INTRODUCCIÓN A LA ESTEGANOGRAFÍA DIGITAL Y CIFRADO HILL CON MATHEMATICA

Ángela Rojas Matas

1.- Introducción

El propósito de la esteganografía es ocultar mensajes en un objeto de cobertura inocuo o inocente como puede ser una imagen digital o una canción.

La información a ocultar puede ser:

- *un mensaje de texto*: por ejemplo, un usuario desea enviar a otro unas instrucciones secretas.
- *una imagen*: por ejemplo, un usuario puede enviar a otro una fotografía secreta con la localización concreta de un punto estratégico dentro de un mapa.
- *un fichero de audio*: por ejemplo, un usuario puede desear enviar a otro una grabación suya donde el emisor explica verbalmente unas instrucciones secretas.

En todos los casos, la información secreta a ocultar se convertirá en una secuencia de bits: ceros y unos. Esta secuencia de bits será almacenada en un fichero digital que nos va servir de cobertura.

La criptografía tiene como objetivo convertir mensajes en mensajes cifrados es decir, ininteligibles para cualquier usuario no autorizado que intercepte dicho mensaje.

Cuando dos usuarios intercambian información secreta codificada puede que un intruso tenga acceso a dicho mensaje cifrado y éste, consciente de que se pretende mantener secreta una información, puede intentar descifrar dicha información. Sin embargo, un intruso puede no sospechar nada si se encuentra con dos usuarios que se intercambian imágenes “inofensivas” o una “simple” canción.

Como vemos, la criptografía y la esteganografía tienen los mismos objetivos: compartir información secreta entre dos usuarios.

En muchas ocasiones, la esteganografía usa también la criptografía para reforzar la seguridad. La información secreta a ocultar se cifra en primer lugar usando un buen método criptográfico, a continuación se convierte en una secuencia de bits que serán ocultados mediante un determinado método esteganográfico en un fichero digital de cobertura.

En este trabajo nos vamos a ocupar de cómo ocultar información en una imagen digital.

Para ocultar el mensaje secreto compuesto por bits, es decir, ceros y unos, la imagen original será ligeramente modificada por el algoritmo de ocultación utilizado obteniendo una nueva imagen que se conoce con el nombre de **estego-imagen**.

Lo ideal de un algoritmo esteganográfico es que tenga una gran capacidad de ocultación, ya que de esta forma podremos ocultar una gran cantidad de bits en una sola imagen y, por otro lado, es deseable que la distorsión que va a provocar en la imagen original no se note mucho a simple vista. Estos dos aspectos generalmente dependen el uno del otro: cuántos más bits se oculten en la imagen mayor

distorsión se produce en ella. Tendremos que procurar un punto de equilibrio entre estas dos cuestiones.

Existen muchos métodos o algoritmos esteganográficos.

El más sencillo y utilizado de este tipo de algoritmos es el conocido como LSB(Least Significant Bit).

Una imagen digital en escala de grises (en blanco y negro) no es más que una matriz de números, donde cada número es un nivel de gris, que varía desde el cero, correspondiente al negro, hasta el 255 correspondiente al blanco.. Para almacenar un nivel de gris necesitamos 8 bits, siendo el último bit el menos significativo.

De manera que poner un cero ó un uno en este último bit altera el nivel de gris del píxel en una cantidad que es inapreciable por el ojo humano. Sin embargo, podemos utilizar ese bit para ocultar información.

Algunas de las herramientas de software de esteganografía más famosas existentes en el mercado como Steganos, S-tools y Hide4PGP se basan en el método LSB.

Por otro lado, es usual usar la criptografía en esteganografía. El mensaje a ocultar se cifra en primer lugar y es el mensaje cifrado el que se oculta después.

En este trabajo se estudiará un método de cifrado matricial, llamado cifrado Hill con el que cifraremos un mensaje que será ocultado en una imagen usando el método LSB.

