

# Primi passi in steganografia

Andrea Centomo

Liceo “F. Corradini” di Thiene (VI)

EMAIL: `andrea.centomo@istruzione.it`

5 febbraio 2007

---

In questo breve articolo si discute un esempio didattico di steganografia per immagini bitmap. Lo scopo principale che ci siamo prefissi non è quello di proporre un metodo steganografico efficiente quanto di evidenziare alcune idee e concetti matematici che intervengono in questo ambito.

---

## Introduzione

Il termine steganografia è composto dalle parole greche *στέγω* (nascondere) e *γραφή* (scrittura) e individua una tecnica molto antica di comunicazione segreta dei messaggi tramite il loro occultamento.

Alcuni esempi di steganografia sono riportati dallo storico greco Erodoto nella sua opera intitolata *Storie*<sup>1</sup>. In essa si narra ad esempio che Demarato, per informare gli Spartani dell'arrivo delle truppe di Serse, avesse inciso l'informazione su una tavoletta di legno coprendola successivamente di cera. La tavoletta fu quindi spedita a Sparta dove il messaggio segreto venne letto dopo che la cera fu rimossa. Diverse altre tecniche più o meno ingegnose di steganografia sono state scoperte nel corso dei secoli fino a giungere ai giorni nostri dove sono stati escogitati diversi interessanti metodi di occultamento di messaggi sfruttando la natura digitale di file multimediali di vario genere (immagini, audio e video).

In queste pagine esamineremo, attraverso un esempio, i principi generali alla base di alcune tecniche di steganografia delle immagini digitali *bitmap*.


## 1. Struttura dell'immagine bitmap

La prima questione che deve essere approfondita per poter affrontare la steganografia delle immagini bitmap consiste nel sapere come è fatta un'immagine digitale con formato *bitmap*.

Per poter affrontare la questione è necessario avere la nozione matematica di *sistema numerico esadecimale*. Il sistema numerico esadecimale è un sistema posizionale in base 16, che utilizza 16 simboli invece dei 10 del sistema numerico decimale tradizionale. Per l'esadecimale si usano in genere simboli da 0 a 9 e poi si prosegue con le lettere dell'alfabeto da *A* a *F*. La trasformazione di un numero da esadecimale a decimale avviene sfruttando la natura posizionale del sistema. Così, ad esempio, avremo

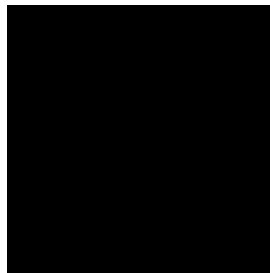
$$4AF = 4 \cdot 16^2 + A \cdot 16^1 + F \cdot 16^0 = 4 \cdot 16^2 + 10 \cdot 16 + 15 \cdot 1 = 1199.$$

---

 Questo documento è stato realizzato usando GNU T<sub>E</sub>X<sub>M</sub>A<sub>C</sub>S (vedi <http://www.texmacs.org>).

1. Si veda ad esempio Erodoto, *Storie*, Libri VII-XI, Volume 4, Edizioni Rizzoli, 2002.

A questo punto possiamo avventurarci nello studio completo della nostra prima immagine bitmap in toni di grigio `quadratonero.bmp` che rappresenta un semplice quadrato nero.



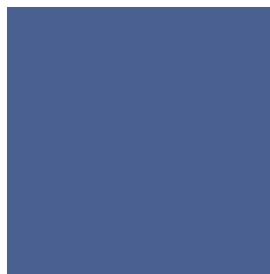
**Figura 1.** Quadrato nero  $100 \times 100$  pixel

Se apriamo il file `quadratonero.bmp` con un visualizzatore di immagini<sup>2</sup> bitmap quello che vediamo è il quadrato nero di Figura 1. Se invece apriamo lo stesso file con un visualizzatore di file esadecimali<sup>3</sup> noteremo che esso in realtà contiene 11078 numeri esadecimali a due cifre. Strano! La questione diventa chiara solo se si comprende il rapporto che intercorre tra ciascuno degli 11078 numeri esadecimali a due cifre e il quadrato nero che vediamo con il visualizzatore di immagini. Se si consultano le specifiche del formato *bitmap* si scopre che:

- a) i primi 1078 numeri esadecimali definiscono l'*header* dell'immagine ossia un certo numero di caratteristiche generiche su di essa (grandezza, offset, risoluzione e via di seguito);
- b) i rimanenti 10000 rappresentano l'immagine vera e propria che è formata da 10000 quadratini - tecnicamente detti *pixel* - di cui il numero esadecimale a due cifre specifica la tonalità di grigio in una scala di 256 livelli che va dal nero (00) al bianco (FF)<sup>4</sup>.

In altri termini quello che sullo schermo vediamo come un quadrato nero è formato da una moltitudine di pixel di colore nero. Da un punto di vista didattico è istruttivo osservare che il visualizzatore di immagini *grosso modo* non fa altro che convertire le informazioni contenute in ciascun numero esadecimale in punti colorati visibili sullo schermo.

