



Maresciallo Capo Raffaele Olivieri

**RAGGRUPPAMENTO CARABINIERI INVESTIGAZIONI SCIENTIFICHE
REPARTO TECNOLOGIE INFORMATICHE
SEZIONE TELEMATICA**

STEGANOGRAFIA: DESCRIZIONE ED IMPLEMENTAZIONE DI UN ALGORITMO

Roma, 24 ottobre 2007

INDICE

1. PREMESSA.....	3
2. CENNI STORICI.....	3
3. CLASSIFICAZIONE	4
4. STEGANOGRAFIA INIETTIVA SOSTITUTIVA.....	7
5. IMPLEMENTAZIONE DI UN ALGORITMO STEGANOGRAFICO.....	9
A. STRUTTURA GENERALE DI UN FILE.....	9
B. CONOSCENZA APPROFONDATA DEL FILE CONTENITORE	9
C. STRUTTURA DEL FORMATO BMP	10
a. <i>BITMAPFILEHEADER:</i>	10
b. <i>BITMAPINFOHEADER:</i>	11
c. <i>RGBQUAD aColors[]:</i>	11
d. <i>BYTE aBitmapBits[]:</i>	12
D. SPECIFICHE DI REALIZZAZIONE DELL'ALGORITMO	12
E. ALGORITMO STEGANOGRAFICO - HEADER CON LE PRINCIPALI DEFINIZIONI.....	18
F. ALGORITMO STEGANOGRAFICO - FUNZIONI DI GESTIONE LETTURA E SCRITTURA BMP	19
G. FUNZIONE STEGANOGRAFICA	21
H. FUNZIONE STEGANOGRAFIA INVERSA	22
6. TECNICHE DI STEGANALISI	14
7. CONCLUSIONI E CAMPI DI APPLICAZIONE.....	17

1. PREMESSA

Il termine *steganografia*, di origine greca, deriva etimologicamente dall'unione dei vocaboli *στéγανος* (*stèganos* = nascosto) e *γράφειν* (*gràfein* = scrittura). Con questo termine ci si riferisce ad una tecnica che consente di nascondere messaggi, che devono essere intelligibili al solo destinatario, inserendoli all'interno di un contesto del tutto estraneo, che funge da mero contenitore.

La steganografia si differenzia dalla crittografia in quanto si prefigge scopi differenti: infatti, mentre la crittografia ha come obiettivo quello di rendere indecifrabile un messaggio a chi non è autorizzato a conoscerne il contenuto, la steganografia ha, oltre allo scopo appena citato, anche quello di nascondere completamente, a qualsiasi osservatore, il fatto che in un dato momento stia avvenendo una comunicazione riservata.

La steganografia basa la propria sicurezza appunto sulla segretezza del metodo e non solo su una chiave, quindi necessitando di trasmissioni sicure ci si avvale della crittografia sia per rafforzare la propria capacità di occultamento rendendo più difficile l'individuazione del messaggio, sia per complicarne la comprensione in caso sia stata svelata l'esistenza del messaggio.

2. CENNI STORICI

I primi "studi" di steganografia risalgono già agli antichi egizi, ove si narra che i faraoni spedissero, in tempi di guerra, i messaggi segreti alle proprie truppe facendoli incidere sulla testa di uno schiavo, preventivamente rasata a zero. Avvenuta la ricrescita dei capelli, si inviava lo schiavo a destinazione. Chiunque lo avesse osservato, al suo passaggio, non avrebbe mai immaginato che egli fosse latore di messaggi d'importanza vitale. In questo modo egli era in grado di passare inosservato qualsiasi controllo e giungere felicemente a destinazione, dove i generali, ritagliando di nuovo i capelli allo schiavo, erano in grado di ricevere gli ordini impartiti dal proprio faraone.

Analogamente si narra che il filosofo Aristotele comunicasse con il nipote Callistene, storico al seguito di Alessandro Magno durante la sua spedizione in Asia, attraverso lettere aventi argomentazioni futili e piuttosto generiche, ma che applicata una griglia su tali pagine, queste lettere rivelassero messaggi di natura ben differente. Tali griglie, note oggi come griglie di Cardano, consistono in fogli su cui vengono ricavati dei fori ad intervalli irregolari.

Nel corso dei secoli, queste tecniche sono state perfezionate con mezzi svariati, ricorrendo all'utilizzo dei cosiddetti inchiostri simpatici, sostanze di uso comune che non lasciano tracce visibili e che rivelano la propria esistenza solo in determinate condizioni. Il più noto di tutti è il succo di limone che, qualora utilizzato per scrivere un messaggio su un foglio di carta, lo rileva solo se la pagina viene posta in prossimità di una fonte di calore.

Più recentemente si è ricorso a metodi più sofisticati, che prevedono l'uso dei cosiddetti "micropunti fotografici" nascosti all'interno di una immagine. In tali piccolissimi punti, impercettibili all'occhio umano, si possono nascondere messaggi segreti anche molto elaborati.

3. CLASSIFICAZIONE

Esistono diverse modalità di classificazione delle tecniche steganografiche.

Esse possono essere raggruppate in base del rapporto che lega il messaggio al contenitore; in tal caso si parla di:

- **steganografia iniettiva**, la tecnica più utilizzata, che consente di inserire (iniettare) il messaggio all'interno di un contenitore già esistente e da esso indipendente, che viene appositamente modificato in modo da apparire uguale a quello originale (privo di messaggio). In tal senso, anche lo schiavo utilizzato dagli antichi egizi può essere considerato un esempio di tecnica iniettiva;
- **steganografia generativa**, in cui il contenitore viene generato a seguito del messaggio, ovvero è il messaggio stesso a pilotare la creazione del contenitore. In questo caso, un esempio è la tecnica utilizzata da Aristotele, in cui dapprima veniva scritto il messaggio

segreto, ponendo la griglia su un foglio bianco, successivamente veniva rimossa la griglia e scritto il “contenitore” partendo dalle lettere distribuite su tutto il foglio.

