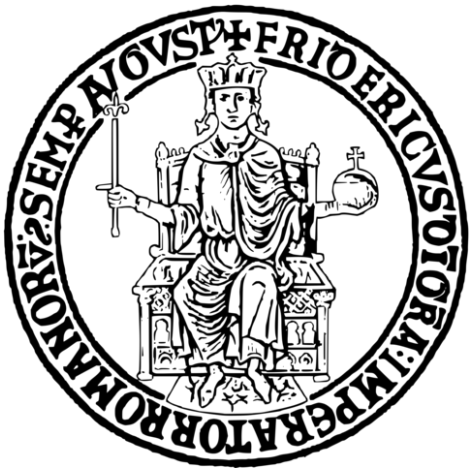
Università degli studi di Napoli

Federico II



Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Ingegneria Elettrica e delle tecnologie dell'informazione

Elaborato in

Network Security

Docente:
Simon Pietro Romano

Candidato:
Vincenzo Carandente
M63001229

Anno accademico 2021/2022

Introduzione

La circolazione e la condivisione delle informazioni ha, nella nostra epoca basata sulla comunicazione, un'importanza cruciale. Spesso però c'è il desiderio o la necessità di mantenere queste comunicazioni riservate, in modo che nessuno, tranne il legittimo destinatario, possa acquisire informazioni ritenute sensibili. L'approccio più utilizzato per rendere privata la conversazione è quello di rendere il messaggio incomprensibile a chi non è a conoscenza delle tecniche necessarie a renderlo nuovamente leggibile. La cosiddetta rivoluzione digitale ha portato anche nel settore della segretezza (o sicurezza) delle informazioni nuovi paradigmi di implementazione di teorie e tecniche già note.

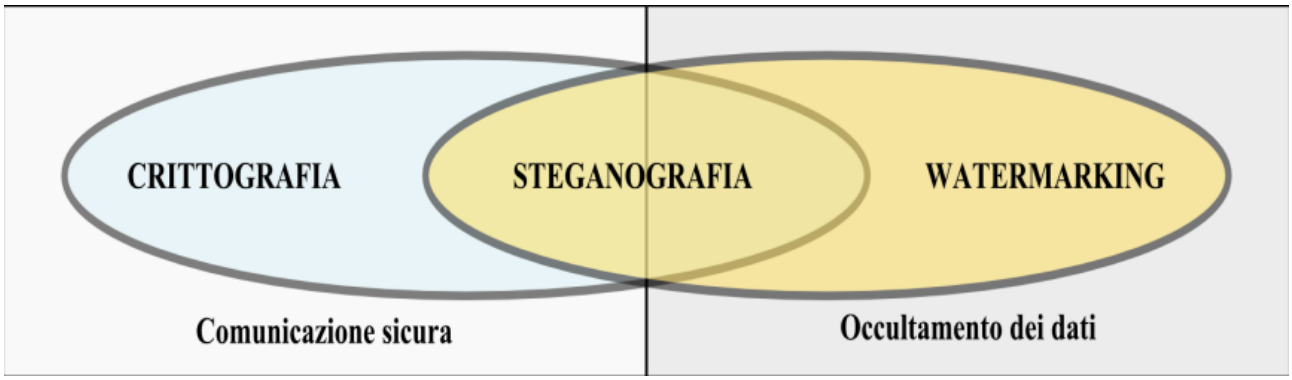


Figura 1: Crittografia-Watermarking-Steganografia

La scienza che si occupa di questo problema è la crittografia, il cui obiettivo principale è quello di garantire la segretezza attraverso la codifica del messaggio. Il messaggio è visibile, ma è codificato mediante appositi algoritmi di cifratura che lo rendono incomprensibile a chi non è a conoscenza dei relativi sistemi di decodifica.

Il watermarking (letteralmente "filigranatura") invece è una tecnologia grazie alla quale è possibile inserire opportune informazioni in un segnale, in particolare su file multimediali quali immagini e video, magari per segnalarne l'originalità o il titolare dei diritti di proprietà. Un watermark, proprio come le

filigrane delle banconote, deve essere visibile solo in certe condizioni – per esempio dopo l'applicazione di opportuni algoritmi.

La Steganografia, che possiamo dire fonda un po' le due cose, nasconde l'esistenza stessa del messaggio, includendolo in un mezzo che potremmo definire "neutrale" e garantendo quindi la segretezza della comunicazione.

Steganografia

La Steganografia, quindi, è la tecnica che permette di nascondere dati all'interno dei dati stessi, con lo scopo di celare la comunicazione tra due interlocutori. Essa rappresenta una tecnica di crittografia che può essere utilizzata come metodo sicuro per proteggere i dati. Deriva dalle parole greche steganos (che significa nascosto o coperto) e dal grafo della radice greca (che significa scrivere).

Tracce della steganografia si hanno già nell'antichità. Non mancano infatti riferimenti a chi per nascondere un messaggio importante ricoprì la tavoletta con cera per poi scriverci sopra messaggi innocui oppure andando più indietro ancora nel tempo a chi scriveva messaggi all'interno dello stomaco dei conigli.

Le tecniche steganografiche possono essere applicate a immagini, file video o file audio. Il contenuto da nascondere tramite la steganografia, chiamato "testo nascosto", viene spesso crittografato prima di essere incorporato nel file di testo della copertina o nel flusso di dati che sembra apparire innocuo.

Esistono online diversi tools e strumenti per utilizzare la steganografia, in modo da nascondere i file all'interno di altri file sui computer.

Il contrario della steganografia è invece la Steganalisi. Essa non è altro che un insieme di tecniche di analisi per la rivelazione di messaggi nascosti (anche senza necessariamente procedere alla decifratura). Possibili motivazioni per cui si attuano tecniche di steganalisi sono: contro-spionaggio, antiterrorismo, controllo dell'opinione pubblica in regimi totalitari, raccolta di dati (anche sensibili) per motivazioni commerciali o per fini illeciti. Indipendentemente dalle motivazioni lo sviluppo di tecniche di steganalisi è indispensabile allo studio delle stesse tecniche di steganografia.

Una prima classificazione dei modelli steganografici si divide in steganografia iniettiva e steganografia generativa:

- La steganografia iniettiva è la più utilizzata, consiste nell'inserire (iniettare) il messaggio segreto all'interno di un altro messaggio che funge

da contenitore, in modo tale da non risultare visibile all'occhio umano e da essere praticamente indistinguibile dall'originale.

- La steganografia generativa consiste nel prendere il messaggio segreto e costruirgli un opportuno contenitore in modo tale da nascondere il messaggio nel miglior modo possibile.

Una seconda classificazione che si può fare è la seguente:

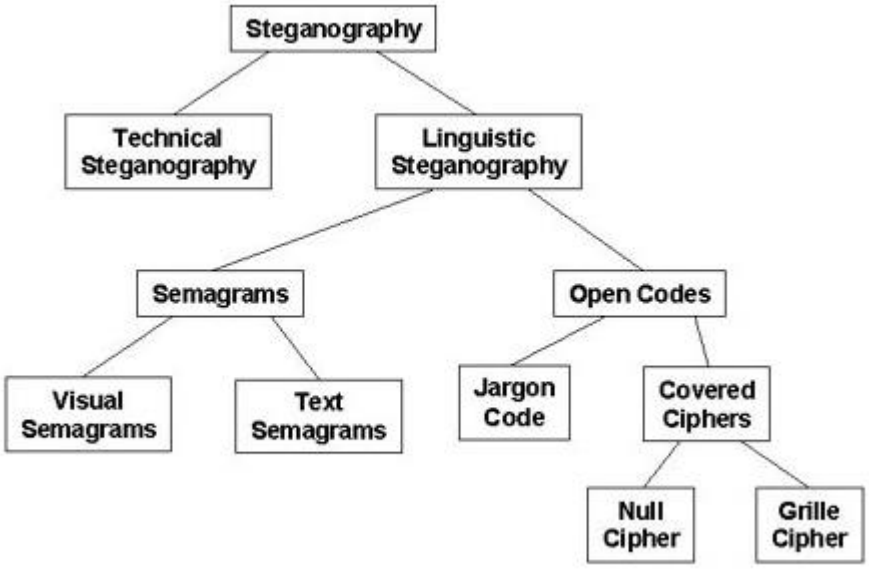


Figura 2: Classificazione steganografia

La steganografia tecnica consiste nel nascondere messaggi di testo con l'aiuto di metodi fisici o chimici. Alcuni di questi sono:

1. **Inchiostro invisibile**;
2. **Microdot**: è un testo o un'immagine che viene considerevolmente ridotta per non essere riconosciuta (tradotto infatti in *micropunti*);
3. **Steganografia digitale**: avvalendosi quindi di un computer per nascondere il testo.

Nella Steganografia Linguistica viene invece nascosto il messaggio in vettori o stringhe di testo. Si divide a sua volta in:

1. **Semagrammi** (*ndb:* semagram in inglese): vengono utilizzati simboli o simboli differenti per nascondere il messaggio. Si compone da:
    - **Visuali**: si utilizzano oggetti fisici per trasmettere il messaggio;
    - **Testuali**: nel testo stesso viene nascosto un messaggio con stringhe particolari (ad esempio lettere rimpicciolite o più spazio tra le due).

