UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in
Sicurezza dei Sistemi e delle Reti Informatiche

Tecniche Avanzate di Crittografia Visuale e Steganografia per la Condivisione di Immagini Mediche

Tesi di:
Christian CODURI
Matricola: 964862

Relatore: Prof. Stelvio CIMATO

Anno Accademico 2022/2023

Dedica

"A me stesso e alla mia incessante sete di conoscenza.
Agli obiettivi falliti e raggiunti.
A mio padre e a mia madre, ai loro sacrifici e al loro supporto.
A mia nonna e alle sue torte deliziose.".

"Alla mia Maddy McGear e a tutti coloro che hanno creduto in me chiamandomi 'dottore' dal primo anno."

Indice

Introduzione

La protezione dei dati e della privacy delle persone rappresenta un aspetto di crescente rilevanza in tutti i settori in cui l'informatica viene applicata, compreso il campo della medicina.

Ospedali, farmacie, centri di cura e altre organizzazioni sanitarie sono sempre più nel mirino di cybercriminali. Ciò è dovuto principalmente al fatto che tali organizzazioni gestiscono grandi quantità di dati personali che possono avere un valore notevole per gruppi criminali.

Inoltre, gli istituti operanti nel campo della sanità spesso non possono permettersi di investire nelle più recenti e avanzate tecnologie di sicurezza, rendendole un bersaglio facile per ogni tipo di crimine informatico, dagli attacchi di phishing ai più sofisticati attacchi di ransomware.

A complicare ulteriormente questa situazione, il settore sanitario ha subito notevoli pressioni negli ultimi anni, dovendo affrontare sfide senza precedenti durante una pandemia mondiale. In questo contesto, i cybercriminali hanno cinicamente sfruttato la pandemia da COVID-19, agendo sia come attori individuali che come membri di gruppi malevoli sostenuti da nazioni.

Secondo il report "*Trends in Ransomware Attacks on US Hospitals, Clinics, and Other Health Care Delivery Organizations, 2016-2021*" [1], il numero degli attacchi annuali agli ospedali statunitensi dal 2016 al 2021 è raddoppiato. In tale finestra temporale, sono stati documentati 374 attacchi di notevoli dimensioni, effettuati mediante ransomware. A causa di questi attacchi, l'esposizione delle informazioni personali sanitarie (PHI), è passata da circa 1.3 milioni di persone nel 2016 ad oltre 16.5 milioni nel 2021.

In Europa e in Italia, la situazione non è migliore. Nella seconda metà del 2021, secondo alcune analisi, l'Italia è stata il terzo paese europeo più colpito da cyberattacchi nel settore sanitario.

Stando a quanto affermato dall'ENISA, ovvero l'Agenzia dell'Unione europea per la cibersicurezza, il settore sanitario europeo rappresenta il 53% degli incidenti totali. I ransomware sono emersi come una delle principali minacce in tale settore (54% degli incidenti). Tra gli obiettivi di tali attacchi, gli asset più presi di mira, sono risultati essere i dati dei pazienti e i dati delle strutture sanitarie stesse.

Obiettivo e risultati

Questa tesi non ha l'obiettivo di prevenire gli attacchi alle strutture sanitarie, ma vuole presentare un meccanismo innovativo, basato su tecniche di crittografia visuale e steganografia, che mira a garantire la protezione e la sicurezza delle immagini mediche nel formato standard DICOM. In questo modo, anche nel caso in cui si verificasse un attacco e tali file venissero estratti, questi sarebbero inutilizzabili dai cybercriminali.

Il tool sviluppato vuole quindi consentire la memorizzazione e la condivisione sicura di immagini DICOM, contribuendo così a preservare la riservatezza dei dati medici in forma visuale e a mitigare i rischi legati agli attacchi informatici nel settore sanitario.

L'utilizzo di questo meccanismo richiede un'architettura diversa rispetto a quella attualmente in uso. In breve, il processo prevede la suddivisione dell'immagine medica risultante da una diagnosi in due parti. Una di queste viene memorizzata nella struttura sanitaria, mentre l'altra viene consegnata al paziente. Entrambe le parti, se prese singolarmente, non permettono di ricavare alcuna informazione. Per poter effettivamente visualizzare e comprendere il referto, è necessario avere entrambe le parti a disposizione e combinarle insieme.

I risultati, che verranno esaminati nel capitolo conclusivo, sono indubbiamente positivi. L'obiettivo prefissato è stato raggiunto con successo, e il progetto costituisce una solida base per lo sviluppo di una tecnologia in grado di garantire la sicurezza delle immagini mediche. Tuttavia, è importante sottolineare che è richiesta una maggiore attenzione nella conservazione delle due parti di referto generate. La perdita di una delle due, che sia del paziente o dell'istituto sanitario, comporta l'impossibilità di ricostruire il referto.

Organizzazione della tesi

Capitolo 1

Il concetto fondamentale su cui si basa l'intera tesi è quello del Visual Secret Sharing (VSS), ovvero un'insieme di tecniche che permettono di condividere un segreto visuale tra più parti. Questo processo richiede la collaborazione di un determinato numero di parti coinvolte per poter ricostruire il segreto.

Il primo capitolo introdurrà il concetto di Crittografia Visuale (VC), che consiste in uno dei due possibili approcci per realizzare uno schema di VSS. Questa tecnica, introdotta da Naor e Shamir nel 1994, prevede la creazione di "shares" che possono essere stampati su carta trasparente e sovrapposti fisicamente per ricostruire il segreto senza alcun calcolo computazionale.

Nel medesimo capitolo, verrà presentata la versione base della crittografia visuale, che utilizza immagini in bianco e nero. In seguito, si esamineranno versioni più avanzate, come quelle che consentono agli share di non sembrare semplicemente rumore, o che lavorano su immagini in scala di grigi o a colori. In merito a queste ultime, si parlerà dei processi di halftoning e dithering sulle immagini e successivamente verranno descritti approcci che consentono di nascondere più immagini segrete all'interno dei due share.

Si concluderà il capitolo evidenziando le limitazioni associate all'uso di queste tecniche basate sulla stampa fisica degli share, che rendono tali approcci non adatti ad il nostro obiettivo.

Capitolo 2

Prima della crittografia visuale, nel 1987, è stata sviluppata un'altra tecnica di VSS da Kafri e Keren, chiamata "Random Grid" (RG). Questa tecnica, nonostante sia stata inventata prima della VC, è stata oggetto di studi solo in tempi più recenti.

Nel secondo capitolo, si scoprirà che RG riesce a superare le limitazioni della VC, ma, a differenza di quest'ultima, richiederà operazioni computazionali per cifrare e decifrare le immagini.

Si partirà analizzando il funzionamento dalla versione più elementare, che lavora su immagini binarie in bianco e nero. Si discuteranno in seguito le differenze tra l'utilizzo delle operazioni OR e XOR, per poi approfondire le tecniche basate sull'halftoning e sui bitplane sviluppate per la manipolazione di immagini in scala di grigi.

Poiché, per lo sviluppo del progetto, è stato necessario un algoritmo che consentisse una ricostruzione perfetta, ovvero dai due share fosse in grado di ricostruire esattamente l'immagine di partenza, l'autore ha creato una versione molto semplice di tale algoritmo, la quale verrà discussa alla fine del capitolo.

Capitolo 3

Avendo raggiunto l'obiettivo di cifratura utilizzando le random grid, nel terzo capitolo si cambia argomento e ci si concentra sulle tecniche di steganografia per immagini nel dominio spaziale.

La steganografia è un'antica arte di occultare informazioni all'interno di altri dati, con l'obiettivo di nascondere il fatto che una comunicazione segreta stia avvenendo. Nell'era moderna, la steganografia può assumere diverse forme, tra cui quella testuale, come spesso rappresentato nei film di spionaggio, quella visuale (su immagini o video), quella audio o anche sfruttando meccanismi più avanzati basati su protocolli di rete.

In questo capitolo, ci si concentrerà sulle tecniche di steganografia applicate alle immagini. In particolare, verranno illustrate tre tecniche: LSB (Least Significant Bit), BPCS (Bit-Plane Complexity Segmentation), e PVD (Pixel-Value Differencing). Tutte e tre consentono di nascondere il segreto desiderato direttamente nei valori dei pixel dell'immagine stessa, per questo vengono definite come tecniche che agiscono nel dominio spaziale dell'immagine.

Si procederà poi a parlare della steganografia reversibile, cioè una forma di steganografia che, durante la fase di decodifica, permette non solo di recuperare il segreto ma anche di ricostruire esattamente la stessa immagine di copertura utilizzata. Sarà presentato un algoritmo appartenente a questa categoria chiamato "difference expansion."

Il capitolo verrà concluso con alcune brevi considerazioni sulla steganalisi, che rappresenta lo studio di metodi per rompere gli schemi di steganografia.

Ai fini del progetto verrà utilizzata la steganografia più semplice, cioè LSB. Tale tecnica si basa sul nascondere parti di segreto nei soli bit meno significativi dei pixel che formano l'immagine. LSB, verrà resa reversibile grazie a una piccola modifica che verrà ampiamente discussa nel capitolo conclusivo dedicato al progetto.

Capitolo 4

Nel quarto ed ultimo capitolo di background, si passerà da da argomenti informatici ad un ambito più affine alla radiologia.

Saranno esplorate le diverse tipologie di esami diagnostici per immagini e si tratteranno i concetti di immagini volumetriche e voxel derivanti dalla grafia 3D. Successivamente, saranno analizzate le varie fasi di elaborazione delle immagini mediche, che includono filtraggi, normalizzazioni, segmentazioni, quantificazioni e visualizzazioni.

Grande attenzione verrà posta sullo standard di immagini mediche DICOM. Questo formato consente di archiviare non solo l'immagine stessa, cioè i valori dei pixel, ma anche altri dati correlati all'esame, al paziente, al medico e alla struttura sanitaria. Lo standard DICOM sarà esaminato sia dal punto di vista del formato del file sia dal punto di vista del protocollo utilizzato per la memorizzazione, il recupero e la gestione di queste informazioni.

Il capitolo si concluderà affrontando questioni legate alla sicurezza e alle potenziali minacce alla privacy connesse alle immagini mediche. Inoltre, saranno esaminati alcuni articoli del GDPR che stabiliscono normative specifiche per il settore medico.

I concetti relativi alla struttura dei file DICOM e quelli riguardanti la sicurezza e la privacy hanno fornito un prezioso contributo al progetto di questa tesi. Altri concetti più generici, come ad esempio la distinzione tra i vari tipi di esami, sono stati inclusi nel capitolo per rendere il contenuto accessibile anche ai lettori meno esperti in materia.

Capitolo 5

Si concluderà la tesi con il quinto capitolo, che rappresenta il risultato di questo lavoro, ovvero un progetto che combina alcune delle tecniche illustrate in precedenza per cifrare e successivamente decifrare i file DICOM.

Verrà presentata una nuova architettura che, dopo l'esame medico, dividerà il referto in due parti. La struttura sanitaria sarà tenuta a memorizzarne solo una, senza la necessità di archiviare l'intero file DICOM. L'altra parte sarà consegnata al paziente.

Si è menzionato in precedenza che un file DICOM contiene sia l'immagine sia informazioni aggiuntive. Il tool creato, nella fase di cifratura, partirà quindi da un file DICOM da cui estrarrà l'immagine principale. Successivamente, attraverso l'applicazione della steganografia, verranno nascoste informazioni all'interno di questa immagine. Verranno poi applicati i concetti di random grid per ottenere i due share. Nella fase di decifratura, invece, si partirà dalle due random grid per ricostruire l'immagine e recuperare le informazioni nascoste. Nel caso in cui siano attivate determinate opzioni, sarà possibile arrivare anche alla ricostruzione totale del file DICOM di partenza.

In questo modo, sia le persone autorizzate che quelle non autorizzate a visualizzare il referto dovranno necessariamente ottenere il consenso del paziente, rappresentato dallo share in suo possesso. Senza di esso, infatti, non saranno in grado di ricostruire né l'immagine né le informazioni contenute.

Oltre a fornire una introduzione teorica del progetto, verrà esaminata l'implementazione di esso attraverso l'utilizzo di una serie di diagrammi di attività. Si presenterà poi un caso di studio specifico, seguito da un'analisi sulla sicurezza e la privacy del programma.

Infine, la tesi concluderà con un'analisi della prima versione del progetto sviluppato e dei possibili sviluppi futuri.

Capitolo 1

Crittografia Visuale

1.1 Introduzione

Nel articolo scientifico di Weir e Yan [2] viene fatta una panoramica generale sulla Crittografia Visuale (VC) che viene definita come una potente tecnica che combina le nozioni di sicurezza perfetta, secret sharing e raster graphics. Il funzionamento della VC si basa sul prendere un'immagine, che rappresenta il segreto, e dividerla in due o più pezzi chiamati share. Quando gli share vengono stampati su carta lucida e sovrapposti, il segreto diventa visibile senza necessità di computazione infatti, come suggerisce la parola "visuale", è sufficiente il sistema visivo umano.

1.1.1 Secret Sharing

Nel 1979, Adi Shamir ha pubblicato un articolo intitolato "How to share a secret" [3], in cui si pone il seguente quesito:

> "Undici scienziati stanno lavorando ad un progetto segreto. Vogliono mettere al sicuro il documento in un armadio che può essere aperto se e solo se almeno sei di loro sono presenti. Quale è il numero minimo di lucchetti necessari? Quale è il numero minimo di chiavi che ogni scienziato deve avere?"

Nel paper di Sahmir, viene fornita una generalizzazione del problema di condivisione di segreti, attraverso la quale viene introdotto il concetto di schema (k, n).

Sia D il segreto da condividere tra n partecipanti. Un schema (k, n) è un modo di dividere D in n pezzi: $D_1, D_2, \ldots, D_n$, che soddisfano le seguenti condizioni:

1. La conoscenza di k o più pezzi rende D facilmente calcolabile
2. La conoscenza di $k-1$ o meno pezzi, rende D completamente indeterminabile.

Nel contesto considerato, la crittografia visuale rappresenta uno dei possibili schemi crittografici per risolvere il problema di Visual Secret Sharing, ovvero della condivisione segreta di immagini.

1.1.2 Sicurezza perfetta

La VC si basa sulla sovrapposizione di due o più immagini che, se osservate singolarmente, non rivelano alcuna informazione sul segreto. Di conseguenza, anche se un attaccante riuscisse ad ottenere uno o più share, non sarà in grado di recuperare alcuna informazione sul segreto originale, a meno di ottenere tutti i k share necessari.

In altre parole, la crittografia visuale offre quella che viene definita crittografia perfetta, ovvero una protezione completa da avversari dotati di illimitate risorse computazionali. Al contrario, gli schemi crittografici attualmente più utilizzati, sono considerati a sicurezza computazionale, ovvero violabili da attaccanti che hanno a disposizione "abbastanza" tempo (tanto da rendere praticamente trascurabile la probabilità di tale evento).

Ad oggi il Cifrario di Vernam (One Time Pad) rappresenta l'unico cifrario perfetto conosciuto. La caratteristica che gli permette di essere considerato tale, consiste nella scelta casuale di una chiave, lunga quanto il testo da cifrare. Di conseguenza, si dimostra matematicamente impenetrabile agli attacchi di crittanalisi (attacco forza bruta compreso) poiché il testo cifrato rispecchia la casualità della chiave utilizzata. Nell'ambito della crittografia visuale, l'immagine stessa diventa il "testo da cifrare", mentre l'insieme di share generato rappresenta la chiave. Questa analogia

tra il one-time padding e la crittografia visuale costituisce un ulteriore vantaggio della crittografia visuale rispetto ad altri algoritmi crittografici applicabili alle immagini.

1.2 Basic Visual Cryptography

Nella prima implementazione di Naor e Shamir, l'immagine segreta consisteva in un'insieme di pixel bianchi e neri che venivano trattati singolarmente. Il colore bianco, nelle stampe su lucido, era rappresentato come trasparente. Formalmente, partendo da un'immagine segreta bianca e nera, vengono generate n immagini (share), in modo tale che l'immagine originale sia visibile solamente se almeno k share sono sovrapposti.

1.2.1 (2,2)-VCS

Nel documento di Bhosale e Patil [4], è presentato un esempio di schema (2,2)-VCS che segue le regole stabilite da Naor e Shamir. Si tratta di uno schema di crittografia visuale nel quale con $k = n = 2$.

