TRABAJO PRÁCTICO DE IMPLEMENTACIÓN: ESTEGANOGRAFÍA

Criptografía y Seguridad - 72.44

Grupo: 1

Profesores:

Pablo Eduardo Abad
Ana María Arias Roig
Rodrigo Ramele

Integrantes:

Federico Viera, 62022
Ramiro Luis García, 61133
Ian Franco Tognetti, 61215

Viernes, 14 de noviembre

1. Introducción

La esteganografía es la disciplina que permite ocultar la existencia de un mensaje dentro de otro objeto digital, denominado portador. A diferencia de la criptografía, que busca hacer ilegible un mensaje, la esteganografía intenta que el mensaje pase inadvertido. Esto se puede lograr insertando información dentro de imágenes, audios o videos, de modo que el contenido oculto no pueda ser detectado a simple vista.

En este trabajo práctico se implementó el programa *stegobmp*, capaz de ocultar y extraer archivos dentro de imágenes *.bmp*, utilizando tres métodos de esteganografía: LSB1, LSB4 y LSBI. Además, se realizaron pruebas comparativas y un análisis sobre archivos especiales otorgados por la cátedra, aplicando técnicas de estegoanálisis para descubrir los mensajes ocultos.

2. Análisis del documento sobre LSBI

El desarrollo de este trabajo practico se basó principalmente en analizar e implementar el algoritmo estudiado en el paper "*An Improved LSB Image Steganography Technique using bit-inverse in 24 bit colour image*" de Mohammed Abdul Majeed y Rossilawati Sulaiman, el cual propone una mejora al método LSB tradicional mediante la inversión de bits en patrones específicos.

LSBI logra un equilibrio entre capacidad y calidad: reduce la distorsión visual de la imagen al minimizar los cambios innecesarios de bits. Esta optimización se basa en el análisis de patrones de los bits menos significativos y la decisión de invertir solo aquellos que aportan mejoras perceptuales.

2.1. Organización del documento

Desde la introducción hasta la discusión y conclusiones, el documento se encuentra estructurado en 8 secciones. Si bien esta separación favorece la lectura, podría haber sido deseable agrupar las propuestas en secciones diferenciadas a las de explicación de contenidos previos, diferenciando marco teórico por un lado y todo lo referido al nuevo método por otro. De esta forma, aquellos familiarizados con los temas pueden dirigirse directamente a aquella sección del paper.

2.2. Descripción del algoritmo

La descripción del algoritmo es clara y su extensión es adecuada. Se hace un paso a paso lo cual facilita el entendimiento y provee imágenes y cuadros que facilitan la comprensión.

Un punto en contra se destaca en la sección de resultados, cuando se muestran las imágenes sin una comparación con su versión previa al algoritmo.

2.3. Notación utilizada

La notación y el lenguaje utilizado a lo largo del documento es correcta. El uso de acrónimos como LSB, PSNR, MSB están correctamente explicados. Los algoritmos están explicados con seudocódigo, lo cual vuelve más dinámica la explicación.

Un error para destacar en el texto se encuentra en la página 5 cuando nombra que la cantidad de bits de un texto de 226 caracteres es 2881. En realidad, si asumimos que cada carácter es un byte y cada byte son 8 bits, la cuenta debe dar 1808 bits.

3. Desarrollo del programa

El programa *stegobmp* fue desarrollado en lenguaje *C* y su funcionamiento se basa en dos modos principales:

- **Modo Embed (-embed)**: inserta un archivo dentro de una imagen BMP.
- **Modo Extract (-extract)**: recupera el archivo oculto dentro de una imagen BMP.

3.1. Estructura general del código

El código se organiza en módulos independientes:

- *bmp_handler.c*: manejo de lectura y escritura del formato BMP.
- *file_management.c*: lectura y escritura de archivos binarios.
- *encryption_manager.c*: cifrado y descifrado utilizando OpenSSL.
- *operations.c*: coordinación de las operaciones de embed y extract.
- *lsb1.c, lsb4.c, lsbi.c*: implementación de los tres algoritmos de esteganografía.

La compilación se realiza mediante un Makefile que genera el ejecutable *stegobmp*.