2. **Codici aperti**: il messaggio sembra innocente, mentre viene nascosto all'interno di esso. Si compone da:
    - **Codice jargon**: è un linguaggio che un gruppo ristretto di persone può capire ma è senza significato per altre. Questo codice è composto da segnali, terminologie e conversazioni che hanno un preciso significato solo per un gruppo di persone;
    - **Cifrari nascosti**: il messaggio è nascosto in un vettore di stringhe, in modo che solo chi conosce la regola può decifrare il testo (ad esempio, *leggi ogni 50 parole*). Si divide a sua volta in:
        - **Cifrario a griglia;**
        - **Cifrario nullo.**

Le principali tecniche di steganografia sono classificate in sei gruppi principali. Essi sono:

1. **Tecniche di sostituzione**: in questa tecnica vengono sostituiti i bit non utilizzati con bit composti dal messaggio segreto. Se il destinatario conosce la posizione di questi bit può estrarre il messaggio;
2. **Tecnica di trasformazione**: l'informazione viene nascosta mediante trasformazioni dell'immagine (o video, audio) tagliando, comprimendo o applicando altri processi di modifica. Si può effettuare su un'intera immagine o solo su blocchi specifici;
3. **Tecnica dello spettro espanso**: questa tecnica è principalmente utilizzata nelle comunicazioni radio militari, proprio per la sua robustezza e difficoltà di estrazione. Può essere descritto come un metodo nel quale un segnale rilasciato a una determinata frequenza di banda viene trasformato poi utilizzando una larghezza di banda più alta (e quindi reso irriconoscibile).
4. **Tecnica della distorsione**: vengono effettuate una serie di modifiche al file. Il destinatario può confrontare il file originario con quello ricevuto, in modo da ricostruire la sequenza di modifiche;
5. **Generazione di una copertura**: in questo caso viene generata una copertura ad-hoc per nascondere il messaggio.

Steganalisi

Le tecniche che invece si prefiggono di individuare la presenza di un messaggio occultato attraverso l'uso di tecniche steganografiche rientrano sotto il nome di **steganalisi**. L'obiettivo almeno in prima istanza non è quello di individuare e

decodificare il contenuto del messaggio segreto, ma semplicemente determinare se un mezzo contiene un messaggio oppure no. Pertanto, la steganalisi può essere formulata come un test sull'ipotesi che il mezzo contenga un messaggio segreto.

Formalmente, dato un insieme di osservazioni Y= {y1, y2, …, yn}, o di loro funzioni dette feature o statistiche si formulano due ipotesi alternative:

- Il file non contiene un messaggio segreto;
- Il file contiene un messaggio segreto.

La decisione tra le due ipotesi viene presa in base ad un criterio di ottimalità.

Il tentativo di determinare la presenza di un messaggio segreto è detto attacco. È possibile distinguere gli attacchi a seconda delle parti dell'equazione note all'analista. Si avrà quindi:

- **Stego-only-attack**: l'analista ha intercettato solo frammento stego e può analizzarlo.
- **Known message attack**: il mittente ha usato la stessa cover ripetutamente per nascondere dati. La differenza fra i messaggi stego, quindi, dipende solo dal messaggio originale che si vuole trasmettere.
- **Known carrier attack**: l'attaccante ha intercettato il frammento stego e sa quale cover è stato usato per crearlo.
- **Known stego attack**: l'attaccante ha "tutto": ha intercettato il frammento stego, conosce la cover usata e il messaggio segreto nascosto nel frammento stego;
- **Manipulating the stego data**: l'attaccante è in grado di manipolare i frammenti stego. Il che significa che l'attaccante può togliere il messaggio segreto dal frammento stego (inibendo la comunicazione segreta);
- **Manipulating the cover data**: l'attaccante può manipolare la cover e intercettare il frammento stego. Questo può significare che con un processo più o meno complesso l'attaccante può risalire al messaggio nascosto

Immagini

La tecnica di steganografia iniettiva su immagini sicuramente più diffusa anche grazie alla sua relativa facilità di implementazione è quella che si basa sulla modifica del bit meno significativo (LSB).

Quando si va a parlare di immagini digitali è molto comune descrivere i colori riferendosi allo spazio colore RGB (red, green, e blue). Lo spazio RGB è basato sul fatto che ogni colore possa essere rappresentato da una "miscela" dei tre colori primari red, green, e blue. I vari contributi sono assunti indipendenti l'uno dall'altro.

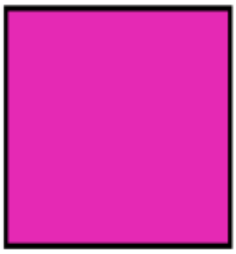
Supponiamo di voler utilizzare come contenitore un file di tipo bitmap con una profondità di colore a 24 bit, quindi una matrice MxN di pixel codificata in modalità RGB. Quindi per esempio un file bitmap a 24 bit di dimensione 640x480 occuperà uno spazio di 640x480x3=921600 byte. Una possibile operazione di steganografia sostitutiva su questi file consiste nel sostituire i bit meno significativi dei singoli byte (LSB).

11100001 **00000100** **00010111**

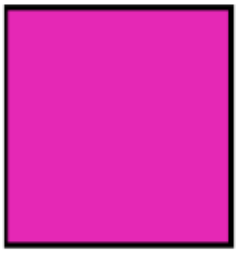
E volessimo codificare: 110 Il pixel diventerà:

1110000<u>1</u> **0000010<u>1</u>** **0001011<u>0</u>**

Queste semplici operazioni fanno sì che le variazioni di colore siano praticamente impercettibili ad occhio nudo.



R: 11100101
G: 00101001
B: 10110100



R: 11100101
G: 00101000
B: 10110101

Quindi dato che un solo pixel può contenere un'informazione segreta di 3 bit, un'immagine di dimensione MxN può contenere un messaggio segreto lungo fino a (MxNx3) /8 byte.
Se devo ad esempio inserire il carattere "A" che in binario posso tradurre in 01000001 e voglio usare gli ultimi due bit di ogni colore per codificarli allora avrò che le prime 6 cifre andranno in un pixel e le ultime due nell'altro.
Utilizzando gli ultimi due bit la differenza di colori ad occhio nudo non è ancora apprezzabile.
Non si deve eccedere nell'utilizzo di più LSB in un unico pixel, infatti, andando ad usarne di più si va sì ad aumentare la capacità della tecnica ma andandone a inficiare la trasparenza (ovvero la capacità di non essere rilevata ne dall'uomo ne dà strumenti di steganalisi) e viceversa.

Altra tecnica interessante per la steganografia basata su immagini è quella BPCS (Bit-Plane Complexity Segmentation) che invece di considerare solamente i bit meno significativi come bit da sostituire si vanno a determinare proprio zone dell'immagine dove è possibile sostituire in maniera sicura senza compromettere l'immagine.
Un'immagine P costituita da pixel ad n-bit può essere decomposta in un insieme di n immagini binarie. Per esempio, se l'immagine è una immagine n-bit gray, la possiamo descrivere come: P = (P1, P2, ..., Pn). Un'immagine RGB, invece, può essere vista come: P= (PR1, PR2, …, PRn; PG1, PG2, ..., PGn; PB1, PB2, ..., PBn) Dove PR1, PG1, PB1 è la bit-plane più significativa (immagine formata dai bit più significativi di tutti i pixel dell'immagine) e PRn, PGn, PBn è la bit-plane meno significativa.

Ogni bit-plane può essere segmentato in regioni shape-informative (forma informativa, cioè parti dell'immagine che sono significative sotto il punto di vista visivo) e noise-looking (disturbo visivo, cioè poco rilevanti dal punto di vista visivo). Le regioni semplici (cioè ben distinguibili visivamente) sono delle shape-informative e pertanto non possono essere modificate, quelle complesse, invece, rappresentano delle noise-looking che possono essere rimpiazzate senza deteriorare la qualità dell'immagine nel complesso.

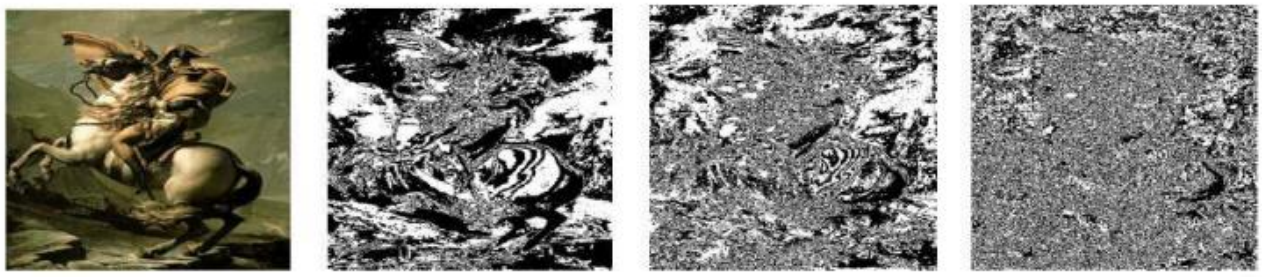


Figura 3: Bit-planes

Uno dei tool più utilizzati per la steganografia su Linux è **Steghide**. Questo programma riesce a fare hiding sia su immagini che su file audio. I formati supportati sono JPEG, BMP, WAV e AU.
Durante la fase di hiding il colore e le frequenze di campionamento dei file non vengono modificate, rendendo l'embedding molto resistente a test statistici di primo ordine.
Steghide utilizza un approccio alla steganografia basato sulla teoria dei grafi ma non è necessario ovviamente conoscerla per utilizzare il tool.
L'algoritmo di embedding funziona più o meno così.In un primo momento i dati segreti vengono compressi e criptati.
Poi viene creata una sequenza di posizioni di pixel nel file di copertura sulla base di un generatore di numeri pseudocasuali inizializzato con la passphrase (i dati segreti saranno incorporati nei pixel in queste posizioni).
Di queste posizioni, quelle che non hanno bisogno di essere modificate (perché contengono già il valore corretto per caso) vengono eliminate dalla lista iniziale.
Quindi un algoritmo di corrispondenza basato su grafo trova coppie di posizioni tali che lo scambio dei loro valori ha l'effetto di incorporare la parte corrispondente dei dati segreti.