2. Método LSB

Como hemos comentado anteriormente, el método LSB es el más usado. Una imagen digital en escala de grises es una matriz de números, donde cada número indica el nivel de gris de cada píxel, habitualmente entre 0 (negro) y 255 (blanco). Para almacenar un nivel de gris necesitamos 8 bits, siendo el último bit el menos significativo:

blanco	00000000
	00000001
⋮	00000010
	00000011
⋮	⋮
negro	11111111

Por ejemplo, una imagen 256×256 se compone de 65536 números o niveles de gris. Si usamos solamente el último bit, es decir, el bit menos significativo, el número máximo de bits que se pueden ocultar en dicha imagen sería:

$$256\times256 = 65536$$

Si usamos los dos últimos bits (los dos bits menos significativos) para ocultar información en esa misma imagen, el número máximo de bits que podremos ocultar será:

$$256 \times 256 \times 2 = 131072$$

Una imagen en color en formato RGB (Red Green Blue), se compone de tres capas ó planos de color: capa de rojo, capa de verde y capa de azul. Cada capa es una imagen del mismo tamaño que la original donde cada dato se interpreta como la cantidad de rojo, verde o azul presente en el píxel de la imagen en color. Los datos de cada capa de color ocupan generalmente un byte: desde el cero hasta el 255.

Una imagen en color va a tener el triple de capacidad para ocultar información que una imagen en escala de grises. Así, por ejemplo para una imagen en color de tamaño 256×256 tendríamos:

$$256 \times 256 \times 3 = 196608$$

Nos vamos a limitar a imágenes en blanco y negro por simplicidad.

Supongamos que deseamos almacenar un bit secreto en el bit menos significativo del nivel de gris de un píxel, suponiendo que el nivel de gris ocupa 8 bits:

- Una forma de hacerlo consistiría en escribir en binario el nivel de gris del píxel, obteniendo una lista de 8 bits. A continuación poner como último bit un 0 si el bit secreto es un cero y poner un 1 si el bit secreto es un uno. Después esta lista de bits se vuelve a poner en decimal y se asigna como nivel de gris al píxel original.

  Ejemplo:

$$\left.\begin{array}{l}\text{nivel de gris original} = 112 = \{0,1,1,1,0,0,0,0\} \\ \text{bit a ocultar} = 1\end{array}\right\} \Rightarrow \text{nivel de gris modificado} = \{0,1,1,1,0,0,0,1\} = 113$$

- Otra forma de hacerlo, equivalente a la anterior, sería la siguiente:
  - Si el nivel de gris es par y el bit a ocultar es 0, no hacer nada:
    nivel de gris modificado= nivel de gris original
  - Si el nivel de gris es par y el bit a ocultar es 1, sumar 1:
    nivel de gris modificado= nivel de gris original+1
  - Si el nivel de gris es impar y el bit a ocultar es 1, no hacer nada:
    nivel de gris modificado= nivel de gris original
  - Si el nivel de gris es impar y el bit a ocultar es 0, restar 1:
    nivel de gris modificado= nivel de gris original-1

A continuación vamos a ver cómo implementar el método LSB con Mathematica:

```
In:lamp= Import["C:/lamp.bmp"]
In:Show[lamp]
```

La primera línea de código lee una imagen llamada lamp.bmp que está en formato BMP. Con la orden `Show` la veremos en pantalla:

Imagen original de cobertura

A continuación creamos una matriz A con los niveles de gris de la imagen original y a continuación, usando el comando `Dimensions` vemos cuál es el tamaño de la matriz A.

```
In: A= lamp[[1, 1]] ;
In: Dimensions[A]
```

Out: {256, 256}

Supongamos que queremos ocultar la siguiente poesía de Antonio Machado:

In: mensaje="Mi infancia son recuerdos de un patio de Sevilla,
y un huerto claro donde madura el limonero;
mi juventud, veinte años en tierras de Castilla;
mi historia, algunos casos que recordar no quiero...".