Abbandoniamo adesso il mondo malinconico delle scale di grigio per dirigerci verso il mondo dei colori e analizziamo la seconda immagine `quadratocelste.bmp` che rappresenta il quadrato celeste di figura.



**Figura 2.** Quadrato celeste  $100 \times 100$  pixel

2. Nell'ambiente Gnu/Linux da noi usato il visualizzatore di immagini è gThumb.

3. Nell'ambiente Gnu/Linux da noi usato il visualizzatore di file esadecimali è Ghex.

4. Osserviamo che il numero esadecimale 00 corrisponde al numero decimale 0 mentre il numero esadecimale FF corrisponde proprio al numero decimale 255.

Se apriamo il file `quadratoceleste.bmp` con il visualizzatore di file esadecimale noteremo che esso questa volta contiene 30053 numeri esadecimale a due cifre. Se guardiamo attentamente i numeri esadecimale contenuti nel file noteremo che gli ultimi 30000 sono 10000 repliche della seguente terna:

91 60 4A

Il numero 4A indica la tonalità di rosso (R), il numero 60 quella di verde (G) e il 91 quella di blu (B) di ciascun pixel dell'immagine.

## 2. Steganografia LSB

Il metodo steganografico che ora proponiamo a mero scopo didattico ha come idea fondamentale quella di modificare un'immagine iniziale in modo che la presenza del messaggio nascosto non sia percepibile ad occhio nudo. In altre parole immagine di partenza e immagine con testo nascosto devono essere visivamente indistinguibili!

*Per ottenere questo effetto diverse tecniche consistono nel nascondere il messaggio segreto nei bit meno significativi (Less Significant Bit) dell'immagine originale.*

Vediamo per cominciare un esempio, che chiarisca che cosa si intende per bit meno significativo. Consideriamo la tonalità di grigio rappresentata dal numero esadecimale 81. Il numero esadecimale 81 corrisponde al numero decimale 129 che, usando le potenze di 2, possiamo scrivere come

$$94 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

e che quindi corrisponde al numero binario di otto cifre 10000001. Nel linguaggio informatico si dice che il numero 10000001 occupa 8 bit e che il bit meno significativo è rappresentato dal suo ultimo valore che è 1. Se modifichiamo l'ultimo bit (l'unica scelta possibile consiste nel sostituire 1 con 0) si ottiene il numero 10000000 che corrisponde alla tonalità di grigio 128 (80 in esadecimale). Come si può notare dalla Figura 2 due quadrati con tonalità di grigio 128 e 129 sono indistinguibili<sup>5</sup>.



**Figura 3.** Quadrati a tonalità di grigio 128 e 129

Nel momento in cui si decide di occultare un messaggio modificando i bit meno significativi di un'immagine è comodo disporre del messaggio nascosto scritto in forma binaria. Per evitare di ricorrere a ulteriori concetti, come ad esempio quello di codifica ASCII, e per rendere leggermente più difficili eventuali tentativi di decrittazione dei nostri messaggi segreti ci costruiamo un alfabeto a 5 bit personalizzato.

<sup>5</sup>. L'occhio umano è in grado di distinguere circa 16 livelli di grigio.

a	11111	i	10111	q	01111	y	00111
b	11110	j	10110	r	01110	z	00110
c	11101	k	10101	s	01101	+	00101
d	11100	l	10100	t	01100	-	00100
e	11011	m	10011	u	01011	*	00011
f	11010	n	10010	v	01010	:	00010
g	11001	o	10001	w	01001	^	00001
h	11000	p	10000	x	01000	□	00000

**Tabella 1.** Alfabeto a 5 bit

dove con il simbolo □ si intende lo spazio vuoto.

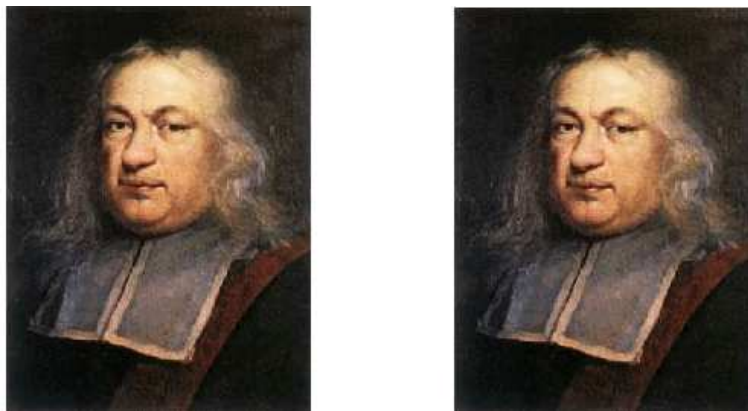
Con questo alfabeto, se il messaggio segreto fosse *ore venti*, esso sarebbe descritto dalla sequenza dei seguenti 9 blocchi di 5 numeri:

10001 – 01110 – 11011 – 00000 – 01010 – 11011 – 10010 – 01100 – 10111

Per nascondere la sequenza all'interno di un'immagine utilizziamo le seguenti **regole di cifratura**:

- si scorre l'immagine verso sinistra e evitando di modificare l'*header*
- al numero 1 si associa la modifica del bit meno significativo (se  $0 \rightarrow 1$ , se  $1 \rightarrow 0$ )
- al numero 0 non corrisponde alcuna modifica.