Anche i pixel nelle posizioni rimanenti (quelle che non fanno parte di una tale coppia) vengono modificati per contenere i dati incorporati (ma ciò avviene sovrascrivendo i pixel, non scambiandoli con altri).
Il fatto che (la maggior parte) dell'incorporazione avvenga tramite lo scambio dei valori dei pixel implica che la statistica del primo ordine (cioè il numero di volte che un colore ricorre nell'immagine) non viene modificata. Per i file audio l'algoritmo è lo stesso, tranne per il fatto che vengono utilizzati campioni audio anziché pixel.
L'algoritmo di crittografia predefinito è AES con una dimensione della chiave di 128 bit in modalità cifratura a blocchi. Il checksum viene calcolato utilizzando l'algoritmo CRC32.
Come possiamo vedere nelle immagini riportate in seguito ecco le varie opzioni che è possibile inserire nel comando steghide.
Queste si dividono in opzioni di embedding e opzioni per l'estrazione di informazioni da un file già lavorato con steganografia.

```
embedding options:
 -ef, --embedfile        select file to be embedded
   -ef <filename>        embed the file <filename>
 -cf, --coverfile        select cover-file
   -cf <filename>        embed into the file <filename>
 -p, --passphrase        specify passphrase
   -p <passphrase>       use <passphrase> to embed data
 -sf, --stegofile        select stego file
   -sf <filename>        write result to <filename> instead of cover-file
 -e, --encryption        select encryption parameters
   -e <a>[<m>]|<m>[<a>]  specify an encryption algorithm and/or mode
   -e none               do not encrypt data before embedding
 -z, --compress          compress data before embedding (default)
   -z <l>                 using level <l> (1 best speed...9 best compression)
 -Z, --dontcompress      do not compress data before embedding
 -K, --nochecksum        do not embed crc32 checksum of embedded data
 -N, --dontembedname     do not embed the name of the original file
 -f, --force             overwrite existing files
 -q, --quiet             suppress information messages
 -v, --verbose           display detailed information
```

Figura 4: Opzioni embedding

```
extracting options:
 -sf, --stegofile        select stego file
   -sf <filename>        extract data from <filename>
 -p, --passphrase        specify passphrase
   -p <passphrase>       use <passphrase> to extract data
 -xf, --extractfile      select file name for extracted data
   -xf <filename>        write the extracted data to <filename>
 -f, --force             overwrite existing files
 -q, --quiet             suppress information messages
 -v, --verbose           display detailed information
```

Figura 5: Opzioni estrazione

Come possiamo vedere si hanno da una parte le opzioni per l'inserimento del file che andrà a fare da cover e l'opzione per il file che invece vogliamo andare a nascondere. Vi sono, inoltre, l'opzione per l'inserimento della passphrase, per un eventuale compressione, un eventuale cambio di algoritmo di criptaggio ed infine l'opzione "verbose" per avere informazioni più precise su cosa si sta eseguendo.
Dall'altra parte abbiamo invece meno opzioni ma quelle fondamentali ovvero il file da cui estrarre le info, la passphrase, il nome per il file estratto e nuovamente l'opzione verbose.
Di seguito possiamo vedere qualche esecuzione e le relative risposte del programma.

```
vincenzo@vincenzo-VirtualBox:~/Scrivania$ steghide embed -cf Tigre.jpeg -ef prov
a.txt -v
Enter passphrase:
Re-Enter passphrase:
reading secret file "prova.txt"... done
reading cover file "Tigre.jpeg"... done
creating the graph... 68 sample values, 251 vertices, 27885 edges
executing Static Minimum Degree Construction Heuristic... 97,2% (1,0) done
```

Figura 6: Opzione verbose

```
vincenzo@vincenzo-VirtualBox:~/Scrivania$ steghide embed -cf Tigre.jpeg -ef prov
a.txt
Enter passphrase:
Re-Enter passphrase:
embedding "prova.txt" in "Tigre.jpeg"... done
```

Figura 7: Embedding semplice

```
vincenzo@vincenzo-VirtualBox:~/Scrivania$ steghide embed -cf Tigre.jpeg -ef prov
a.txt -p 1230
embedding "prova.txt" in "Tigre.jpeg"... done
```

Figura 8: Embedding con passphrase

```
vincenzo@vincenzo-VirtualBox:~/Scrivania$ steghide extract -sf Tigre.jpeg -xf pr
ova1.txt
Enter passphrase:
wrote extracted data to "prova1.txt".
```

Figura 9: Estrazione

Come spesso avviene su Linux le password sono completamente oscurate per motivi di sicurezza.
Possiamo vedere in seguito il file "prova.txt" che è quello che è stato inserito nell'immagine cover per nasconderlo e il file "prova1.txt" che invece è quello

estratto dall’immagine successivamente provando il funzionamento dello strumento.

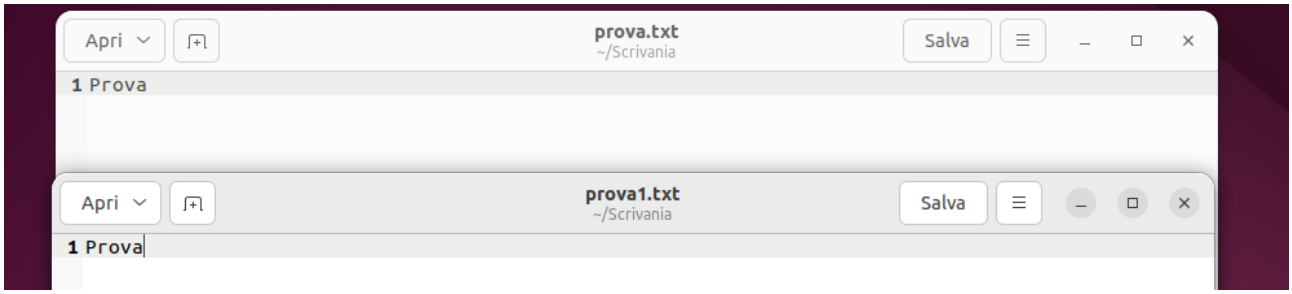


Figura 10: Confronto testi

Si mostra invece ora come la distinzione ad occhio nudo delle due immagini, quella originale e quella post-embedding, siano completamente indistinguibili.

Figura 9: Immagine originale

Figura 11: Immagine dopo l'embedding

Si può vedere dalle prossime immagini come questo metodo di steganografia (grazie anche al fatto che il messaggio da nascondere fosse abbastanza piccolo) sia robusto e passi inosservato a due dei più importanti tool di steganalisi di Linux ovvero Binwalk e Foremost.
Questi ultimi nell'analisi dell'immagine modificata prima non riescono a trovare altro se non normalissimo un file jpeg.

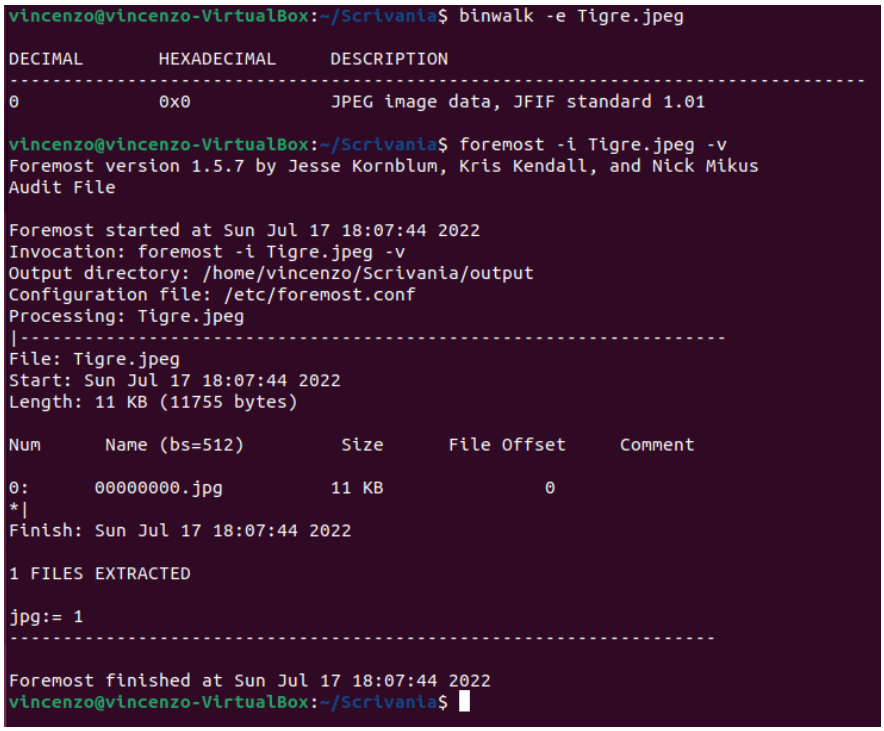


Figura 12: Steganalisi

File audio

I file audio come anche quelli video sono preferiti a file testuali o immagini per la presenza in essi di ridondanza. La stenografia audio però risulta essere più complessa rispetto a quella che riguarda le immagini ed i video poiché lo Human Auditory System (HAS) è molto più sensibile dello Human Visual System (HVS).
Affinché qualsiasi tecnica di steganografia audio sia implementabile, deve soddisfare tre condizioni:

- Capacità
- Trasparenza
- Robustezza.

