Trabajo Práctico - Criptografía y Seguridad

Patrick Dey
Santos Rosati
Matias Lombardi

Índice

Introducción

El objetivo de este trabajo práctico era implementar distintos algoritmos de estenografiado con el fin de poder ocultar y extraer distintos archivos en imágenes *bmp*. Para esto, utilizamos el lenguaje de programación **C** con las librerías de desarrollo de **OpenSSL** para poder encriptar y desencriptar los mensajes a ocultar. Luego, para implementar los algoritmos utilizamos la documentación brindada por la cátedra: el enunciado del Trabajo Práctico y el paper escrito por Nadeem Akhtar, Shahbaaz Khan, Pragati Johri.

En cuanto al paper de **LSBI**, el mismo consiste en una introducción explicando qué es la esteganografía seguido por una parte de implementación, donde explican los algoritmos clásicos y una sección donde se discuten dos nuevos posibles esquemas de esteganografía. Luego posee una sección de resultados (y análisis de los mismos) en donde explica las mejoras de los esquemas propuestos. Finalmente, hace una reflexión sobre la eficacia y seguridad de estos esquemas y qué posibles consideraciones se podrían tener a futuro.

El algoritmo **LSBI** consiste en analizar el tercer y segundo bit menos significativo de todos los píxeles del archivo *bmp*. Por cada uno de estos patrones (en los que existen cuatro combinaciones posibles: 00, 01, 10 y 11), chequear cuántos bits menos significativos cambiaron. Si cambiaron más bits que los que no cambiaron, entonces se invierten, caso contrario, se deja igual.

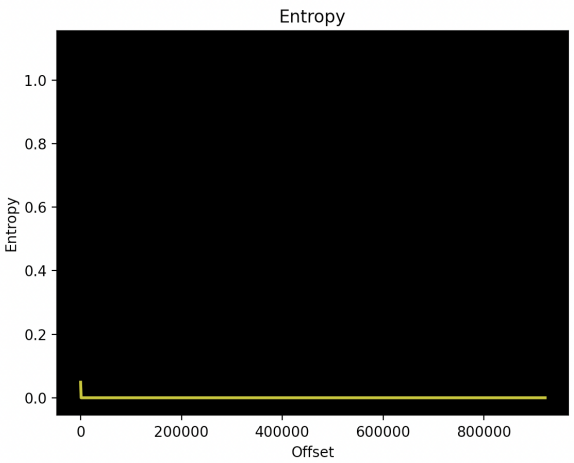
La descripción del algoritmo es mayormente clara, salvo por la explicación sobre si la combinación que hay que considerar es TERCER bit menos significativo - SEGUNDO bit menos significativo o al revés (esta consulta fue respondida por la cátedra).

Comparación de Algoritmos

Para poder hacer una comparación visual de los algoritmos, utilizamos un archivo *gris.bmp* (como indica el nombre, una imagen completamente gris) generado con paint. En esta, embebimos la imagen *charizard.png* con los 3 algoritmos esteganográficos implementados en el TP: **LSB1**, **LSB4** y **LSBI**. Todos sin encripción. Luego para ver que tan bien funciona cada uno para ocultar información, obtuvimos un gráfico de la entropía de los archivos de salida. Finalmente, modificamos el contraste, la saturación y la exposición de las imágenes de salida para poder visualizar las diferencias. Para **LSB1** y **LSBI** a simple vista no había diferencias, mientras que en **LSB4** era notorio que la imagen había cambiado por lo que esta salida está sin modificar. Como se puede observar en las siguientes imágenes, los píxeles alterados se encuentran en la parte inferior de la imagen. Esto se debe a que los archivos bmp se leen de abajo hacia arriba y de izquierda a derecha.