(el mensaje es realmente mucho más largo, ver el fichero LSB.nb)

A continuación, la orden **`ToCharacterCode`** nos convierte cada letra en un número de acuerdo a su código ASCII: la a es el 97, la b es el 98, etc. Por ejemplo, si ejecutamos:

ToCharacterCode[“abc”], Mathematica devuelve {97, 98, 99}

También emplearemos la orden **`IntegerDigits[n,b,m]`** que nos devuelve una lista con el número n escrito en base b con m dígitos.

Por ejemplo: IntegerDigits[127,2,8], nos dará lugar a {0,1,1,1,1,1,1,1}

```
In:mensaje1=ToCharacterCode[mensaje];
len=Length[mensaje1];
mensaje2={};
For[i=1,i≤len,i++,
    AppendTo[mensaje2,IntegerDigits[mensaje1[[i]],2,8]];];
```

Ahora en mensaje2 tenemos una lista del tipo:

{{0, 1, 0, 1, 0, 0, 1, 0}, {0, 1, 0, 0, 0, 1, 0, 1}, ...}

Ya que la primera letra del mensaje era R, que en ASCII es 82, que en binario es {0,1,0,1,0,0,1,0}. La segunda letra del mensaje es E, que en ASCC es 69, que en binario es {0,1,0,0,0,1,0,1}, etc.

Nos interesa dejar una lista con todos los bits a ocultar seguidos. Quitamos las llaves internas con la orden Flatten

```
In: mensaje2=Flatten[mensaje2];
```

Ahora mensaje 2 es: mensaje2={0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, ...}

Resumiendo, hemos cogido el mensaje a ocultar y lo hemos convertido en una secuencia de bits.

A continuación vamos a ocultar un bit en cada píxel de la imagen, modificando el bit menos significativo. Emplearemos la siguiente técnica:

- Si el nivel de gris es par y el bit a ocultar es 0, no hacer nada:
  nivel de gris modificado= nivel de gris original
- Si el nivel de gris es par y el bit a ocultar es 1, sumar 1:
  nivel de gris modificado= nivel de gris original+1
- Si el nivel de gris es impar y el bit a ocultar es 1, no hacer nada:
  nivel de gris modificado= nivel de gris original
- Si el nivel de gris es impar y el bit a ocultar es 0, restar 1:
  nivel de gris modificado= nivel de gris original-1

Para ello, vamos a emplear la orden **`Mod[n, m]`** que nos devuelve el resto de la división entera de n entre m. Entonces **`Mod[n, 2]`** nos devolverá un cero si n es par y un 1 si n es impar.

```
In: An=A;
l=1;
len=Length[mensaje2];
For[i=1,i<=256&&l<=len,i++,
  For[j=1,j<=256&&l<=len,j++,
    bit=mensaje2[[l]];
    If[Mod[A[[i,j]],2]==0&&bit==1,An[[i,j]]++;];
    If[Mod[A[[i,j]],2]==1&&bit==0,An[[i,j]]--;];
    l++;
    ]]
```

La matriz An es la estego-imagen. ¿Se habrá alterado mucho la imagen original?.

Para ver la matriz An como imagen emplearemos la orden **`ListDensityPlot`** que interpreta cada dato de la matriz An como un nivel de gris:

```
In: ListDensityPlot[An,Mesh->False,Frame->False]
```

Obteniendo el siguiente resultado:

Estego-Imagen

Como vemos, no es apreciable a simple vista ninguna modificación.
El programa LSB.nb consigue la estego-imagen anterior.

En el caso de que queramos usar los dos bits menos significativos se puede proceder también de forma similar. La forma quizás más intuitiva consistiría en averiguar los 8 bits de cada nivel de gris, y sustituir los dos últimos bits por los dos bits secretos que toquen ocultar. Sin embargo, hay una forma más sencilla de hacerlo.