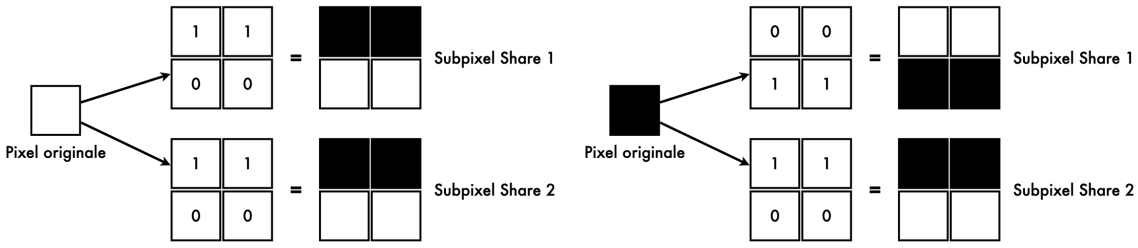
L'immagine iniziale viene sottoposta ad un processo di pixel expansion, mediante il quale ciascun pixel del segreto viene rappresentato con quattro subpixel in ogni share. Si fa notare che, sebbene sarebbero stati sufficienti due subpixel, ai fini di preservare l'aspect ratio dell'immagine originale se ne utilizzano quattro (disposti in righe da due).

Si definiscono le seguenti Matrici (1.1) che permettono di determinare il colore dei subpixel, presenti nei due share, in funzione del colore originale del pixel di partenza. In particolare, la matrice W viene utilizzata nel caso in cui il pixel di partenza sia bianco, mentre la matrice B nel caso in cui questo sia nero.

$$W = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{1.1}$$

Il procedimento adottato prevede l'utilizzo della prima riga, di una delle due matrici, per determinare il colore dei subpixel dello share uno, mentre la seconda riga viene utilizzata per determinare quelli dello share due. Ciascuna riga delle matrici è composta da quattro valori, i primi due consistono nei colori della prima riga di subpixel, mentre i restanti due valori si riferiscono ai colori dei subpixel della seconda riga. Inoltre, è importante precisare che nella crittografia visuale, il colore nero è rappresentato dal valore 1, mentre il colore bianco è rappresentato dal valore 0.

Le Figura (1.1) illustra un esempio di applicazione del processo di pixel expansion e di assegnamento dei colori ai subpixel di un (2,2)-VCS. In particolare, la Figura (1.1a) mostra il risultato del processo quando il pixel di partenza è bianco (applicata matrice W), mentre la Figura (1.1b) mostra il risultato nel caso in cui il pixel di partenza sia nero (applicata matrice B).



(a) Esempio con pixel di origine bianco (b) Esempio con pixel di origine nero

Figura 1.1

Durante la fase di generazione degli share, si effettua una permutazione casuale delle colonne delle matrici W e B per ogni pixel dell'immagine segreta. Al termine del processo, si otterranno due share aventi dimensioni doppie rispetto all'immagine di partenza a causa della pixel expansion.

Possiamo quindi definire:

- *C0* = {*tutte le matrici ottenute permutando le colonne di* W}
- *C1* = {*tutte le matrici ottenute permutando le colonne di* B}

È possibile osservare che, indipendentemente dalla permutazione delle colonne effettuata, le seguenti proprietà rimangono valide. Nel caso di un pixel di origine bianco, le due righe della

matrice W presenteranno gli stessi valori, implicando che i subpixel di entrambi gli share avranno lo stesso pattern. Al contrario, se il pixel di partenza è nero, le due righe della matrice B saranno complementari l'una rispetto all'altra, e di conseguenza i subpixel dei due share presenteranno pattern invertiti. In Figura (1.2) vengono mostrati tutti i possibili pattern che possono essere assunti da ciascuno share.

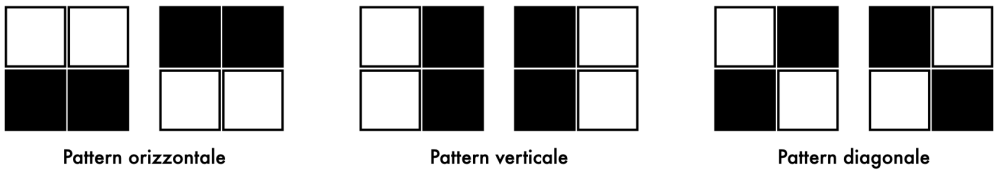
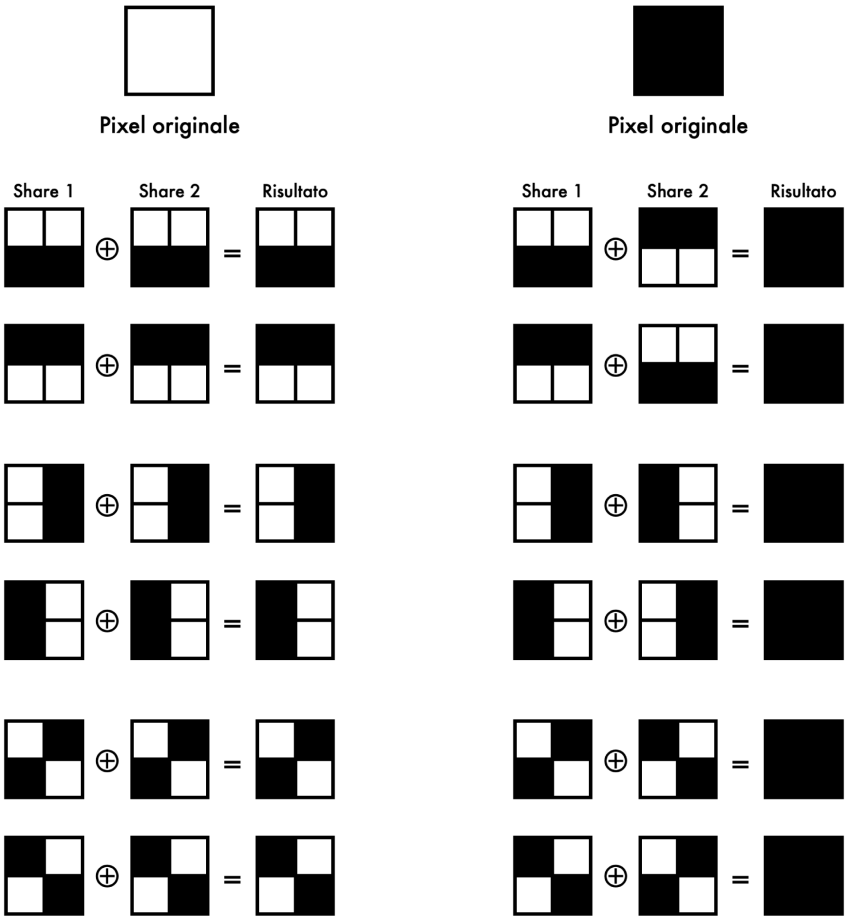


Figura 1.2: Pattern possibili

Si osservi che, poiché un pattern di subpixel di uno share non fornisce alcuna informazione sul colore del pixel di partenza, non è possibile ricostruire l'immagine segreta da un singolo share, indipendentemente dalla potenza computazionale a disposizione. Questa caratteristica è quella che permette di definire la crittografia visuale una cifratura perfetta.

Al termine della generazione degli share, è possibile ottenere l'immagine segreta applicando l'operazione logica OR tra i due share. La Figura (1.3) mostra tutte le possibili combinazioni di share e i relativi risultati.

A causa della pixel expansion, gli share generati sono di dimensioni doppie rispetto all'immagine di partenza. Di conseguenza, l'immagine ricostruita avrà anch'essa dimensioni doppie rispetto all'immagine originale.



(a) Combinazioni con pixel di partenza bianco

(b) Combinazioni con pixel di partenza nero

Figura 1.3: Possibili combinazioni di share e risultato

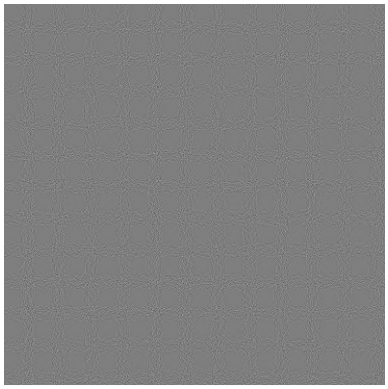
Come evidenziato nella Fig. (1.3), in ogni gruppo di subpixel il contrasto minimo sarà sempre del 50%. Tale effetto è causato dal fatto che, in ciascun gruppo, ci saranno sempre almeno due

subpixel neri, indipendentemente dal pixel di partenza. Se consideriamo un pixel nero nell'immagine segreta, durante il processo di sovrapposizione degli share, si otterranno quattro subpixel neri nell'immagine ricostruita. D'altra parte, se consideriamo un pixel bianco presente nell'immagine segreta, al momento della sovrapposizione, non si otterranno quattro subpixel bianchi nell'immagine ricostruita, ma due subpixel neri e due subpixel bianchi.

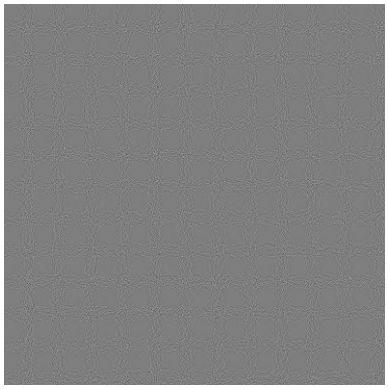
Questa caratteristica, oltre a garantire la sicurezza perfetta di cui si è parlato precedentemente, implica la natura lossy[1] della crittografia visuale. Tuttavia, va notato che la perdita di questa informazione non intralcia il sistema visivo umano utilizzato per riconoscere il segreto.

Di seguito viene mostrato un esempio di (2,2)-VCS ottenuto mediante l'esecuzione di uno script Python, consultabile nell'Appendice (A.1). L'immagine binaria[2] di partenza è mostrata in Figura (1.4a). Successivamente, vengono presentati i due share, ottenuti da tale immagine, nelle Figure (1.4b) e (1.4c). Infine, il risultato della sovrapposizione dei due share è mostrato in Figura (1.4d).

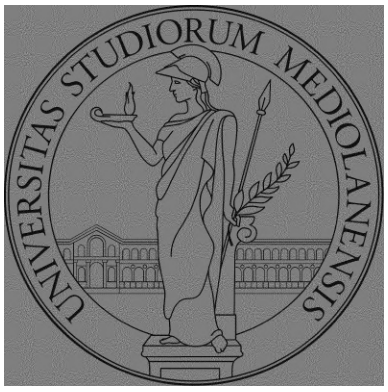
(a) Immagine segreta



(b) Trasparenza 1



(c) Trasparenza 2



(d) Immagine segreta ricostruita

Figura 1.4: Esempio di schema di Crittografia Visuale (2,2)

[1]Processo lossy: un processo caratterizzato dalla perdita di informazioni

[2]Immagine binaria: immagine che ha due possibili valori per ogni pixel, in questo caso 1 per nero e 0 per bianco.

L'utilizzo di un generico (k,n)-VCS presenta due problematiche principali. La prima riguarda l'incapacità di gestire strutture di accesso arbitrarie che consentono o vietano la visualizzazione del segreto a specifiche combinazioni di share. La seconda riguarda il fatto che gli share sono rappresentati come immagini prive di significato, che sembrano rumore. Questa caratteristica può facilmente far sorgere il sospetto di cifratura, anche agli occhi meno esperti.

1.3 Extended Visual Cryptography

Come indicato nel documento [5], la Crittografia Visuale Estesa (EVCS) cerca di risolvere i problemi della VC base. Per questo nella EVCS vengono introdotte delle strutture di accesso (T_{qual}, T_{forb}) e si fa in modo che, dati n partecipanti, ciascuno degli n share generati rappresenterà un'immagine con un significato proprio.

1.3.1 Strutture di Accesso

Nel documento "Visual cryptography for general access structures" [6] la Crittografia Visuale viene estesa ad un insieme $\mathcal{P}$ di n partecipanti, dove ogni membro riceve uno share dell'immagine segreta. Alcuni sottoinsiemi, definiti *qualified* di $\mathcal{P}$, sono autorizzati a ricostruire il segreto mediante sovrapposizione dei propri share, mentre altri sottoinsiemi, chiamati *forbidden*, non possono.

In altre parole, definiamo:

- *Qualified Set* un sottoinsieme p di $\mathcal{P}$ tale che sovrapponendo gli share dei partecipanti in p è possibile ricostruire l'immagine segreta;
- *Forbidden Set* un sottoinsieme f di $\mathcal{P}$ tale che sovrapponendo gli share dei partecipanti in f non è possibile ricostruire il segreto;

Definiamo inoltre:

- T_{qual} come l'insieme che contiene tutti i qualified set p per l'insieme $\mathcal{P}$;
- T_{forb} come l'insieme che contiene tutti i forbidden set f per l'insieme $\mathcal{P}$;

La coppia (T_{qual}, T_{forb}) è chiamata struttura di accesso.

Esempio 1.3.1. Consideriamo di avere un'insieme di partecipanti $\mathcal{P} = \{1,2,3,4\}$ e che i *qualified set* siano tutti i sottoinsiemi p contenenti almeno uno tra gli insiemi $\{1,2\}$, $\{2,3\}$ o $\{3,4\}$. Abbiamo quindi un $T_{qual} = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\}$. Mentre si deduce che i rimanenti sottoinsiemi di $\mathcal{P}$, ovvero $\{1,3\}$, $\{1,4\}$, $\{2,4\}$, siano *forbidden set* e quindi $T_{forb} = \{\{1,3\}, \{1,4\}, \{2,4\}\}$.

1.3.2 Share con significato

Per costruire un EVCS che funzioni è necessario considerare il colore del pixel nell'immagine segreta $(n + 1)$ che si vuole ottenere, in modo che quando gli n partecipanti sovrapporranno i loro share, l'immagine di ciascuno share sparisca al fine di visualizzare il segreto. In particolare C_0 e C_1 dello schema precedente sono estesi a $C_0^{p_1,\dots,p_n}$ e $C_1^{p_1,\dots,p_n}$, dove $p_i \in \{0,1\}, \forall i \in \{1,..,n\}$.

Per ottenere il pixel c dell'immagine segreta, n pixel $(p_1, \dots, p_n)$ vanno considerati, uno per ciascuna immagine. Pertanto le collezioni di matrici che devono essere generate sono $(C_0^{p_1,\dots,p_n}, C_1^{p_1,\dots,p_n})$, una per ogni combinazione di pixel bianchi e neri nelle n immagini. W e B sono rispettivamente le *matrici base* che contengono un insieme di share che definiscono i pixel bianchi e neri.

Definiamo quindi:

- $C_0^{p_1,\dots,p_n}$ come l'insieme delle matrici $n \times m$ ottenute permutando le colonne di $W^{p_1,\dots,p_n}$.
- $C_1^{p_1,\dots,p_n}$ come l'insieme delle matrici $n \times m$ ottenute permutando le colonne di $B^{p_1,\dots,p_n}$.

Al fine di implementare questo schema, sono necessarie 2^n paia di collezioni, una per ogni possibile combinazione di pixel bianchi/neri nelle n immagini.