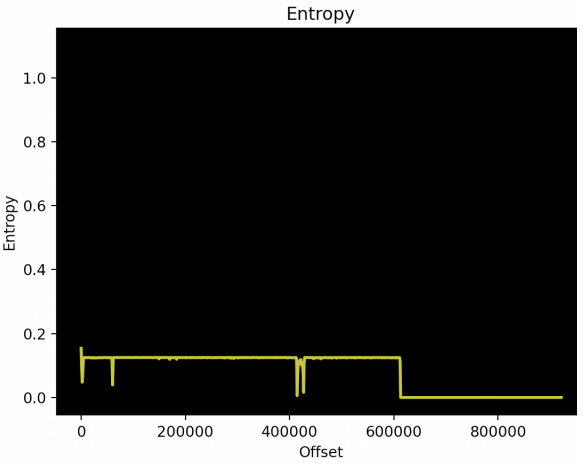
charizard.png

Imagen original



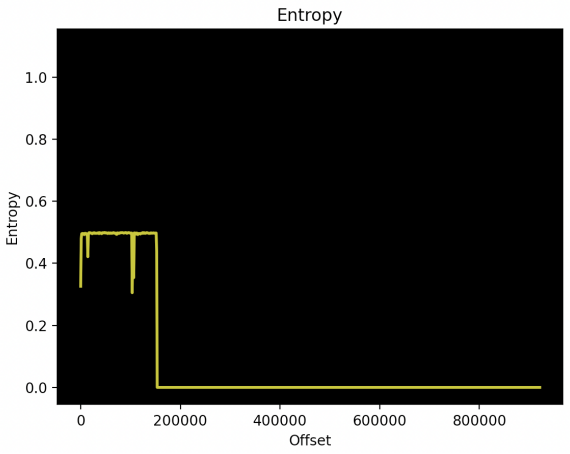
Entropía imagen original

Salida LSB1



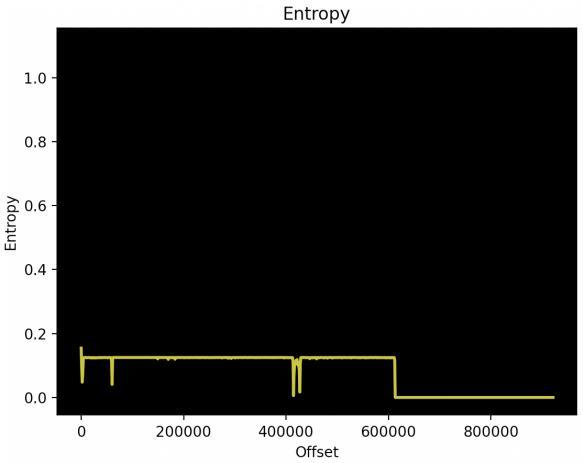
Entropía salida LSB1

Salida LSB4



Entropía salida LSB4

Salida LSBI



Entropía salida LSBI

	LSB1	LSB4	LSBI
Análisis de imagen	A simple vista, los cambios eran imperceptibles. Al modificar la imagen como se detalló anteriormente, se ven cambios uniformes ya que los cambios son siempre de a 1 bit (siendo este el menos significativo)	A simple vista se notan cambios en los píxeles de la parte inferior de la imagen, sin aplicar ninguna modificación, ya que se modifican 4 bits por cada byte de la imagen original	Similar a **LSB1**, no se ven cambios a simple viste y requirió aplicar modificaciones en la imagen
Análisis del gráfico de entropía	Los cambios al archivo se distribuyen a lo largo de los primeros 600.000 bytes de forma parecida. Esto se debe a que se van tomando de a un bit por byte	Los cambios al archivo son más abruptos (mayor entropía, mayor incerteza ya que se modifica la mitad del byte) y están más desplazados hacia la mitad de los primeros 200.000 bytes	Similar a **LSB1** pues el algoritmo solamente invirtió un patrón, por lo tanto el resultado final es muy parecido
Ventajas	Altera muy poca información de la imagen ya que toma 1 bit de cada pixel, es invisible al ojo humano	Al tomar la mitad de un byte por byte del archivo a embeber, ocupa mucho menor espacio que los otros algoritmos	Es el que menor cantidad de bits modifica a la salida. De esta forma modifica muy poco el archivo portador
Desventajas	Se necesitan 8 bytes por cada 1 byte del archivo a embeber por lo que es necesario utilizar archivos grandes	Cómo altera más bits por píxel que los otros algoritmos, hay mayor posibilidad de que sea visto a simple vista	Al igual que en **LSB1**, se necesitan 8 bytes por cada 1 byte del archivo a embeber + 4 bytes por los patrones