Los dos bits secretos a ocultar pueden ser: {00, 01, 10, 11} que se corresponden en el sistema de numeración decimal con los números {0, 1, 2, 3}, que son los posibles restos de la división entera de un número entre 4. Entonces, este número secreto *s* que varía entre 0 y 3, se almacenará de la siguiente forma:
Nivel de gris modificado= $q \times 4 + s$
siendo q el cociente de la división entera del nivel de gris original entre 4 y siendo *s* el número secreto a ocultar.
Por ejemplo, si el nivel de gris es 127 y los dos bits a ocultar fueran 10, entonces, el primer método requeriría escribir 127 en binario, y sustituir los dos últimos bits por 10:

```
In: IntegerDigits[127,2,8]
```

Out: {0, 1, 1, 1, 1, 1, 1, 1}
Por lo tanto, el nivel de gris modificado es {0, 1, 1, 1, 1, 1, 1, 0} en binario que se corresponde con

```
In:FromDigits[{0,1,1,1,1,1,1,0},2]
```

Out: 126
126 en decimal.
Esto es equivalente a hallar *q* el cociente de la división de 127 entre 4:

```
In:Quotient[127,4]
```

```
Out: 31
```

Calulando después el nivel de gris modificado como $31 \times 4 + 3 = 126$

Se deja como trabajo para el alumno hacer un programa en Mathematica que oculte un mensaje en una imagen usando los dos bits menos significativos

Alterar más de dos bits en una imagen generalmente produce graves distorsiones en la imagen original y ya no tiene mucho interés.
El método LSB tiene el inconveniente de que una simple compresión de la estego-imagen destruye la información secreta que ya no se podrá recuperar.

Los ficheros:

- **LSB-ocultar.nb** oculta un mensaje con el método LSB usando el bit menos significativo.
- **LSB-extraer.nb** es el complementario del anterior. Extra el mensaje de la estego-imagen.

4. Cifrado Hill o cifrado matricial

Lester Hill introdujo en 1929 su método de cifrado y descifrado de un mensaje en la revista *The American Mathematical Monthly*. La idea es bastante sencilla, como vamos a exponer a continuación. Debemos fijar previamente el alfabeto que vamos a emplear en la escritura de nuestro mensaje.

Supongamos que en el mensaje vamos a utilizar un alfabeto compuesto por 31 caracteres, que incluyen las 27 letras del abecedario así como el punto, la coma, el símbolo de interrogación y el espacio en blanco, pero se puede utilizar cualquier alfabeto de tamaño *m*. Asignamos a cada uno de los caracteres del alfabeto un número comprendido entre 0 y 30 como se indica en la siguiente tabla:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Q	R	S	T	U	V	W	X	Y	Z	.	,	?	
17	18	19	20	21	22	23	24	25	26	27	28	29	30

De acuerdo a la tabla anterior, un mensaje se transformará en una secuencia de números. Por ejemplo, si el mensaje fuera: “MATRICES Y CRIPTOGRAFIA”, se transformaría, de acuerdo a la tabla anterior en :
{12, 0, 20, 18, 8, 2, 4, 19, 30, 25, 30, 2, 18, 8, 16, 20, 15, 6, 18, 0, 5, 8, 0}

A continuación, agrupamos la secuencia de números en grupos de *n* elementos. Por ejemplo, en grupos de 2 elementos, añadiendo si fuera necesario un espacio en blanco al final. En nuestro caso hay que añadir un espacio en blanco al final, por tener nuestro mensaje un número impar de caracteres, 23 exactamente.

