Criptografía y Seguridad

Trabajo Práctico Especial / 2018
Esteganografía sobre imágenes BMP

Carla Barruffaldi
Luciano Bianchi
Tomás Cerdá
Marcelo Lynch

Introducción

En pos de introducirnos al mundo de la esteganografía y sus aplicaciones, se implementó un programa `stegobmp` en Java que se encarga de: ocultar un archivo cualquiera en un archivo `.bmp` mediante algún método de esteganografiado con o sin password y descubrir archivos ocultos con un método de esteganografiado en archivos `.bmp`.

Se implementaron los siguientes métodos de esteganografía: LSB1, LSB4 y LSB Enhaced (o ELSB). A continuación se detalla un análisis respecto a la implementación del programa y el desafío propuesto para el trabajo práctico.

El enfoque del análisis estuvo puesto en la efectividad de la esteganografía de cada método desde un punto de vista de la aplicación práctica: eficiencia en el transporte de información, y facilidad de detección.

Análisis

Se responden a continuación las preguntas planteadas para el trabajo práctico

0. Para la implementación del programa se pidió que la ocultación incluya los bytes de padding del formato BMP. ¿qué ventajas y desventajas presenta esto frente a no usarlos?[1]

La ventaja viene por dos lugares: por un lado, simplifica bastante la implementación, ya que no hay implementar toda la lógica para manejar este caso (aunque no sería increíblemente complejo teniendo en cuenta que el header incluye el ancho de la imagen en píxeles). Por el otro, agrega espacio para el esteganografiado (de 3 a 9 bytes más para esteganografiar por fila).

La desventaja clara viene por el lado de una posible detección en el estegoanálisis: aunque el estándar de BMP no especifica qué valores deben estar en los bytes de padding, es común encontrar ceros en esos bytes. Si estegoanalizando se encuentra que esos bytes son diferentes de cero es más sencillo detectar que hubo una esteganografía.

1. Para la implementación del programa stegobmp se pide que la ocultación comience en el primer componente del primer píxel. ¿Sería mejor empezar en otra ubicación? ¿Por qué?

Mientras menor sea la regularidad introducida en el esteganografiado, mayor es la dificultad de detectarlo: una mejora obvia sería, en lugar de comenzar por el primer píxel y seguir por los

[1] Esta pregunta no está en la consigna pero fue pedida por CAMPUS

siguientes, tomar los pixeles que indique un un generador pseudoaleatorio (cuidando de no repetir las posiciones)[2].

2. ¿Qué ventajas podría tener ocultar siempre en una misma componente? Por ejemplo, siempre en el bit menos significativo de la componente azul.

Una ventaja podría ser que al adulterar una única componente de un color específico las irregularidades visuales de la imagen no son tan notorias, en contraste a si se adulteraran las 3 componentes. Esto como resultado dificulta los análisis para distinguir si la imagen se trata de una esteganografiada o no. La componente podría elegirse según las características de cada imagen.

3. Esteganografiar un mismo archivo en un .bmp con cada uno de los tres algoritmos, y comparar los resultados obtenidos. Hacer un cuadro comparativo de los tres algoritmos estableciendo ventajas y desventajas.

Es difícil establecer una comparación visual codificando el mismo archivo con los tres métodos, por la característica de que para una imagen dada el método ELSB tiene muchísima menos capacidad: en la tabla 1 se muestra a modo de ejemplo la cantidad máxima en bytes que se puede codificar sobre una imagen de *Composición en amarillo rojo y azul*[3] (Figura 1).

En pos de comparar bien, se codificaron bytes aleatorios (usando siempre un generador con la misma semilla) según la capacidad máxima de cada método en la imagen. La figura 2 muestra los resultados para cada método: se observa que no hay diferencias notables ni con LSB1 ni con ELSB, en cambio LSB4 afecta claramente la paleta de colores y degrada la imagen. La misma observación puede hacerse en la figura 3, donde se repitió este procedimiento sobre una imagen completamente blanca (notar que en este caso ELSB sería igual que LSB1). El efecto sobre imagénes con colores sólidos (generadas digitalmente en lugar de una fotografía) es notorio.