Esempio 1.3.2. Il caso più semplice, che viene analizzato di seguito, è quello di un (2,2)-EVCS. Le collezioni $C_c^{p_1,p_2}$ dove $c, p_1, p_2 \in \{0,1\}$ (0 per il bianco, 1 per il nero), sono ottenute permutado le colonne delle seguenti matrici.

$$S_w^{ww} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad e \quad S_b^{ww} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$S_w^{wb} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad e \quad S_b^{wb} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

$$S_w^{bw} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad e \quad S_b^{bw} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$S_w^{bb} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad e \quad S_b^{bb} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

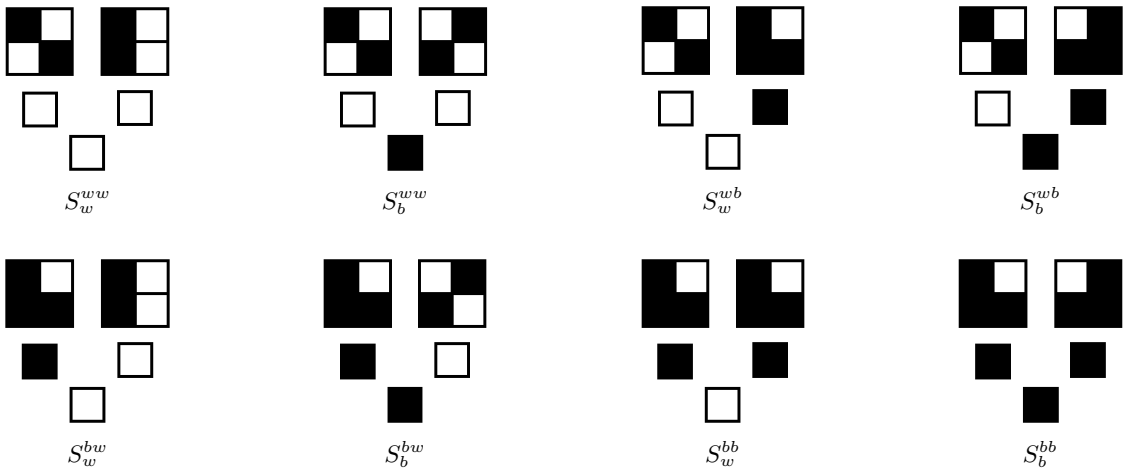
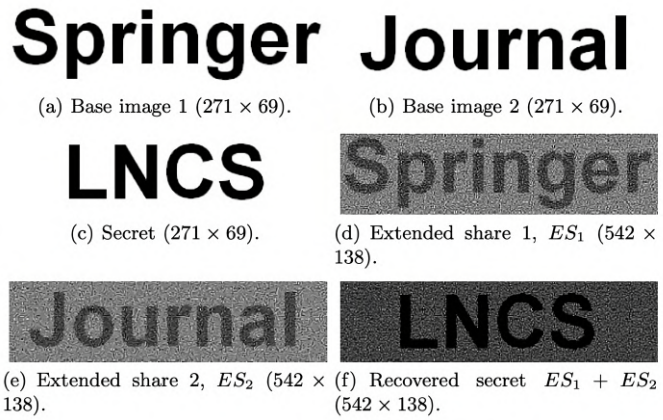


Figura 1.5: In ogni figura, la prima riga rappresenta gli share, la seconda rappresenta i pixel delle immagini di partenza, la terza rappresenta il pixel del segreto

Per un ragionamento analogo a quanto osservato nel caso precedente del (2,2)-VCS, si può facilmente dedurre che questa VC raggiunge un contrasto pari a $\frac{1}{4}$. Considerando i casi estremi, il caso S_b^{bb} produrrà quattro subpixel neri nella sovrapposizione degli share. D'altra parte, S_w^{ww}, che rappresenta il colore bianco in tutte e tre le immagini, avrà solamente un pixel su quattro bianco nella sovrapposizione degli share.



(a) Base image 1 (271 × 69). (b) Base image 2 (271 × 69).

(c) Secret (271 × 69). (d) Extended share 1, ES_1 (542 × 138).

(e) Extended share 2, ES_2 (542 × 138). (f) Recovered secret $ES_1 + ES_2$ (542 × 138).

Figura 1.6: Esempio di applicazione di EVCS [2]

La Figura (1.6) rappresenta un esempio di applicazione di EVCS. Abbiamo in ordine: le due "immagini di copertura" (1.6a) e (1.6b), l'immagine segreta (1.6c), i due share generati che hanno un significato proprio (1.6d) e (1.6e) e infine (1.6f) che rappresenta il messaggio ricostruito mediante la sovrapposizione dei due share.

Sulla base delle evidenze presentate, è possibile affermare che l'adozione di uno schema EVC offre numerosi vantaggi. Tale schema consente di implementare strutture di accesso e di occultare il rumore degli share mediante l'utilizzo di "immagini di copertura". Tuttavia, per progredire ulteriormente nella crittografia visuale, è necessario estenderne l'utilizzo ad immagini in scala di grigi o a colori, superando la limitazione dell'utilizzo esclusivo di immagini a pixel bianchi e neri come osservato finora.

1.4 Gray Scale Visual Cryptography

1.4.1 Immagini in scala di grigi

Le immagini in scala di grigi si basano sull'assegnazione di un valore numerico a ciascun pixel dell'immagine, che rappresenta l'intensità luminosa corrispondente. Nella maggior parte dei casi ogni pixel viene rappresentato su 8bit, dove quindi i valori possono variare da 0 (nero assoluto) a $2^8 - 1 = 255$ (bianco assoluto), consentendo una vasta gamma di sfumature di grigio tra i due estremi.

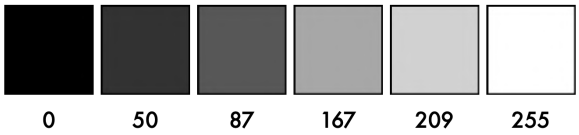


Figura 1.7: Esempio di sfumature di grigio rappresentabili in 8bit

L'utilizzo delle tonalità di grigio consente di catturare e rappresentare dettagli sottili e variazioni di luminosità all'interno dell'immagine, rendendo questa codifica adatta a molteplici applicazioni, come la fotografia, la grafica digitale e l'elaborazione delle immagini.

1.4.2 VC su immagini in scala di grigi

Gli articoli [7] e [8] propongono dei nuovi schemi di VC che consentono di utilizzare immagini in scala di grigi o a colori sfruttando la tecnica dell'halftoning.

L'Halftoning consiste in una tecnica di stampa o visualizzazione che permette di simulare un'immagine a toni continui mediante l'uso di punti che variano in dimensione e spaziatura. Nel contesto di un'immagine in scala di grigi, l'halftoning implica la rappresentazione dell'immagine come una serie di punti neri su uno sfondo bianco. La fedeltà all'immagine originale aumenta all'aumentare del numero di punti utilizzati. Questo effetto si basa sul fatto che il sistema visivo umano non è in grado di individuare numerosi punti di piccole dimensioni all'interno di uno spazio ristretto. Di conseguenza, visivamente, i punti tendono a simulare un tono di grigio, il quale varia in base alla densità dei punti. Tale concetto è ben illustrato in Figura (1.8).

(a) Immagine in scala di grigi

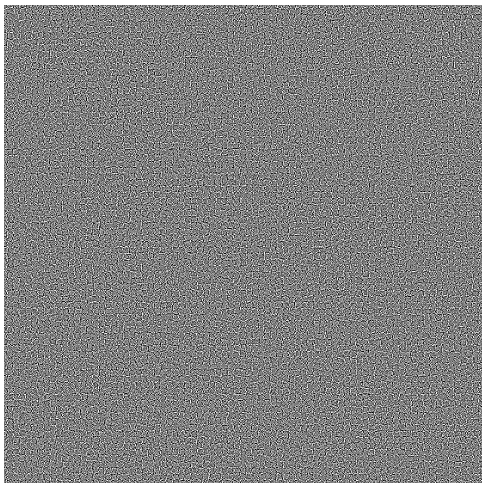
(b) Immagine halftoned

Figura 1.8: Immagine a toni continui (a), immagine halftoned formata da punti bianchi e neri (b)

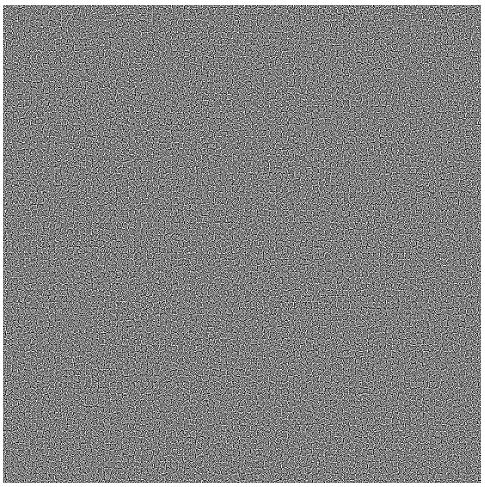
Dopo aver applicato l'algoritmo di halftoning all'immagine in scala di grigi, otterremo un'immagine composta da soli pixel bianchi e neri. Tale trasformazione, oltre a ridurre notevolmente la dimensione dell'immagine (da 8 o più bit per pixel a 1bit per pixel), consente l'applicazione di algoritmi standard di Crittografia Visuale precedentemente presentati.

La Figura (1.9), ottenuta dall'esecuzione di uno script Python consultabile nell'Appendice (A.1), mostra il risultato di questo processo.

(a) Immagine segreta halftoned



(b) Share 1



(c) Share 2

(d) Share 1 + Share 2

Figura 1.9: Esempio di VC in scala di grigio

1.4.3 Halftoning e dithering

Abbiamo definito l'halftoning come la tecnica che sfrutta punti e distanze per rappresentare un'immagine in scala di grigi in una in bianco e nero.

Il dithering è invece una forma specifica di halftoning che si differenzia in quanto effettua un processo di *error diffusion* in cui un certo errore di quantizzazione[3] viene diffuso nei pixel adiacenti a quello considerato. Questo contribuisce a una transizione più graduale delle tonalità dei colori nei pixel, migliorando la qualità dell'immagine risultante.

Tra le tecniche di dithering più diffuse vi è l'algoritmo di Floyd-Steinberg, di cui è riportato lo pseudocodice qui sotto (Algoritmo 1).

Algorithm 1 Algoritmo di Floyd-Steinberg per l'Halftoning di immagini

Per un'immagine in scala di grigi a 8-bit, il valore di grigio va da 0 (nero) a 255 (bianco).
Consideriamo: $b = 0$, $w = 255$, e $t = \text{int}[(b + w)/2] = 128$.
Assumiamo che g sia il valore di grigio dell'immagine in posizione $P(x, y)$, ed e sia l'errore di quantizzazione.

if $g > t$ **then**
    print(white);
    $e = g - w$;
else
    print(black);
    $e = g - b$;
end if

$7/16 \times e$ è aggiunto a $P(x + 1, y)$;
$3/16 \times e$ è aggiunto a $P(x - 1, y + 1)$;
$5/16 \times e$ è aggiunto a $P(x, y + 1)$;
$1/16 \times e$ è aggiunto a $P(x + 1, y + 1)$;

L'algoritmo in questione viene eseguito seguendo un approccio bottom-up, left-to-right, ed applicato su ciascun pixel dell'immagine. Questo ordine di elaborazione rappresenta una convenzione adottata nel contesto dell'elaborazioni delle immagini.

Esempio 1.4.1. Consideriamo un pixel di valore $g = 130$. Dato che l'intensità delle immagini in scala di grigi varia in modo continuo, si suppone che i pixel adiacenti, nella regione circostante, avranno valori vicini a 130.

In conformità all'algoritmo di Floyd-Steinberg, il pixel viene convertito in bianco poiché il suo valore, 130, supera la soglia di 128. È importante notare che il valore effettivo del bianco è 255, significativamente distante da 130 Successivamente, i valori dei pixel adiacenti vengono modificati per compensare questo errore introdotto.

Nel caso specifico, la differenza e tra il valore del pixel ($g = 130$) e il valore reale ($w = 255$) è di $e = 130 - 255 = -125$, e quindi verrà aggiunto al pixel successivo ($P(x + 1, y)$) un valore di -55, che è la differenza (e) moltiplicata per il coefficiente $7/16$ definito dell'algoritmo.

Data la continuità dei toni nell'immagine, è altamente probabile che l'aggiunta di -55 al valore del pixel $P(x + 1, y)$ faccia sì che questo scenda al di sotto della soglia di 128, risultando nella stampa in un pixel in nero. L'alternanza tra bianco e nero nei pixel adiacenti crea l'illusione di una gradazione di grigio.

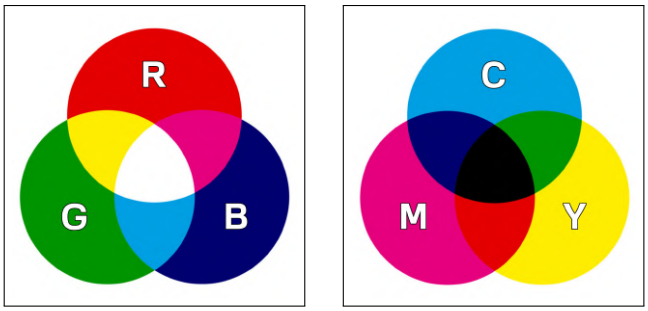
Esempio 1.4.2. Consideriamo un pixel con un valore di $g = 250$, che si avvicina molto al bianco ($w = 255$). In questo caso, l'algoritmo produrrà correttamente un pixel bianco nella nuova immagine, con un errore di approssimazione di -5. Questo avrà un impatto decisamente minore sui pixel adiacenti, rispetto a quello dell'esempio precedente.

[3]Errore di quantizzazione: si riferisce alla discrepanza tra un valore continuo e la sua approssimazione discreta. Nel contesto delle immagini, rappresenta la differenza tra i valori di colore originali e quelli approssimati.

1.5 Color Visual Cryptography

1.5.1 Immagini a colori

Sulla base di quanto affermato nel documento "Visual cryptography for color images" di Hou [8] viene fatta un'introduzione ai modelli comunemente usati per descrivere la composizione dei colori, ovvero i modelli additivi e sottrattivi.



(a) (b)

Figura 1.10: (a) Modello Additivo (RGB), e (b) Modello Sottrattivo (CMY)

Nel contesto del sistema additivo, i colori primari utilizzati sono rosso, verde e blu (RGB), colori differenti possono essere ottenuti combinando quantità diverse di questi tre. In questo modello, ciascun componente (R, G e B) è rappresentato da 8 bit, ovvero ognuno può assumere valori compresi tra 0 e 255. Pertanto, il numero totale di colori ottenibili è di $255^3 = 16.581.375$.

Mentre il modello additivo viene utilizzato su schermi e display, il modello sottrattivo viene utilizzato in stampanti, plotter e dispositivi simili. Quest'ultimo si basa sulla sovrapposizione di inchiostri ciano (C), magenta (M) e giallo (Y) in quantità diverse per creare colori colori differenti.

Nel sistema RGB, la somma dei valori massimi dei tre canali produce il colore bianco, mentre nel modello CMY, la somma dei valori massimi di ciano, magenta e giallo genera il colore nero. Questo è uno dei motivi che giustifica l'utilizzo del modello CMY anziché RGB nei processi di stampa, che utilizzano inchiostri applicati su fogli bianchi.

Relazione tra i modelli

Essendo (R, G, B) e (C, M, Y) colori complementari vale la seguente relazione:

$$(R, G, B) = (255, 255, 255) - (C, M, Y)$$

Infatti abbiamo che:

- Il valore (0,0,0) rappresenta il bianco in (C,M,Y) ed il nero in (R,G,B);
- Il valore (255,255,255) rappresenta il nero in (C,M,Y) ed il bianco in (R,G,B);

1.5.2 Processo di stampa di immagini a colori

La memorizzazione delle immagini avviene comunemente in codifica RGB. Nel caso in cui si desideri stampare un'immagine, è necessario eseguire un processo di decomposizione dei colori, che consiste nella separazione dei componenti ciano, magenta e giallo da ciascun pixel dell'immagine, generando così tre immagini monocromatiche. Tali immagini sono paragonabili a quelle in scala di grigi, ma saranno in "scala di ciano", "scala di magenta" e "scala di giallo".

Molte tipologie di stampanti possono controllare solo se un singolo pixel viene stampato oppure no, e non hanno controllo diretto sul tono di colore dell'immagine stampata. Per questo motivo si può applicare un algoritmo di halftoning (o dithering), su ciascuna immagine monocromatica, che deciderà se per un determinato pixel verrà rilasciato inchiostro o meno. Le tre immagini halftoned monocromatiche saranno infatti composte rispettivamente da pixel ciano-bianco, magenta-bianco e giallo-bianco.

Sovrapponendo queste tre immagini o, nel contesto della stampa, sovrapponendo gli strati di inchiostro corrispondenti, l'immagine originale sarà visibile.

Per completezza di informazioni, si noti che nelle stampanti la cartuccia di colore nero è presente al fine di ridurre i costi di stampa (il nero verrebbe ottenuto sovrapponendo l'inchiostro di tutti gli altri colori) e previene possibili problemi di "sbavature".

