Instituto Superior Politécnico José Antonio Echeverría

Facultad de Ingeniería Informática

******

Informe

Esteganografia en BMP

Autor

Camilo José del Real Martell

La Habana

Mayo, 2013

De acuerdo a algunas definiciones, esteganografía es el arte de cifrar, o en caracteres, que no son inteligibles excepto a personas que tienen la clave (criptografía). En términos computacionales, la esteganografía ha evolucionado en la práctica de esconder un mensaje en uno más grande de forma que los demás no puedan saber de la presencia o contenidos del mensaje oculto. Actualmente, la esteganografía ha evolucionado en la forma de esconder un archivo en forma de multimedia, como una imagen, un archivo de audio (.wav o .mp3) o incluso un archivo de video.

A continuación se expone como llevar a cabo esteganografía en un fichero BMP.

Esta es la estructura de un Bmp (wikipedia):

0, 1 Tipo de fichero "BM"

2, 3, 4, 5 Tamaño del archivo

6, 7 Reservado

8, 9 Reservado

10, 11, 12, 13 Inicio de los datos de la imagen

14, 15, 16, 17 Tamaño de la cabecera del bitmap

18, 19, 20, 21 Anchura (píxels)

22, 23, 24, 25 Altura (píxels)

26, 27 Número de planos

28, 29 Tamaño de cada punto

30, 31, 32, 33 Compresión (0=no comprimido)

34, 35, 36, 37 Tamaño de la imagen

38, 39, 40, 41 Resolución horizontal

42, 43, 44, 45 Resolución vertical

46, 47, 48, 49 Tamaño de la tabla de color

50, 51, 52, 53 Contador de colores importantes

Método empleado: Less Significant Bit (Bit menos significativo)

El proceso consta de modificar el bit menos significativo de la información de color; esto es tan sencillo como colocarse al inicio de los bytes que describen los pixeles y comenzar a modificar el bit menos significativo de cada color. Por cada 3 bytes son un pixel, pero a la hora de programar esto no es importante pues al modificar cada color se modifica cada byte leído, por lo que es suficiente con leer un byte y modificarle su bit menos significativo en caso necesario.

En la siguiente fracción de código binario, el bit marcado es el menos significativo:

1011010**0**

1110101**1**

1001010**0**

Se tuvo en cuenta a la hora de programar el algoritmo, guardar una firma (texto LSB) de tamaño 3 bytes (24 bits que serían 24 bytes de colores necesarios) para poder identificar que el fichero contiene información guardada, así como un entero de 32 bits para almacenar la cantidad de bytes escritos en el fichero, los bytes restantes serán la información a ocultar.

Implementación propuesta

void Esteganografia\_LSB::LSB\_BMP\_write( BMP \*bmp, string text )

{

RandomAccessFile \*file = bmp->getRAF();

string bitsLSB = toBinary("LSB");

string bitsLength = toBinary(text.size());

string bitsText = toBinary(text);

//representaion binaria

string str = bitsLSB + bitsLength + bitsText;

if(bmp->getBitsPerPixel() != 24)

{

InconsistentData ex = \*(new InconsistentData("El fichero no es de 24 bits por pixels"));

throw ex;

}

file->end();

int lengthfile = file->bytePosition();

if(lengthfile - 54 < str.size())

{

NoBytes ex = \*(new NoBytes("No hay espacio suficiente"));

throw ex;

}

file->seek(54);

for(int i = 0; i < str.size(); i++)

{

uint8\_t byt = file->readUnsignedInt8();

if(byt & 1 && str.at(i) == '0')

byt -= 1;

else if( !(byt & 1) && str.at(i) == '1')

byt += 1;

file->skipBytes(-1);

file->write(byt);

}

}

string Esteganografia\_LSB::LSB\_BMP\_read( BMP \*bmp )

{

RandomAccessFile \*file = bmp->getRAF();

file->seek(54);

uint8\_t \*lsb = new uint8\_t[24];

memset(lsb, 0, 25);

file->read(lsb, 24);

string bitsLSB = toBinary("LSB");

string isLSB;

for(int i = 0; i < 24; i++)

{

if(lsb[i] & 1)

isLSB.append("1");

else

isLSB.append("0");

}

if(isLSB != bitsLSB)

{

NoSignature ex = \*(new NoSignature("El fichero no contiene firma"));

throw ex;

}

uint8\_t \*size = new uint8\_t[32];

memset(size, 0, 33);

file->read(size, 32);

string bitsLength;

for(int i = 0; i < 32; i++)

{

if(size[i] & 1)

bitsLength.append("1");

else

bitsLength.append("0");

}

int length = toInt(bitsLength, 32);

uint8\_t \*data = new uint8\_t[length\*8];

memset(data, 0, length\*8+1);

file->read(data, length\*8);

string text;

for(int i = 0; i < length \* 8; i++)

{

if(data[i] & 1)

text.append("1");

else

text.append("0");

}

string finalText = toString(text);

return finalText;

}