3.2. Formato de datos de esteganografía

Cada método oculta la información siguiendo el formato:

T*AMAÑO (4 bytes, big-endian)* || *DATOS DEL ARCHIVO* || *EXTENSIÓN (".txt\0", etc)*

Cuando se utiliza encripción, la secuencia completa se cifra y se guarda como:

TAMAÑO CIFRADO (4 bytes) || *DATOS CIFRADOS*

3.3. Algoritmos implementados

- **LSB1**: reemplaza el bit menos significativo de cada byte del portador por un bit del mensaje.
- **LSB4**: utiliza los cuatro bits menos significativos de cada byte.
- **LSBI**: basado en el paper de Majeed y Sulaiman (2018), realiza una inserción LSB normal y luego analiza patrones de bits para decidir qué bytes deben invertir su bit menos significativo, mejorando la calidad perceptual de la imagen resultante.

3.4. Encriptación

Se implementó encriptación con **AES (128, 192, 256 bits)** y 3DES, utilizando *PBKDF2* con *SALT* fijo en *0x0000000000000000* y los modos *ECB*, *CBC*, *CFB*, *OFB*.

4. Comparación de resultados experimentales

Como su nombre indica, estos métodos de esteganografía se basan en la modificación de los bits menos significativos de cada byte dentro de un píxel de una imagen de tipo bitmap, para ir almacenando un mensaje o archivo oculto. Cada píxel se representa con tres bytes: uno para cada color RGB (azul, verde y rojo en ese orden).

Los métodos LSB1 y LSB4 se pueden generalizar a la familia LSB-N, donde N determina la cantidad de bits menos significativos que se modifican. Es importante destacar que mientras mayor sea el N, más información podrá abarcar el píxel, pero más se notará visualmente la alteración de la imagen respecto de la original. En cuanto al algoritmo LSBI, se modifican menos bytes que en LSB1 dado que no se alteran los bits del color rojo dentro del píxel. Esto hace que sea incluso más difícil notar la alteración sobre la imagen original, pero requiere de un archivo portador más grande para esconder el mensaje.

A modo de ejemplo se comparan 2 imágenes de color azul antes y después de ser esteganongrafiada.

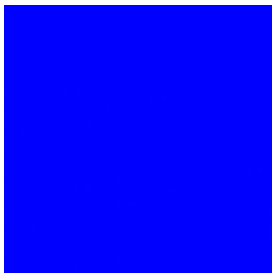

Figura 4.1: Imagen original

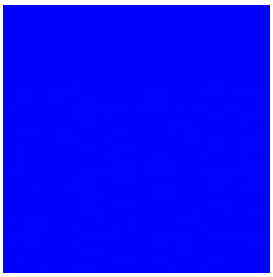

Figura 4.2**:** Imagen esteganografiada con LSB1

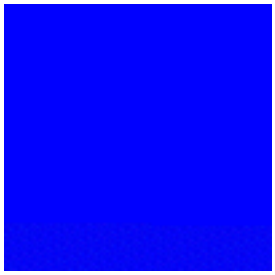

Figura 4.3: Imagen esteganografiada con LSB4

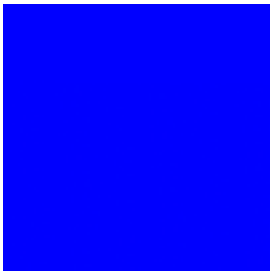

Figura 4.4: Imagen esteganografiada con LSBI

Si nos detenemos en la imagen esteganografiada con LSB4 podemos notar como la zona inferior posee un tono distinto de color, dando evidencia que fue alterada. En cuanto a las LSB1 y LSBI es más difícil diferenciarlas de la original.

A continuación, presentamos las ventajas y desventajas de cada algoritmo.

Método	Ventajas	Desventajas
LSB1	Difícil de detectar cambios visualmente sobre el archivo.	Al usar únicamente un bit por byte, se requiere de un portador 8 veces más grande que el mensaje oculto.
LSB4	Se usan 4 bits por cada byte del portador, solo necesito un portador 2 veces más grande que el mensaje oculto.	Es fácil de detectar los cambios visualmente sobre el archivo.
LSBI	Es más difícil de detectar cambios en el archivo en comparación con LSB1 dado que no cambia los bytes del rojo.	Como solo se usa el bit menos significativo del byte azul y el byte verde, se necesita de un portador doce veces más grande que el mensaje oculto.

