

# INVISIBLE INK

Breve documentazione sul funzionamento di Invisible Ink e del suo algoritmo

Invisible Ink è un software che permette di crittografare e steganografare dati all'interno di immagini in formato bmp.

*“Il termine **steganografia** è composto dalle **parole greche** *stèganos* (nascosto) e *gràfein* (**scrittura**) e individua una **tecnica** risalente all'antica **Grecia** che si prefigge di nascondere la **comunicazione** tra due interlocutori, e fu teorizzata dall'abate **Tritemio** attorno al **1500** nell'omonimo libro. La "Steganographia" di Tritemio si proponeva di poter inviare messaggi tramite l'uso di linguaggi magici, sistemi di apprendimento accelerato e senza l'utilizzo di simboli o messaggeri. L'opera iniziò a circolare in corrispondenze private e suscitò reazioni tanto allarmate che l'autore decise di non darla alle stampe e ne distrusse addirittura larghe parti, ritenendo che non avrebbero mai dovuto vedere la luce. Continuò comunque a circolare in forma di minuta e fu pubblicata postuma nel **1606**.”*

Fonte: Wikipedia.it

## L'algoritmo Invisible Ink

Come modificare i colori delle immagini, in modo impercettibile all'occhio umano ma senza perdere dati nell'informazione da immagazzinare? Niente di più facile nell'era dell'informatica.

Spesso ci si dimentica che tutte le cose complesse sono composte da sottosistemi semplici ... per farla un po' più breve, complessi calcoli matematici, finestre di windows, filmati, giochi, immagini (si proprio loro), devono, prima o poi, al livello più basso, passare per circuiti elettronici, che per semplicità, nella loro costruzione, sono pensati per gestire solo due stati .... E così, grazie al matematico G.Boole, nacque il sistema binario .... quindi .... se tutto è 1 o 0 .....

Esaminando la struttura del codice di una bmp, si può vedere che è composta da 2 parti, un'intestazione e la parte riservata ai dati dei colori. (Non prenderemo in considerazione le bmp a 8, 16 e 256 colori che utilizzano il concetto di "palette" e che non si prestano bene al nostro utilizzo in quanto hanno un numero limitato e predefinito di colori utilizzabili .... perchè a noi piace dipingere liberamente ;) ).

Struttura generale bmp



Nell'intestazione abbiamo dati che ci indicano quanto è alta , larga, come si chiama, quanto pesa, dove va e cosa pensa la nostra immagine ... e noi la salteremo a piè pari.

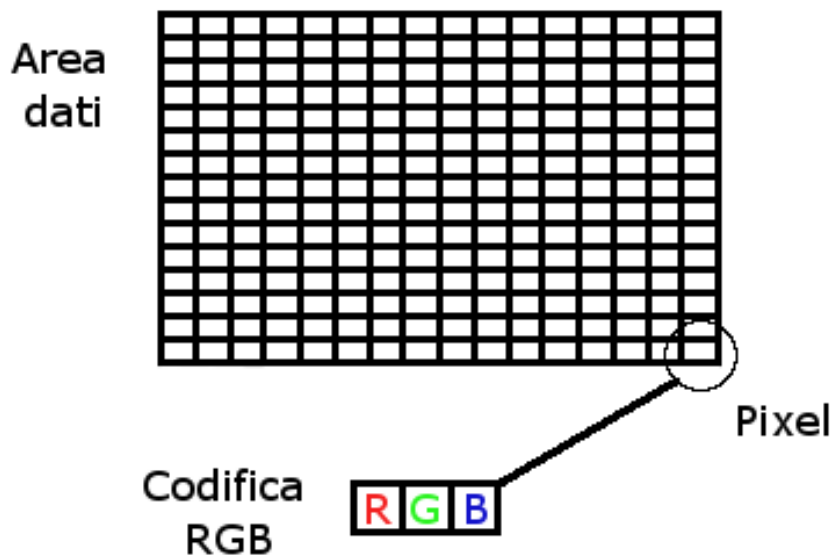
Passiamo direttamente ad esaminare la struttura dati dei colori...

Abbiamo parlato di sistemi complessi composti da sottosistemi semplici.....trattiamo quindi la nostra bitmap come un sistema complesso e iniziamo a dividerlo in sottosistemi sempre più semplici.

Il primo passaggio è scomporre l'immagine in pixel.....ed ecco il nostro primo sottosistema.....ma è abbastanza semplice? No.

Dobbiamo arrivare ai nostri 0 e 1 ... continuiamo quindi la scomposizione....cos'è un pixel? La parte più piccola di un immagine che ne rappresenta un colore ... e per gli standard bitmap il colore verrà rappresentato in RGB, i canali Rosso Verde e Blu.....