La capacità è la quantità di informazioni segrete che può essere incorporata all'interno del messaggio ospite mentre trasparenza significa quanto bene il

messaggio segreto è incorporato nel suo file cover. La robustezza di una tecnica indica la capacità del messaggio segreto incorporato di resistere agli attacchi.
Per essere certi che il messaggio da nascondere riesca ad essere coperto dal file host deve essere necessariamente rispettata questa regola:

$$\text{Samples of Host Message} = 8 * \text{Bits of Secret Message}$$

Il metodo dell'LSB viso in precedenza per le immagini è ovviamente valido anche per i file audio ma recenti studi hanno cercato di migliorare questo approccio come vedremo ora. Infatti, questa tecnica pur andando a modificare l'ultimo bit quindi quello meno significativo va comunque ad introdurre una minima fonte di rumore. Di per sé questo non sarebbe un problema ma andando a modificare una grande quantità di bit il rumore indotto sulla fonte diventerebbe significativo e quindi sarebbe rilevabile dagli strumenti di steganalisi facendo così fallire l'operazione di hiding.

Per andare a rendere ancora più sicuro l'LSB per quanto riguarda la steganografia audio si è studiato un metodo chiamato Secure LSB che non fa altro che aggiungere operazioni di RSA e XOR per andare a rendere più sicuro l'embedding.
Si va infatti prima di tutto a crittografare con RSA il dato prima di andare a fare l'embedding su un file cover. Poi dato che LSB convenzionale è vulnerabile a pattern recognition da parte degli attaccanti si va a prendere i Most Significant Bits del campione che sto leggendo per andare a scegliere quale sarà il prossimo campione da modificare. Questo aggiunge una notevole randomicità alla steganografia.
Se ad esempio considero "I" il sample che sto correntemente leggendo allora seguendo questa tabella che si basa sui 3 bit più significativi allora scoprirò quale è il prossimo campione da modificare.

1st MSB	2nd MSB	3rd MSB	Next Sample
0	0	0	I+1
0	0	1	I+2
0	1	0	I+3
0	1	1	I+4
1	0	0	I+1
1	0	1	I+2
1	1	0	I+3
1	1	1	I+4

Figura 13: Tabella MSB per scegliere il prossimo campione

Per aggiungere infine un altro livello di randomicità oltre al prossimo campione da modificare si va a scegliere anche quale fra il primo, il secondo e il terzo tra gli LSBs deve essere modificato.
Anche in questo caso si sceglie in base agli MSBs ma in questo caso solo con i primi due come possiamo vedere in tabella.

1st MSB	2nd MSB	Bit Selected
0	0	3rd LSB
0	1	2nd LSB
1	0	1st LSB
1	1	1st LSB

Figura 14: Tabella MSB per decidere il prossimo LSB

Anche la sostituzione del bit non è così banale e contribuisce alla robustezza di questa tecnica: essa si basa infatti su quello che è chiamato metodo Kali che presenta una operazione XOR che coinvolge anche la chiave a 256 bit scelta per l'RSA ed il bit del messaggio che vogliamo andare a nascondere.

$$B_s(n) = B_o(m) \oplus B_k((n \bmod 8) + 1)$$

Figura 15: Metodo Kali

Da risultati sperimentali si ottiene che questo metodo, nonostante la grande complessità dietro all'algoritmo di conversione dei bit risulta non aggiungere rumore (valutato con l'indice SNR) rispetto ad un'implementazione più semplice e quindi "prevedibile" come l'Enhanced LSB che è già comunque una tecnica un po' più sofisticata rispetto al semplice LSB in quanto consiste nell'implementazione di una sola delle due tecniche (Sample Selection e Bit Selection) che sono state prima analizzate.

Ma quale è il più significativo LSB che si può andare a modificare prima che la steganografia sia facilmente rilevabile?
Come si può vedere nelle immagini sottostanti il cambiamento del terzo LSB non incide molto sulla forma dell'audio originale, il cambiamento del quarto invece già lascia intravedere qualche differenza, mentre l'ottavo cambia completamente i valori assunti dall'audio.
In sostanza, quindi, è universalmente accettato che il terzo LSB è il più significativo LSB che può essere modificato mantenendo integro il file originale.

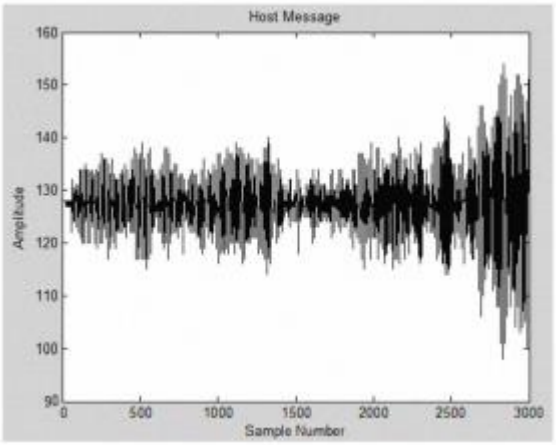


Figura 16: Audio originale

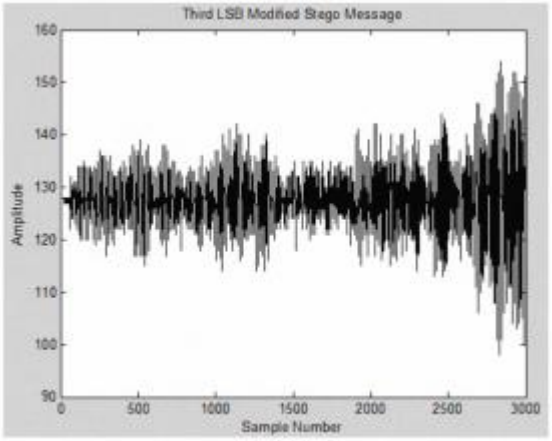


Figura 17: 3° LSB

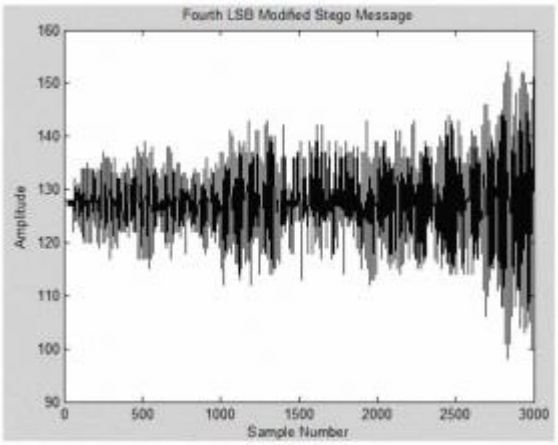


Figura 18: 4° LSB

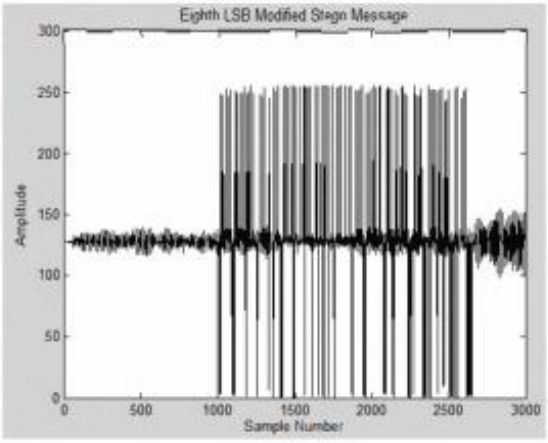


Figura 19: 8° LSB

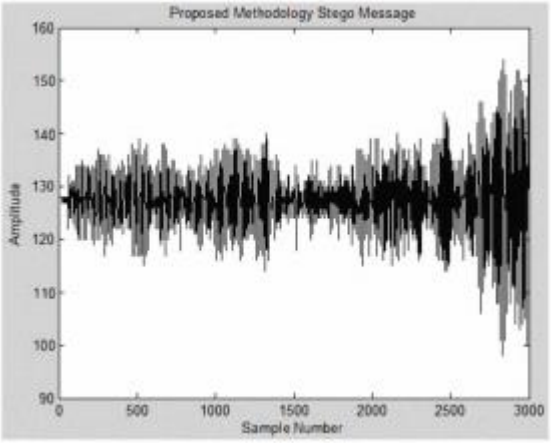


Figura 20: Metodologia proposta

Altre tecniche interessanti per la steganografia sui file audio sono:

- Parity coding: si utilizza il bit di parità per codificare
- Phase coding: La fase dell'audio è modificata in base al messaggio da nascondere.
- Spread spectrum: Il messaggio è codificato all'interno dello spettro dell'audio

- Echo hiding: L'informazione è nascosta nell'eco del segnale originale

Per quanto riguarda il funzionamento sullo strumento steghide per i file audio basta semplicemente cambiare l'opzione da –sf a –cf per poter inserire come file cover un file di tipo WAV.

```
vincenzo@vincenzo-VirtualBox:~/Scrivania$ steghide embed -cf audio.wav -ef prova
.txt
Enter passphrase:
Re-Enter passphrase:
embedding "prova.txt" in "audio.wav"... done
```

Figura 21: Embedding Audio

Questa volta il confronto fra la fonte originale e quella modificata è stata fatta usando un programma musicale di nome Audacity.
Come nel caso delle immagini nemmeno in questo caso è possibile apprezzare differenze fra le due onde audio.