Estegoanálisis de archivos ocultos

La cátedra nos brindó 4 archivos *bmp* a sabiendas de que cada uno tenía alguna información oculta. En el enunciado se especificaba que había un archivo que utilizaba **LSB1**, otro **LSB4** y otro **LSBI**. Por último, un cuarto archivo que tenía información oculta con otro método.

Lo primero que hicimos fue probar con **LSB1** sin encripción sobre todos los archivos y dimos con que la imagen *avatar.bmp* contenía la imagen *buscamina.png*.

Luego probamos el mismo método con **LSB4** sobre los 3 archivos faltantes y no ocurrió nada, así que por descarte, volvimos a probar con **LSBI** sin encripción y obtuvimos que el archivo *quito.bmp* tenía oculto un archivo .pdf que tenía escrito dentro un mensaje: **"La password es sorpresa"**. Esto nos indicaba que uno de los dos archivos faltantes tenía que estar estenografiado con **LSB4** y que el archivo oculto había sido encriptado utilizando la password "sorpresa". El problema es que no sabíamos cuál era el algoritmo y el modo que se había utilizado para encriptarlo.

Para ello, recurrimos a probar distintos métodos vistos en clase como por ejemplo, ver si en los archivos faltantes (*buenosaires.bmp* y *budapest.bmp*) había alguna cadena de caracteres estática que pudiera tener información. Utilizando el comando **strings** sobre *buenosaires.bmp* obtuvimos un texto plano que decía **"mal .png cambiar extensión por .zip y descomprimir"**.

Este método estenográfico es muy simple ya que agrega información al archivo.bmp sin alterar la salida pero a su vez, es muy fácil de detectar si utilizamos programas de detección de cadenas estáticas como **strings** o si hacemos un **hexdump**.

Como el único archivo *png* que teníamos era el buscaminas, le cambiamos la extensión a *zip* y luego lo descomprimimos utilizando el comando **unzip** de la terminal. Otra posible forma de extraer el *zip* es utilizando el comando **binwalk**, que prueba en un determinado archivo todas las extensiones conocidas y así obtiene todos los archivos que estaban dentro del mismo.

Cambiar la extensión de un archivo y obtener otro tipo de información se puede hacer ya que en un mismo archivo pueden haber distintos conjuntos de datos en distintos formatos sin pisarse unos a otros. Por ejemplo, los datos que representan la imagen *png* están acompañados de un header con la información necesaria para saber cómo leer la imagen y una marca de fin que le indica al sistema operativo que todo lo que viene a continuación no debe leerlo. Luego, la información del *zip* estará acompañada con su propio header y su marca de fin. De esta manera, cuando le pedimos abrir el archivo con la extensión *zip* por ejemplo, el SO va a buscar el header correspondiente a un *zip* y no a un *png*.

El *zip* que estaba oculto en *buscaminas.png* contiene un archivo llamado *sol.txt* que, al leerlo, nos daba la clave para interpretar la imagen y obtener otro texto oculto dentro de la misma, con el cual llegaríamos al algoritmo y modo de cifrado. Con ello obtuvimos que el algoritmo de encripción era **aes** en modo **ecb**. Como el texto decía que la clave era de 128 bits, el algoritmo utilizado es **aes128**.

0	1	0	0	0	0	0	1	A
0	1	1	0	0	1	0	1	e
0	1	1	1	0	0	1	1	s
0	1	0	0	0	1	0	1	E
0	1	1	0	0	0	1	1	c
0	1	1	0	0	0	1	0	b

Secreto de buscaminas.png

Finalmente, nos quedaba un último archivo sin investigar. Hicimos un *extract* con **LSB4** utilizando **aes128**, **ecb** y la *password* obtenida para desencriptar sobre el archivo *budapest.bmp* y obtuvimos un video *wmv*.