5. Obtención del mensaje oculto

Para probar el correcto funcionamiento del sistema y a modo de desafío, la catedra otorgó a cada grupo 4 imágenes *.bmp* esteganografiadas con diferentes métodos para descifrar sus mensajes.

El procedimiento, al no tener ningún tipo de información relevante sobre el proyecto, comenzó con un ataque al estilo "*brute force*". En el cual probamos todas las posibles combinaciones entre archivos y algoritmos esteganográficos con un Script automatizado.

Esto, nos llevó a la conclusión de que los archivos *loimposible.bmp* y *buenosaires.bmp*, estaban escondiendo información no encriptada, utilizando los algoritmos de LSB4 y LSBI respectivamente. Este pseudo-ataque de Brute Forcing, fue,

por defecto, utilizando extensiones de archivo *.png*. Esto, dio resultados inmediatos con *loimposible.bmp*, que reveló la siguiente imagen:

```
./stegobmp -extract -p loimposible.bmp -out loimposible.bmp -steg LSB4
```

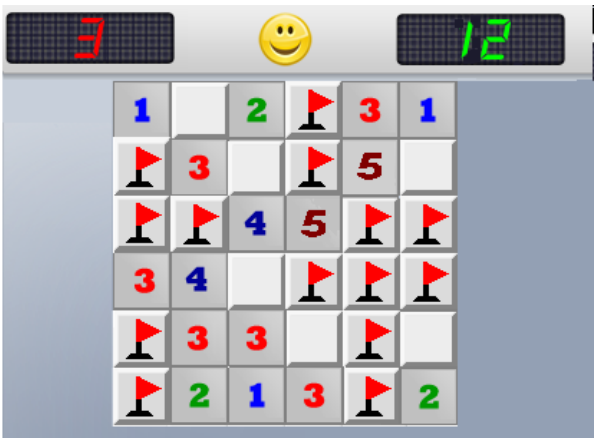

Figura 5.1: Imagen PNG de Buscaminas

Al analizar esta imagen de manera literal (.i.e: observándola) y analizando su hexdump con un editor hexadecimal, notamos que claramente nos faltaba más información. Entonces, decidimos enfocarnos en *buenosaires.bmp*.

Primero, al notar que el *.png* extraído estaba "corrupto", decidimos investigar el hexdump de este archivo utilizando un editor hexadecimal, lo cual evidenció que la extensión correcta del output de extraer el contenido de este *.bmp*, era *.pdf*.

Luego de re-extraer buenosaires.bmp, pero con extensión *.pdf*, obtuvimos un archivo *.pdf* (Obviamente) de una página, con la siguiente instrucción:

al .png cambiarle la extension por .zip y descomprimir

Figura 5.2: PDF Extraído de buenosaires.bmp

Esto, nos llevó a duplicar el archivo *.png* del juego buscaminas y cambiarle, a mano, la extensión a *.zip*. Luego, con una herramienta de Linux, "unzip", extrajimos un archivo *.txt* con las siguientes instrucciones:

“cada mina es un 1.
cada fila forma una letra.
Los ascii de las letras empiezan todos en 01.
Asi encontraras el algoritmo y el modo
La password esta en otro archivo
Con algoritmo, modo y password hay un .wmv encriptado y oculto.”

Esto, nos llevó a resolver dicho juego de buscaminas y con las instrucciones previamente obtenidas, conseguir la siguiente secuencia binaria:

01100100
01100101
01110011
01001111
01100110
01100010

La cual, fue procesada por un conversor de binario a ASCII, para devolvernos el siguiente output:

desOfb

Con el conocimiento de que en alguno de nuestros otros 2 archivos estaba las password necesaria para conseguir el *.wmv*, notamos que, ambos archivos estaban encriptados, sin que nosotros supiéramos sus respectivas contraseñas. Entonces, empezamos a buscar información por medios alternativos.