Una ulteriore catalogazione della steganografia può essere fatta sulla base dell’approccio tecnico:

- **steganografia sostitutiva:** tale tecnica, tra le più diffuse, si basa sulla ricerca di “rumori” all’interno di un contenitore (linee telefoniche, trasmissioni radio, immagini, ecc.) e, dopo averne osservato la struttura, sostituzione diretta di questo rumore con un altro segnale contenente il messaggio segreto, avente le stesse caratteristiche statistiche del rumore stesso, in modo da poter essere trasmesso senza destare sospetti;
- **steganografia selettiva:** ha valore puramente teorico e apparentemente, non viene utilizzata in pratica ma solo per studi. L’idea su cui si basa è quella di procedere per tentativi, ripetendo una stessa misura fintanto che il risultato non soddisfi una certa condizione. La criticità di questa tecnica, che la rende poco utilizzabile in pratica è la sua dispendiosità in termini di contenitori rispetto alla scarsa quantità di informazione utile trasmissibile in maniera nascosta. Ad esempio, se si considera una funzione di parità sul numero di bit che costituisce un contenitore, quest’ultimo potrebbe comunicare il valore 0 o 1 a seconda della sua parità. Pertanto, per memorizzare una sequenza di bit si dovrebbero inviare un insieme di contenitori le cui rispettive applicazioni della funzione di parità restituissero la sequenza del messaggio da nascondere. Un altro esempio consiste nel generare dei contenitori mediante creazione di immagini con uno scanner cercando iterativamente di verificare la condizione richiesta. L’immagine così ottenuta conterrebbe effettivamente un’informazione segreta, ma, trattandosi di un’immagine “naturale” (senza valori aggiunti, rimanipolazioni successive o inserimento di rumori), sarebbe immune da eventuali indagini statistiche anti-steganografiche.
- **steganografia costruttiva:** essa tenta di sostituire il rumore presente nel contenitore utilizzato con l’informazione segreta opportunamente modificata in modo da imitare le

caratteristiche statistiche del rumore originale. Secondo quest'idea, un buon sistema steganografico dovrebbe basarsi su un modello del rumore e adattare i parametri dei suoi algoritmi di codifica in modo tale che il falso rumore contenente il messaggio segreto sia il più possibile conforme al modello. Questo approccio è senza dubbio valido e pratico, ma presenta anche alcuni svantaggi quali:

- la difficoltà nella costruzione di un modello di rumore che richiede grossi sforzi, anche nella fase di testing;
- se il modello del rumore utilizzato dal metodo steganografico dovesse cadere nelle mani del “nemico”, questi lo potrebbe analizzare per cercarne possibili difetti e quindi utilizzare proprio il modello stesso per controllare che un messaggio sia conforme a esso; così, il modello, che è parte integrante del sistema steganografico, fornirebbe involontariamente un metodo di attacco particolarmente efficace proprio contro lo stesso sistema.

Un'ultima categorizzazione delle tecniche steganografiche può essere fatta in base al tipo di modifiche effettuate sul contenitore all'atto della realizzazione:

- **substitution systems**: consiste nella sostituzione di una parte ridondante del contenitore con un messaggio segreto;
- **trasform domain techniques**: consiste nell'inserire informazioni segrete in uno spazio trasformato ottenuto dal dominio del segnale originale, da usare come contenitore (ad esempio nel dominio delle frequenze o di Fourier);
- **spread spectrum techniques**: l'idea alla base è tipica del processo di spread spectrum communication, in cui un segnale con uno spettro in frequenza, a banda limitata, viene distribuito su un intervallo di frequenze molto ampio (di solito si combina il segnale con una forma d'onda che abbia uno spettro di frequenze molto vasto come il rumore bianco, idealmente uniforme su tutte le frequenze); a seguito dello processo di spread, il contributo dato dal segnale in ogni frequenza è molto piccolo e quindi difficile da rilevare. Nella

steganografia spread spectrum, dopo che i dati sono stati combinati con il rumore, sono distribuiti nel contenitore, e poiché l'intensità dei dati risulterà molto bassa, essi non sono visivamente percepibili ma rilevabili solo con un confronto con il contenitore originale;

- **statistical methods**: codificano l'informazione cambiando diverse proprietà statistiche del contenitore ed utilizzano le tecniche di hypothesis testing nel processo di estrazione;
- **distortion techniques**: memorizzano l'informazione distorcendo il segnale e misurando la deviazione del contenitore originale nel processo di decodifica;
- **cover generation methods**: l'informazione è codificata creando un apposito contenitore per nascondere la comunicazione segreta.

4. STEGANOGRAFIA INIETTIVA SOSTITUTIVA

Si focalizzerà ora l'attenzione sul meccanismo con cui opera un particolare tipo di steganografia, quella iniettiva sostitutiva, in vista di un'implementazione software di tale algoritmo in un caso specifico.