El video contiene una escena de la película **WANTED** en la cual, Morgan Freeman le explica al personaje que en el hilar de la fábrica están ocultos nombres de personas los cuales la fraternidad deberá asesinar. Esto se relaciona con la materia ya que el método de la fraternidad de asesinos para ocultar estos nombres es, básicamente, un método estenográfico. A simple vista, nadie podría detectar esta información, pero sabiendo buscar los puntos de costura que se explica en el video, es simple.

Algoritmo LSBI

El algoritmo **LSBI**, propuesto por *Akhtar, Khan y Johri*, es realmente una mejora comparado con **LSB1** o **LSB4** porque reduce la cantidad de bits alterados de la imagen original. Al utilizar **LSB1** y verificar si la cantidad de bits alterados (por patrón) es mayor que la cantidad de bits sin alterar y luego invertirlos, la imagen de salida es más parecida a la original comparado con **LSB** común.

La primera implementación propuesta de este algoritmo guarda en los primeros 4 bytes al estilo **LSB1**, un bit por cada patrón posible indicando si el bit menos significativo fue invertido (1) o no (0). Una alternativa que se nos ocurre es utilizar **LSB4** únicamente para esta información y así utilizar 3 bytes menos. Además sería más fácil obtener esta información.

Segundo esquema LSBI

Las ventajas del segundo esquema con respecto al primero es que altera incluso menos bits que la primera implementación ya que tiene en cuenta más patrones, al agregar el bit menos significativo. En este caso, tendremos 8 patrones en vez de 4. Las 4 combinaciones posibles del segundo y tercer bit menos significativo y las 4 combinaciones con respecto al bit menos significativo. Si era 0 y se mantuvo, si era 1 y se mantuvo, si era 0 y pasó a ser 1 o si era un 1 y pasó a ser 0.

Por otro lado, este algoritmo conlleva una desventaja que es que el receptor debe tener una copia de la imagen original para poder ejecutar el algoritmo. Esto es un paso extra que la primera implementación no sufre. Además, si no se cuenta con un canal seguro y un atacante puede interceptar las imágenes, se podrá dar cuenta de que hay un mensaje oculto (y además si conoce las técnicas de estegoanálisis lo podrá obtener). A su vez, al estar buscando más patrones y realizando más operaciones, la complejidad se verá un poco deteriorada.

Dificultades

En cuanto a las dificultades encontradas para implementar el algoritmo **LSBI** no fue tanto la codificación del mismo, sino entender bien cuales eran los bits que debíamos alterar. En el paper no queda muy claro que hay que analizar los 4 patrones posibles para el segundo y tercer bit menos significativo, tuvimos que recurrir a preguntar en clase cual era la forma correcta de hacerlo.

Por otro lado, tuvimos que pensar bien cómo almacenar los bits que especifican si cada patrón había sido invertido o no a la hora de embeber un archivo. Nos ocurrió que por un mal manejo de memoria, las imágenes de salida tenían los colores distorsionados como en la siguiente captura:

lado.bmp alterada

Mejores y futuras extensiones

En cuanto a la implementación nos quedaron algunas mejoras pendientes como mejorar la modularización de código y la limpieza de código. En cuanto al programa, se podría agregar la segunda implementación de **LSBI** que se encuentra en el papel o agregar una funcionalidad para graficar la entropía como el programa **binwalk** o que realice una búsqueda de todas las extensiones dentro de un mismo archivo (de la misma manera que se ocultó el *zip* en el *png*).

Otra extensión que se nos ocurre sería agregar la posibilidad de que, dado un archivo que sabemos que tiene algo embebido, que stegobmp nos diga qué algoritmo de estenografía utilizó, aunque sería mucho más costoso ya que debería hacer un análisis de la imagen o en el peor caso, debería probar todas las combinaciones de **LSB** posibles.