Dado a que, a lo largo del procedimiento, utilizamos el editor hexadecimal, decidimos analizar el hexdump de ambos archivos. Esto, nos llevó al final del archivo bogota.bmp, donde, al final del hexdump del mismo, se indicaba lo siguiente:

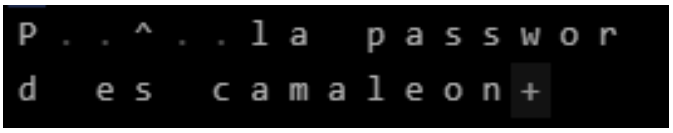

Figura 5.3: Screenshot de hexdump de bogota.bmp

Entonces, a este punto del procedimiento conocíamos, el algoritmo de encriptación (3des), el modo de encadenamiento (OFB), la contraseña (camaleon) y por descarte, el algoritmo de esteganografía utilizado[1] (LSB1) y finalmente, el archivo que tenía escondido el *.wmv* (quitoG1_.bmp). Lo único que quedaba a este punto, era simplemente utilizar toda la información recolectada para obtener el *.wmv*, con el siguiente comando:

```
./stegobmp -extract -p quitoG1_.bmp -out video.wmv -steg LSB1 -a 3des -m ofb -pass "camaleon"
```

El resultado de este comando fue el muy buscado video.wmv que mostraba un clip de 0:53 segundos de la película *Wanted (2008)* sobre el "*loom of fate*".

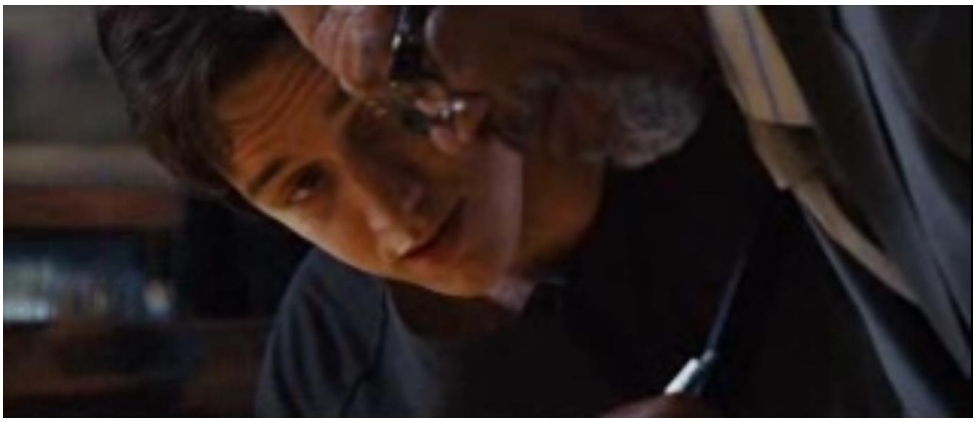

Figura 5.4: Screenshot clip de Wanted (2008), "Loom of Fate"

5.1 Explicación del fragmento de video

Una vez obtenido el mensaje oculto en *quitoG1_.bmp* se puede ver un video de aproximadamente un minuto donde se explica un antiguo método de esteganografía descubierto por un grupo de tejedores llamado La Hermandad.

Es un método que, mediante la forma de entrelazar las tramas de un tejido, y en función de la posición del hilo vertical, se puede interpretar el mensaje oculto. Cuando el hilo esta enhebrado por debajo se obtiene un 0 y si esta enhebrado por arriba se obtiene un 1. Mediante esta simple regla se podían intercambiar mensajes ocultos a simple vista utilizando como portador el tejido realizado por el telar.

5.2 Método alternativo

Se detectó un cuarto método donde se coloca el mensaje en formato literal al final del archivo. Como podemos suponer, este método cuenta con varias falencias comenzando por colocar el contenido secreto en formato literal, logrando que con un análisis muy superficial se pueda llegar al valor y, en segundo lugar, colocando este contenido al final del documento, se vuelve aún más fácil su obtención. Sin dudas este método resulta menos efectivo en términos de ocultamiento respecto a los demás métodos implementados.

6. Análisis del algoritmo LSBI

Esta variación “mejorada” del LSB expuesta en el Paper de Majeed y Sulaiman, 16Vol80No2, es indiscutiblemente superior a otras versiones anteriores de LSB (*léase, LSB1 y LSB4*) en cuanto dificulta a un atacante detectar el mensaje oculto. Sin embargo, no logra compensar su mal rendimiento en términos de tiempo de cómputo.

Al analizar la propuesta del paper, es notable denotar que, al revertir los cambios en las situaciones de mayor reemplazo de bits, se logra que el archivo no cambie de forma innecesaria y solo reporte en bits cual fue el tratamiento provocando. De esta forma la imagen, de manera visual y estadística, no sufre cambios visibles al ojo humana, o al menos, no tanto como ocurre otros métodos. Un agregado y optimización notable de LSBI, es que el canal de rojo no sea modificado, dado a que este es el color más fácilmente detectado para el ojo humano.