Vediamo allora di operare la manipolazione dell'immagine a sinistra di Figura 3 che consideriamo come immagine di partenza e che deve essere posseduta sia da chi spedisce il messaggio (Alice) che da chi lo riceve (Bob). Alice utilizza l'immagine per manipolarla nascondendo il messaggio segreto *ore venti* mentre Bob la utilizzerà per la decifrazione.



**Figura 4.** Immagine di Fermat originale (a sinistra) e immagine con testo occultato

Se apriamo l'immagine originale con un editor esadecimale noteremo che gli ultimi<sup>6</sup> 45 numeri esadecimali a due cifre (raggruppati per convenienza a gruppi di 5) sono i seguenti:

6. La scelta di modificare gli ultimi 45 è casuale, si potrebbe anche scegliere di modificare 45 numeri scelti casualmente procedendo tuttavia sempre dal fondo dell'immagine muovendosi verso sinistra.

```

39 39 4A 31 42 - 4A 31 39 4A 31 - 39 4A 31 39 39
31 39 4A 29 31 - 39 29 31 39 31 - 39 39 31 39 39
31 39 39 31 39 - 39 31 39 39 31 - 39 39 42 4A 52

```

Se li modifichiamo usando le regole di cifratura che ci siamo dati otterremo:

```

38 39 4A 31 43 - 4A 30 38 4B 31 - 38 4B 31 38 38
31 39 4A 29 31 - 39 28 31 38 31 - 38 38 31 38 38
30 39 39 30 39 - 39 30 38 39 31 - 38 39 43 4B 53

```

Salvate le modifiche otterremo l'immagine di Figura 3 a destra, che sembra - almeno a vista - esattamente uguale a quella originale. Se però Bob confronta immagine originale e modificata con l'editor esadecimale o con altri strumenti<sup>7</sup> potrà vedere quali numeri sono stati modificati e quindi potrà ricostruire il messaggio.

### 3. Metrica di Hamming

Per concludere la nostra trattazione introduciamo il concetto di *metrica di Hamming* che ci permette di quantificare la differenza tra immagine di partenza  $x$  e immagine modificata  $y$ . Posto che sia  $x$  che  $y$  hanno lo stesso numero di bit diremo che loro distanza secondo Hamming  $d_{\mathcal{H}}(x, y)$  è rappresentata dal *numero di bit corrispondenti in cui le due immagini differiscono*.

Non è difficile verificare che la metrica di Hamming soddisfa le usuali proprietà di una distanza:

- a)  $d_{\mathcal{H}}(x, y) \geq 0$ : la distanza è non negativa;
- b)  $d_{\mathcal{H}}(x, y) = 0$  se e solo se  $x = y$ : ossia la distanza tra due immagini è nulla se e solo se esse sono uguali;
- c)  $d_{\mathcal{H}}(x, y) = d_{\mathcal{H}}(y, x)$ : scambiando l'ordine delle immagini la distanza non cambia;
- d)  $d_{\mathcal{H}}(x, y) \leq d_{\mathcal{H}}(x, z) + d_{\mathcal{H}}(z, y)$ : disuguaglianza triangolare.

Con riferimento all'esempio di occultamento del messaggio *ore venti* visto sopra, dal momento che la sequenza di 45 bit che traduce il messaggio in sequenza di 0 e 1 contiene 23 volte 1 e per le regole di cifratura stabilite, avremo

$$d_{\mathcal{H}}(x, y) = 23$$

dove  $x$  è l'immagine di Fermat originale e  $y$  quella modificata.

### Sitografia

1. [http://it.wikipedia.org/wiki/Windows\\_bitmap](http://it.wikipedia.org/wiki/Windows_bitmap) voce di Wikipedia relativa al formato per immagini bitmap.

<sup>7</sup> Nell'ambiente Gnu/Linux il confronto si può effettuare usando il comando di shell `cmp -l` che restituisce in uscita i numeri esadecimali in cui le due immagini differiscono.

2. <http://www.filosofico.net/erodotostorie7.htm> sito contenente una traduzione del Libro VII delle *Storie* di Erodoto.
3. T. Morkel, J.H.P. Eloff, M.S. Olivier, An overview of image steganography, Information and Computer Security Architecture (ICSA), Research Group Department of Computer Science University of Pretoria, 0002, Pretoria, South Africa. Disponibile online.
4. [http://it.wikipedia.org/wiki/Distanza\\_di\\_Hamming](http://it.wikipedia.org/wiki/Distanza_di_Hamming) voce di Wikipedia relativa alla metrica di Hamming.