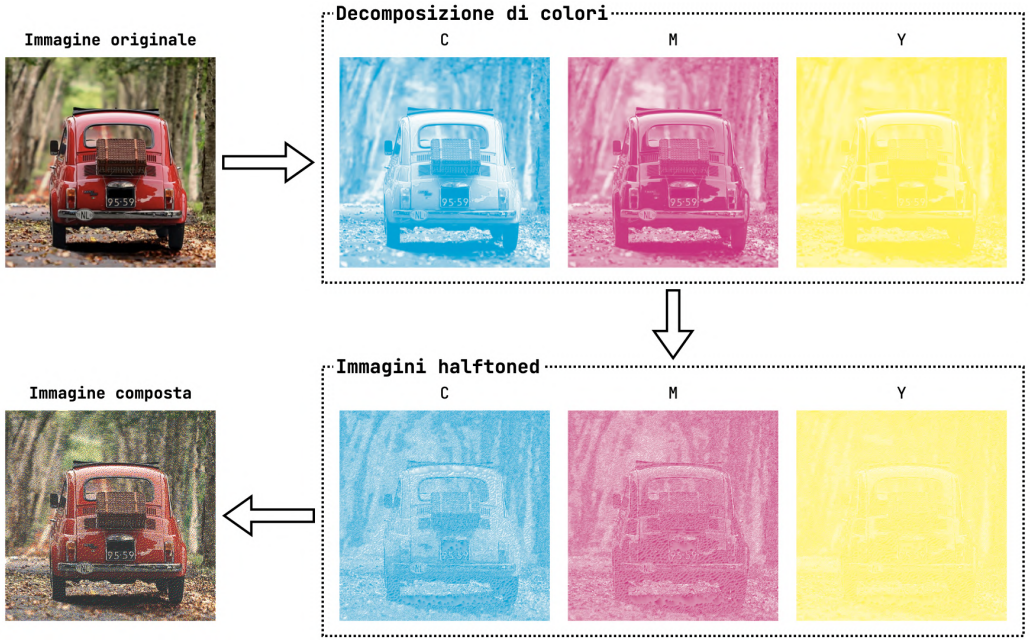


Figura 1.11: Processo di stampa delle immagini

Dalla Figura (1.11), che illustra l'intero processo di stampa descritto, si osserva che ogni pixel P_{ij} dell'immagine composta è ottenuto combinando i corrispettivi pixel C_{ij}, M_{ij} e Y_{ij} delle immagini halftoned. Queste immagini saranno, a loro volta, composte da pixel che avranno valore 0 a denotare bianco, o 255 a denotare il corrispondente colore. Per adottare una rappresentazione binaria con valori 0 e 1, è possibile sostituire il valore 1 al posto di 255. Di conseguenza ogni pixel P_{ij} avrà uno dei seguenti valori CMY: (0,0,0), (1,0,0), (0,1,0), (0,0,1),(1,1,0),(1,0,1),(0,1,1) o (1,1,1).

Le varie immagini in Fig. (1.11) sono il risultato dell'esecuzione di uno script Python consultabile nell'Appendice (A.2).

1.5.3 VC su immagini a colori: Metodo 1

In accordo con l'articolo [7], un metodo semplice che può essere sviluppato segue i primi due passi descritti in Figura (1.11), ovvero la decomposizione del colore e l'halftoning.

Successivamente, ogni immagine C, M ed Y può essere impiegata come immagine segreta nel processo di crittografia visuale standard. Adottando uno schema (2,2), saranno generati due share da ciascuna immagine, per un totale di 6 share.

Lo step finale consiste nel selezionare tre share di colori diversi, e combinarli tra loro per ottenere tre share colorati denominati S1, S2 e S3.

1.5.4 VC su immagini a colori: Metodo 2

Un altro approccio, suggerito dall'articolo [8], produce tre share (SC, SM ed SY) ed anche uno share nero (SK), partendo sempre delle immagini halftoned C, M ed Y. Di seguito si riportano i passaggi:

1. Trasforma l'immagine a colori in tre immagini monocromatiche halftone: C, M e Y;
2. Per ogni pixel $P_{i,j}$ dell'immagine composta P, con componenti colore ($C_{i,j}$, $M_{i,j}$, $Y_{i,j}$), esegui i seguenti passaggi:
   a Seleziona una maschera di subpixel nera (MK) di dimensione 2x2 in modo casuale, lasciando le altre posizioni vuote. I pattern tra cui si può scegliere sono stati presentati precedentemente in Figura (1.2);
   b Valuta se il valore ciano deve comparire o meno nel corrispettivo share (SC). Genera una maschera di subpixel vuota 2x2:
      - Se $C_{i,j} = 1$: il componente ciano sarà rivelato. Riempi, con il colore ciano la maschera creata, nelle posizioni corrispondenti ai pixel bianchi in MK. Lascia le altre posizioni vuote;

- Se $C_{i,j} = 0$: il componente ciano sarà nascosto. Riempi, con il colore ciano la maschera creata, nelle posizioni corrispondenti ai pixel neri in MK. Lascia le altre posizioni vuote;

c Aggiungi il blocco di subpixel alla posizione corrispondente nello Share 1 (SC);

d In modo analogo, valuta i blocchi di subpixel nello Share SM (magenta) con il valore di $M_{i,j}$ e nello Share SY (giallo) con il valore di $Y_{i,j}$;

e Aggiungi la maschera MK nello Share 4 (SK);

3. Ripeti il passaggio 2 per ogni pixel dell'immagine composta;

Si ottengono in questo modo quattro trasparenze (ciano, magenta, giallo e nero), ed è necessario sovrapporre le quattro immagine per poter ricostruire il segreto. Nel caso in cui si desideri creare uno schema diverso, ad esempio (2,2), è possibile combinare gli share a coppie, riducendo così il numero totale da quattro a due.

In Figura (1.12) si può vedere come vengono colorati i subpixel degli share SC, SM ed SY al variare del colore di $P_{i,j}$ nel caso in cui sia stata scelta come maschera di subpixel nera quella rappresentata.

Mask	Colore P_{ij}	Share 1 (SC)	Share 2 (SM)	Share 3 (SY)	Sovrapposizione
	(0,0,0)				
	(1,0,0)				
	(0,1,0)				
	(0,0,1)				
	(1,1,0)				
	(0,1,1)				
	(1,0,1)				
	(1,1,1)				

Figura 1.12: Pattern possibili al variare di $P_{i,j}$

1.5.5 Importanza della error diffusion

Al fine di sottolineare l'importanza della error diffusion (vedi Sezione 1.4.3), la Figura (1.13) presenta due immagini elaborate utilizzando l'algoritmo di Floyd-Steinberg, dove la prima ha subito la diffusione dell'errore prevista dall'algoritmo, mentre la seconda ha omesso questa fase.

(a) Con error diffusion

(b) Senza error diffusion

Figura 1.13: Confronto tra un'immagine halftoned con e senza error diffusion

1.6 Multiple Secret Sharing

Gli schemi precedenti sono tutti focalizzati sulla condivisione di un solo segreto. Una naturale evoluzione di questo è riuscire a nascondere più segreti senza aumentare il numero di share.

Tra le tecniche descritte nel documento [2] Wu e Chen propongono un metodo mediante il quale un segreto è ottenuto sovrapponendo lo share S1 e lo share S2, mentre l'altro segreto è ottenuto ruotando S1 di 90 gradi in senso antiorario e sovrapponendolo ad S2.

Un'altra opzione è quella di creare share dalla forma circolare in modo che non ci siano limitazioni sull'angolo di rotazione. In questo caso i segreti sono rivelati quando S2 viene ruotato di un certo angolo compreso tra 0 e 360 gradi.

Un ulteriore metodo suggerisce di rivelare segreti diversi ad ogni nuovo share sovrapposto.

Segreti intersecati e disgiunti con master key

Un altro approccio, illustrato nel paper [9], prevede l'utilizzo di una master key. I due segreti da nascondere vengono trasformati in share mediante l'utilizzo della VC e di una master key. Successivamente, i due share vengono combinati per formare un nuovo ed unico share $S1$, e viene modificata la master key per generare un key share $S2$. $S1$ ed $S2$ sono i due share effetivi che consentono di recuperare i segreti semplicemente spostando il key share S2 in posizioni diverse su S1. Il flowchart di questa tecnica è illustrato nella Figura (1.14).

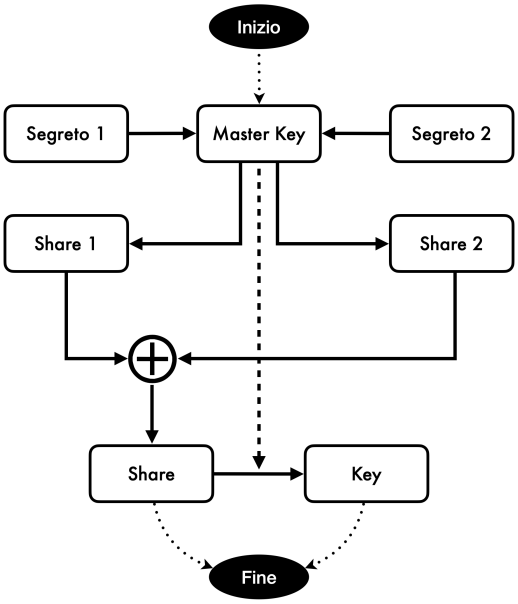


Figura 1.14: Flowchart di Multiple Secret Sharing

Il processo di combinazione dei due share può avvenire in due modalità distinte, a seconda che i segreti contenuti siano disgiunti o si intersechino.

Nella tecnica *disjoint VC*, i due (o più) segreti sono cifrati con la stessa master key e posizionati, all'interno dello share, uno vicino all'altro verticalmente, orizzontalmente o diagonalmente, senza che si sovrappongano.
Per rivelare i segreti, la chiave ($S2$) deve essere sovrapposta e spostata ad una distanza pari all'altezza o alla larghezza dello share $S1$, in modo da rendere visibile solo uno dei segreti alla volta.

La Figura (1.15) presenta un esempio di applicazione di questa tecnica. Le immagini (a) e (b) rappresentano le due immagini segrete, mentre (c) è lo share $S1$ e (d) è la chiave $S2$. Le immagini (e) ed (f) mostrano invece le sovrapposizioni della chiave sullo share in posizioni differenti.

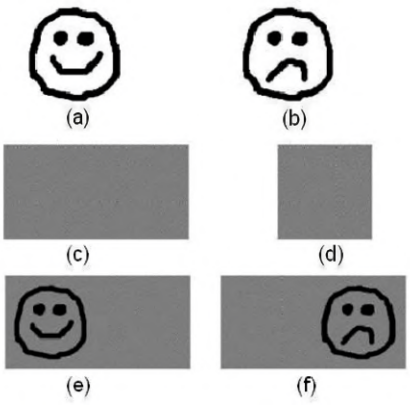


Figura 1.15: Segreti disgiunti [10]

Al contrario della precedente, nella tecnica *joint VC*, i due segreti sono sovrapposti in alcune parti, creando un'intersezione parziale tra loro. Questo rende il processo più complesso. L'unione dei due share, che devono essere della stessa dimensione, può essere fatta in più modi.

Tra le tecniche possibili vi è il meccanismo di *even-odd combination*, che consiste nel nascondere il primo segreto nelle righe pari e il secondo nelle righe dispari dello share $S1$. Attraverso questo processo si ottiene uno share di dimensioni doppie rispetto ai singoli share di partenza.

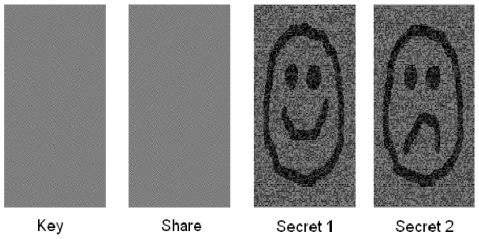


Figura 1.16: Segreti intersecati con tecnica di combinazione even-odd [10]

Poiché il segreto è stato precedentemente cifrato con la VC tradizionale che mappa un pixel in una matrice di subpixel 2x2, la suddivisione tra righe pari e dispari avviene con un gap di due pixel. Ad esempio le righe 0 e 1 saranno considerate righe pari e conterranno pixel relativi al segreto uno, le righe 2 e 3 conterranno invece pixel del segreto due e si procede in questo modo.

Per recuperare i segreti, la chiave $S2$ deve essere costruita in base alle righe pari e dispari della master key, in modo da ottenere una chiave di dimensioni pari a $S1$. Una volta ottenuta la chiave, è sufficiente sovrapporla allo share $S1$ e spostarla di due pixel per mostrare l'altro segreto.

È stato dimostrato che la condivisione di segreti multipli è praticabile, tuttavia bisogna considerare che la dimensione dello share aumenta proporzionalmente al numero di segreti che si nascondono.

È inoltre possibile combinare questa tecnica con l'utilizzo dell'halftoning, come descritto nelle sezioni precedenti, per condividere più segreti utilizzando immagini in scala di grigi o a colori.

1.7 Ambiti applicativi

I due principali ambiti applicativi in cui la crittografia visuale può essere utilizzata si dividono in:

- **Sistema di "difesa"**: l'applicazione più semplice che si può pensare per la VC consiste nel suo uso per cifrare le immagini in due parti. Una prima parte viene fornita direttamente alla persona interessata, mentre l'altra può essere memorizzata ed inviata o sfruttata in un secondo tempo. La decodifica dei dati può essere eseguita solo dalla persona che possiede il primo share una volta che riceve il secondo, garantendo così la sicurezza delle informazioni.
- **Watermarking**: al fine di garantire i diritti d'autore in modo discreto, una filigrana viene suddivisa in share tramite l'utilizzo della crittografia visuale. Successivamente, uno degli share viene posizionato o nascosto nel documento o all'interno di un'altra immagine, mentre l'altro share viene mantenuto segreto dal proprietario. Si fornisce in questo modo all'autore un mezzo per dimostrare la sua legittima proprietà in caso di violazione dei suoi diritti.

Esempio di sistema di difesa: ambito medico

Va notato che la crittografia visuale non rappresenta l'unica forma di cifratura applicabile alle immagini. Nel documento [11], viene esaminato l'utilizzo degli algoritmi di cifratura *AES (Rijndael)* e *RC6* per la protezione di alcune immagini mediche. L'impiego di tali algoritmi crittografici su queste immagini può svolgere un ruolo significativo nella protezione dei dati altamente sensibili dei pazienti.

Tuttavia, è importante sottolineare che gli algoritmi menzionati richiedono l'utilizzo e la memorizzazione di una chiave di cifratura. Pertanto, un'alternativa da considerare potrebbe essere lo sviluppo di un algoritmo di crittografia visuale che tratti immagini mediche e che sia in grado di ricostruire fedelmente l'immagine originale.

Un esempio concreto potrebbe essere l'acquisizione di un'immagine di una radiografia. Dopo l'acquisizione, l'immagine potrebbe essere divisa in due share, di cui uno viene immediatamente consegnato al paziente e l'altro viene conservato presso l'ospedale. Questo approccio garantirebbe che nessuna persona possa visualizzare il referto senza richiedere al paziente la sua porzione di immagine, assicurando così una maggiore sicurezza e riservatezza dei dati medici.

1.8 Stato dell'arte

In questo capitolo, è stato esaminato come affrontare il problema del *Visual Secret Sharing* utilizzando la Crittografia Visuale. Si è partiti dalla VC più semplice, su immagini a pixel bianchi e neri, per poi espandere gradualmente l'approccio ad immagini più complesse o a situazioni con segreti multipli.

Nonostante le evoluzioni descritte, rimangono tre problematiche principali: la pixel expansion, che va a raddoppiare le dimensioni dell'immagine segreta; il contrasto introdotto, che degrada la qualità dell'immagine una volta ricostruita; e la necessità di avere codebook diversi in base ai parametri k ed n (parametri diversi comportano la gestione di matrici di forma differente).

Ridurre le dimensioni

Per quanto riguarda la dimensione degli share, sono stati ottenuti notevoli risultati mediante l'utilizzo di approcci come la *Size Invariant Visual Cryptography*, che sfrutta vettori booleani per evitare la pixel expansion o la *Size Adjustable Scheme*, che consente agli utenti di specificare la dimensione degli share. Queste tecniche consentono di adattare gli share alle esigenze specifiche di un'eventuale applicazione che li utilizza.

Limitare la degradazione dell'immagine

L'utilizzo dell'operatore XOR al posto dell'OR, per ricostruire il segreto, permette di ottenere un'immagine totalmente fedele all'originale.

Sebbene tale schema possa risultare utile per specifici scopi, si discosta da ciò su cui si fonda la VC, ovvero l'impiego del sistema visivo umano. Infatti, con una crittografia visuale basata su XOR non è possibile stampare gli share su carta trasparente e sovrapporli per ricostruire il segreto, ma è necessaria computazione [2].

