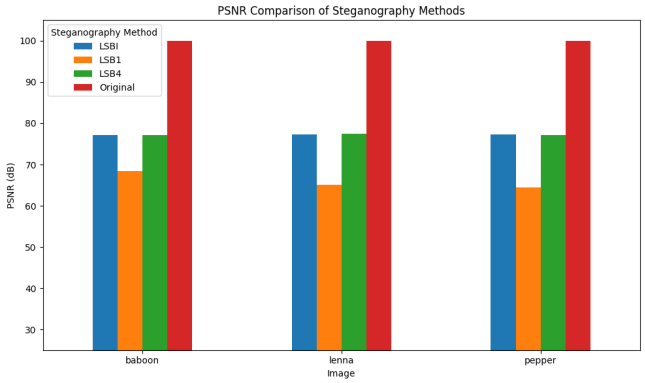


Fig. 2. PSNR para cada imagen con cada algoritmo comparando con original.

y su tabla correspondiente:

	LSB1	LSB4	LSBI
baboon	77.20	68.39	77.10
lenna	77.35	65.00	77.42
pepper	77.34	64.38	77.11

Cuadro 1. Comparación de PSNR para los algoritmos LSB1, LSB4 y LSBI.

Se puede concluir que LSB4, al modificar más significativamente los píxeles, tiene un menor PSNR, mientras que LSB1 y LSBI, con mejor desempeño, afectan de manera muy similar. Esto es lógico, ya que tanto LSB1 como LSBI siempre alteran el bit menos significativo y, aunque modifiquen diferentes canales, afectan el mismo número de canales RGB.

Posteriormente, tomamos una imagen en particular, `pepper.bmp`, para analizar cómo los distintos algoritmos afectan la imagen. A modo de ejemplo, generamos los histogramas para cada archivo alterado y el original.

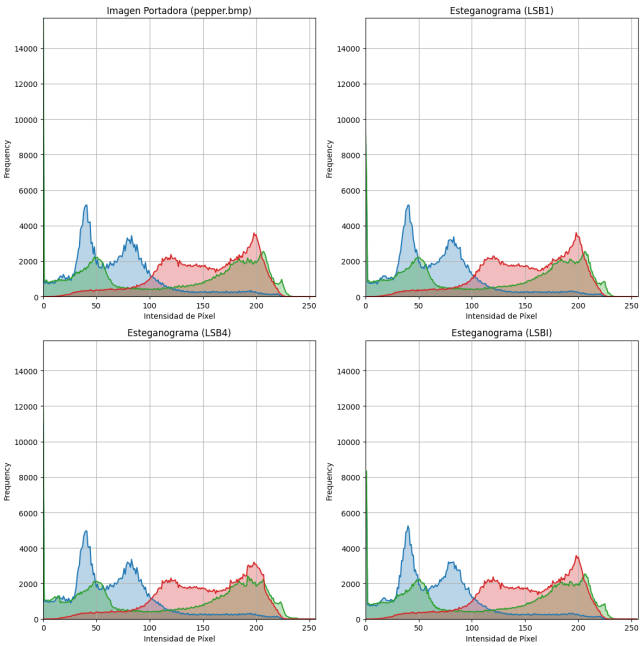


Fig. 3. Histograma de `pepper.bmp` con cada algoritmo.

A partir de este gráfico, optamos por analizar como varían los distintos canales con los diferentes algoritmos. Para esto usamos la métrica de χ^2 obteniendo la siguiente tabla:

	LSB1	LSB4	LSBI
Blue	39098.20	2108.78	61623.02
Green	40426.28	5541.40	62314.09
Red	272.73	788.81	0.00

Cuadro 2. Resultados de χ^2 usando `pepper.bmp` con cada algoritmo.