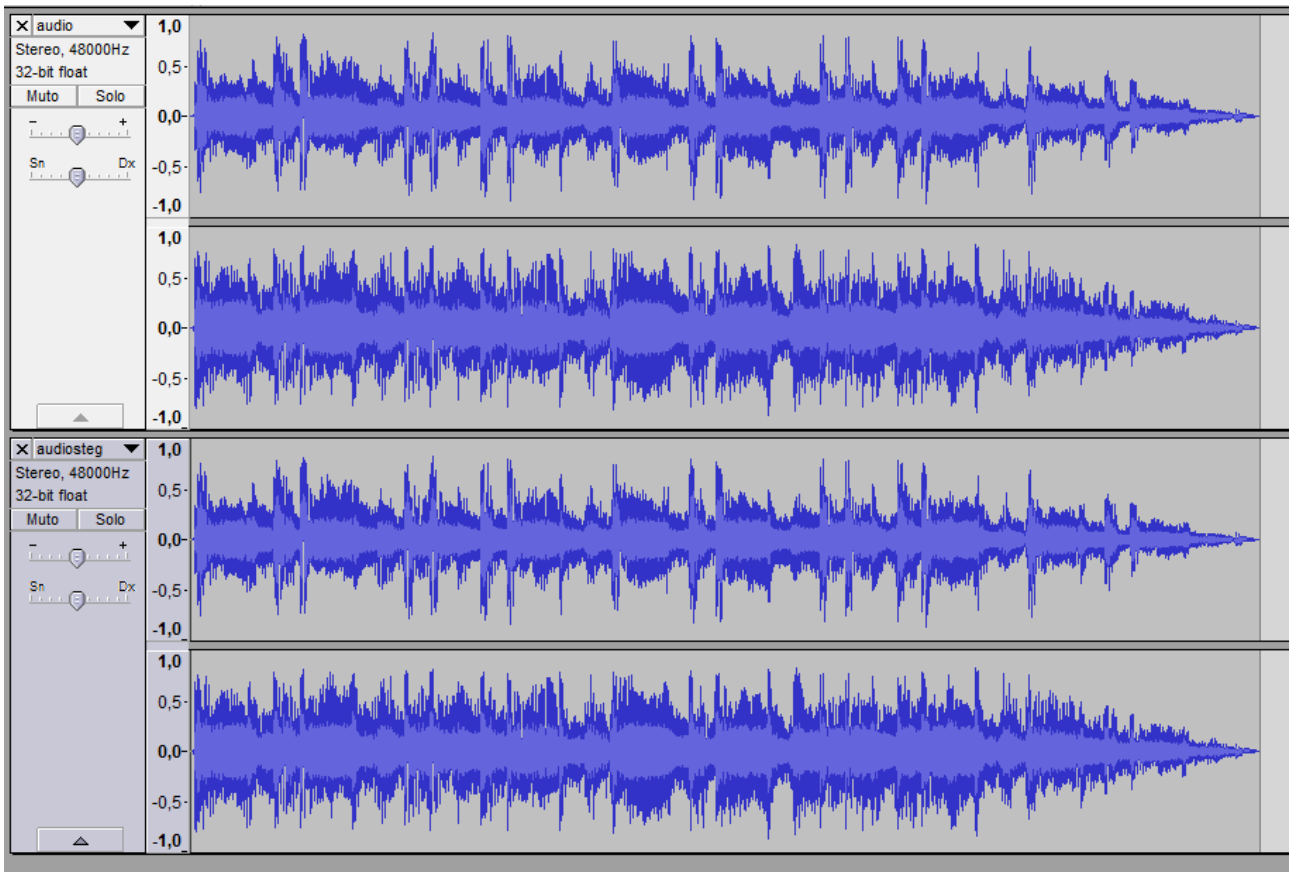


Figura 22: Confronto onde audio

Anche all'ascolto delle due tracce, come è possibile fare seguendo i link sotto, non è possibile apprezzare differenze.

Audio originale:
https://drive.google.com/file/d/1NetSBJOm8nOkFpLGSpl_wlN3IL4o_7Ws/view?usp=sharing

Audio post-embedding:

https://drive.google.com/file/d/12uMNNXH-ovyt27yfaMGnWFZTx4NdMmFr/view?usp=sharing

Video

Un file video è un file multimediale in senso stretto. Infatti, esso contiene il video (cioè una sequenza di immagini), audio (sia dialoghi che musica) e testo (sottotitoli, titoli di testa e di coda, scritte in sovraimpressione). Inoltre, i file video vengono compressi a causa delle grandi dimensioni, utilizzando numerosi schemi di codifica/decodifica (CODEC).

Sfruttando la poliedricità dei file video, è possibile inserire messaggi nascosti all'interno di una qualunque delle sue parti (flusso di immagini statiche, audio, testi), utilizzando gli efficienti algoritmi già noti per ciascuna parte. L'uso dei file video come cover consente di avere più bit a disposizione per il messaggio nascosto, e rende più complicato l'attacco: infatti il messaggio potrebbe essere nascosto in uno qualunque tra gli elementi del video, o addirittura suddiviso tra di essi.

Dei veri e propri tool Linux per la steganografia via video non esistono però come vedremo in seguito ve n'è uno su Windows che presenta caratteristiche molto interessanti.

Come si può immaginare quando si va a parlare di video tutto può farsi più complicato, sono moltissime le tecniche che sono state ideate a scopo scientifico. Molte di queste si basano sul Machine Learning e vengono usate reti neurali con reti pre-allenate ad hoc come la 3D-CNN, oppure non sono altro che una combinazione di LSB o Secured LSB con tecniche come quella del fuzzing. Altre volte ancora si va a giocare sulla ridondanza che si va a lavorare nelle tecniche di compressione dei file.

Tutte le strategie nominate fino a questo momento sono tutte di tipo sostitutive. Per la parte riguardante il video si vuole andare ad analizzare

invece una tecnica molto recente (2020) che invece non va a toccare alcun bit del file di cover (che a questo punto di cover non è ma che serve solamente come “indice” per decifrare il messaggio). Questa tecnica prende il nome appunto di “UN-Substituted Video Steganography”. L'abbreviazione è EMA - RKDD, dove EMA è l'algoritmo di Exact Matching Algorithm e RKDD è la Random Key - Dependent Data.

L'idea principale di EMA - RKDD è quella di trovare il testo segreto (messaggio) tra i valori RGB dei pixel dei fotogrammi video estratti. Generare quindi un Random Key - Dependent Data (RKDD) che viene utilizzato come parte del processo di recupero del testo segreto.

Il sistema è composto da due parti. Il sistema parte dal destinatario che genera una chiave pubblica utilizzando la crittografia a curva ellittica. Il mittente riceverà la chiave pubblica. La seconda parte è il sistema di trasmissione che prevede tre fasi principali.

La prima fase è la fase di inizializzazione e preelaborazione. L'utente seleziona i video e i dati segreti necessari. Il sistema inizia quindi con il processo di estrazione dei fotogrammi, poi converte il testo segreto nella sua rappresentazione in codice ASCII, quindi lo divide in più parti.

Tutto questo è fatto grazie ad uno Pseudo-Random Number Generator (PRNG) che, dopo che il video è stato splittato in frame, seleziona quali sono quelli che dovremo usare per la steganografia. La quantità invece di frame da considerare la sceglie il mittente inserendo un numero m come parametro.

Il messaggio invece viene diviso in chunk, ogni chunk viene assegnato ad un frame ed il numero di chunk è commisurato al numero di frame.

Il numero di chunk è deciso in base a questa formula:

$$\text{Number of Chunks} = \text{Total Numbers of Characters}/m\text{-}1$$

Number of Characters rappresenta il numero di caratteri del messaggio da nascondere. Un elemento del chunk non è altro che un carattere del messaggio.

La seconda fase si concentra sulla ricerca di corrispondenze esatte tra i valori RGB dei fotogrammi video selezionati e il testo segreto. Il risultato di questa

fase è un elenco di pixel corrispondenti, valori RGB e la posizione del testo segreto.

Nel dettaglio si vanno ad abbinare, tramite tecniche di randomizzazione che rendono il tutto più casuale, i chunk ai frame. Successivamente si cercano nella "traduzione" in ASII dell'RGB i caratteri, sempre in ASCII, presenti nel chunk e si va a salvare la coppia (x,y) che non sono altro che le coordinate sull'asse orizzontale e verticale del pixel dove si è trovata la corrispondenza. Da sottolineare che nella ricerca si parte prima dallo strato B (Blue) dell'RGB, poi si passa al G (Green) fino ad arrivare al R (Red).

Fatto ciò, non resta che creare la chiave di decriptazione di questa steganografia.