Finalmente, cabe destacar, que, aunque LSBI sea una notoria mejor en comparación a sus contrapartes, no logra completamente imperceptible al ojo humano.

6.1. Almacenamiento de patrones

La mejora al LSB presentada en el paper, introduce la novedad del almacenamiento de patrones invertidos antes del mensaje. Esta metodología provee mejoras como disminuir la complejidad computacional o más claridad en el algoritmo. Pues no se deben recalcular estos patrones, ya que son calculados al principio y reutilizados más adelante. Se podrían considerar como reemplazo viable las siguientes dos opciones:

- **Almacenar Patrones Originales al Final del Documento**: Como indica el título, almacenaríamos los valores de los patrones, sin invertir, al final del documento. Así, podríamos consultar a los 4 últimos bits del archivo cada vez que necesitemos derivar información de los patrones.
- **Utilizar Header del Archivo para Almacenar Información**: También, como indica el título, utilizaríamos 4 bits del Header del archivo para almacenar a los patrones invertidos. Esto debe ser implementado con mucho cuidado, dado a que un cambio en el Header, puede implicar cambios al formato del archivo.

6.2. Dificultades en la implementación

Una gran complicación a nivel de desarrollo fue la de no considerar desde el principio a **LSB1**, **LSB4** y **LSBI** como variantes del mismo método. De esta manera, pensar las 3 implementaciones de forma separada trajo demoras en el desarrollo del proyecto.

En el caso puntual de LSBI, la mayor complicación fue resolver problemas derivados de desacoples entre la lectura y la escritura de datos. A continuación, listamos cada error:

- El *pattern_map* se estaba leyendo desde componentes distintos a los que se usaban en el embed.
- Los offsets no se manejaban de forma centralizada, causando que la extracción empezara en posiciones erróneas.
- No se tenía en cuenta el padding propio del formato *.bmp*, lo cual generaba desalineamientos sistemáticos.
- El orden de los patrones y su correspondencia con los bits del *pattern_map* estaba invertido o no era consistente.
- La primera versión mezclaba la lectura del *pattern_map* y de los datos, complicando el flujo y dificultando el debugging.

La versión final del proyecto resuelve estas dificultades mediante:

- El uso homogéneo de *extract_bits_lsb*1 para leer el *pattern_map*.
- Manejo centralizado de offsets.
- La utilización de *get_component_by_index* para respetar padding y formato *.bmp*.
- Unificación del orden de los patrones.
- La separación modular del proceso de extracción.

Además, le dimos mucha vuelta a archivos rotos que trabaron el avance del análisis final. Por este motivo dedicamos tiempo a revisar todo el proyecto y la lógica de los algoritmos cuando tendríamos que simplemente haber consultado con la catedra.

7. Mejoras a futuro

Como mejoras futuras al programa *stegobmp*, se le podría agregar una opción para controlar el uso del padding en los algoritmos de encriptación. Además, también se le podría agregar el manejo de más extensiones de archivo más allá de la extensión *.bmp*. Finalmente, como extensión de *stegobmp* se podría agregar una herramienta que encuentre posibles mensajes embebidos en el archivo portador, probando con cada método conocido.

8. Conclusiones

El algoritmo LSBI representa una mejora efectiva frente a los métodos LSB tradicionales, ya que optimiza la cantidad de bits modificados y mantiene una mayor

fidelidad visual. Sin embargo, su implementación requiere mayor complejidad y una correcta gestión de los patrones de inversión.

En cuanto al desarrollo, a lo largo del trabajo se pudieron entender los diferentes mecanismos de esteganografía, notar las mejoras entre algoritmos, identificar los tiempos de cómputo de cada uno.

Durante la implementación se presentaron dificultades para reproducir exactamente el comportamiento descripto en el paper y obtener los resultados esperados. Sin embargo, el desarrollo del proyecto sirvió como herramienta para poner en práctica los temas vistos durante la cursada.

Bibliografía

- Majeed, M.A. & Sulaiman, R. (2015). *An Improved LSB Image Steganography Technique using bit-inverse in 24 bit colour image*. Journal of Theoretical and Applied Information Technology.