Como era de esperarse, la tabla muestra que LSBI afecta más los canales azul y verde en comparación con el rojo. La relación se mantiene de acuerdo a:

$$\chi^2_{azul,LSB1} \approx \frac{2}{3}\chi^2_{azul,LSBI}$$

y de manera análoga para el verde. LSBI tiende a afectar significativamente los canales azul y verde, lo cual puede dete-

riorar la imagen si contiene una alta proporción de estos colores. Sin embargo, en la mayoría de los casos, las imágenes están equilibradas, y evitar alterar el color rojo es una buena práctica para asegurar la **imperceptibilidad** de los cambios. Esto se debe a que el color rojo es aquel que el ojo humano detecta con mayor facilidad.
En cuanto a la **robustez**[2], es importante medirla en relación con el área susceptible de daño. Mientras más bits se utilicen para almacenar la información, mayor será la fragilidad del método. Esto está relacionado con qué posiciones de los bits se usan, ya que no es lo mismo utilizar los últimos 4 bits que el último bit. Al cambiar los bits más significativos, se aumenta la entropía, haciendo que los cambios sean más notorios. Por esta razón, LSB1 es el más robusto, ya que solo utiliza el último bit; LSBI le sigue en robustez (ya que su área de susceptibilidad comprende los 3 bits menos significativos debido al patrón de inversión); y LSB4 es el menos robusto debido a su uso de más bits significativos.
En cuanto a la **seguridad**, los métodos LSB1 y LSB4 son ampliamente conocidos, lo que facilita su detección y extracción con herramientas comunes, además de que utilizan la totalidad de los píxeles (a diferencia de LSBI, que evita el canal rojo). LSBI, por otro lado, agrega un nivel de complejidad al invertir bits, lo que hace que la información oculta no esté directamente embebida en el archivo portador, y requiere de un dato adicional para su extracción, lo que dificulta los ataques.
La comparación final se resume en la siguiente tabla:

	LSB1	**LSB4**	**LSBI**
Capacidad	3 bits/píxel	12 bits/píxel	2 bits/píxel
Robustez	3 bits/píxel	12 bits/píxel	6 bits/píxel
PSNR	$\sim 70-80$	$\sim 60-70$	$\sim 70-80$
Impercep.	Media	Baja	Alta
Seguridad	Baja	Baja	Alta

Cuadro 3. Comparación de los algoritmos LSB1, LSB4 y LSBI.

La seguridad de los algoritmos está estrechamente relacionada con su **complejidad**. A diferencia del mundo de la criptografía, donde el **Principio de Kerckhoffs** [3] establece que el atacante tiene conocimiento de los algoritmos utilizados, en el ámbito de la esteganografía, si el atacante conoce el algoritmo empleado, el sistema se ve fuertemente comprometido. Por esta razón, la inversión de bits y el desuso del píxel rojo presentan una mejora en la seguridad, ya que dificultan la identificación del algoritmo. Lógicamente, la mejor opción es ocultar mediante esteganografía información previamente encriptada (aunque esto requiere una clave y abre otro abanico de problemas).

3. Procedimiento de Extracción y Análisis de Archivos Ocultos

En resumen, si bien el orden puede cambiar dependiendo de algunos factores de decisión aleatoria, los pasos que seguimos para completar el proceso fueron los siguientes:

1. Extraer de avatar.bmp con LSBI un PDF.
2. Extraer de montevideo.bmp un con LSB1 un PNG.
3. Cambiar la extensión del PNG extraído a .zip.
4. Descomprimir el ZIP y obtener el archivo de texto con instrucciones para interpretar el acertijo de buscaminas.
5. Resolver el acertijo para obtener la primitiva AES128 y el modo de encadenamiento ECB.
6. Obtener de paris.bmp la contraseña "gloria".
7. Extraer de anillo2.bmp con desencripción un video .wmv.