La struttura della chiave si presenta così:

X-axis		**Y-axis**		**Frame** (f)		**Character (c)**		(1.....n) similar characters at the same frame	**Next frame**	Same representation for the next matches
#of digits	x-axis value	#of digits	y-axis value	# of digits	Frame - NO.	#of digits	C1 location on text		00	

Figura 23: Struttura chiave

Come si può vedere per ogni carattere del messaggio vanno inserite nella chiave le coordinate di corrispondenza del frame, il numero del frame e la posizione di quel carattere nel messaggio.

La chiave, quindi, non è altro che la concatenazione di tutte queste informazioni.

Ne consegue però che la chiave ha una lunghezza notevole e servono delle strategie di compressione di essa.

Un primo passo è quello di evitare di inserire elementi ridondanti; quindi, se una lettera nel messaggio compare più volte verrà presa una sola corrispondenza con un unico frame, utilizzando una sola volta lo spazio dedicato alle coordinate e al numero del frame e verrà semplicemente aggiunta un'altra voce per quanto riguarda la posizione di quel carattere ne messaggio.

Oltre a questa accortezza possono essere usati i più noti algoritmi di ridondanza sui bit della chiave in sé.

Possiamo vedere dalle seguenti tabelle come l'ottimizzazione della chiave vada ad influire sulle performance di mappatura del messaggio all'interno del video.

Nella prima tabella ci troviamo nel test senza ottimizzazione e vediamo come per nascondere un messaggio di soli 10 bytes ci vogliano 14 secondi.

#	Secret Message Size (Bytes)	Time (Seconds)	Selected Video Name
1	10	14.36	MVI_3572
2	30	16.5	
3	50	17.30	
4	100	25.46	
5	500	31.6	
6	1000	35.23	
7	2000	38.8	
8	3000	43.78	

Mentre nella seconda tabella vediamo come questi tempi diminuiscano drasticamente.

#	Number of Char.	Time (Sec.)
1	15	0.85
2	25	0.66
3	45	0.13
4	85	0.171
5	105	0.191
6	154	0.134
7	254	0.173

In quest'ultima tabella vediamo invece quali sono i tempi di Decoding delle informazioni celate.

Secret Text Size (Char.)	Retrieve Time (Sec.)	Accuracy (%)
20	0.40	100%
30	0.41	100%
90	0.55	100%
130	0.58	100%
210	0.66	100%
300	0.75	100%
450	0.84	100%
600	1.1	100%
725	1.32	100%
829	1.45	100%
971	1.60	100%
4733	2.35	100%
7624	3.15	100%

In quest'ultimo grafico abbiamo invece una rappresentazione di come i metodi di compressione della chiave possano influire sulla sua grandezza al variare della grandezza del file segreto.

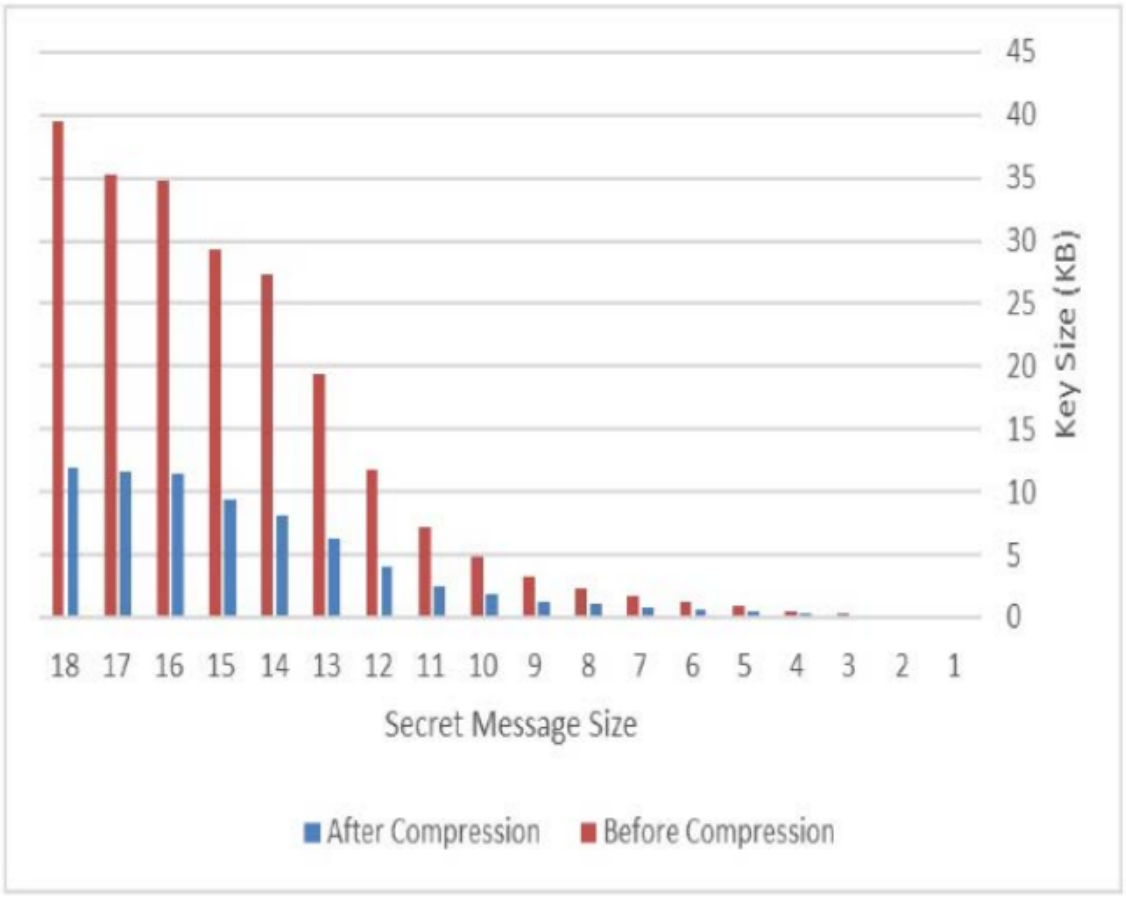


Figura 24: Confronto pre e post ottimizzazione

Fra le ultime considerazioni da fare vi è il fatto che in questa tipologia di steganografia la grandezza del file da nascondere può essere potenzialmente infinita e non è più correlata alla grandezza del file di cover, vi è solamente da valutare la sempre più crescente dimensione della chiave segreta.

Infine, si può dire che questa metodologia è altamente sicura in quanto:

- Non va a modificare il file cover
- La selezione randomica dei frame aumenta la complessità di criptaggio
- La divisione in chunk è un ennesimo elemento di randomicità
- Nella fase di matching, se si trovano dei match multipli, si sceglie un solo elemento da inserire nella chiave e questo è assolutamente casuale

Windows tool

Per completezza di trattazione facciamo qualche esempio anche per quanto riguarda invece il sistema operativo Windows.

Per quanto riguarda la trattazione delle immagini abbiamo si utilizza un programma di nome OpenStego con interfacce grafiche molto intuitive che possiamo vedere di seguito.