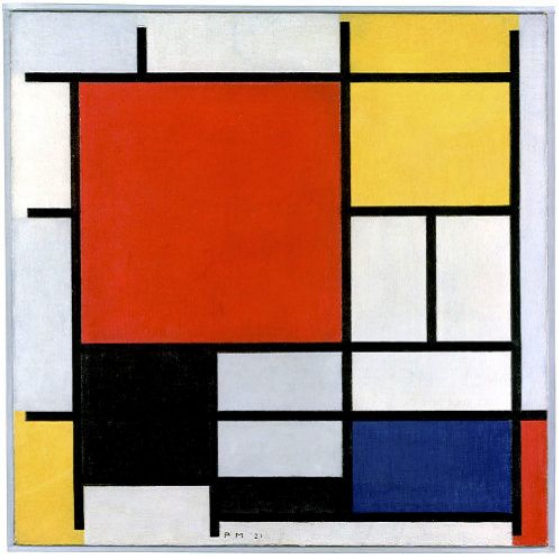

Figura 1. Imagen usada para analizar capacidad

[2] Esto implica también, por supuesto, comunicar la semilla del generador para poder realizar la decodificación, lo que puede ser un problema en sí mismo.
[3] Se elige esta imagen por tener notoriamente colores "brillantes", que son en principio los adecuados para codificar ELSB: si bien resultó "mejor" en este sentido que otras imagenes que se probaron, embargo, solo el 3,8% de los bytes resultan con valores mayores a 253 que son los que se utilizan para codificar.