A continuación, detallamos el flujo completo del análisis y extracción de los archivos.
Una vez que implementamos todos los algoritmos de esteganografiado y comprobamos su correcto funcionamiento con los métodos de encripción/desencripción, procedimos a hacer el análisis de los cuatro archivos (avatar, montevideo, paris, anillo2). Nos empezamos a preguntar qué había que hacer y lo primero que hicimos fue pasar todos los BMPs a hexdumps para ver su contenido. Esto lo hicimos con la utilidad **xxd** y luego de analizar brevemente los archivos no llegamos a conclusiones e intentamos hacer extracciones por fuerza bruta (debería haber sido nuestra primera acción). Probando con cada imágen y cada algoritmo sin cifrado pudimos extraer, con LSBI sobre avatar.bmp, un PDF con una oración breve que decía que cambiemos la imagen con extensión **.png** a **.zip**. Con esto nos dimos cuenta que se trataba de una especie *capture the flag* y empezamos a usar distintas técnicas para extraer información. Como aún no teníamos la imagen nos confundimos un poco y probamos analizar la imagen que extrajimos de los archivos de prueba (la imagen del logo del ITBA) sin éxito. Luego seguimos con el proceso de fuerza bruta y logramos obtener un **.png** con el siguiente contenido.

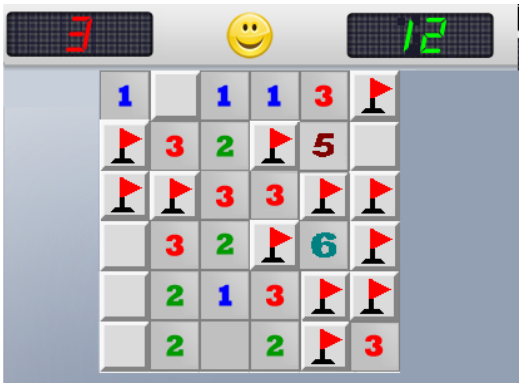


Fig. 4. PNG extrarído del BMP montevideo con LSB1.

La figura 4 muestra un pequeño acertijo que debe ser resuelto junto a un **.txt** que contiene instrucciones. Este archivo de texto se obtiene cambiando la extensión del PNG a ZIP. Haciendo esto se puede *unzipear* el archivo y obtener el archivo

[2] Usamos las definiciones de debilidad y robustez (entre otros conceptos) tomando como referencia la presentación provista por la cátedra: Steganography and Digital Watermarking
[3] Se consideran deseables en la criptografía los Principios de Kerckhoff

de texto en cuestión. Resolviendo el acertijo se obtienen los caracteres ASCII que conforman el método y el modo de cifrado que se utilizará para extraer información de otro archivo. En este punto nos quedan dos archivos no explorados: paris.bmp y anillo2.png. Como nos faltaba una contraseña para poder extraer de uno de los archivos procedimos a leer los hexdumps. No nos costó mucho encontrar la contraseña dado que por experiencia de otros casos es normal que se guarde en texto plano algo por el estilo de 'contraseña: <contraseña>'. Probamos buscando la palabra contraseña en ambos archivos y no estaba pero buscando 'pass' logramos encontrar el texto 'password es gloria' en el hexdump del BMP paris. Entonces ya con el algoritmo de encripción, el modo de cifrado, la contraseña y un único archivo para explorar (anillo2.bmp) procedimos a extraer, con LSB4, un archivo **.wmv** que es un archivo de video que podría considerarse nuestro *flag* que estabamos buscando.[4]

4. Mensajes Anidados en Archivos Ocultos

En uno de los archivos, el mensaje oculto contenía a su vez otro archivo comprimido dentro del mensaje extraído. Es el caso de montevideo.bmp que contiene embebido un archivo **.png** que a su vez tiene embebido un archivo **.zip**. Es el paso 2 a 4 que se describe en la pregunta 3. Como el ejercicio daba instrucciones claras de qué hacer con el **.png** no hicieron falta herramientas adicionales pero en un caso más real, hacer uso de `binwalk` es recomendado ya que facilita identificar información oculta o embebida. Si ejecutamos `binwalk -e montevideo.bmp`, el programa extraerá automáticamente los archivos embebidos dentro de la imagen.

```
$ binwalk stegoanalysis/out/montevideo.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 413 x 302, 8-bit/color RGBA, non-interlaced
91            0x5B            Zlib compressed data, compressed
42829         0xA74D          Zip archive data, at least v2.0 to extract, compressed
size: 173, uncompressed size: 260, name: sols/sol9.txt
43156         0xA894          End of Zip archive, footer length: 22
```

Fig. 5. Clara presencia de un zip embebido en el png.