Elegimos una matriz cualquiera A de tamaño 2×2, por ejemplo: $A=\begin{pmatrix}9 & 4\\ 3 & 2\end{pmatrix}$

y vamos transformando cada par de números de la secuencia anterior en otro par diferente, mediante un producto matricial y trabajando con congruencias módulo 31. Así los dos primeros números {12, 0} se transforman en {15,5}:

$$A\begin{pmatrix}12\\ 0\end{pmatrix}=\begin{pmatrix}15\\ 5\end{pmatrix}\ (\text{Mod. } 31)$$

mirando en nuestra tabla, estos números corresponden a los caracteres OF. Repitiendo para el resto, obtendremos la siguiente secuencia:

{O, F, E, D, R, ,, S, S, ?, P, , I, I, H, Z, E, Z, H, W, O, A, . , ?}

El mensaje que enviamos a nuestro receptor será: "OFEDR ,SS?P BIIHZEZHWOA.?".

¿Cómo se hacen estas operaciones con Mathematica?

Los ficheros:

- Hill-mensaje-cifrar.nb coge la frase " MATRICES Y CRIPTOGRAFIA " y la cifra con el método Hill.
- Hill-mensaje-descifrar.nb es el complementario del anterior.

Para descifrar nuestro mensaje haremos justo el proceso inverso. Transformaremos el mensaje recibido en una secuencia de números de acuerdo a la tabla anterior, agrupamos de dos en dos y multiplicamos por la matriz A^{-1} (módulo 31) que sólo el receptor del mensaje conoce:

$$A^{-1}=\begin{pmatrix}21 & 20\\ 15 & 17\end{pmatrix}(\text{mod. } 31)$$

Ya que:

$$A=\begin{pmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{pmatrix}\Rightarrow\ A^{-1}=\frac{1}{|A|}\begin{pmatrix}a_{22} & -a_{12}\\ -a_{21} & a_{11}\end{pmatrix}=6^{-1}\begin{pmatrix}2 & -4\\ -3 & 9\end{pmatrix}=26\begin{pmatrix}2 & -4\\ -3 & 9\end{pmatrix}=\begin{pmatrix}21 & 20\\ 15 & 17\end{pmatrix}\ (\text{mod. } 31)$$

Sólo hay que tener cuidado, por lo tanto, en escoger una matriz inversible. Para ello hay que asegurarse que el determinante de la matriz A elegida sea un número que admita inverso cuando se trabaja con

congruencias módulo m. En nuestro caso, con $m = 31$ tenemos pocos problemas porque todos los números (salvo el cero) tienen inverso cuando se trabajan con congruencias módulo 31.

En general, si se trabajan con congruencias módulo m, un número d admitirá inverso módulo m si y sólo si $m\,c\,d\,(m, d) = 1$.

Si por ejemplo el módulo fuera $m = 256$, entonces la matriz anterior no se podría invertir en $\mathbb{Z}_{256}$ ya que el determinante de A vale 6 y el mcd(256, 6) no es 1, por lo que 6 no admite inverso. En este caso, si usamos esa matriz para cifrar el mensaje luego no podremos descifrarlo porque para poder hacerlo necesitamos la matriz inversa.

Para $m = 256$, podríamos utilizar por ejemplo la matriz $\begin{pmatrix} 11 & 15 \\ 23 & 40 \end{pmatrix}$ que sí que admite inversa en $\mathbb{Z}_{256}$ ya que su determinante es 95 que es primo relativo con 256.

Como usamos un alfabeto con 31 caracteres, cada carácter a codificar es un número entre 0 y 30. Con 5 bits tenemos suficiente para almacenar dicho carácter. Por lo tanto, para almacenar un solo carácter usaremos el bit menos significativo de cinco píxeles.

Si usamos la imagen de la lámpara de tamaño 256x256, con esta estrategia podemos ocultar un máximo de 65536/5 .

En la figura a se presenta una imagen y a su derecha la imagen alterada ocultando en el bit menos significativo un mensaje compuesto por 1646 caracteres, cifrado previamente con el método de Hill. Como puede observarse en la figura 1, no se detectan a simple vista diferencia entre ambas imágenes.

Imagen original

Imagen con mensaje oculto

El fichero LSB-Hill.nb cifra un mensaje y lo oculta en una imagen con el método LSB.