Scomposizione Immagine -> Pixel ->



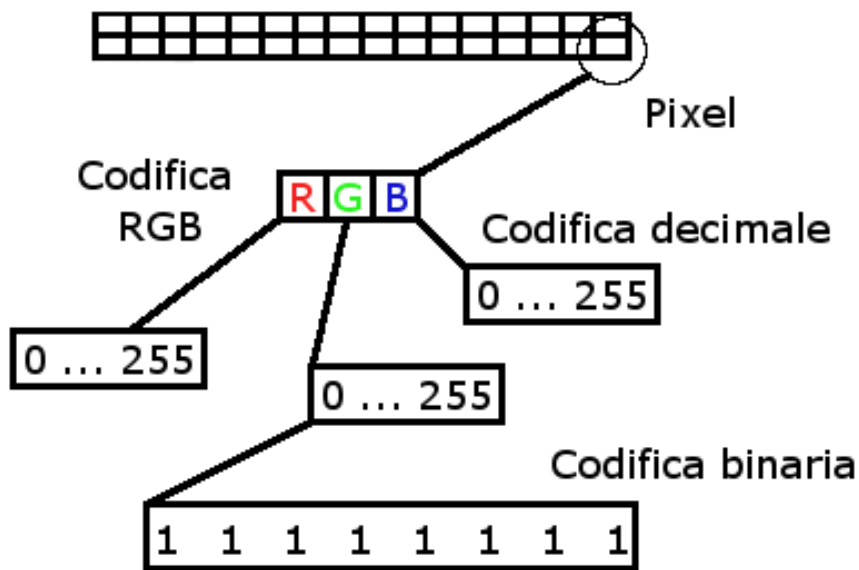
RGB

Ok, ora abbiamo una griglia di colori scomposti in RGB ... ma ancora non basta ... proseguiamo sempre nello stesso modo.

I canali R, G e B vengono indicati con valori che vanno da 0 a 255 (come si può anche vedere da Paint nella schermata di selezione dei colori personalizzati).

Ed è un numero veramente interessante, perché sono 256 valori, e ....., in binario, 256 valori sono rappresentabili con 1 byte, ovvero 8 bit .... Ora ci siamo ...

Scomposizione immagine pixel -> RGB -> decimale -> binario



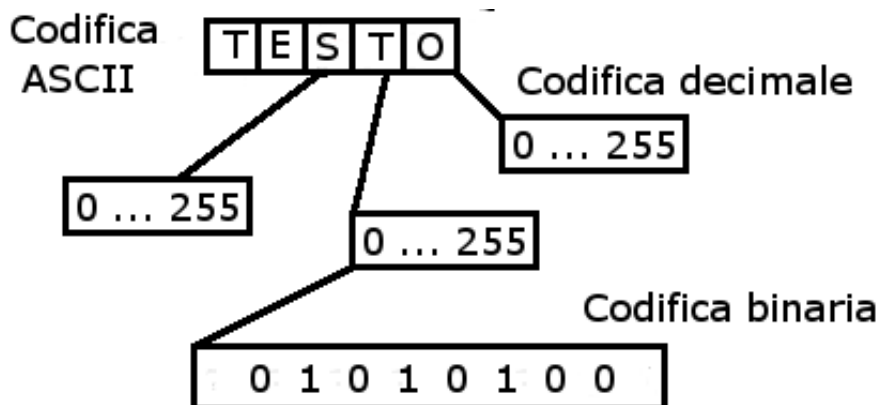
Torniamo per un secondo sul nostro obiettivo.

Dobbiamo nascondere del testo nell'immagine modificandone i colori , abbiamo quindi portato quest'ultimi al loro stato più semplice ottenendo da un immagine una griglia di valori binari che a tre a tre compongono i colori dei pixel (che tutti insieme compongono l'immagine).

Facciamo ora lo stesso con il nostro testo.

Questo diventa ora il nostro sistema complesso da scomporre in sottosistemi fino ad arrivare al sistema binario.

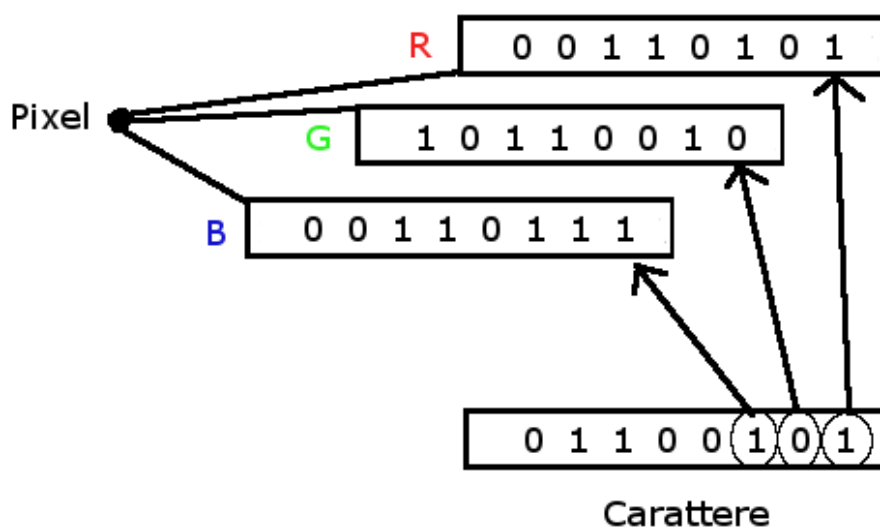
Il nostro testo è quindi scomponibile in caratteri, e ogni carattere (considereremo qui solo il sistema di codifica ASCII, ignorando l'UNICODE, anche se il ragionamento non cambierebbe di una virgola.) è rappresentato da un numero decimale che, tramite una tabella (ASCII), prende il legame con il simbolo corrispondente .... ed indovinate un po'... ogni carattere è rappresentato da un numero che va da 0 a 255 (in realtà nemmeno usati tutti, ma non ci dilunghiamo sui dettagli :D), che sono 256 valori, che sono un 1byte.....e si ... sono 8 bit.