Anche se è possibile mitigare le problematiche, non è possibile eliminarle completamente.

È importante sottolineare che la Crittografia Visuale non rappresenta l'unico approccio per creare uno schema di Visual Secret Sharing (VSS). Nel prossimo capitolo, verrà esplorata un'alternativa conosciuta come Random Grid (RG).

Capitolo 2

Random Grid

Nel 1987, Kafri e Keren hanno ideato uno schema di *Visual Secret Sharing* (VSS) basato su Random Grid (RG), che si contrappone quello basato su Crittografia Visuale [12]. Sebbene RG e VC risolvano lo stesso problema, ovvero quello di condivisione di un'immagine segreta, questo nuovo approccio risolve le problematiche, discusse in precedenza, riguardanti la crittografia visuale.

Il concetto di Random Grid non è altro che una matrice di pixel, i cui valori vengono generati in modo casuale. In sostanza, la cifratura avviene attraverso la suddivisione dell'immagine in due random grid indipendenti, le quali non forniscono alcun dettaglio sull'immagine originale. La decifratura, invece, si realizza sovrapponendo tra loro le due griglie. Di conseguenza se non si possiedono entrambe l'operazione di decifratura non è possibile.

2.1 Basic VSS con Random Grid

Consideriamo un'immagine binaria di partenza A di dimensione $h \times w$, generiamo due random grid R_1 ed R_2, entrambe della stessa dimensione di A.

Si riporta di seguito l'algoritmo di uno dei tre possibili approcci tradizionali, ideati da Kafri e Keren, per implementare uno schema di questo tipo:

1. La random grid R_1 di dimensione $h \times w$ viene generata selezionando casualmente il valore 0 o 1 (bianco o nero) per ciascun pixel;

2. Successivamente, in base al valore del pixel $A[i, j]$ (dell'immagine segreta A) e del corrispondente pixel $R_1[i, j]$ (della prima random grid R_1), viene determinato il valore del pixel $R_2[i, j]$ (della seconda random grid R_2) come:

$$R_2[i,j] = \begin{cases} R_1[i,j], & \text{se } A[i,j] = 0 \\ \overline{R_1[i,j]} & \text{altrimenti} \end{cases} \tag{2.1}$$

   dove $[i, j]$ è la posizione del pixel nell'immagine ($i = 1, 2, ...h$ e $j = 1, 2, ...w$) e dove $\overline{R_1[i,j]}$ indica il complementare di $R_1[i, j]$;

3. Si ripete il Punto 2 finché non si esauriscono i pixel di A;

Se si considera un pixel di A con valore 1, i corrispondenti pixel di R_1 ed R_2 si complementeranno tra loro. Nel caso in cui abbia invece valore 0 i valori dei pixel di R_1 ed R_2 saranno uguali.

Gli altri due approcci si distinguono unicamente per l'equazione utilizzata nel calcolo dei valori dei pixel di R_2:

$$R_2[i,j] = \begin{cases} R_1[i,j], & \text{se } A[i,j] = 0 \\ \text{RANDOM\_PIXEL}(0,1) & \text{altrimenti} \end{cases} \tag{2.2}$$

$$R_2[i,j] = \begin{cases} \text{RANDOM\_PIXEL}(0,1), & \text{se } A[i,j] = 0 \\ \overline{R_1[i,j]} & \text{altrimenti} \end{cases} \tag{2.3}$$

Indipendentemente dall'algoritmo a cui si ricorre, i valori di una singola random grid non forniscono informazioni sui valori dell'altra.

Decifratura con OR o XOR

Come menzionato in precedenza, la decifratura, secondo Kafri e Keren, viene effettuata attraverso la sovrapposizione delle griglie. Da un punto di vista informatico, questo si traduce nell'operazione di OR, che viene tipicamente utilizzata anche nella crittografia visuale.

Nella sezione conclusiva del capitolo precedente, è stata accennata la potenzialità dell'operatore XOR, che consente, sia per la VC che per la RG, di ottenere una ricostruzione del segreto con una qualità superiore.

Pertanto, è opportuno effettuare un confronto tra l'utilizzo delle Random Grid con entrambi gli operatori. Le Figure (2.2c) e (2.2d) presentano un esempio di sovrapposizione con l'operatore OR e l'operatore XOR delle due Random Grid mostrate in Figura (2.2a) e (2.2b), le quali sono state ottenute a partire dall'immagine rappresentata in Figura (2.1).

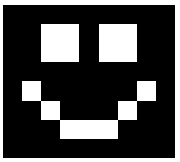


Figura 2.1: Immagine di partenza - Dimensioni: 9×8px

(a) R_1 (b) R_2 (c) R_1 OR R_2 (d) R_1 XOR R_2

Figura 2.2: VSS con Random Grid (OR e XOR) - Dimensioni: 9×8px

Si procede ora a spiegare la ragione per cui la ricostruzione ottenuta tramite l'operatore XOR risulta essere fedele all'originale, mentre quella ottenuta con l'operatore OR presenta delle "imperfezioni", attraverso un'analisi della Tabella (2.1).

A	B	A OR B	A XOR B
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	0

Tabella 2.1: Tabella di verità operatori OR e XOR

La differenza tra i due operatori si può vedere nell'ultima riga della tabella di verità, dove $1 \vee 1 = 1$, mentre $1 \bigoplus 1 = 0$.

Si supponga di avere un pixel di $A[i, j]$ con valore zero (bianco) e un pixel generato casualmente $R_1[i, j]$ con valore uno (nero). Seguendo l'Equazione (2.1), allora $R_2[i, j]$ sarà uguale ad $R_1[i, j]$, ovvero $R_2[i, j] = 1$.

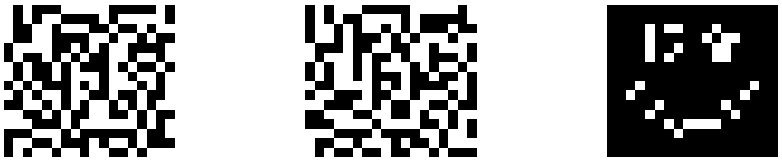
Il problema dell'operatore OR si presenta in questa casistica. Infatti, nella ricostruzione del segreto si avrà: $R_1[i, j] \vee R_2[i, j] = 1 \vee 1 = 1$ (nero), mentre il pixel corrispondente dell'immagine originale aveva valore 0 (bianco). A conferma di ciò, osservando la Figura (2.2c) si nota chiaramente che il problema dell'operatore OR riguarda la stampa dei pixel che dovrebbero essere bianchi, e non il viceversa.

Al contrario, l'operazione XOR produce: $R_1[i, j] \bigoplus R_2[i, j] = 1 \bigoplus 1 = 0$, stampando correttamente il pixel come bianco.

Nonostante l'operatore OR influisca negativamente sulla qualità del segreto, è spesso preferito perché consente la stampa delle random grid (o degli share se si considera la VC) su carta trasparente, permettendo la rivelazione del segreto mediante sovrapposizione. L'operatore XOR, invece, richiede calcoli per ricostruire il segreto e non può fare affidamento esclusivamente sul sistema visivo umano come accade per l'OR.

Confronto con la Crittografia Visuale

Per completare il confronto tra Random Grid e Crittografia Visuale applicate ad immagini binarie, la Figura (2.3) mostra la cifratura dell'immagine rappresentata in Figura (2.1) utilizzando la CV.



(a) Share 1 (b) Share 2 (c) S_1 OR S_2

Figura 2.3: VSS con Crittografia Visuale - Dimensioni: 18×16px

Le differenze tra le Figure (2.2) e (2.3) evidenziano come l'utilizzo della tecnica Random Grid abbia risolto le principali problematiche della VC: non esistono le matrici di base per la cifratura e le dimensioni degli share rimangono invariate rispetto all'immagine di partenza.

Invece, per quanto riguarda la qualità del segreto ricostruito, in entrambe le tecniche, è possibile ricorrere allo XOR per ottenere una ricostruzione del segreto di migliore qualità.

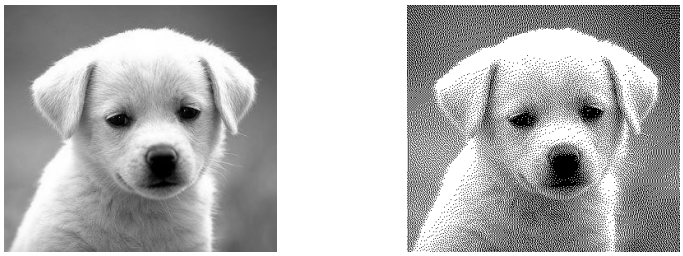
Gli script utilizzati per generare le Figure (2.2) e (2.3) sono reperibili nell'Appendice (A.1) e (B).

2.2 Gray Scale VSS con Random Grid

2.2.1 Tecnica dell'halftoning

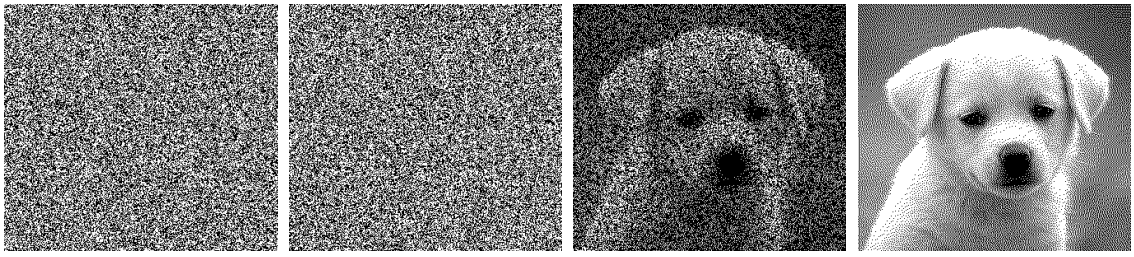
Per le immagini in scala di grigi, Shyong Jian Shyu propone di trasformare l'immagine sorgente in una in bianco e nero mediante l'utilizzo di una tecnica di halftoning con diffusione dell'errore. Una volta ottenuta un'immagine binaria, è possibile applicare l'algoritmo di Kafri e Keren precedentemente descritto.

Al fine di migliorare ulteriormente il risultato, Kumar e Sharma propongono di sfruttare lo XOR anziché l'OR per la ricostruzione del segreto [13].



(a) Scala di grigi (b) Halftoned

Figura 2.4: Immagini di partenza - Dimensioni: 332×300px



(a) Random Grid 1 (b) Random Grid 2 (c) Overlap (OR) (d) Overlap (XOR)

Figura 2.5: VSS con Random Grid - Dimensioni: 332×300px

Partendo dall'immagine in scala di grigi di Figura (2.4a), si è applicato l'algoritmo di dithering di Floyd-Steinberg per ottenere l'immagine binaria in Figura (2.4b). Questa viene data come input al processo di VSS con Random Grid descritto precedentemente.

Per quanto visto nel capitolo sulla Crittografia Visuale, è facile intuire che grazie all'halftoning e ad alcune accortezze si può sfruttare questo approccio anche per immagini a colori.

È utile notare che, come ci si aspetterebbe, l'immagine ricostruita con lo XOR è identica a quella halftoned fornita come input all'algoritmo.

Il punto debole di questa tecnica è il processo di halftoning stesso, in quanto, a causa di esso, alcuni dettagli dell'immagine originale verranno persi. Pertanto, la fedeltà dell'immagine ricostruita rispetto all'originale è maggiormente influenzata dalla tecnica di halftoning piuttosto che dall'impiego delle Random Grid per la cifratura.

2.2.2 Tecnica bit-planes

Per diminuire ulteriormente la degradazione dell'immagine dovuta al halftoning, nel documento [13], viene illustrato un algoritmo che si basa sul concetto di *bit planes*.

Bit planes

Il bitplane di un segnale discreto, è un insieme di bit corrispondenti ad una determinata posizione in ciascuno dei numeri binari che rappresentano il segnale.

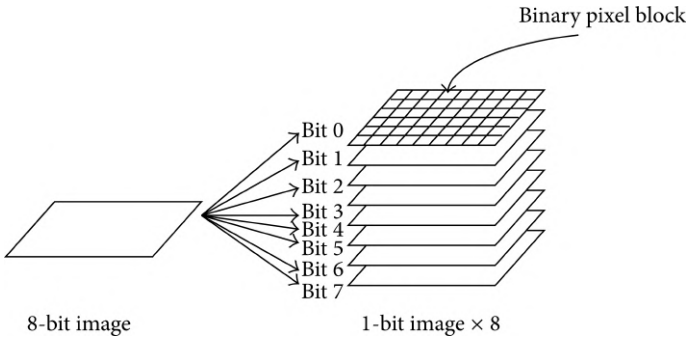


Figura 2.6: Concetto di bit-planes [14]

Ad esempio, per rappresentare un'immagine in scala di grigi a 8 bit si usano 8 bitplane, dove il primo contiene i bit più significativi di tutti i pixel, mentre l'ottavo contiene i bit meno significativi di tutti i pixel. Una schematizzazione di questo concetto è visibile in Figura (2.6).

Un discorso analogo vale per tutte le rappresentazioni, anche non ad 8 bit.

Ogni layer può essere considerato come un'immagine binaria indipendente. L'immagine derivante dal bitplane costituito dai bit più significativi sarà quella che maggiormente si avvicina all'originale. Man mano che si procede verso i bit meno significativi, l'immagine risultante dal singolo bitplane diventerà sempre più simile a rumore.

La Figura (2.7) mostra un'immagine a 8 bit insieme alle rispettive immagini dei bitplane.

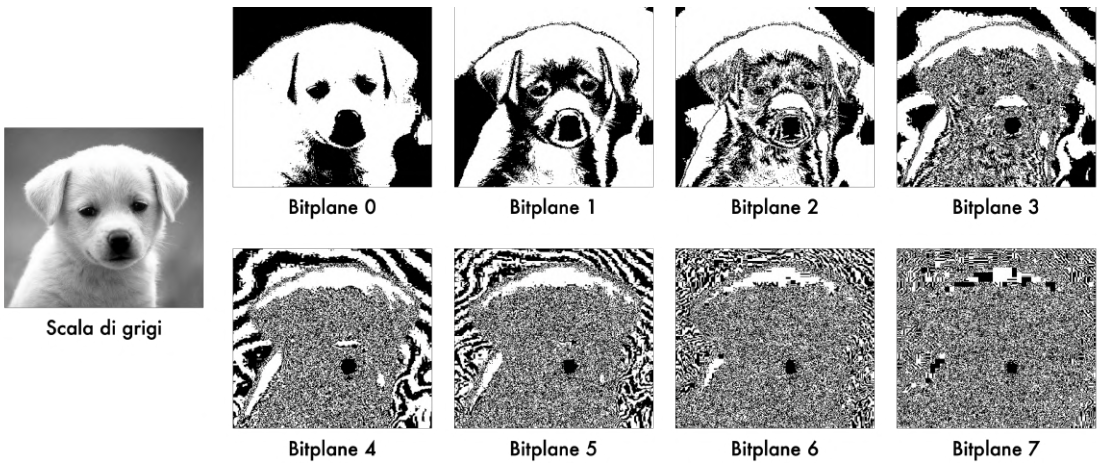


Figura 2.7: Decomposizione in bitplane di un'immagine ad 8bit

Come detto, il primo bitplane, preso singolarmente, è un'approssimazione grezza dell'immagine. Man mano che si sommano bit planes meno significativi, si può notare che il contributo aggiunto

diventa sempre più piccolo e trascurabile rispetto alla somma dei bit planes più significativi. Questa osservazione è verificabile in Figura (2.8), dove il processo inizia con il bitplane più significativo e procede sommando un numero crescente di bitplanes.