5. Ocultamiento del Fragmento de Video Extraído

En el video[5] se detalla un código binario donde el portador físico es una tela. Para definir los ceros y los unos se sigue un hilo horizontal, si el hilo vertical que interseca esta por encima es un 1, en cambio, si esta por debajo es un 0.
Por ende, haciendo una lectura de las tramas horizontales se puede obtener una cadena de bits, la cual en este caso es representativa de un carácter ASCII. Luego se utilizan varias tramas horizontales para generar una cadena de caracteres la cual indica el nombre del objetivo a **asesinar**.

[4] Si se desea revisar el paso a paso exacto con ejecución incluida ver README en GitHub.
[5] Video de referencia Loom of fate

6. Métodos Alternativos de Esteganografía

Además de los métodos LSB1, LSB4 y LSBI que son técnicas de **sustitución**[6], se implementó un método de esteganografía por **adición**, también conocido como concatenación. En este método, como se menciona en la Sección 4, no se modifican los datos internos de la imagen PNG, sino que simplemente se agrega el archivo ZIP al final de la imagen. Una manera sencilla de lograr esto es utilizando el comando `cat`, el cual por definición concatena archivos. El siguiente comando muestra un ejemplo de cómo se realiza esta operación:

```
cat portador.png mensaje_secreto.zip > esteganograma.png
```

Fig. 6. Comando para lograr esteganograma como el provisto en el *capture the flag*.

Este método de concatenación es particularmente eficaz cuando el archivo portador es un formato que, al ser manipulado por una aplicación convencional, no detecta o ignora los datos adicionales agregados al final del archivo. En este caso, el archivo ZIP se oculta de manera efectiva, ya que el archivo PNG puede abrirse sin problemas en cualquier visualizador de imágenes, mientras que el ZIP puede extraerse y descomprimirse con un simple cambio de extensión. Esto le otorga al método un nivel de eficacia considerable, ya que no altera la estructura interna del archivo portador, manteniendo su funcionalidad original intacta. Sin embargo, su principal desventaja radica en que este tipo de ocultación puede ser relativamente fácil de detectar si se examina el tamaño del archivo o su estructura con herramientas especializadas como editores hexadecimales o comparadores binarios.
Otro método de esteganografía utilizado fue la inserción directa de texto en el archivo portador mediante la modificación de su contenido hexadecimal. En este caso, el mensaje secreto "la password es gloria"fue añadido en secciones no críticas del archivo, sin alterar su funcionalidad. Este método pertenece a la categoría de adición, ya que se incluye el mensaje en un área que las aplicaciones que procesan el archivo pueden ignorar, pero es visible al examinarlo con un editor hexadecimal.

```
00493e20: 2b49 492b 4949 2b49 492b 4949 2b49 492b  +II+II+II+II+II+
00493e30: 4a4a 2c4a 4a2c 6c61 2070 6173 7377 6f72  JJ,JJ,la passwor
00493e40: 6420 6573 2067 6c6f 7269 61              d es gloria
```

Fig. 7. Output si ejecutamos `tail hexdump_de_paris.hex -lines 3`.

Ambos métodos son eficaces en diferentes escenarios. La concatenación es ideal cuando se busca ocultar archivos de

[6] En la presentación del Dr. Roberto Gómez Cárdenas dada para la Maestría en Seguridad Informática se hace una breve clasificación de técnicas esteganográficas que se usan como referencia, ver página 5 de la presentación.

manera discreta sin modificar el comportamiento del archivo portador, mientras que la inserción directa es menos eficaz y más propensa a ser detectada. En términos de seguridad, la concatenación ofrece una mejor estrategia de ocultación, pero ambos métodos son significativamente menos robustos en comparación con las técnicas avanzadas como LSB1, LSB4 o LSBI.

7. Mejoras del Algoritmo de Majeed y Sulaiman

Las principales mejoras que tiene el algoritmo son explicadas por los mismos autores y fueron referenciadas en la sección 2.