Ogni informazione all'interno di un sistema digitale è rappresentata sotto forma di sequenze di bit. Quindi un valore, ad esempio il numero decimale 54, è si rappresenta secondo la notazione binaria posizionale come segue:

$$0 * 2^7 + 0 * 2^6 + 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 00110110$$

I bit associati alla base 2 con esponente più piccolo, sono detti bit meno significativi, in quanto una loro variazione incide con una minima variazione del valore che il byte rappresenta, mentre i bit associati alla base 2 con esponente più grande sono detti bit più significativi per il motivo opposto.

Ad esempio, cambiando il solo bit meno significativo della sequenza binaria 00110110 (decimale 54) si otterrà la nuova sequenza 00110111 corrispondente al numero decimale 55; cambiando invece il bit più significativo, quindi quello più a sinistra, la sequenza corrispondente sarà 10110110, ossia il numero 182 in decimale.

Come si può vedere, nonostante siano stati eseguiti cambiamenti di un solo bit in entrambi i casi, il risultato ottenuto è estremamente diverso, con un lieve cambiamento nel primo a fronte di un consistente cambiamento nel secondo, per cui il valore dell'informazione che il byte rappresenterà dopo una variazione dipende da una relazione esponenziale rispetto alla posizione del bit in esame. Il modello di steganografia che agisce sul bit meno significativo prende appunto il nome di Steganografia LSB (Less Significant Bit).

Su questa considerazione si basano molti degli algoritmi di steganografia per inserire messaggi all'interno di contenitori quali immagini e suoni, poiché è possibile cambiare il valore di gruppi di bit meno significativi senza stravolgere troppo il significato dell'informazione originaria.

All'interno di una immagine con profondità di colore 24 bit (8 bit per ciascun colore fondamentale Rosso Verde Blu), il cambiamento degli ultimi due bit nel byte che rappresenta il ogni singolo colore di un pixel non apporta variazioni significative al colore del pixel stesso, percettibili all'occhio umano e l'immagini si presenterà nel suo complesso con le medesime caratteristiche.

Un discorso analogo vale per i suoni campionati ove nelle codifiche a 16-bit, è possibile cambiare sino ad 8 meno significativi senza che il cambiamento della qualità del suono campionato sia percettibile all'orecchio umano.

A tal punto è possibile fare una valutazione sulla bontà del mezzo steganografico (contenitore) in modo da avere un'indice valutativo. Se con N si indica il numero di bit che compone ogni unità di memorizzazione del mezzo steganografico (ad esempio 8 bit nel caso di immagini e 16 per nel caso dei suoni) ed M il numero dei bit steganografabili, variabile a seconda nel modello steganografico utilizzato e cioè modificabili tale che ad ogni variazione non si verificano variazioni percettibili nella qualità (ad esempio 2 bit per le immagini e 8 per i suoni), per stimare la bontà del mezzo steganografico si calcola un parametro b che sarà proporzionale a M e inversamente proporzionale ad N , secondo la formula che segue:

$$b = (2^N / 2^M) (M / N)$$

Il risultato di tale calcolo, associando al valore $b = 16$ per le immagini e $b = 128$ per i suoni precedentemente citati, secondo il modello steganografico LSB, fa emergere che il suono risulta mediamente un mezzo steganografico migliore delle immagini.

5. IMPLEMENTAZIONE DI UN ALGORITMO STEGANOGRAFICO

Prima di procedere alla seguente implementazione di un algoritmo steganografico in linguaggio C è doveroso rivedere bene due aspetti fondamentali:

- struttura generale di un file;
- conoscenza approfondita del contenitore su cui si deve operare.

a. Struttura generale di un file

Un file, per quanto di interesse in questo lavoro, costituisce una sequenza di byte, ove ciascun byte può assumere un valore rappresentabile da 8 bit e quindi compreso nell'intervallo $[0, 255]$. Il limite di byte che un file può contenere è dettato dal file system su cui tale file viene memorizzato.

Diverse tipologie di file possono considerarsi composte da due parti fondamentali:

- header: consiste in una porzione del file in cui sono memorizzate, in modo ben strutturato, le informazioni inerenti il file, come dimensione, risoluzione, tipo di compressione etc.;
- corpo: contiene l'insieme dei dati che costituiscono la parte informativa del file.

b. Conoscenza approfondita del file contenitore

Si è scelto, in fase di realizzazione pratica dell'algoritmo in questione, di utilizzare, come file contenitore un formato particolarmente diffuso di immagini: il file Bitmap ovvero bmp (noto anche con l'estensione .dib). Questa estensione si riferisce ad uno standard grafico introdotto con il sistema operativo Windows 3.0 e successive versioni. Esso è nato per la memorizzazione delle immagini in formato punto per punto, ed in alcune versioni può implementare forme di compressione.

La struttura base del formato BMP può essere mappata sulle seguenti strutture dati:

- BITMAPFILEHEADER: contiene informazioni inerenti la struttura fisica del file BMP;
- BITMAPINFOHEADER: contiene informazioni inerenti l'immagine Bitmap come dimensione, numero di colori, risoluzione grafica etc.;
- RGBQUAD aColors[]: è un array contenente la tabella dei colori;
- BYTE aBitmapBits[]: è un array contenente il corpo del file, quindi il resto dei byte raggruppati in gruppi di tre byte per ogni pixel da rappresentare, in cui ogni byte del singolo gruppo costituisce il valore di un colore fondamentale RGB da rappresentare;

c. Struttura del formato BMP

La seguente tabella consente l'esatta interpretazione di ogni singolo byte all'interno di un header BMP, in base alla propria posizione (Offset), con i valori riferiti ad un file bitmap avente le dimensioni 100x100x256 colori, senza compressione, rispetto alla seguente legenda:

- Offset rappresenta la posizione relative del primo byte del campo della struttura rispetto all'inizio del file;
- Dim rappresenta la dimensione in byte del campo;
- nomeAPI rappresenta il nome assegnato all'elemento della struttura secondo la Microsoft API documentation;
- Stdvalue rappresenta il valore standard del campo in esadecimale.

a. BITMAPFILEHEADER:

Offset	Dim	nomeAPI	stdvalue	Descrizione
0	2	bfType	0x4D42	È sempre impostato con valori standard che rappresentano i caratteri ASCII 'BM' per indicare che si tratta di un file .bmp.
2	4	bfSize	??	Specifica la dimensione totale del file in byte.
6	2	bfReserved1	0x00	Riservato – sempre impostato a 0.
8	2	bfReserved2	0x00	Riservato – sempre impostato a 0.
10	4	bfOffBits	0x0436	Specifica l'offsets dal quale ha inizio il vettore dei dati aBitmapBits[] rispetto l'inizio del file.

b. BITMAPINFOHEADER:

Offset	Dim	nomeAPI	stdvalue	Descrizione
14	4	biSize	0x28	specifica la dimensione in byte della struttura BITMAPINFOHEADER.
18	4	biWidth	0x64 (100)	specifica la larghezza dell'immagine in pixels.
22	4	biHeight	0x64 (100)	specifica l'altezza dell'immagine in pixels..
26	2	biPlanes	1	specifies the number of planes of the target device, must be set to zero.
28	2	biBitCount	0x08	Specifica la profondità di colore per pixel ¹ .
30	4	biCompression	0x00	Specific ail tipo di compressione utilizzata, normalmente settato a zero per indicare senza compressione.
34	4	biSizeImage	0x00	Specifica la dimensione in byte del vettore dei dati "aBitmapBits[]" nel caso di BMP con compressione.
38	4	biXPelsPerMeter	0x00	specifica i pixel orizzontali per il tester sul dispositivo, solitamente impostato a zero.
42	4	biYPelsPerMeter	0x00	specifica i pixel verticali per il tester sul dispositivo, solitamente impostato a zero.
46	4	biClrUsed	0x00	Specifica il numero di colori utilizzati nella bitmap; se il valore è zero il numero dei colori viene calcolato usando il campo biBitCount.
50	4	biClrImportant	0x00	Specifica il numero di colori considerato importanti per la bitmap; se impostato a zero tutti i colori sono importanti.

c. RGBQUAD aColors[]:

La seguente tabella mostra la struttura di un singolo elemento del vettore RGBQUAD

aColors[]:

Offset	Dim	name	stdvalue	purpose
0	1	rgbBlue	-	Specifica il valore della componente Blu

¹ Il campo *biBitCount* accetta i seguenti valori: 1 (bianco/nero), 4 (16 colors), 8 (256 colors), 24 (16.7 million colors), da ciò ne deriva l'eventuale presenza, all'interno del file, della Tabella dei Colori nonché la sua dimensione e la dimensione in bit preposti alla rappresentazione di ogni pixel (1 = 1 bit per pixel – 4 = 1 byte per 2 pixel – 8 = 1 byte per pixel – 24 = 3 byte RGB per pixel).

1	1	rgbGreen	-	Specifica il valore della componente Verde
2	1	rgbRed	-	Specifica il valore della componente Rosso
3	1	rgbReserved	0x00	Sempre settato a zero.

d. BYTE aBitmapBits[]:

La sua dimensione nonché il modo in cui i dati in essa contenuti devono essere interpretati per la produzione dei pixel dipendono fondamentalmente dalla struttura BITMAPINFOHEADER.

È doveroso evidenziare che per la visualizzazione dell'immagine, i dati all'interno del vettore aBitmapBits[], costituenti i pixel, sono memorizzati per riga e che ogni riga deve avere una dimensione multipla di 4 (in caso contrario si procede con l'aggiunta di valori di riempimento); inoltre l'immagine viene memorizzata capovolta come nell'esempio sottostante:

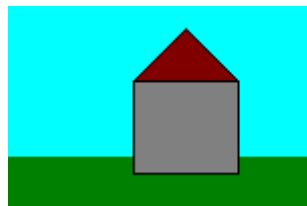


Immagine visualizzata a schermo

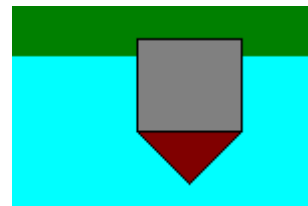


Immagine memorizzata nel file .bmp

d. Specifiche di realizzazione dell'algoritmo

Gli algoritmi di seguito riportati implementano una funzione steganografica iniettiva in un file BMP, in grado di inserire un file scomponendolo nei vari bit ed inserendone ognuno di essi nel LSB del colore Blue di ogni Pixel, sino alla fine del file.

L'algoritmo si compone dei seguenti file:

- bmp.h: contiene le strutture del formato BMP, i prototipi delle funzioni e le costanti;
- bmp_handle.c: contiene le funzioni per la lettura e la scrittura di un file BMP;
- stega_bmp.c: contiene la funzione steganografia adottata;
- destega_bmp.c: contiene la funzione steganografia inversa.