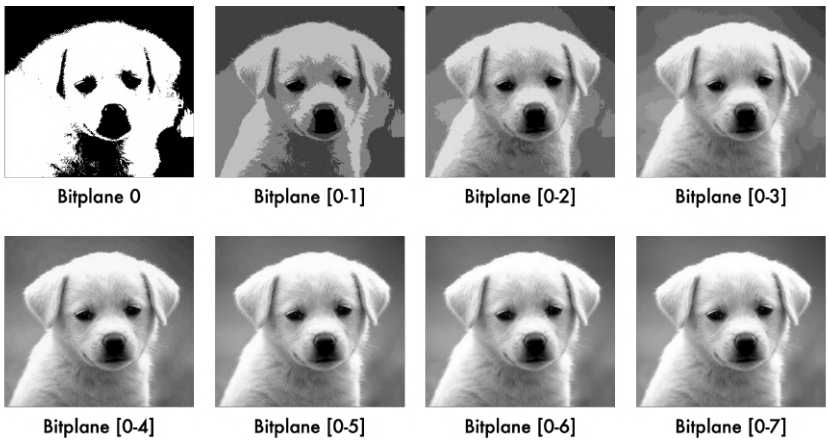


Figura 2.8: Risultato dell'unione di più bitplane

Se si considera un bit dell'n-esimo bitplane di un'immagine composta da m bitplane, se tale bit ha valore 1, contribuirà con un valore pari a $2^{(m-n)}$; in caso contrario, non avrà alcun effetto. Pertanto, un bitplane può fornire la metà del contributo del bitplane precedente [15].

Questo equivale a dire che se prendiamo in considerazione un bit in posizione n di un certo pixel in un'immagine in scala di grigi con m bit, e se tale bit ha valore 1, allora contribuirà, rispetto agli altri bit di tale pixel, con un peso pari a $2^{(m-n)}$.

Esempio 2.2.1. Si esamini il caso di un pixel di un'immagine in scala di grigi a 8 bit che ha valore 11010001 (ovvero 209 in decimale).

La Tabella (2.2) riporta tutti i calcoli necessari per capire i contributi delle singole bitplane.

Bitplane	**Valore**	**Contributo**	**Totale parziale**
Primo	1	$1 \times 2^7 = 128$	128
Secondo	1	$1 \times 2^6 = 64$	192
Terzo	0	$0 \times 2^5 = 0$	192
Quarto	1	$1 \times 2^4 = 16$	208
Quinto	0	$0 \times 2^3 = 0$	208
Sesto	0	$0 \times 2^2 = 0$	208
Settimo	0	$0 \times 2^1 = 0$	208
Ottavo	1	$1 \times 2^0 = 1$	209

Tabella 2.2: Contributo di ciascun bitplane

In Figura (2.9), è illustrato il modo con cui il colore viene "distribuito", permettendo di comprendere se il pixel assumerà il colore bianco o nero negli otto bitplane.

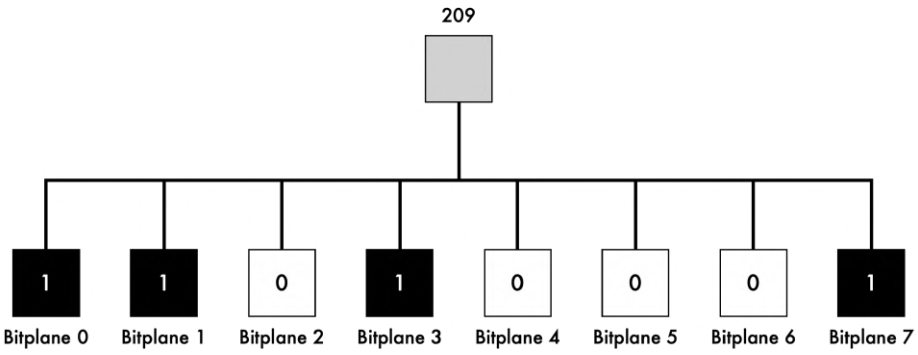


Figura 2.9: "Distribuzione" del colore nei bit plane

Cifratura

Tornando all'applicazione di questo concetto nell'algoritmo Random Grid, il processo di cifratura è composto da quattro fasi.

La prima fase è la decomposizione, in cui l'immagine in scala di grigi, supponiamo a 8 bit, viene suddivisa nei suoi otto bit plane. Nella seconda fase avviene la selezione delle tre bit plane più significativi, ovvero quelli che contengono i dettagli più rilevanti. Nel caso di immagini ad 8bit verranno scelti i bitplane 0, 1 e 2. La terza fase è l'encoding, in cui i tre bit plane selezionati vengono cifrati individualmente utilizzando l'algoritmo Random Grid di Kafri e Keren, ottenendo sei random grid: R_{01}, R_{02}, R_{11}, R_{12}, R_{21} e R_{22}.

Nell'ultima fase, avviene la combinazione delle sei random grid generate nella fase tre, al fine di ottenere due random grid in scala di grigi. R_1 è composta dalla combinazione di R_{01}, R_{11} e R_{21}, mentre R_2 da R_{02}, R_{12} e R_{22}.

Decifratura

La decifratura segue il processo inverso alle fasi di cifratura. Le due random grid R_1 ed R_2 vengono decomposte nuovamente nelle sei random grid sopracitate. Successivamente, l'operatore XOR viene applicato alle coppie (R_{01}, R_{02}), (R_{11}, R_{12}) e (R_{21}, R_{22}) per ricostruire i bit plane.

Una volta ottenuti i bit plane 0, 1 e 2, è possibile combinarli per ricreare l'immagine.

Valutazione

Sempre nel documento [13], si afferma che sono stati sviluppati vari metodi di cifratura basati sul concetto di Random Grid per immagini in scala di grigi, ma tutti questi approcci richiedono una conversione dell'immagine in una forma binaria. Tale trasformazione, come detto, comporta sempre inevitabilmente una perdita di informazione visiva. Tuttavia, nel caso della tecnica descritta, che si basa sull'uso di bit planes, la perdita di dettagli è significativamente inferiore rispetto all'approccio che fa uso dell'halftoning.

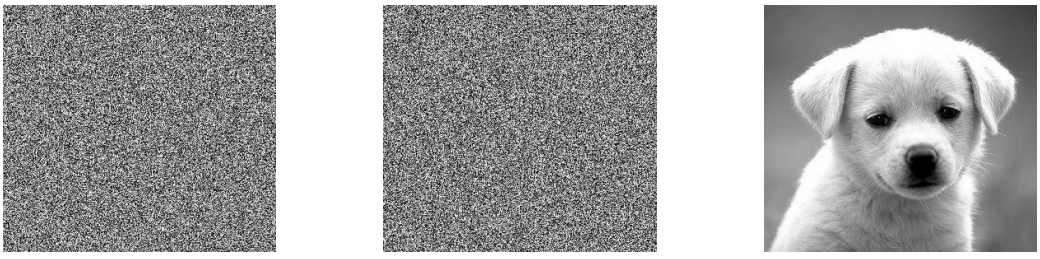
2.3 Perfect Reconstruction

Un approccio diverso, che non è illustrato in alcun documento scientifico inerente al VSS, si potrebbe basare su una semplice operazione di sottrazione tra pixel.

Si supponga di avere un'immagine I di dimensione $h \times w$. La prima grid R_1, di dimensioni $h \times w$, può essere creata con valori di pixel compresi tra 0 e 255 generati in modo randomico. La seconda grid R_2, delle medesime dimensioni, verrà creata con valori di pixel calcolati come: $R_2[i,j] = R_1[i,j] - I[i,j]$

Con questi due step basilari ho generato perfettamente due random grid, in quanto si tratta di una griglia generata casualmente e un'altra generata come sottrazione dei valori dell'immagine di partenza a dei valori randomici.

Per riottenere l'immagine, una volta ottenuti i due share, basta fare il passaggio algebrico, ovvero $I[i,j] = R_1[i,j] - R_2[i,j]$. La Figura (2.10) riporta un esempio di applicazione di questo schema di ricostruzione perfetta.



(a) Random Grid 1 (b) Random Grid 2 (c) Overlap

Figura 2.10: Perfect Reconstruction con Random Grid - Dimensioni: 332 × 300px

Questo meccanismo si allontana molto dal concetto di sovrapposizione di share e sfruttamento del sistema visivo umano, e probabilmente, per tale motivo, non è stato approfondito in alcun paper.

2.4 Conclusioni su RG

In questo capitolo, sono state esaminate le soluzioni ai problemi presentati dalla crittografia visuale. Inoltre, è stato illustrato un potenziale algoritmo che consente di ottenere, nella fase di decifratura, la ricostruzione perfetta dell'immagine in scala di grigi di partenza.

Considerate le finalità di questo capitolo, non è necessario dirigere l'attenzione verso implementazioni più avanzate che sfruttano Random Grid, come ad esempio i casi di immagini a colori o in situazioni che coinvolgono segreti multipli.

Legame tra RG e VC

È già stato affermato precedentemente che la crittografia visuale, introdotta da Naor e Shamir, e le random grid, introdotte da Kafri e Keren, sono due tipi di implementazione di uno schema di Visual Secret Sharing. Un ulteriore elemento da menzionare è la stretta connessione che vi è tra queste due tecniche.

Nel documento [16], viene dimostrato che ogni schema di RG può essere associato ad uno schema di VC senza pixel expansion e, viceversa, ad ogni schema di VC corrisponde uno schema di RG con espansione. Questo legame apre la possibilità di sfruttare i risultati e le proprietà di un modello all'interno dell'altro.

Capitolo 3

Image Steganography in Spatial Domain

In questo capitolo, con l'aiuto del documento intitolato "*A Survey on different techniques of steganography*" [17] come linea guida, verrà inizialmente presentata una panoramica del campo della steganografia. Successivamente, ci si concentrerà sulle tecniche Spatial Domain applicate alle immagini.

3.1 Introduzione

La steganografia è l'arte di rendere invisibili, a chi non autorizzato, comunicazioni di informazioni segrete dentro altre informazioni. All'atto pratico consiste nel data embedding in file multimediali come immagini, testo, audio o video.

Un sistema steganografico è composto da tre oggetti:

- *Cover-object*: oggetto di copertura utilizzato per nascondere al suo interno il messaggio;
- *Secret message*: messaggio che si vuole nascondere e tenere segreto;
- *Stego-object*: cover-object con il secret message incorporato.

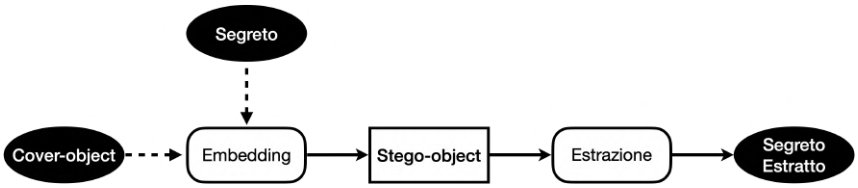


Figura 3.1: Processo di steganografia

3.1.1 Steganografia vs Crittografia

Steganografia e crittografia agiscono entrambe nell'ambito della sicurezza informatica, ma hanno obbiettivi ben distinti. L'obiettivo della crittografia è garantire una comunicazione sicura attraverso la trasformazione dei dati da trasmettere in una forma non facilmente comprensibile per un attaccante.

La steganografia, invece, si concentra sul nascondere una comunicazione in atto. A tale scopo, sono tre i parametri di valutazione principali:

- Capacità: indica la massima quantità di informazioni che possono essere nascoste all'interno del cover-object;
- Impercettibilità: indipendentemente dal cover-object scelto, le modifiche apportate devono risultare impercettibili;
- Robustezza: si riferisce alla capacità di incorporare dati e mantenere uno stego-object che resista a varie trasformazioni e compressioni.

Poiché la steganografia e la crittografia hanno finalità diverse, l'impiego di una non esclude necessariamente l'utilizzo dell'altra; anzi, l'adozione di una forma di cifratura è spesso essenziale per garantire la sicurezza della trasmissione dello stego-object stesso.

È pertanto possibile considerare l'opzione di applicare la cifratura all'intero stego-object, al segreto da nascondere o persino ad entrambi.

Nel primo scenario, l'estrazione del segreto risulterà possibile soltanto per chi è in possesso della chiave per decifrare lo stego-object. Nel secondo caso, sarà sempre possibile estrarre il segreto cifrato, ma quest'ultimo sarà comprensibile solamente da chi possiede la chiave di decifratura. Nell'ultimo caso, invece, sarà necessario possedere entrambe le chiavi.

La seconda modalità di integrazione tra cifratura e steganografia descritta è illustrata in Figura (3.2).

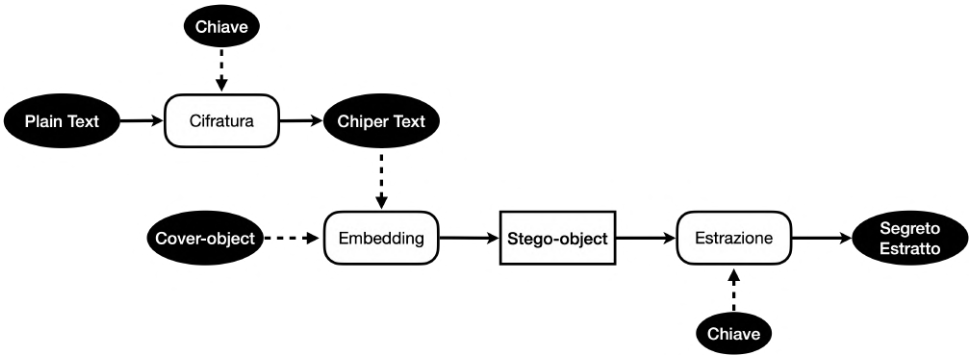


Figura 3.2: Combinazione di steganografia e crittografia

3.1.2 Tipi di Steganografia

Esistono svariati tipi di cover object, ossia oggetti in cui le informazioni vengono nascoste, ad esempio: testi, immagini, video, audio o addirittura header di protocolli. A seconda del tipo di cover-object impiegato e della tecnica di steganografia scelta, si ottengono diverse forme di steganografia, ognuna con caratteristiche differenti.

Text Steganography

Nel contesto della steganografia testuale, il cover-object utilizzato è di tipo testuale. In altre parole, l'obiettivo è nascondere del testo all'interno di un altro testo preesistente.

Esistono diverse tecniche che si basano su varie strategie, come il conteggio di spazi, tabs, lettere maiuscole e altre caratteristiche all'interno del testo ospite. Conteggi di questo tipo, simili al principio utilizzato nel codice Morse, vengono sfruttati per codificare e nascondere le informazioni desiderate all'interno del testo di copertura.

Image Steganography

Nel contesto in cui l'oggetto di copertura è un'immagine, per occultare le informazioni si sfrutta l'intensità dei pixel. Le modalità in cui ciò avviene variano in base al formato dell'immagine. Inoltre, maggiore è la dimensione dell'immagine, maggiore sarà la quantità di informazioni che potrà essere nascosta al suo interno.

Network Steganography

In questa categoria, un protocollo di rete viene utilizzato come cover-object. Ad esempio, all'interno dei protocolli TCP/IP esistono header non utilizzati che potrebbero essere impiegati per attività di steganografia.

Audio Steganography

Mentre nell'ambito delle immagini si agisce sfruttando le limitazioni del sistema visivo umano nell'identificare cambiamenti minimi, nell'ambito dell'audio occorre considerare il sistema uditivo umano, il quale possiede frequenze udibili e non udibili.

Video Steganography

Qui si sfruttano video come se fossero una sequenza di singole immagini. L'approccio si basa su tecniche come la Trasformata Discreta del Coseno (DCT), che permette di variare alcuni valori all'interno di ciascuna immagine del video.

3.1.3 Tecniche di Steganografia

Come accennato, esistono diverse tecniche di steganografia che si basano su approcci differenti e che sono più o meno adatte a determinati tipi di cover-object. Tra questi approcci, troviamo quelli che operano nello *spatial domain* o nel *transform domain*, o che utilizzano analisi statistiche, distorsioni, vector embedding e spread spectrum.

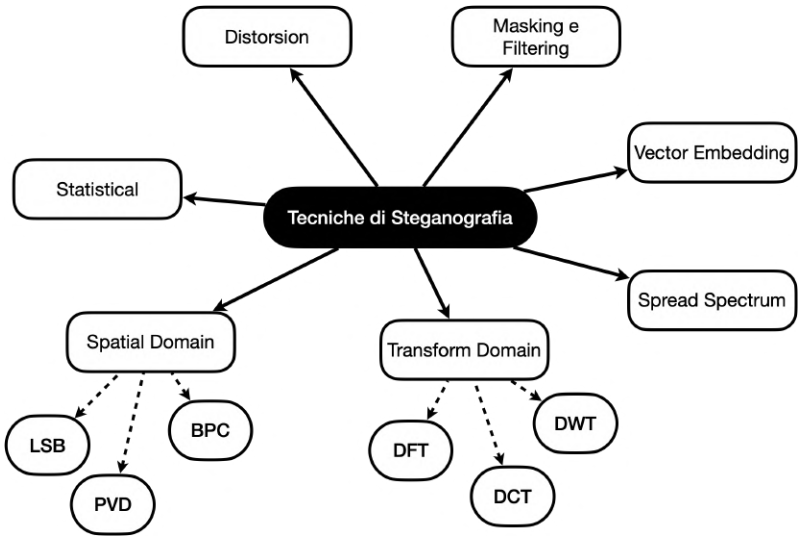


Figura 3.3: Tecniche di steganografia

Spatial Domain

La rappresentazione in pixel di un'immagine è chiamata spatial domain.