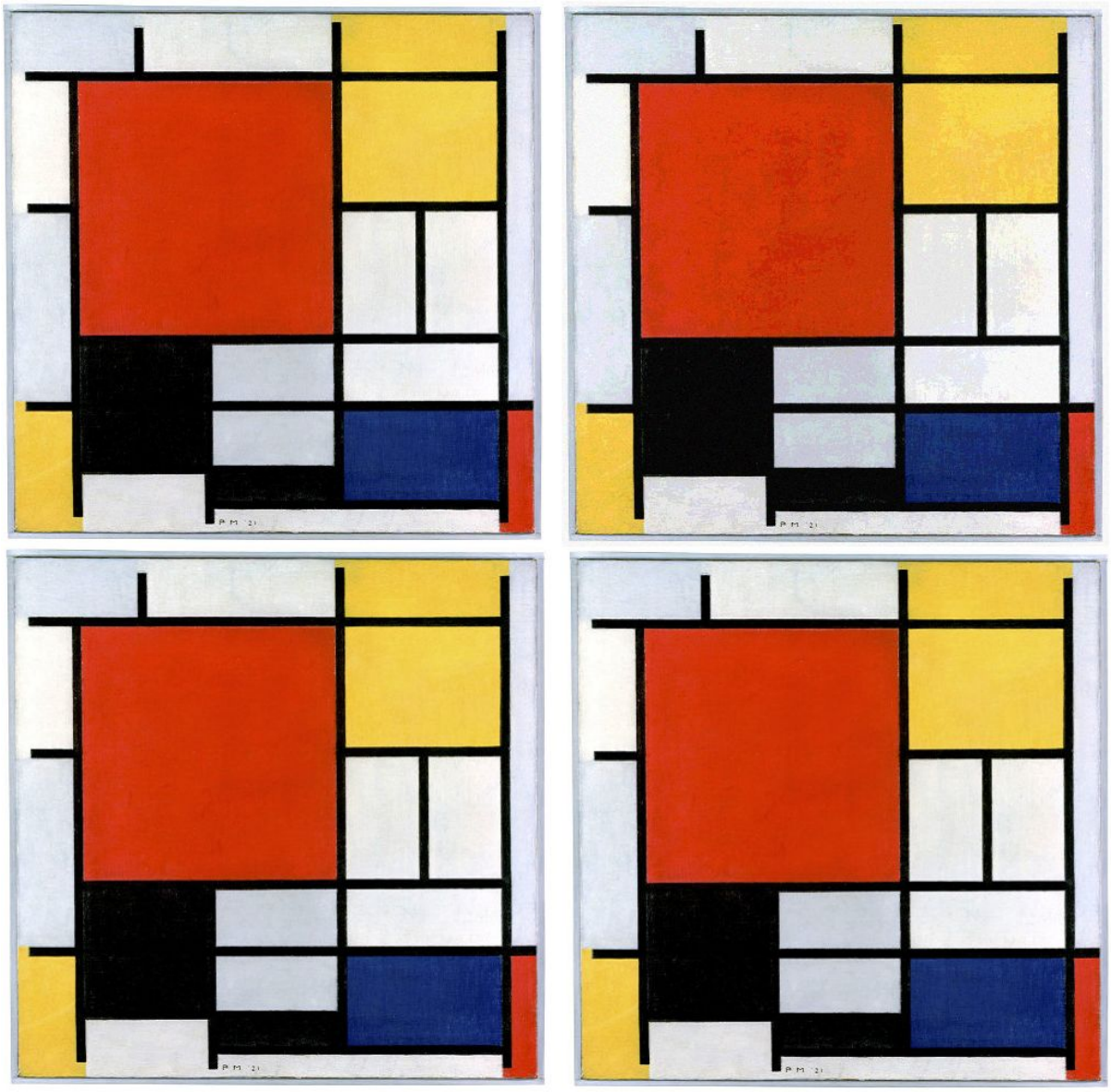

Figura 2. Superior: izquierda, original; derecha, LSB4
Inferior: izquierda, LSB1. derecha: ELSB

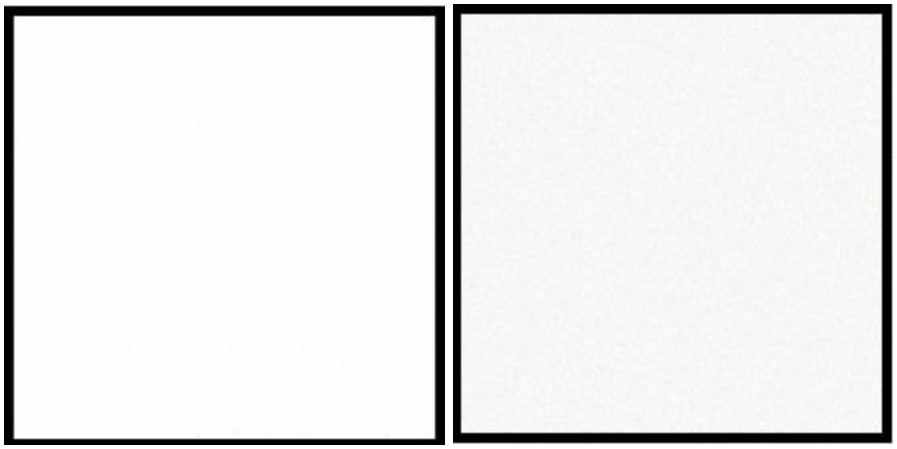

Figura 3. Imagen blanca con ruido esteganografiado: LSB1 (izquierda) y LSB4 (derecha).
Los bordes negros se agregaron después para visibilidad del contenido blanco

Método	LSB1	LSB4	ELSB
Capacidad	Mediana (12.5%)	Alta (50%)	Baja
Capacidad en imagen de ejemplo	93750	375.000	3.597 (3,8%)
Estegoanálisis	Propenso	Más propenso	Mucho menos propenso
Notoriedad	Baja	Mediana/Alta	Muy baja

Tabla 1. Tabla comparativa entre los distintos algoritmos

4. Para la implementación del programa `stegobmp` se pide que la extensión del archivo se oculte después del contenido completo del archivo. ¿Por qué no conviene ponerla al comienzo, después del tamaño de archivo?