La funzione steganografica biiettiva è stata realizzata con i seguenti criteri:

- a. si procede alla lettura di un singolo byte per volta del messaggio da occultare;
- b. si definisce una maschera "MASK" con 00000001 per le successive operazioni;
- c. viene prelevato dal byte letto (chiamato ch) un singolo bit alla volta, partendo da quello meno significativo; tale operazione eseguita attraverso l'istruzione $((ch \gg i) \& (MASK))$ esegue ad ogni ciclo una rotazione di i bit verso destra del byte letto;
- d. il byte ottenuto dal punto c. viene successivamente sottoposto ad una operazione AND, bit a bit, con la maschera, in questo modo otteniamo il valore del bit da occultare nel contenitore;
- e. a tal punto viene azzerato il valore del bit meno significativo del colore Blue del Pixel in trattazione attraverso una operazione di AND bit a bit con il complemento a 1 della MASK $((Bitmap \rightarrow Pixel[k].Blue \& (\sim MASK))$;
- f. infine viene eseguita un'operazione di OR con il bit prelevato al punto d.

Esempio:

ch = 1010 1011, MASK = 00000001, Blue = 10110110, i=3 (bit da prelevare in ch)

CH ₀₁₀₁₀₁₀₁₁ >> i ₄ =	00010101 AND
MASK =	<u>00000001</u>
Bit Prelevato	00000001
Blue ₁₀₁₁₀₁₁₁	10110110 AND
~MASK(complemento a 1)	<u>11111110</u>
Blue =	10110110 OR
Bit Prelevato	<u>00000001</u>
Blue _{definitivo}	10110111

La funzione steganografica inversa è stata realizzata con i seguenti criteri:

- a. si imposta ch=0 (carattere da ricomporre);
- b. si esegue una operazione di AND fra il valore del colore Blue del Pixel in questione e la nostra MASK corrispondente sempre a 00000001, in questo modo preleviamo il valore contenuto nell'ultimo bit meno significativo di Blue;

- c. attraverso l'istruzione (bit $\ll i$) si procede ad eseguire una rotazione del bit prelevato di i posti verso sinistra, in modo da posizionarlo in corrispondenza del bit in cui dovrà essere inserito nel carattere da ricomporre;
- d. infine attraverso l' OR con ch, viene inserito il bit prelevato nel corrispondente bit di ch.

Esempio:

ch = 00000011, MASK = 00000001, Blue = 10110111, $i=5$ (posizione del bit in ch)

Blue ₁₀₁₁₀₁₁₁ $\gg i_4$ =	10110111 AND
MASK =	<u>00000001</u>
Bit Prelevato	00000001

Bit $\ll i_5$	00100000 OR
CH ₀₀₀₀₀₀₁₁	<u>00000011</u>
CH _{finale} =	00100011

(Sorgenti alla fine del documento)

6. TECNICHE DI STEGANALISI

Se la steganografia si propone di nascondere la presenza di informazioni e quindi garantire la comunicazione, la steganalisi ha lo scopo esattamente opposto, ovvero quello di rilevare la presenza di informazioni nascoste e quindi sottoporre il mezzo contenitore ad analisi ed attacchi, alla stessa stregua dei sistemi crittografici.

Individuando i seguenti elementi all'interno di un sistema di steganografia:

- secret_data: il messaggio segreto che si vuole inviare;
- clean_box: il contenitore, in cui nascondere il messaggio segreto, tale da non destare sospetti sulla presenza del secret_data;
- stego_box: il contenitore con all'interno già inserito il messaggio segreto,

si può immaginare l'operazione di steganografia come la seguente relazione che lega i suddetti elementi:

$$\text{stego_box} = F(\text{G(clean_box)}, \text{H(secret_data)})$$

dove:

- **F()** rappresenta una funzione a due variabili di tipo biiettivo;
- **G()** rappresenta una funzione di preparazione della `clean_box`;
- **H()** rappresenta una eventuale funzione crittografica del messaggio segreto, per rendere più difficoltosa l'individuazione di uno `stego_box`.

Data la biiettività della funzione di steganografia F , si può ricavare la funzione inversa di steganografia inversa F^{-1} come l'applicazione che dal dominio degli `stego_box` conduce a quello dei `clean_box`.

Quindi essendo le applicazioni di steganografia e di destenografia entrambe funzioni biettive per il transito fra due domini, è possibile, attraverso la steganalisi, applicare le tecniche di attacco crittografico basate sugli elementi noti all'avversario.

Le tipologie di attacco possono essere classificate come segue:

- **stego_only attack** quando si è in possesso esclusivamente dello `stego_box`;
- **known_cover attack** quando si ha sia lo `stego_box` che il `clean_box`; quindi la presenza di "secret_data" può essere evidenziata mediante confronto diretto;
- **known_message attack** quando si conoscono i "secret_data" ed il corrispondente `stego_box`; dalla analisi congiunta si possono ottenere informazioni sul funzionamento del sistema steganografico utilizzato;
- **chosen_stego attack** quando sono conosciuti la funzione del sistema steganografico e lo `stego_box`;
- **chosen_message attack** quando dato uno specifico messaggio segreto, si genera il corrispondente `stego_box`;
- **known_stego attack** quando sono noti la funzione steganografica, il `clean_box` e lo `stego_box`.

Un esempio di steganalisi applicabile efficacemente al codice steganografico iniettivo sostitutivo realizzato in questa relazione è quella di individuare i file modificati e di confrontarli

con l'originale, ottenendo "secret_data" per differenza. Questo metodo è appunto noto come known_cover attack, in cui sono note sia la clean_box che la stego_box.

Per procedere con tale attacco però bisogna conoscere a priori la clean_box, in modo da identificarla in maniera univoca e distinguerla dalla stego_box; a tal proposito è possibile fare uso delle funzioni di HASH, con algoritmi standard quali MD5 e/o SHA-1.