Nel contesto della steganografia, le tecniche di questo tipo agiscono direttamente sui valori dei pixel dell'immagine di copertura per nascondere le informazioni segrete.

Tra i metodi, che operano nel dominio spaziale, più diffusi vi sono il Least Significant Bit (LSB), il Bit Plane Complexity Segmentation (BPC) e il Pixel Value Differencing (PVD).

Transform Domain

Le tecniche steganografiche di questo genere incorporano il segreto all'interno dei coefficienti del dominio di trasformazione dell'immagine. Alcune tecniche, a differenza di quelle nel dominio spaziale, sono indipendenti dal formato dell'immagine e possono sopportare compressioni lossless e lossy, così come elaborazioni e ritagli dell'immagine.

Tra i metodi del transform domain si trovano: la Trasformata Discreta di Fourier (DFT), la Trasformata Discreta del Coseno (DCT) e la Trasformata Discreta di Wavelet (DWT).

Rapporto tra Steganografia e Watermarking

È possibile distinguere la steganografia in due ambiti distinti: uno orientato al data hiding, che costituisce l'oggetto di analisi di questo capitolo, e un altro che si focalizza sul document marking. In quest'ultimo contesto, si adoperano tecniche di *masking* e *filtering* che mirano a "marcare" l'immagine con informazioni come ad esempio il copyright, i diritti d'autore e la licenza.

Nelle prossime sezioni, ci si concentrerà sulle tre tecniche steganografiche più conosciute di data hiding nel dominio spaziale: LSB, BPCS e PVD.

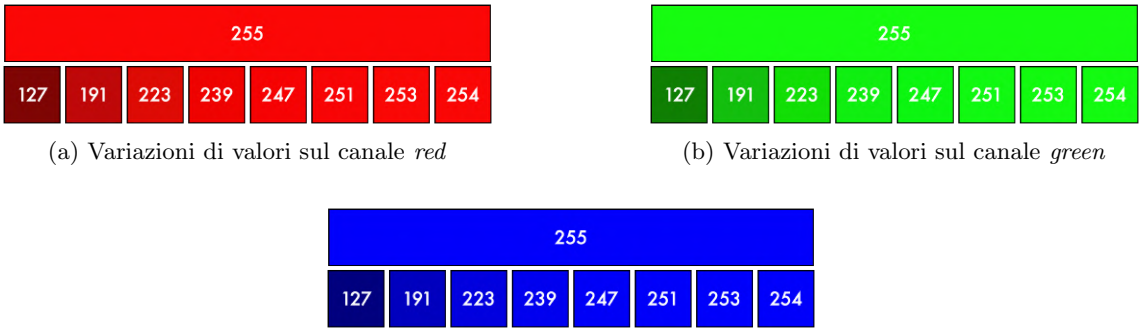
3.2 Least Significant Bit (LSB)

Essendo la tecnica LSB il metodo di steganografia su immagini più semplice da realizzare, si può partire direttamente ad analizzare la sua applicazione su immagini a colori.

Si considerino i byte dei tre canali di un singolo pixel:

$$R = r_7\ r_6\ r_5\ r_4\ r_3\ r_2\ r_1\ r_0 \text{ — } G = g_7\ g_6\ g_5\ g_4\ g_3\ g_2\ g_1\ g_0 \text{ — } B = b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0.$$

I bit meno significativi saranno i bit con indice zero, quindi r_0, g_0 e b_0. La modifica di questi bit crea una differenza di colore impercettibile all'occhio umano, come si può osservare in Figura (3.4) confrontando i valori 255 e 254 in ogni canale.



(a) Variazioni di valori sul canale *red*

(b) Variazioni di valori sul canale *green*

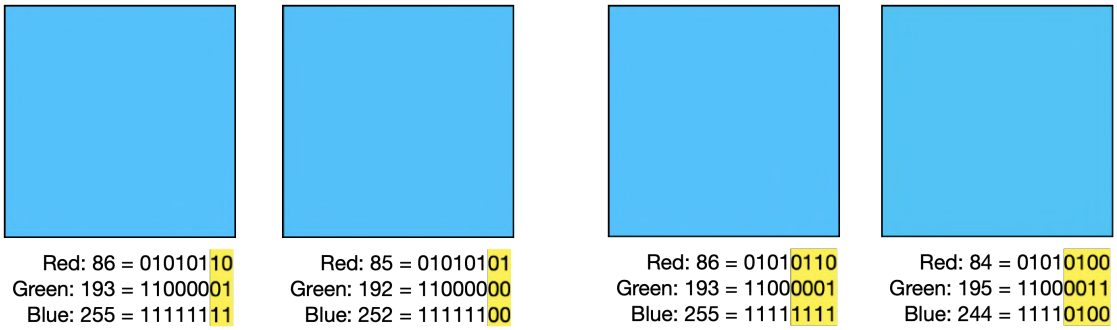
(c) Variazioni di valori sul canale *blue*

Figura 3.4

Tale impercettibilità permette di utilizzare l'ultimo bit di tutti e tre i canali, in ogni pixel dell'immagine, per nascondere un messaggio.

Un approccio semplice per la codifica del segreto potrebbe sfruttare la tabella ASCII Estesa, dove ogni carattere è rappresentato da 8 bit. Dato che un singolo pixel può nascondere 3 bit, con tale codifica, sono sufficienti 3 pixel per nascondere un singolo carattere.

Per mantenere modifiche che non alterino drasticamente l'immagine, è possibile considerare l'utilizzo di più di un bit per canale.



(a) Utilizzo di 2bit per canale

(b) Utilizzo di 4bit per canale

Figura 3.5: Valori RGB prima e dopo l'inserimento di 6 e 12 bit di segreto

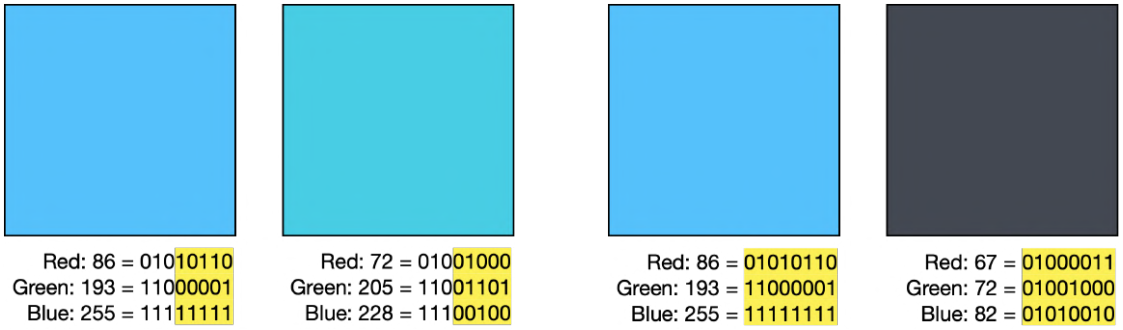
Nell'esempio in Figura (3.5a), sono impiegati due bit per canale al fine di occultare i primi 6 bit che costituiscono il carattere "C", ossia il valore numerico 67, rappresentato come 01000011. La differenza tra i due colori risulta invisibile all'occhio umano, tuttavia, il numero di bit riservati per nascondere le informazioni è stato raddoppiato.

In Figura (3.5b), invece, vengono impiegati quattro bit per canale, per nascondere l'intero carattere "C" e altri 4 bit del carattere "H" (01001000).

3.2.1 Compromesso tra impercettibilità e capacità

Se si estremizza questo concetto, ricorrendo all'utilizzo di troppi bit, si rischia di causare una modifica troppo intensa ai colori del pixel, generando un'alterazione significativa nell'immagine.

Ad esempio, nella Figura (3.6b), è evidente che l'impiego di tutti e 24 bit per occultare il messaggio, includendo i caratteri "C" (01000011), "H" (01001000) e "R" (01010010), comporta una modifica radicale nel colore del pixel.



(a) Utilizzo di 5bit per canale (b) Utilizzo di 8bit per canale

Figura 3.6: Valori RGB prima e dopo l'inserimento di 15 e 24 bit di segreto

Esiste quindi un trade-off tra impercettibilità e capacità: utilizzando meno bit, il cambiamento risulterà meno percettibile, ma allo stesso tempo potrò nascondere meno informazioni.

3.2.2 Problemi e criticità

Il documento [18], descrive una serie di problematiche della steganografia LSB di seguito riportate.

Semplicità

Data la sua intrinseca semplicità costruttiva, è plausibile che, un attaccante a conoscenza della "comunicazione invisibile", sia in grado di estrarre il segreto. Per contrastare ciò, si può procedere in due modi:

1. Cifrare il messaggio, in modo che se anche venisse estratto il secret message non sarà comprensibile;
2. Randomizzare i pixel su cui vengono scritti i bit del messaggio, mediante l'utilizzo di random function. Questa randomizzazione rende impossibile la ricostruzione del messaggio senza la conoscenza del seme della funzione.

In questo modo il messaggio gode di confidenzialità, e anche integrità in quanto risulta decisamente più difficile la contraffazione.

Oltre alle preoccupazioni legate alla cifratura, l'obiettivo principale è stabilire una comunicazione nascosta. Di conseguenza, la selezione dell'immagine e del suo formato (24 bit, 8 bit compressi o non compressi) è da effettuare accuratamente.

Selezione dell'Immagine

L'immagine di copertura scelta deve possedere le seguenti caratteristiche:

- Avere un senso per essere scambiate tra la sorgente e la destinazione;
- Presentare diversi colori (o sfumature) e avere del rumore in modo che il rumore aggiunto sarà coperto da quello già presente.

Selezione del Formato

Nei contesti di immagini compresse tramite algoritmi lossy come JPEG, l'approccio LSB non risulta applicabile. In tali circostanze, è necessario adottare tecniche che operano nel transform domain per nascondere i bit del segreto nei coefficienti, anziché nei singoli bit.

D'altra parte, la condivisione di file di grandi dimensioni e non compressi tra utenti potrebbe suscitare sospetti di una comunicazione nascosta.

Considerando un'immagine di 1024×768 pixel (=786.432 px), si ha che se l'immagine è:

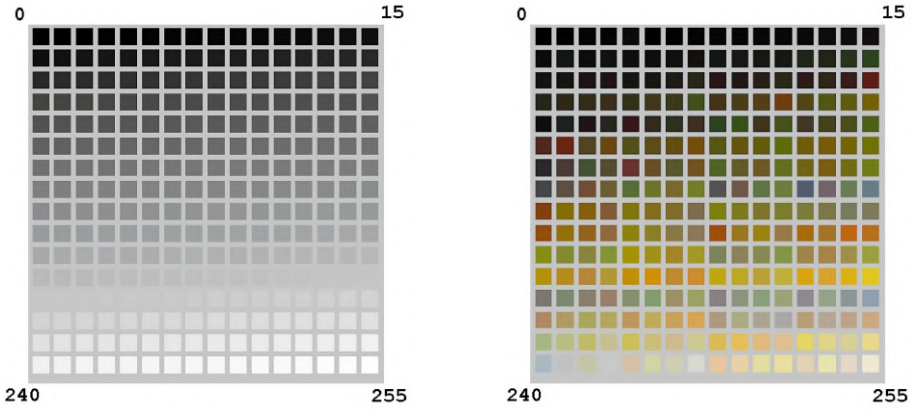
- a colori RGB (24 bit) = $1024 \times 768 \times 24 = 18.874.368$ bit = 2.359.296 byte $\approx$ 2,36MB
- scala di grigi (8 bit) = $1024 \times 768 \times 8 = 6.291.456$ bit = 78.6432 byte $\approx$ 0,79MB

Per queste ragioni, quando si utilizza LSB come tecnica di steganografia, è consigliabile impiegare immagini a 8 bit per mantenere la percezione di dimensioni "normali".

Palette a 8 bit

Le immagini in scala di grigio risultano preferibili poiché le sfumature cambiano gradualmente da byte a byte. Minore è la variazione dei colori all'interno della palette, minore sarà l'alterazione visiva dell'immagine una volta nascosto il segreto.

In realtà, con 8 bit, l'uso di una scala di grigi non è obbligatorio. È possibile creare, per ogni immagine, una palette indicizzata di 256 colori utilizzati in essa.



(a) Palette in scala di grigi (b) Palette di Reinor

Figura 3.7: Palette di 256 colori [18]

La modifica di un singolo bit tramite LSB comporta una variazione del colore di un pixel da un indice nella palette a un indice adiacente (precedente o successivo). Se i colori adiacenti nella palette presentano contrasti accentuati, un pixel contenente solo un bit di segreto potrebbe subire una spiccata variazione di colore, rilevabile all'occhio umano.

Diverse strategie sono state concepite per affrontare questa problematica. L'opzione più semplice è popolare ed ordinare la palette ponendo colori che non generino contrasti eccessivi vicini tra loro.

Questo problema dei "salti di colore" si aggrava se si riduce la dimensione delle palette, ad esempio in un'immagine a 5 bit si utilizzeranno palette a 32 colori e, molto probabilmente, la differenza tra questi sarà accentuata.

3.3 Bit Plane Complexity Segmentation (BPCS)

La tecnica BPCS, nota anche come BPC (Binary Pattern Complexity), è un approccio che identifica aree di rumore nelle diverse bit-plane, al fine di nascondere informazioni in tali aree.

In un'immagine possiamo distinguere delle *regioni informative*, cioè parti rilevanti dal punto di vista visivo, e delle *regioni noise-like*, ovvero aree complesse e poco significative.

Figura 3.8: Esempio di regioni informative e noise-like

Se si considera l'immagine di un deserto, come quella in Figura (3.8), la parte rappresentante la sabbia costituirà un'area ad alta complessità e potrà essere assimilata a rumore. Infatti, questa area ha una scarsa rilevanza visiva, dal momento che l'occhio umano percepirà abbastanza uniformemente tutta la porzione di spiaggia. Queste aree di alta complessità diventano utili

per occultare informazioni, in quanto piccole modifiche non alterano significativamente l'aspetto complessivo dell'immagine.

Al contrario, la porzione del cielo rappresenta un'area a bassa complessità. Qualsiasi modifica minima a quest'area sarà più facilmente individuabile. Proprio per questa ragione, le regioni informative, ovvero quelle a bassa complessità, non vengono sfruttate per nascondere informazioni, poiché le alterazioni sarebbero troppo evidenti.

In realtà, la distinzione tra le zone viene fatta sui singoli bitplane che compongono l'immagine. Ad esempio, in Figura (3.9) è mostrato il settimo bit-plane estratto dalla versione in scala di grigi dell'immagine in Figura (3.8).

Si può notare come in tale bitplane le aree rumorose ad alta complessità, in cui è possibile nascondere bit del segreto, e le zone uniformi a bassa complessità, dove non conviene agire, sono facilmente distinguibili. Ciò significa che il settimo bit, perché si tratta della settima bitplane, dei pixel nelle aree più rumorose può essere sfruttato per nascondere segreto.

Questo ragionamento verrà applicato su tutti i bitplane dell'immagine e permette di determinare su quali bit dei pixel agire e su quali no.

Figura 3.9: Bitplane meno significativo

Mentre nella tecnica LSB i dati sono occultati nella (o nelle) bit-plane meno significative, nella tecnica BPCS i dati sono nascosti in tutti i livelli, dal più elevato (MSB plane) al più basso (LSB plane), sempre all'interno delle regioni rumorose.

3.3.1 Distinguere regioni ad alta e bassa complessità

La parte fondamentale di BPCS consiste nel individuare correttamente le regioni noise-like nella cover image.

Metodo tradizionale

Il metodo tradizionale divide ogni bit-plane dell'immagine di copertura in dei blocchi quadrati di pixel binari (bianco e nero). Successivamente, su ciascun di questi blocchi, viene eseguito un test che consente di quantificare la loro complessità. In maniera semplificata si può affermare che blocchi con pattern complessi verranno considerati come regioni rumorose.