Una de las características deseadas a la hora de ocultar información es que la adulteración de los bits sea lo menos regular posible y así sea más difícil de detectar. Colocar la extensión al principio no promueve esto pues suele tener siempre el mismo formato: *.txt*, *.zip*, *.pdf*, *.html*, etc.

Si dos agentes intercambian regularmente archivos esteganografiados con la misma extensión a través de un canal público que está siendo investigado por un tercer agente, este último podría comparar las imágenes intercambiadas y distinguir patrones/similitudes en los bytes siguientes al del tamaño del archivo. Esto no sería tan sencillo si la extensión se encuentra al final, pues el tamaño del contenido de los archivos es variable. Además, una vez conocido el formato del archivo, se puede usar para buscar patrones en el resto de los bytes de la imagen; por ejemplo, si se determina que es un *.html*, se pueden usar tags conocidos como * * o *<body>* para reconocer patrones en el esteganografiado.

5 + 6. Explicar detalladamente el procedimiento realizado para descubrir qué se había ocultado en cada archivo y de qué modo. ¿Qué se encontró en cada archivo?

Por la consigna se sabía que había un esteganografiado LSB1, uno LSB4, uno ELSB y uno en un modo que había que descubrir.

Por lo tanto, se procedió por "tanteo" a intentar des-esteganografiar algún archivo que hubiera sido codificado sin encriptación utilizando las imágenes provistas, considerando que es obvio si la decodificación es exitosa pues se debe obtener algo "con sentido". Además de eso, existe otra condición que permite descartarlo todavía más rápido: asumiendo un esteganografiado, al recuperar el tamaño del archivo (los primeros 4 bytes) se puede verificar si realmente "cabe" ese esteganografiado en el bitmap: si ese número es mayor al máximo posible, entonces no hay forma de que sea un esteganografiado válido.

Con estas consideraciones, se descubrieron archivos en `back.bmp` (con LSB4) y en `budapest.bmp` (con ELSB). Los resultados de este "tanteo" con los archivos `madridoso.bmp` y `bogota.bmp` se discuten más adelante.

El archivo descubierto en `back.bmp` tiene el formato PNG (su hash md5 es `c5e2ed148d1536eb879cdd64568a18a5`), y corresponde a la imagen de la figura 5.

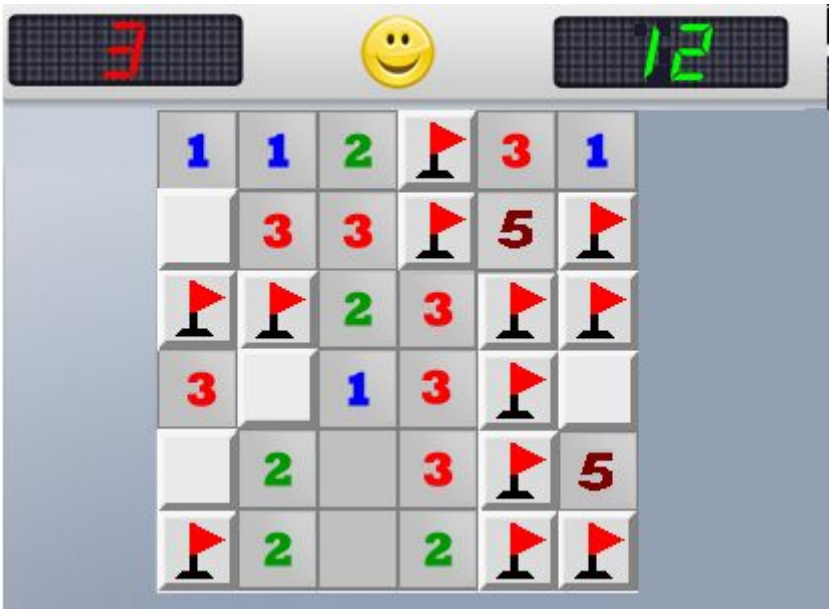

Figura 5. Imagen descubierta en back.bmp

El archivo descubierto en `bogota.bmp (md5: 4215166f788c68bffe1b18774f6c36f9)` es un PDF con la frase:

> al .png cambiarle la extension por .zip y descomprimir

En efecto, al cambiarle la extensión a `.zip` al PNG obtenido a partir de back.bmp, y descomprimiendo ese archivo, se obtiene un archivo `sol3.txt` con el contenido:

```
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.
```

Utilizando la imagen del buscaminas (y terminando de resolverlo, pues hay minas no marcadas en la imagen), interpretando cada fila como los seis bits menos significativos de un carácter ASCII (los dos más significativos, como dice la pista, son `01`), donde una mina es un 1 y lo demás es un 0, obtenemos:

Binario	Hexadecimal	ASCII
01000100	44	D
01100101	65	e
01110011	73	s
01000011	43	C
01000011	62	B
01100011	63	c

Con esto descubrimos que el algoritmo de encripción para el `.wmv` encriptado es DES y el modo de encriptación es CBC.

A la hora de encontrar el archivo cifrado hay que remontarse a los resultados del "tanteo" sobre `madridoso.bmp` y `bogota.bmp,` y a un descubrimiento hecho mientras se desarrollaba el programa, antes de este estegoanálisis. En esa etapa, utilizando los archivos provistos por la cátedra para realizar pruebas, se comprobó que lo que estaba siendo esteganografiado en el caso del cifrado no era

```
tamaño del texto cifrado || texto cifrado
```

donde, si **`Enc`** es el algoritmo de cifrado, es:

```
texto cifrado = Enc(tamaño del archivo || archivo || extensión)
```

sino

```
tamaño del texto cifrado || texto cifrado || extensión
```

Esto es interesante porque así, el esteganografiado queda con el mismo formato que el esteganografiado en plano, es decir:

```
tamaño || contenido || extensión
```

Esto resulta en que si se usa el desesteganografiador de texto plano sobre una cosa que tiene este formato, se obtiene como salida el binario del archivo encriptado (con la extensión del propio archivo).

En efecto, cuando se utilizó el método LSB1 sobre `madridoso.bmp` la salida fue un archivo `.wmv` que no se pudo reproducir. Con la información obtenida en los archivos exitosamente descubiertos, sumado a lo recién explicado, es lógico suponer que la salida obtenida es la encripción de el verdadero `.wmv` (obviamente eso no se puede reproducir), y además que está hecho con LSB1.

Finalmente, cuando se probó con el archivo `bogota.bmp` no pudo obtenerse ningún archivo válido (con cualquier estrategia de esteganografiado se obtenía un tamaño -al decodificar los supuestos primeros 4 bytes- inválido). Considerando la pista `"La password esta en otro archivo"` se intentó adivinar la password a partir de lo que contenía el archivo (las palabras del

cartel, la ciudad que muestra la foto), y finalmente se procedió a explorar directamente el binario de los distintos archivos.

La contraseña se encontró, como era de esperarse, en `bogota.bmp`, insertada directamente en el binario como texto ASCII, al final del archivo (en definitiva, una manera de esteganografiar un texto).