Principalmente, el algoritmo mejora la **seguridad** en comparación a otros métodos debido a dos factores. Primero, el desuso del pixel rojo, este actúa como ruido y aumenta la complejidad para el atacante a la hora de obtener el mensaje secreto. Segundo, la inversión de bits, esta técnica añade una capa adicional evitando que el mensaje oculto sea embebido directamente.

El documento también hace mención de resultados experimentales favorables cuando se evaluó su método contra otros del campo bajo la métrica de PSNR. Lo que implica que es más difícil detectar las imágenes cuando la información se oculta por medio de LSBI.

8. Posibilidades de Almacenamiento de Patrones de Inversión

En la implementación actual de LSBI, los patrones invertidos (o el mapa de inversión) se almacenan antes del mensaje en los primeros bits del archivo BMP. Esto permite que el receptor pueda conocer cómo se invirtieron ciertos bits durante el proceso de ocultamiento y así recuperar el mensaje correctamente. A continuación, se describen otras técnicas posibles para almacenar los patrones de inversión, analizando sus ventajas y desventajas.

A. Almacenar el patrón al final del mensaje. En vez de poner el mapa de inversión al inicio se lo puede almacenar al final de los datos ocultos. Esto complica un poco el proceso de decodificación ya que se necesitaría leer primero el mensaje y luego aplicar los patrones o guardarse un *offset* al principio que indique donde está definido el mapa de inversión.

El principal beneficio de esta técnica radica en que disminuye la exposición del mapa de inversión, ya que al estar ubicado al final del mensaje es menos probable que sea detectado por un atacante que analice los primeros bits del archivo. Por otro lado una desventaja significativa es la mayor complejidad en la decodificación, que requiere modificar la secuencia estándar de lectura del archivo. Además, este método es inherentemente frágil, dado que los últimos bits del archivo pueden ser susceptibles a alteraciones o ataques, lo que podría comprometer tanto los patrones de inversión como el mensaje mismo.

B. Utilizar metadatos o secciones no visuales de la imagen. Para un 24-bit BMP es posible el almacenamiento de datos adicionales en los metadatos del archivo o en secciones no visibles de la imagen. Esta técnica aprovecha estos espacios para guardar los patrones de inversión sin afectar la apariencia visual del archivo. En particular se puede usar los campos `bfReserved1` y `bfReserved2` del `BITMAPFILEHEADER` que son campos que no suelen ser usados ni chequeados (hay otros que se pueden considerar también). Además se pueden usar los *padding bytes* del BMP ya que estos no son información que afectan visualmente la imagen.

La ventaja de esta opción radica en la facilidad de implementación, ya que los metadatos o áreas no visibles del archivo ofrecen un espacio accesible y directo para almacenar el mapa de inversión. Además, este enfoque no afecta en absoluto la calidad visual de la imagen, ya que no se modifican los píxeles visibles. Sin embargo, este método también tiene sus desventajas. Los metadatos son fácilmente accesibles y pueden ser analizados por terceros, lo que aumenta el riesgo de que un atacante descubra los patrones ocultos. Además, los metadatos suelen ser eliminados o modificados durante conversiones de formato o procesos de compresión, lo que hace que esta técnica sea frágil en muchos casos.

C. Basada en Coeficientes de Media Frecuencia DCT. Esta técnica es la más compleja y difícil de implementar, característica que la hace más segura también.

C.1. Conversión de RGB a YCbCr. Para poder realizarla el primer paso es convertir los píxeles de la imagen BMP desde RGB a YCbCr, utilizando la conversión ITU-R BT.601[7], que convierte los canales de 8 bits a este nuevo espacio de color.

$$
\begin{aligned}
Y &= 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B \\
Cb &= 128 + (-0{,}168736 \cdot R - 0{,}331264 \cdot G + 0{,}5 \cdot B) \\
Cr &= 128 + (0{,}5 \cdot R - 0{,}418688 \cdot G - 0{,}081312 \cdot B)
\end{aligned}
$$

C.2. Selección de bloque de píxeles. Seleccionamos un bloque de 8×8 píxeles en uno de los canales Y, Cb o Cr. En este bloque se embebe el patrón de 4 bits utilizando DCT. Se utilizan los canales de croma, ya que son los menos perceptibles al ojo humano.