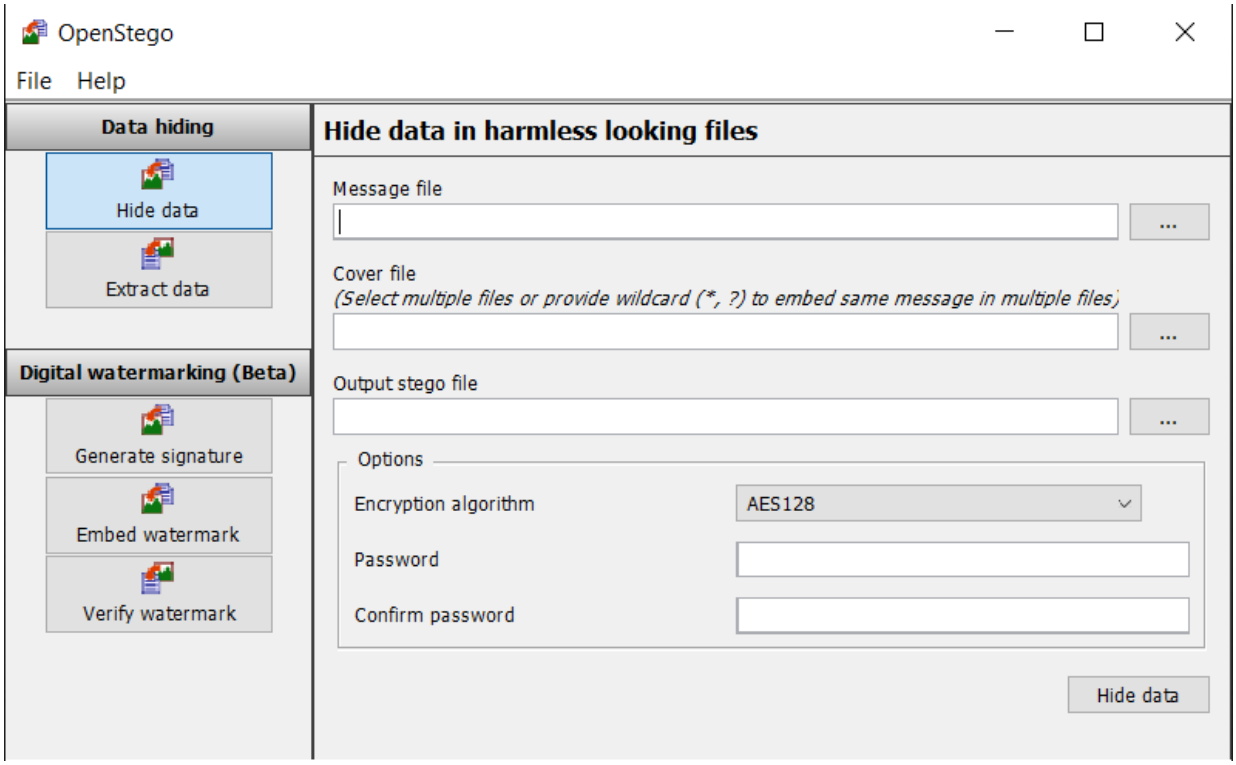


Figura 25: Fase di embedding

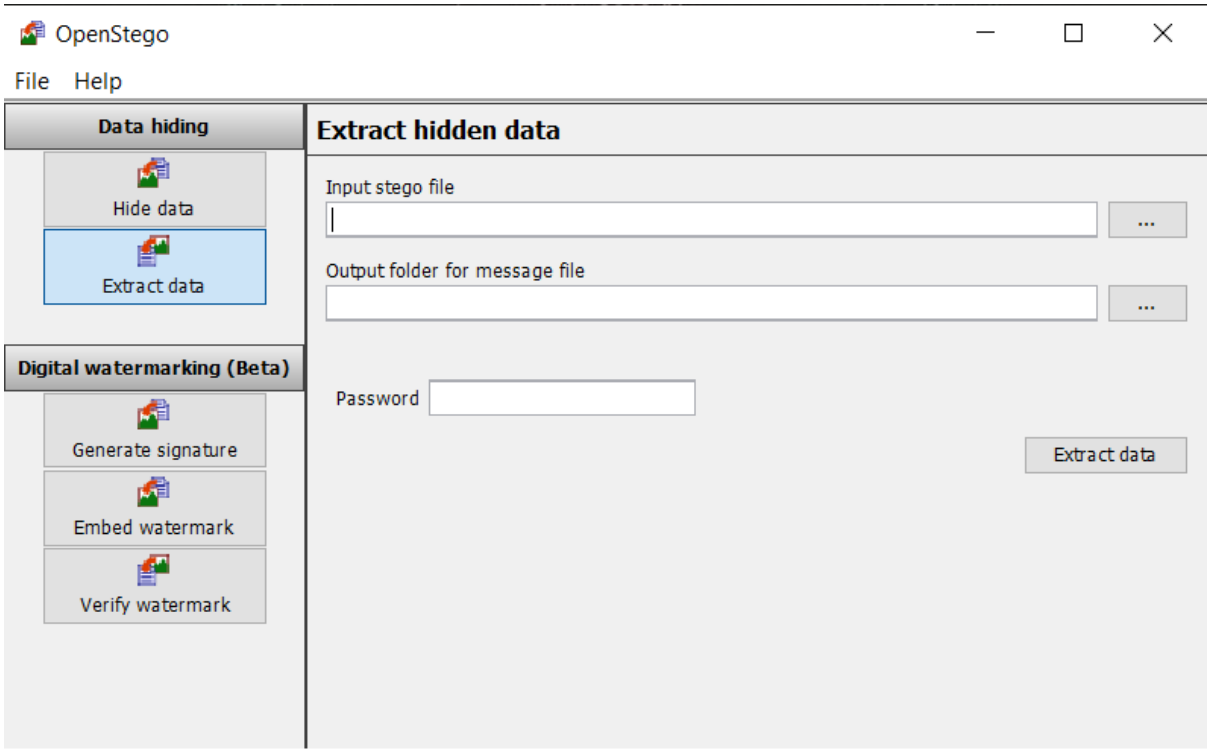


Figura 26: Fase di estrazione

Nulla da aggiungere sul funzionamento rispetto alle implementazioni dei tool Linux.

Per quanto riguarda invece i file audio abbiamo due tool che sono quelli preponderanti.
Uno di questi è **S-Tool**, facilmente reperibile in rete presso i più comuni siti di software freeware. L'uso del programma è infatti completamente gratuito e illimitato nel tempo. L'hiding dei file WAV, ad esempio, avviene tramite lo schema LSB. I Cover Bits vengono scelti in base alla passphrase che si utilizza; infatti, questo tool utilizza uno generatore di numeri pseudo-casuale per decidere il prossimo bit della cover da rimpiazzare. Alternativa a S-Tool è **Mp3Stego** che invece è un tool che effettua l'embedding sfruttando la tecnica dell'Echo-data-hiding. L'algoritmo che usa è di tipo lossy, infatti esso comprime i dati in input con perdita di informazioni fornendo un output molto simile ma non identico all'input. Il programma è nato per funzionare da linea di comando anche se poi si è sviluppata una interfaccia grafica che è ancora però da migliorare.

Di particolare interesse è invece lo strumento che si vuole analizzare per l'hiding tramite video ovvero **OpenPuff.**

OpenPuff è un software gratuito e open-source per nascondere messaggi e documenti nei video. Ha un'interfaccia utente molto ben organizzata e funziona abbastanza velocemente. Utilizzando OpenPuff, è possibile selezionare qualsiasi tipo di supporto come immagini, audio, flash e naturalmente video (3GP, MP4, MPG e VOB) per nascondere i propri dati. Si può però aggiungere solo un file segreto di dimensioni non superiori a 256 MB.

Una caratteristica molto interessante di questo software gratuito di steganografia video è che si può nascondere un documento in più file video. In questo caso, una parte del documento viene memorizzata in ciascun file video e per recuperare il file completo, l'utente deve disporre di tutti i file video. Si tratta quindi di una funzione piuttosto utile che protegge ulteriormente i file che si intende nascondere nei file video.

OpenPuff offre tecniche di crittografia con chiave simmetrica a 256+256 bit, scrambling dei dati con chiave simmetrica a 256 bit e whitening dei dati con chiave simmetrica a 256 bit.

Lo scrambling è il processo di offuscamento o rimozione dei dati sensibili. Questo processo è irreversibile e i dati originali non possono essere ricavati dai dati criptati mentre il whitening non è altro che la tecnica mediante la quale si cerca di incrementare la sicurezza di un cifrario a blocchi iterato attraverso passi successivi durante i quali vengono combinati i dati da cifrare con porzioni della chiave (in genere mediante semplici operazioni di XOR) prima del primo passaggio e dopo l'ultimo passaggio del processo di cifratura.

Altra funzione molto interessante di questo tool è la possibilità di fare Steganografia Negabile. Questa è una tecnica basata sull’uso di un’esca che permette di negare in maniera convincente di stare nascondendo dati sensibili, anche se gli attaccanti possono dimostrare che si sta nascondendo qualcosa. Basta semplicemente fornire un’esca sacrificabile che plausibilmente deve rimanere confidenziale. Verrà rivelata all’attaccante, sostenendo che questa è l’unico contenuto.
Un normale testo cifrato può essere decifrato ottenendo un unico testo in chiaro e l'utilizzatore può sostenere di non avere cifrato un messaggio differente. La crittografia negabile permette di cifrare il testo e di ottenere, se necessario, un testo in chiaro alternativo che ha il ruolo di esca. L'attaccante, anche se in grado di forzare l'utilizzatore a produrre un testo in chiaro, non può rilevare alcuna differenza fra il vero testo in chiaro e quello alternativo.

L'utilizzatore può sempre difendersi, scegliendo volta per volta se produrre, a partire dallo stesso testo cifrato, il vero testo in chiaro o quello di copertura.

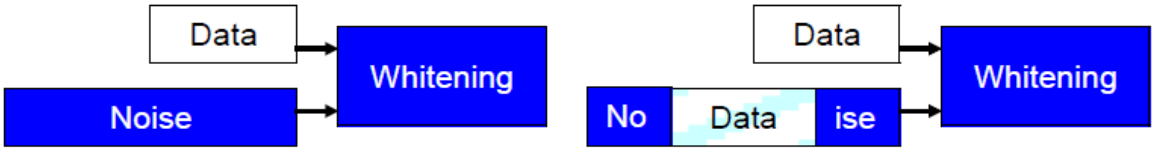


Figura 27: Steganografia negabile

Come possiamo vedere dalle immagini che seguono con questo tool è possibile creare tre password, due per la crittografia ed una per lo scrambling che devono essere lunghe almeno 8 caratteri (preferibilmente >16) e la distanza di Humming tra le tre deve essere almeno del 25% fra tutte le coppie di password.