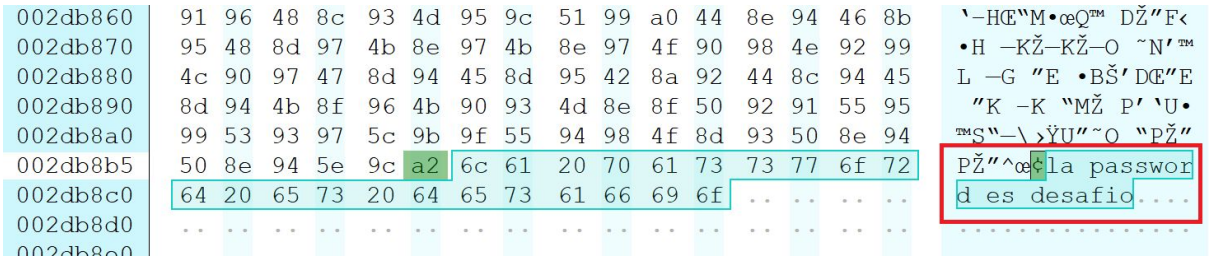

Figura 6. La contraseña en `bogota.bmp`, como la muestra Hex Editor Neo

Finalmente se procede a obtener el archivo oculto en `madridoso.bmp` utilizando la decodificación con desencripción. Se obtiene efectivamente un archivo WMV (`md5: 0f89d7fb8c178f06a77d6bc5cf0fdd0c`), con una escena de algún programa policial de FOX; se sospecha que se trata de BONES:

Figura 7. Captura de pantalla del video WMV obtenido

7. Algunos mensajes ocultos tenían, a su vez, otros mensajes ocultos. Indica cuál era ese mensaje y cómo se había ocultado.

El archivo PNG obtenido a partir de `back.bmp`, como ya se explicó, tenía oculto un archivo ZIP. En realidad, los datos binarios corresponden a los datos del PNG y los del ZIP uno a continuación del otro:

```
binario_nuevo = binario_PNG || binario_ZIP
```

El archivo puede interpretarse como PNG pues este formato tiene una cabecera al principio del archivo (los primeros bytes) como el BMP, que indica las dimensiones, etc. de la imagen (luego no

se leen los bits del ZIP, solamente se toman los datos de la imagen). Para descomprimir un archivo ZIP, en cambio, se obtiene la información de los *últimos* bytes del archivo, luego también se puede interpretar esto como un ZIP y descomprimirlo correctamente.

8. Uno de los archivos ocultos era una porción de un video, donde se ve ejemplificado una manera de ocultar información ¿cuál fue el portador?

El portador era `madridoso.bmp`. En la escena se discute la ocultación de información en un archivo, detectada porque el archivo tiene un tamaño incorrecto. Esto se alinea por ejemplo con el modo en que se ocultó el `.zip` en un `.png` (por ejemplo, se podría ocultar de esa forma un ZIP de 500MB y mostrarlo como una imagen PNG de 50x50 píxeles, que sería sospechoso que tuviera ese tamaño).

9. ¿De qué se trató el método de estenografiado que no era LSB? ¿Es un método eficaz? ¿Por qué?

Se utilizó el método ELSB o Enhanced LSB1, que codifica igual que LSB1 (utilizando un solo bit, el menos significativo, de las componentes), pero únicamente sobre las componentes que tienen valor 254 ó 255.

Esta técnica tiene el objetivo de reducir la posibilidad de detección del esteganografiado, ya que no codifica con una frecuencia (o bitrate) fija, algo que la mayoría de los algoritmos de detección basados en análisis estadísticos asumen. Al eliminar parámetros fijos en el esteganografiado se dificulta la tarea de un atacante a la hora de encontrar patrones que le permitan encontrar la información oculta en la imagen. Además, vuelve inútil a la estrategia de usar un método de fuerza bruta que pruebe todos los bitrates (fijos) posibles.

Por otro lado, la desventaja es que se reduce la capacidad de almacenamiento de una imagen, ya que solamente se pueden alterar las componentes que tengan ese valor (es más: mientras la imagen tenga mayor cantidad de componentes 254 / 255, los análisis estadísticos que funcionan sobre LSB1 se vuelven también más efectivos).

10. ¿Qué mejoras o futuras extensiones harías al programa stegobmp?

Extensiones:

- Agregar soporte para más tipos de bitmap
- Poder ocultar información en otro tipo de archivos portadores además de bitmaps como por ejemplo audio, video, documentos o incluso headers de protocolos de comunicación como UDP.
- Extender el método ELSB para que además de codificar los bytes 254 y 255, que pueda recibir por parámetro los valores de los bytes a codificar, dando así mayor flexibilidad acerca de las características que deben cumplir las imágenes a usar como portadoras.
- Agregar estocasticidad al encodeo: por ejemplo, se puede generar la secuencia con un PRNG (cuidando de no repetir bytes).