C.3. Aplicación de la DCT. Aplicamos la DCT al bloque seleccionado de 8×8 para convertir los valores espaciales de los píxeles en coeficientes de frecuencia.

C.4. Modificación de los coeficientes de media frecuencia. Vamos a distribuir cada uno de los bits del patrón en diferentes posiciones de media frecuencia[8] en el bloque DCT. El coeficiente se convierte en impar si el bit a embeber es 1 y

[7] La conversión debe ser para canales de 8-bits ITU-R BT.601 conversión

[8] Las posiciones de media frecuencia son elegidas estratégicamente para no estar ni en las frecuencias bajas (que afectan mucho la calidad visual) ni en las altas (que son más susceptibles a la compresión).

en par si el bit es 0. Por ejemplo, se utilizan los coeficientes en las posiciones `[4, 3]`, `[3, 4]`, `[5, 2]` y `[2, 5]` para embeber los bits.

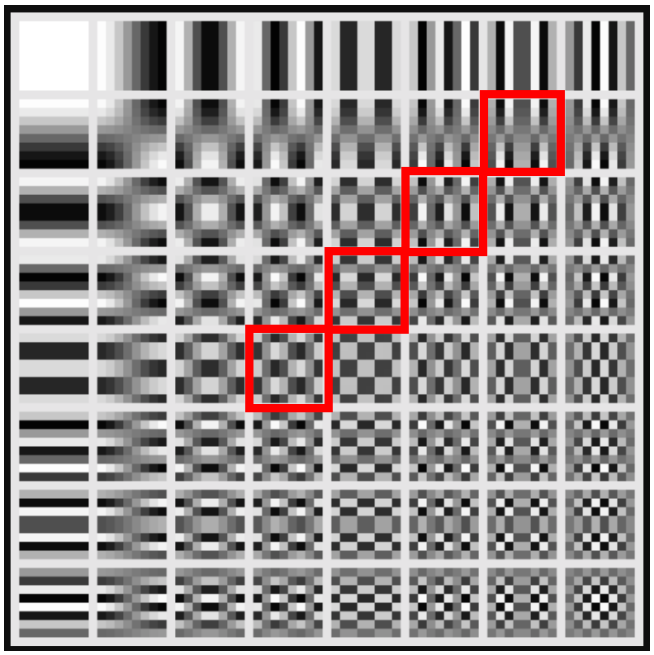


Fig. 8. En rojo las posiciones que se usarían para guardar el patrón.

C.5. Aplicación de la IDCT. Después de modificar los coeficientes de media frecuencia de la figura 8, aplicamos la IDCT (inversa de DCT) para volver a convertir el bloque modificado al dominio espacial.

C.6. Reinserción del bloque modificado. Luego insertamos el bloque modificado de vuelta en la imagen.

C.7. Conversión de YCbCr a RGB. Convertimos la imagen de nuevo a RGB para poder continuar con el proceso normal de LSBI con:

$$R = Y + 1{,}402 \cdot (Cr - 128)$$
$$G = Y - 0{,}344136 \cdot (Cb - 128) - 0{,}714136 \cdot (Cr - 128)$$
$$B = Y + 1{,}772 \cdot (Cb - 128)$$

C.8. Consideración del espacio modificado. Es importante recordar que a esta altura los primeros 8×8 píxeles han sido modificados. Al continuar con LSBI, se debe considerar este espacio como ocupado.
Nota: Tiene sentido hacer esto con el esteganografiado completo evaluando distintas técnicas además de LSBI. Este detalle fue omitido debido a que la pregunta se centraba en la modificación de la forma o posición del patrón de bits en LSBI.

9. Desafíos en la Implementación del Algoritmo de Inversión de Bits

La complejidad añadida por el algoritmo LSBI radica en su lógica de inversión de bits, lo que incrementa significativamente la complejidad tanto para insertar como para extraer la información.
Si hubiéramos implementado LSBI como primer algoritmo, la dificultad habría sido mucho mayor. Sin embargo, comenzar con LSB1 y LSB4 como pasos progresivos nos permitió adaptarnos de manera más gradual y facilitó la implementación de LSBI.
El mayor reto que enfrentamos fue el manejo del patrón de inversión durante la extracción de los datos. Consideramos que el documento no describe de manera óptima este proceso, como ya se mencionó en la Sección 1. Afortunadamente, el enunciado proporcionado por la cátedra ayudó a aclarar esta situación.