Per la comparazione dei due file, in modo da ricavare per differenza "secret_data", è possibile utilizzare il concetto che sta alla base della metrica di Hamming, che viene definita, date due sequenze binarie di uguale lunghezza, come il numero di posizioni per le quali i corrispondenti simboli sono differenti². Ciò consente di quantificare la differenza tra due entità x ed y, aventi la stessa dimensione.

Siano x ed y due immagini aventi lo stesso numero di bit, diremo che la loro distanza secondo la metrica Hamming è rappresentata dalla funzione $d_H(x,y)$ che fornisce il numero di bit corrispondenti in cui le due immagini differiscono.

La metrica di Hamming rispetta le seguenti proprietà:

1. $d_H(x,y) \geq 0$: la distanza è non negativa;
2. $d_H(x,y) = 0$ se e solo se $x = y$: ossia la distanza tra le due immagini è nulla se e solo se esse sono uguali;
3. $d_H(x,y) = d_H(y,x)$: scambiando l'ordine delle immagini la distanza non cambia;
4. $d_H(x,y) \leq d_H(x,z) + d_H(z,y)$: disuguaglianza triangolare.

Quindi, nel caso specifico, applicando ad ogni tripla di byte, rappresentanti il pixel RGB dell'immagine BMP, la tecnica della distanza implementabile attraverso una funzione di XOR, otterremo un risultato come segue:

	Red	Green	Blue
x = pixel clean_box:	01001000	00110010	11010101
y = pixel stego_box:	01001000	00110010	11010100

² esempio: tra 1011101 e 1001001 è pari a 2; tra "toned" e "roses" è pari a 3.

$d_H(x,y)$ 00000000 00000000 00000001

$$d_H(x,y) = 1$$

Ripetendo in maniera iterativa, il processo di cui sopra, per tutti i pixel dell'immagine, si ottiene il "secret_data" composto dalla sequenza delle distanze ottenute.

7. CONCLUSIONI E CAMPI DI APPLICAZIONE

La steganografia, nelle sue varie forme, ha trovato una vasta applicazione nell'ambito dell'informatica e delle comunicazioni digitali, in particolare viene utilizzata per due applicazioni fondamentali, ovvero:

- la privacy delle comunicazioni, vista la sempre crescente necessità di tutelare le comunicazioni digitali, da migliaia di occhi indiscreti che in ogni istante sono presenti in rete;
- nel settore commerciale il cosiddetto "watermarking" che consiste nel marchiare digitalmente un contenuto per garantirne la proprietà intellettuale. Un esempio di "watermarking" sono i sistemi di protezione inseriti all'interno di immagini, video e documenti per evitarne la copia e la distribuzione illegale, oppure il codice che ogni fotocopiatrice e/o stampante a colori inseriscono nelle loro stampe, sotto forma di sequenze di puntini gialli distribuiti, i quali permettono di risalire alla marca, modello e numero seriale dell'apparato che ha effettuato la stampa, al fine di prevenire l'illecito utilizzo di tali apparecchiature per la produzione di banconote false.

Nel caso del "watermarking" il marchio deve rispettare ancor di più il concetto di robustezza del mezzo steganografico, in quanto deve essere in grado di resistere al maggior numero di trasformazioni che possono essere applicate al contenitore senza che il messaggio nascosto possa subire perdite o danneggiamenti.

Algoritmo steganografico - Header con le principali definizioni

```

/*****
* Raggruppamento Carabinieri Investigazioni Scientifiche
* Reparto Tecnologie Informatiche - Sezione Telematica
* Realizzato dal Mar. Ca. Raffaele Olivieri
*
* Header contenente le principali definizioni
* Per la compilazione su linux con gcc
* gcc bmp_stega.c bmp_handle.c -o stega.exe
* gcc bmp_destega.c bmp_handle.c -o destega.exe
* DISCLAIMER:
* L'autore rende disponibile a tutti liberamente il presente codice e non si assume
* nessuna responsabilità circa il suo utilizzo nè di eventuali danni hardware/software
* che ne possano derivare dalla sua compilazione, linking, esecuzione, impiego illecito
*****/
typedef unsigned char      byte;
typedef unsigned short int word;
typedef unsigned long int  dword;
#define BMPFILETYPE 0x4D42
#define MASK 00000001

/* Strutture per l'Header del Windows bitmap files (.BMP) version 3*/
typedef struct strBMPFILEHEADER
{ word ImageFileType; /*2*/
  dword FileSize; /*4*/
  word Reserved1; /*2*/
  word Reserved2; /*2*/
  dword ImageDataOffset; /*4*/
} BMPFILEHEADER; /*Totale 14*/

typedef struct strBMPINFOHEADER
{ dword HeaderSize; /*4*/
  dword ImageWidth; /*4*/
  dword ImageHeight; /*4*/
  word NumberOfImagePlanes; /*2*/
  word BitsPerPixel; /*2*/
  dword CompressionMethod; /*4*/
  dword SizeOfBitmap; /*4*/
  dword HorizontalResolution; /*4*/
  dword VerticalResolution; /*4*/
  dword NumberOfColorsUsed; /*4*/
  dword NumberOfSignificantColors; /*4*/
} BMPINFOHEADER; /*Totale 40*/

typedef struct strRGBTRIPLE
{ byte Blue;
  byte Green;
  byte Red;
} RGBTRIPLE;

typedef struct strRGBQUAD
{ byte Red;
  byte Green;
  byte Blue;
  byte Reserved;
} RGBQUAD;

typedef struct strBITMAP
{ BMPFILEHEADER FileHeader;
  BMPINFOHEADER BmpHeader;
  dword DimPalette; /* 0 per Immagini True Color */
  RGBQUAD *Palette;
  dword NumPixel;
  RGBTRIPLE *Pixel;
} BITMAP;

/* Prototipi di funzioni */
void ReadBitmap (BITMAP *Bitmap, FILE *file_in);
void WriteBitmap (BITMAP *Bitmap, FILE *file_out);