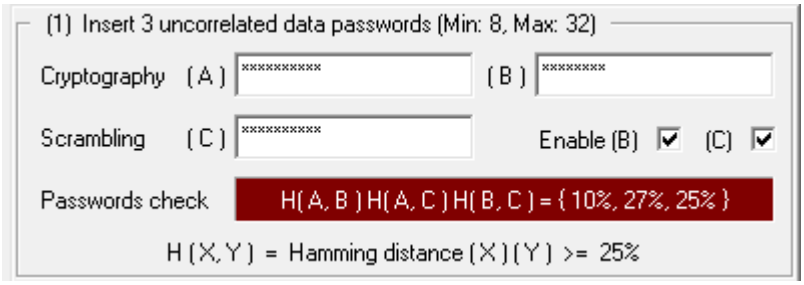


Figura 28: Distanza insufficiente

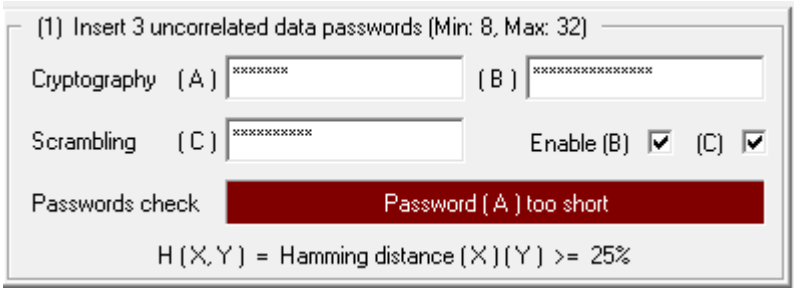


Figura 29: Password troppo corta

Le chiavi B e C possono essere tranquillamente disabilitate se non se ne necessita. Una volta impostate tutte le password si dovranno andare a selezionare i file che andranno a fare da cover ad i nostri dati.

Si va infine a selezionare il file da voler nascondere e selezionare quanti bit su otto dedicare ai dati e quanti al whitening seguendo la seguente tabella.

(Minimum)	1/8 dati, 7/8 whitening.
(Very Low)	1/7 dati, 6/7 whitening.
(Low)	1/6 dati, 5/6 whitening.
(Medium)	1/5 dati, 4/5 whitening.
(High)	1/4 dati, 3/4 whitening.
(Very High)	1/3 dati, 2/3 whitening.
(Maximum)	1/2 dati, 1/2 whitening.

Da come possiamo vedere nell'immagine sottostante nella sezione dei file di cover necessitiamo di soli 45 byte per poter nascondere il nostro file di testo di prova e dato che in input abbiamo dato tanti file .flv allora il programma stesso ne userà solo uno perché è sufficiente.

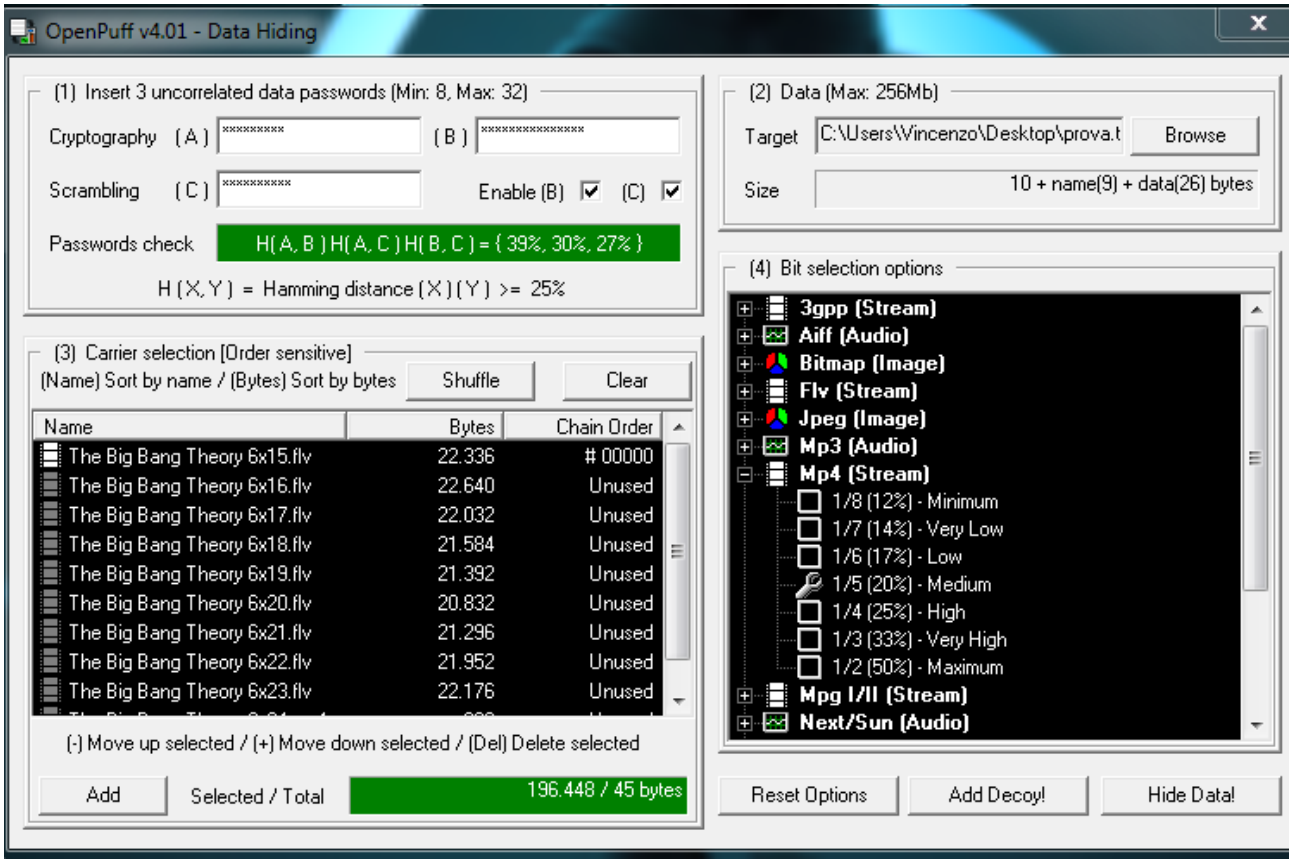


Figura 30: Password corrette e file cover in eccesso

In questo secondo caso invece il file da nascondere è un mp4 quindi i file di cover non bastano per poterlo nascondere tutto.

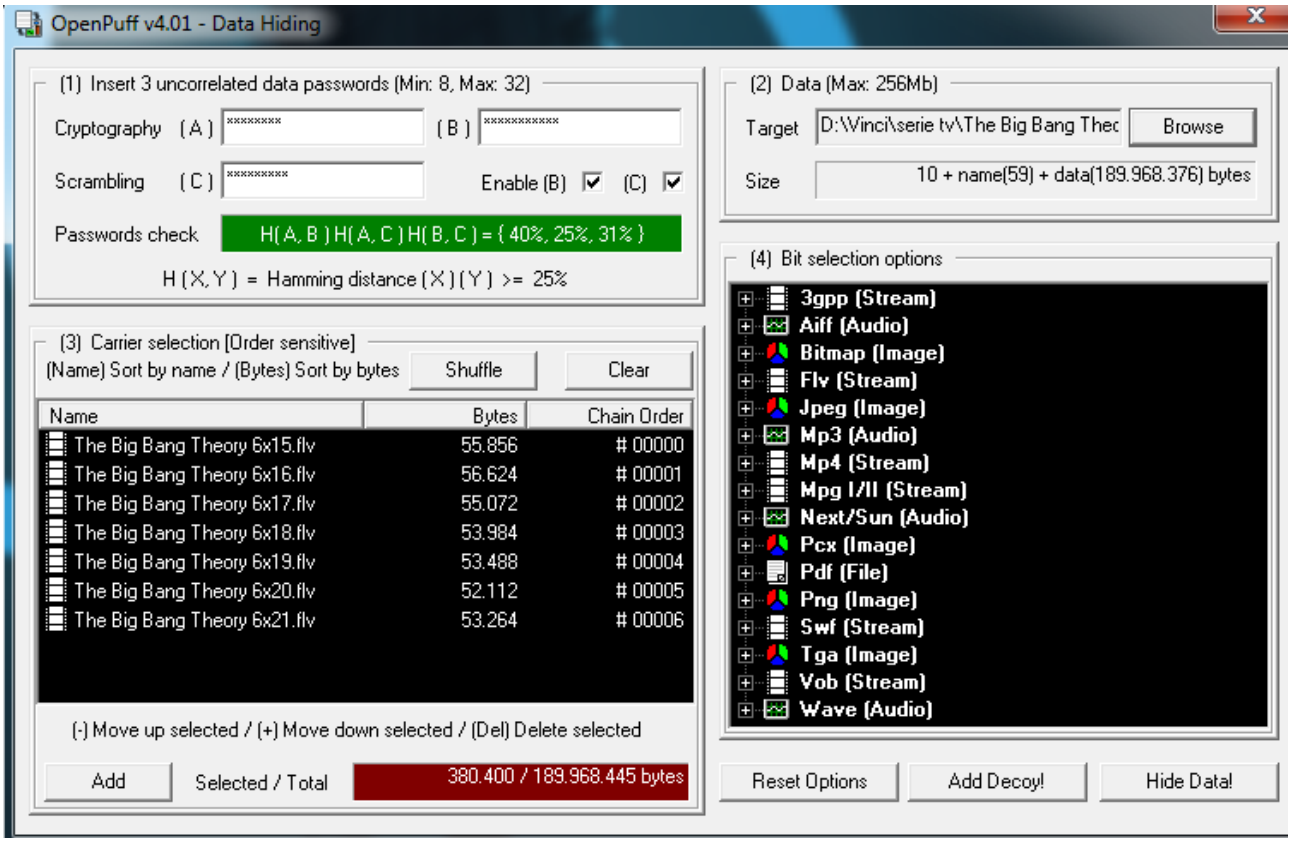


Figura 31: file di cover insufficienti