10. Mejoras Futuras para el Programa Stegobmp

- **Optimización del método de ejecución**: Se sugiere modificar la estructura de comandos para que no sea necesario detallar todos los flags en una única línea. Utilizar valores predeterminados o una interacción tipo diálogo puede mejorar la usabilidad. También se podría desarrollar una interfaz gráfica que permita visualizar todas las permutaciones posibles de forma más intuitiva.

- **Expansión a LSBX**: Dado que los métodos LSB1 y LSB4 son similares, se propone parametrizar los bits utilizados para que el usuario pueda seleccionar cuales considerar. Por ejemplo, la entrada 00000001 representaría LSB1, 00001111 representaría LSB4 y 00001000 podría representar un nuevo método que utilice el cuarto bit menos significativo para insertar o extraer información. Esta mejora también podría ampliarse a los 24 bits de un píxel, permitiendo patrones más complejos. Por ejemplo, 00000001 00000001 00000000 (BGR) sería similar a LSBI, pero sin la lógica de inversión.

- **Comando de fuerza bruta**: Incluir un comando 'force-extract', paralelo a 'extract' y 'embed', que intente obtener un mensaje oculto probando todos los métodos conocidos por el programa. Si se implementa la expansión mencionada en el punto anterior, el número de patrones posibles sería 2^{24} más los patrones únicos, como LSBI y otros métodos con etapas adicionales.

- **Información adicional**: Tras la inserción de información, el programa podría mostrar la métrica PSNR (Peak Signal-to-Noise Ratio) como parte de los resultados, así como el espacio disponible no utilizado y otra información relevante para el usuario.

- **Compatibilidad ampliada**: Se propone incluir soporte para más formatos de archivo en los cuales se pueda ocultar información, no solo imágenes, sino también texto plano, audio y video, expandiendo así el alcance de la esteganografía a otros medios.

- **Paralelización**: Añadir la capacidad de distribuir la ejecución en múltiples *threads* permitiría aprovechar mejor los recursos del CPU. Esta mejora sería especialmente útil en el método de fuerza bruta, que se beneficiaría significativamente de la paralelización.

Conclusiones

Este informe presentó un análisis comparativo de tres algoritmos de esteganografía basados en LSB (LSB1, LSB4 y LSBI). Los resultados demuestran que, si bien los métodos LSB1 y LSB4 son más simples y permiten un mayor almacenamiento de datos, LSBI proporciona una seguridad significativamente mejorada al introducir la inversión selectiva de bits. Sin embargo, esta mejora en seguridad viene acompañada de una mayor complejidad en su implementación y un menor rendimiento en términos de capacidad de almacenamiento.

En situaciones donde la seguridad es crítica, LSBI se presenta como la mejor opción, mientras que LSB1 y LSB4 pueden ser preferibles en casos donde la simplicidad y la capacidad sean más importantes. Las aplicaciones prácticas de estos métodos abarcan desde la protección de datos en sistemas de marcas de agua digitales hasta el ocultamiento de información en medios de comunicación.

El estudio se limitó a la evaluación de archivos BMP, por lo que se sugiere que futuros trabajos exploren la efectividad de estos algoritmos en otros formatos de archivo, así como la integración de técnicas avanzadas de cifrado que puedan potenciar aún más la seguridad de los métodos basados en LSB. Además, se proponen mejoras en el programa desarrollado, como la compatibilidad con más formatos de archivos y la automatización de procesos de *decoding* mediante fuerza bruta.

Bibliografía

1. Mohammed Abdul Majeed and Rossilawati Sulaiman. An Improved LSB Image Steganography Technique Using Bit-Inverse in 24 Bit Colour Image. *Journal of Theoretical and Applied Information Technology*, 80(2):342–348, October 2015. ISSN 1992-8645.