```

e. Algoritmo steganografico - Funzioni di gestione Lettura e Scrittura BMP

```

/*****
* Raggruppamento Carabinieri Investigazioni Scientifiche
* Reparto Tecnologie Informatiche - Sezione Telematica
* Realizzato dal Mar. Ca. Raffaele Olivieri
*
* Funzioni di gestione Lettura e Scrittura File BMP
* Per la compilazione su linux con gcc
* gcc bmp_stega.c bmp_handle.c -o stega.exe
* gcc bmp_destega.c bmp_handle.c -o destega.exe
* DISCLAIMER:
* L'autore rende disponibile a tutti liberamente il presente codice e non si assume
* nessuna responsabilità circa il suo utilizzo nè di eventuali danni hardware/software
* che ne possano derivare dalla sua compilazione, linking, esecuzione, impiego illecito
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmp.h"

void ReadBitmap(BITMAP *Bitmap, FILE *file_in)
{
    byte fill;
    dword Palette;
    int i,j, k, row, height, width;
    dword FillByte;

    /* legge le intestazioni */
    fread(&Bitmap->FileHeader.ImageFileType, 2, 1, file_in);
    fread(&Bitmap->FileHeader.FileSize, 4, 1, file_in);
    fread(&Bitmap->FileHeader.Reserved1, 2, 1, file_in);
    fread(&Bitmap->FileHeader.Reserved2, 2, 1, file_in);
    fread(&Bitmap->FileHeader.ImageDataOffset, 4, 1, file_in);

    if (Bitmap->FileHeader.ImageFileType != BMPFILETYPE)
    {
        printf ("Il file non è una immagine bitmap\n");
        exit(EXIT_FAILURE);
    }

    fread(&Bitmap->BmpHeader, sizeof(BMPINFOHEADER), 1, file_in);

    if (Bitmap->BmpHeader.CompressionMethod != 0)
    {
        printf ("Immagine compressa, funzione non supportata\n");
        exit (EXIT_FAILURE);
    }

    switch (Bitmap->BmpHeader.BitsPerPixel)
    {
        case 1: /* Monocromatica */
            printf ("Immagine BMP Monocromatica - funzione non supportata\n");
            exit (EXIT_FAILURE);
            break;
        case 4: /* 16 colors */
            printf ("Immagine BMP a 16 colori - funzione non supportata\n");
            exit (EXIT_FAILURE);
            break;
        case 8: /* 256 colors */
            printf ("Immagine BMP a 256 colori - funzione non supportata\n");
            exit (EXIT_FAILURE);
            break;
        case 24: /* 16,7M colors - True Colors */
            height = Bitmap->BmpHeader.ImageHeight;
            width = Bitmap->BmpHeader.ImageWidth;
            Bitmap->DimPalette = 0;
            Bitmap->NumPixel = height * width;
            Bitmap->Pixel = (RGBTRIPLE *) malloc (Bitmap->NumPixel * sizeof(RGBTRIPLE));
            if (Bitmap->Pixel == NULL)
            {
                printf ("Errore di allocazione memoria per il vettore dei pixel\n");
                exit (EXIT_FAILURE);
            }
    }
}

```

```

    fill = width % 4;
    for (i = 0; i < height; i++)
    { row = (i * width);
      fread (&Bitmap->Pixel[row], sizeof(RGBTRIPLE)*width, 1, file_in);
      /* Legge i byte di riempimento delle righe e scarta */
      /* il numero dei byte è forzato ad essere multiplo di 4 */
      if(fill!=0)
        fread (&FillByte, sizeof(byte), fill, file_in);
    }
    printf ("%d Pixels Letti\n", row+width);
    break;

    default:
      printf ("Combinazione di colori non supportata\n");
      exit (EXIT_FAILURE);
      break;
  }
}
return;
}

void WriteBitmap (BITMAP *Bitmap, FILE *file_out)
{
  int i, j, k, fill, height, width, row;
  byte FillByte = 0;

  width = Bitmap->BmpHeader.ImageWidth;
  height = Bitmap->BmpHeader.ImageHeight;

  /* Scrittura del BmpFileHeader */
  fwrite(&Bitmap->FileHeader.ImageFileType, 2, 1, file_out);
  fwrite(&Bitmap->FileHeader.FileSize, 4, 1, file_out);
  fwrite(&Bitmap->FileHeader.Reserved1, 2, 1, file_out);
  fwrite(&Bitmap->FileHeader.Reserved2, 2, 1, file_out);
  fwrite(&Bitmap->FileHeader.ImageDataOffset, 4, 1, file_out);

  /* Scrittura del BmpInfoHeader */
  fwrite (&Bitmap->BmpHeader, sizeof (BMPINFOHEADER), 1, file_out);

  /* il numero di byte in una riga deve essere multiplo di 4 */
  fill = width % 4;
  for (i = 0; i < height; i++)
  {
    row = (i * width);
    fwrite (&Bitmap->Pixel[row], sizeof(RGBTRIPLE)*width, 1, file_out);
    /* Scrive i byte di riempimento delle righe % 4 */
    if(fill!=0)
      fwrite (&FillByte, sizeof(byte), fill, file_out);
  }

  printf ("%d Pixels Scritti\n", row+width);

  return;
}