Adesso che abbiamo sia il testo che l'immagine sotto la stessa codifica, bisogna sostituire i bit del colore con quelli dei caratteri.

Per rendere la modifica dei colori impercettibile all'occhio umano, gli unici bit che verranno modificati saranno quelli "meno significativi", ovvero il bit più a destra di uno canale del colore di ogni pixel (finché c'è testo da nascondere). Questo causerà, nel 50% dei casi una modifica di un solo valore (es. da 133 a 134) per singolo canale di colore (una modifica non visibile quindi), e nell'altro 50% delle probabilità nessuna modifica (qualora si sostituisse un bit con uno dello stesso valore).

Una volta sostituiti i bit del colore con quelli del testo ci basterà ricomporre i canali RGB, quindi il pixel e reimpacchettare tutto ricomponendo l'immagine, che risulterà completamente identica all'originale, sia nelle dimensioni su disco che ad occhio, ma che conterrà l'informazione del testo iniettato.



Una piccola modifica all'algoritmo

Mantenendo questa bassissima frequenza di inserimento dati nei colori ci troviamo a poter inserire tranquillamente 1 byte di informazione ogni 9 byte di immagine, quindi, su una bmp di 6mb ci staranno senza problemi quasi 600.000 caratteri di testo.

Dato che difficilmente si inseriranno tanti dati, ho deciso di inserire una piccola variante all'algoritmo per rendere il tutto un po' più elegante, ovvero il calcolo automatico della frequenza di inserimento dei byte di testo. Questo calcolo permette di "spalmare" i byte lungo tutta la griglia RGB dell'immagine, in modo da non avere tutti i byte modificati all'inizio ma equamente suddivisi su tutta la superficie.

A questo punto sorge un ultimo problema.

Se, a seconda della lunghezza del testo e della dimensione dell'immagine, la posizione dei byte modificati varia, diventa impossibile, in fase di rilettura, ricavare da che byte rileggere le informazioni.

Per ovviare a questo problema i primi 4 pixel dell'immagine sono riservati dall'algoritmo per l'inserimento di un'informazione di "testata", ovvero verranno usati per contenere l'informazione della lunghezza del testo che verrà steganografato nell'immagine, in modo tale da poter rieffettuare il calcolo inverso sulla frequenza di inserimento e quindi ricavare quali byte andare a leggere...

### **L'algoritmo al contrario**

E in rilettura?

Beh...ormai dovrebbe essere tutto abbastanza chiaro...

Prima di tutto vengono letti i primi 4 pixel (la testata) della griglia di colori dell'immagine.

Vengono scomposti nuovamente in binario, vengono prese le posizioni meno significative (il bit più a destra di ogni canale di colore), ricomposti e convertiti in un numero decimale, ottenendo così la "quantità" di testo steganografata nell'immagine.

Da questa viene ricalcolata la frequenza di inserimento (ogni quanti pixel ne è stato modificato uno) che è stata utilizzata durante la fase di inserimento, quindi recuperati ad uno ad uno i pixel incriminati sulla quale verrà effettuata l'operazione inversa a quella di inserimento, riottenendo quindi, dai colori dei pixel, l'informazione originaria.

### **Perché crittografare i dati allora?**

Perché se l'inserimento di testo è così irrilevabile è stata inserita un'opzione per la crittografia?

E' vero che è impossibile stabilire se nell'immagine ci sia veramente un testo steganografato o meno, in quanto in questa continuano a rimanere solo le informazioni dei colori e a seconda di come si interpretano i bit possono essere colori, testo, musica e quant'altro (data is what you define it to be. Randall Hyde), ma è anche vero che basterebbe effettuare un tentativo di rilettura con un'altra copia dello stesso programma (che quindi interpreta i bit nello stesso modo) per poter vedere se si ottiene un testo leggibile o no.

Ed è per questo che è meglio effettuare una crittatura del testo prima dell'inserimento, in modo tale che non sia possibile sapere se si tratti di un testo crittato o di una sequenza casuale di bit.

L'algoritmo scelto per la crittatura è un algoritmo simmetrico, che ha quindi bisogno di una chiave già conosciuta da entrambe le parti (Rijndael a 256byte per l'esattezza), e questo perché non ripongo abbastanza fiducia negli algoritmi asimmetrici, per quanto risolvano ottimamente il problema dello scambio delle chiavi.