```

f. Funzione Steganografica

```

/*****
* Raggruppamento Carabinieri Investigazioni Scientifiche
* Reparto Tecnologie Informatiche - Sezione Telematica
* Realizzato dal Mar. Ca. Raffaele Olivieri
*
* Funzione per occultare un file di testo da un file BMP 24 bit
* Per la compilazione su linux con gcc
* gcc bmp_stega.c bmp_handle.c -o stega.exe
* DISCLAIMER:
* L'autore rende disponibile a tutti liberamente il presente codice e non si assume
* nessuna responsabilità circa il suo utilizzo nè di eventuali danni hardware/software
* che ne possano derivare dalla sua compilazione, linking, esecuzione, impiego illecito
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmp.h"

void stega_bmp(BITMAP *Bitmap, FILE *file_txt)
{ int k = 0, i;
  unsigned int dim = Bitmap->BmpHeader.ImageWidth * Bitmap->BmpHeader.ImageHeight;
  byte ch, bit;
  do /* while (ci sono bit nel messaggio) */
    { ch = fgetc(file_txt);
      for (i=0; i<8; i++)
        { /* preleva l'iesimo bit da ch (Shift >> + And con maschera 1 */
          bit = ((ch >> i) & MASK);
          /* inserisce il bit prelevato al bit 0 del colore blue */
          Bitmap->Pixel[k++].Blue = ((Bitmap->Pixel[k].Blue & (~MASK)) | bit);
          if (k >= dim)
            { printf ("spazio terminato, il file da steganografare è troppo lungo\n");
              return -5;
            }
        }
    } while ((char) ch != EOF);
  return;
}

int main (int argc, char *argv[])
{ int errore = 0;
  FILE *file_in, *file_out, *file_txt;
  BITMAP Bitmap;
  if(argc != 4)
    { printf("ARGOMENTI: <input file.bmp> <output file.bmp> <file di testo>\n");
      errore = -4;
    }
  if((file_in = fopen(argv[1], "rb")) == NULL)
    { printf("Errore nell'apertura del file di input.bmp\n");
      exit (EXIT_FAILURE);
    }
  if((file_out = fopen(argv[2], "wb")) == NULL)
    { printf("Errore nell'apertura del file output.bmp\n");
      exit (EXIT_FAILURE);
    }
  if((file_txt = fopen(argv[3], "r")) == NULL)
    { printf("Errore nell'apertura del file di testo\n");
      exit (EXIT_FAILURE);
    }
  ReadBitmap(&Bitmap, file_in);
  fclose(file_in);
  stega_bmp(&Bitmap, file_txt);
  WriteBitmap(&Bitmap, file_out);
  printf("Messaggio Steganografato\n");
  free(Bitmap.Pixel); /* Libera il vettore dei pixel */
  fclose(file_txt); /* Chiude il file di Testo */
  fclose(file_out); /* Chiude il file BMP di Output */
  return EXIT_SUCCESS;
}

```

g. Funzione Steganografica Inversa

```

/*****
* Raggruppamento Carabinieri Investigazioni Scientifiche
* Reparto Tecnologie Informatiche - Sezione Telematica
* Realizzato dal Mar. Ca. Raffaele Olivieri
*
* Funzione di estrazione di un file di testo da un file BMP 24 bit
* Per la compilazione su linux con gcc
* gcc bmp_destega.c bmp_handle.c -o destega.exe
* DISCLAIMER:
* L'autore rende disponibile a tutti liberamente il presente codice e non si assume
* nessuna responsabilità circa il suo utilizzo nè di eventuali danni hardware/software
* che ne possano derivare dalla sua compilazione, linking, esecuzione, impiego illecito
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmp.h"

void destega_bmp (BITMAP *Bitmap, FILE *file_txt)
{
    unsigned int k = 0, i;
    byte ch, bit;

    /* while (!EOF && ci sono ancora Pixel) */
    do
    {
        ch=0;
        for (i = 0; i < 8; i++)
        {
            /* leggi il j-esimo bit di ch dal bit 0 di Bitmap->Pixel[k].Blue */
            bit = (Bitmap->Pixel[k].Blue & MASK);
            /* imposta il j-esimo bit del carattere alla posizione j) */
            ch = (ch | (bit << i));
            k++;
        }
        if ((char) ch != EOF)
        {
            printf ("%c", ch);
            fwrite (&ch, sizeof(byte), 1, file_txt);
        }
    } while (( (char) ch != EOF) && (k < Bitmap->NumPixel));
    return;
}

int main (int argc, char *argv[])
{
    FILE *file_in, *file_txt;
    BITMAP Bitmap;
    if (argc != 3)
    {
        printf ("ARGOMENTI: <input file.bmp> <file di testo di output>\n");
        exit (EXIT_FAILURE);
    }
    if ((file_in = fopen (argv[1], "rb")) == NULL)
    {
        printf ("Errore nell'apertura del file Bitmap\n");
        exit (EXIT_FAILURE);
    }
    if ((file_txt = fopen (argv[2], "w")) == NULL)
    {
        printf ("Errore nell'apertura del file di output\n");
        exit (EXIT_FAILURE);
    }
    ReadBitmap(&Bitmap, file_in);
    fclose(file_in);
    destega_bmp (&Bitmap, file_txt);
    fclose (file_txt);

    printf("\n\nMessaggio Decodificato\n");
    free(Bitmap.Pixel); /* Libera il vettore dei pixel */
    return EXIT_SUCCESS;
}

```