Fondamentalmente, esistono tre metodi di misura della complessità, il più comune si basa sul misurare la "lunghezza del bordo". Tale lunghezza corrisponde alla somma del numero di cambi di colore lungo le righe e le colonne del blocco.

(a) (b)

Figura 3.10: Esempi di blocchi da valutare. (a) è un Blocco a bassa complessità (regione informativa), mentre (b) è un blocco ad alta complessità (regione noise-like)

Ad esempio nella Figura (3.10a), tutti i pixel sono bianchi, ciò significa che non vi è cambio di colore tra righe e colonne e quindi la lunghezza del bordo è zero.

Invece, nella Figura (3.10b), vi è un'alternanza di bianco e nero continuo. In ogni riga e colonna vi sono tre cambi di colore; di conseguenza il totale di cambi di colori è 24 [19].

Successivamente, tale valore viene confrontato con una determinata soglia (parametro variabile che può dipendere dall'immagine) e in base al risultato si classificherà il blocco come informativo o rumoroso. Definendo in questo modo anche se i pixel del blocco possono essere sfruttati per nascondere del segreto, oppure no.

Metodi avanzati

Ci sono altre accortezze che si possono mettere in atto, ad esempio esiste una formula che permette di calcolare matematicamente e normalizzare la lunghezza del bordo riportandola ad un valore sempre compreso tra zero e uno.

Oppure il fatto che pattern periodici come a scacchiera o a righe, anche se identificati come complessi effettuando il conteggio, non dovrebbero essere utilizzati per l'embedding di dati, altrimenti l'immagine si deteriorerebbe in modo ovvio [14].

Tuttavia, l'obiettivo di questa sezione era semplicemente quello di fornire un'introduzione alla tecnica più diffusa che si pone in contrasto con l'LSB, piuttosto che esaminarne tutti i dettagli.

3.4 Pixel Value Differencing (PVD)

Per completezza si illustra l'idea base su cui si fonda il metodo PVD (Pixel Value Differencing), ovvero l'ultima delle tecnica più diffuse nell'ambito della steganografia nel dominio spaziale.

Considerando un'immagine in scala di grigi, questa viene suddivisa in blocchi non sovrapposti di due pixel consecutivi P_i e P_{i+1}. Successivamente, per ogni blocco, viene calcolata una differenza $d_i = |p_i - p_{i+1}|$, dove d_i può assumere valori da 0 a 255.

- Se d_i è "piccolo", indica che la differenza tra i due pixel è limitata, ovvero si tratta di un'area a bassa complessità. In questo caso, sarebbe meglio non nascondere informazioni, o eventualmente poche.

- Viceversa, se d_i è "grande", si tratta di un'area complessa in cui è possibile nascondere più informazioni.

Senza entrare troppo nello specifico, la formula $m = \log_2(u - l + 1)$ permette di calcolare il numero di bit utilizzabili per nascondere segreto in un blocco. I parametri u ed l rappresentano un limite inferiore e superiore che vengono indicati in delle apposite tabelle [20].

3.5 Reversible steganography

In tutte le tecniche analizzate la stego-image, dopo l'estrazione del segreto, risulta inutile, poiché è completamente diversa dall'originale. Per affrontare tale problema, sono stati proposti degli schemi di steganografia reversibile (RSS), noti anche come metodi di steganografia lossless [21].

Un RSS è quindi formato da tre procedure: una di *embedding* e una di *extraction*, come qualsiasi schema, e una di *recovery* dove ripristina la stego-image al suo stato originale.

In base al tipo di tecniche utilizzate, gli schemi RSS possono essere ulteriormente suddivisi in diverse categorie, come *difference expansion* (DE), *histogram-shifting* (HS), *pixel-value-ordering* (PVO), e così via.

3.5.1 Difference Expansion (DE): Tian's method

Al fine di avere un'idea di come una tecnica reversibile possa funzionare, e delle criticità che può presentare, viene riportata lo tecnica ideata da Tian, che rientra nella categoria di *difference expansion*.

Questa categoria richiama in parte il concetto di differenza di pixel utilizzato nella PVD. In questo schema, due pixel adiacenti possono essere impiegati per occultare una parte del segreto. La media del testo nascosto sarà di un bit ogni due pixel, equivalente a 0.5 bit per pixel.

Embedding

Si supponga che i pixel della cover image siano $p_1 = 20$ e $p_2 = 15$, dove $0 \leq p_1, p_2 \leq 255$, e che il bit di segreto che si vuole nascondere sia $s = 1$.

Per prima cosa, il metodo Tian calcola la differenza $d = p_1 - p_2 = 20 - 15 = 5$ e successivamente calcola la media di p_1 e p_2 come: $\alpha = \lfloor \frac{(p_1+p_2)}{2} \rfloor = \lfloor \frac{(20+15)}{2} \rfloor = 17$.

Il dato segreto viene nascosto sfruttando la formula $d' = 2 \times d + s = 2 \times 5 + 1 = 11$, che permette di calcolare gli stego-pixel come: $p'_1 = \alpha + \lfloor \frac{(d'+1)}{2} \rfloor = 17 + \lfloor \frac{(11+1)}{2} \rfloor = 23$ e $p'_2 = \alpha - \lfloor \frac{d'}{2} \rfloor = 17 - \lfloor \frac{11}{2} \rfloor = 12$.

Extraction e recovery

Per la procedura di estrazione, la differenza espansa è calcolata come $d' = p'_1 - p'_2 = 23 - 12 = 11$, e la media come $\alpha = \lfloor \frac{(p'_1+p'_2)}{2} \rfloor = \lfloor \frac{(23+12)}{2} \rfloor = 17$.

La differenza originale d è misurabile come $d = \lfloor \frac{d'}{2} \rfloor = 5$, mentre il messaggio segreto lo si ricava da $s = d' - \lfloor \frac{d'}{2} \rfloor \times 2 = 11 - 10 = 1$. I pixel originali sono recuperabili mediante i calcoli $p_1 = \alpha + \lfloor \frac{(d+1)}{2} \rfloor = 17 + \lfloor \frac{(5+1)}{2} \rfloor = 20$ e $p_2 = \alpha - \lfloor \frac{d}{2} \rfloor = 17 - \lfloor \frac{5}{2} \rfloor = 15$.

Casi di overflow e underflow

Nel caso dell'esempio precedente, non si sono verificati problemi di overflow o underflow e per questa ragione, i pixel p_1 e p_2 in questione, vengono definiti "espandibili".

Al fine di affrontare i casi di pixel non espandibili Tian propone una strategia alternativa denominata "*changeable pixels*" Se sono soddisfatte le regole: $0 \leq \alpha - (2 \times \lfloor \frac{d}{2} \rfloor + s) \leq 255$ e $0 \leq \alpha + (2 \times \lfloor \frac{d}{2} \rfloor + s) \leq 255$, allora i pixel sono *changeable* e di conseguenza l'equazione per nascondere segreto cambia in $d' = 2 \times \lfloor \frac{d}{2} \rfloor + s$.

In questo modo il ricevente non ha modo di comprendere se il segreto è stato nascosto con l'equazione $d' = 2 \times d + s$ o $d' = 2 \times \lfloor \frac{d}{2} \rfloor + s$.

Date le due possibili equazioni è necessario generare anche una *location map* che registra la strategia usata per nascondere il segreto in ogni coppia di pixel. Inoltre, se i bit sono *changeable*, allora sarà necessario memorizzare anche l'*original least significant bit* (OLSB) della differenza d per poter fare il recovery dell'immagine. La *location map* e gli OLSB verranno concatenati ai dati segreti per essere memorizzati nel cover-object [21].

3.6 Steganalysis

Come per la crittografia esiste la crittoanalisi, ovvero lo studio di metodi per decifrare testo cifrato senza avere la chiave, per la steganografia, esiste la steganalisi, ovvero la scienza che si occupa del rilevamento dei messaggi nascosti con steganografia [22].

Possibili attacchi

Di seguito vengono presentati i possibili scenari di attacco:

- *Known cover attack*: sia cover-object che stego-object sono conosciuti;
- *Known message attack*: l'informazione segreta e lo stego-object sono conosciuti e si analizza quest'ultimo per trovare un pattern che aiuterà attacchi futuri al sistema;
- *Chosen message attack*: attaccante può generare degli stego-object partendo da un messaggio scelto. È il tipo di attacco più forte, anche qui viene ricercato un pattern;
- *Known stego attack*: attacco dove è conosciuto solo lo stego-object.

Detection

L'incorporazione di un messaggio segreto in un'immagine potrebbe non essere visibile all'occhio umano, ma rappresenta comunque una distorsione rilevabile.

Attacchi di tipo known cover e know stego sono spesso condotti al fine di individuare una firma specifica, che può suggerire la tecnica di steganografia adottata.

Ad esempio, nel riconoscimento di uno schema LSB, un segno distintivo facilmente individuabile è l'elevato livello di rumore nell'immagine, oppure un altro indicatore sono i "cambi di colore veloci", magari dovuti ad una palette mal strutturata.

Per poter risalire al messaggio segreto può tornare utile, in primo luogo, riuscire ad identificare il tool utilizzato. Ad esempio, due segni distinguibili sono il padding o crop "strano" dell'immagine dovuto ai requisiti massimi e minimi di dimensoni delle immagini dati ad alcuni software che permettono di fare steganografia. Di conseguenza se l'immagine è troppo piccola subirà padding, se è troppo grande subirà un crop.

Disabling information

La rilevazione della tecnica o dello strumento ci aiuta a avvicinarci all'informazione. Tuttavia, vi è un'altra categoria di attacchi che mira invece a distruggere o disabilitare l'informazione anziché ottenerla.

I metodi di attacchi attivi includono:

- *Blur*: ammorbidisce i cambiamenti e il contrasto tra pixel vicini dove i cambiamenti sono significativi;
- *Noise*: inserisce del rumore, ovvero colora in modo casuale alcuni pixel;
- *Noise reduction*: riduce il rumore aggiustando i colori e facendo la media tra pixel vicini;
- *Sharpen*: è il contrario di Blur, aumenta il contrasto tra pixel adiacenti dove c'è un contrasto di colori importante (di solito sui contorni degli oggetti);
- *Rotate*: ruota un'immagine attorno a un punto centrale.

Capitolo 4

Immagini Mediche

Dalla scoperta dei raggi X di Wilhelm Conrad Röntgen nel 1895, il campo delle immagini mediche ha avuto un grande sviluppo.

Il presente capitolo intende fornire una panoramica su: alcune tipologie di esami diagnostici per immagini, gli attuali standard e il quadro normativo relativo alla privacy delle informazioni contenute in tali immagini.

4.1 Esami diagnostici per immagini

La diagnostica per immagini è una branca della medicina che consente di ottenere immagini dell'interno del corpo, aiutando i medici nelle diagnosi, nelle valutazioni sulla gravità di patologie e nel monitoraggio dei pazienti. Gli esami diagnostici possono utilizzare diverse tecniche tra cui:

1. *Radiazioni*: utilizzate per ottenere immagini dei tessuti e degli organi interni, come ad esempio nelle radiografie e tomografie computerizzate;
2. *Onde radio e campi magnetici*: utilizzati per creare immagini dettagliate dei tessuti molli e delle strutture anatomiche, come nel caso delle risonanze magnetiche;
3. *Onde sonore*: ad esempio nelle ecografie vengono sfruttate onde sonore ad alta frequenza per produrre immagini dei tessuti e degli organi;

Questi esami forniscono ai medici informazioni preziose sull'anatomia e sullo stato dei tessuti e delle strutture interne, aiutandoli a formulare diagnosi accurate e a definire piani di trattamento adeguati per i pazienti.

Differenza tra onde

Anche se ci sono diverse tipologie di imaging, si tratta sempre di energia inviata dentro il corpo per capire cosa succede al suo interno. Il concetto di onde che attraverso il corpo è qualcosa che si verifica continuamente; come nel caso dell'energia del sole, le onde della luce visibile e le onde sonore.

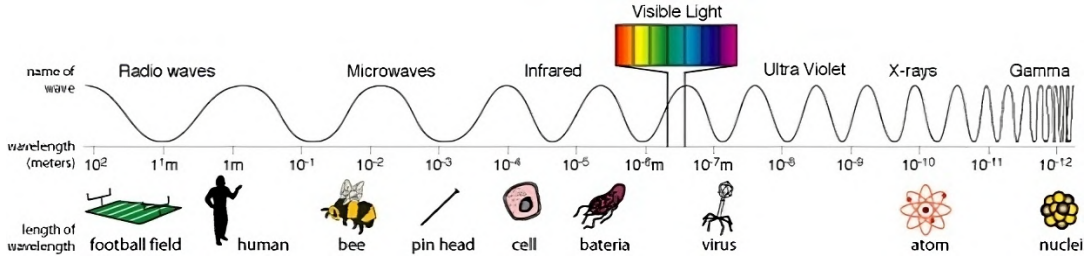


Figura 4.1: Spettro elettromagnetico [23]

All'interno dello spettro elettromagnetico, Figura (4.1), vengono rappresentate le diverse forme di onde elettromagnetiche, tra cui i raggi X e le onde radio. Si noti che le onde sonore non fanno parte di questo spettro in quanto sono onde meccaniche e non hanno natura elettromagnetica. Tuttavia, basti sapere che proprio come non possiamo vedere i raggi elettromagnetici con la più alta e bassa energia, non possiamo sentire le frequenze più alte (utilizzate nell'ecografia) e più basse delle onde sonore.

4.1.1 Radiografia

Una radiografia, comunemente chiamata raggi X, trasmette delle radiazioni al corpo. Le aree con un alto livello di calcio, come denti e ossa, bloccano le radiazioni, risultando bianche nell'immagine finale. Al contrario, tessuti morbidi, fanno passare le radiazioni, risultando in grigio o nero. Questo esame diagnostico è il più veloce e risulta molto efficacie nei casi di problematiche con le ossa, ad esempio fratture e lussazioni [24].

Il termine "immagine" non viene usato solo per riferirsi ad immagini 2D tradizionali, come quelle risultanti da una radiografia. In casi come quello della tomografia computerizzata e della risonanza magnetica si trattano sequenze di immagini acquisite nel tempo che vengono successivamente, attraverso elaborazione, ricondotte ad un'unica immagine.

4.1.2 Tomografia Computerizzata

La tomografia computerizzata (TC)[1] è una tecnica diagnostica che sfrutta la trasmissione di radiazioni ionizzanti in forma di raggi X per ottenere immagini dettagliate e tridimensionali di un dato distretto anatomico del corpo umano [25].

Il suo funzionamento si basa sulla trasmissione di raggi X sul corpo e sulla misurazione, da diverse angolazioni, della quantità di energia assorbita dalle varie strutture. La quantità di radiazioni assorbite è proporzionale alla densità e alla composizione del tessuto corporeo attraversato, in altri termini materiali diversi assorbono quantità di radiazioni diverse.

Nella procedura di TC il paziente viene posizionato supino e scorre attraverso l'apertura circolare di uno scanner. Durante la scansione, un tubo a raggi X emette un fascio di radiazioni che attraversano il corpo della persona. Mentre i tessuti del corpo assorbono, uno specifico rilevatore circolare acquisisce le informazioni sulle energie assorbite e vengono create più immagini denominate immagini assiali o trasversali.

Mentre una normale radiografia mostra i bordi dei tessuti molli sovrapposti l'uno sull'altro, il computer in una TC è in grado di determinare come quei bordi si relazionano tra di loro in profondità, rendendo l'immagine molto più utile per analizzare tali tessuti. Questo è possibile proprio perché la tomografia ha un livello di dettaglio maggiore trattandosi di uno scanning a 360 gradi della struttura corporea. L'immagine finale ottenuta sarà una combinazione delle varie immagini trasversali acquisite.

Le TC impiegano più tempo di una semplice radiografia, ma si tratta comunque di un'esame rapido e utilizzabile nei casi di traumi per individuare problematiche quali coaguli di sangue fratture ossee, incluse quelle più sottili non visibili nelle radiografia, lesioni degli organi e altre anomalie.

Scala di Hounsfield

La scala Hounsfield (HU) si occupa di effettuare la misurazione delle quantità di radiazioni assorbite. Nella Tabella (4.1), ottenuta dal documento [26], è possibile osservare i valori approssimativi in unità HU di alcuni tessuti e materiali.

Tessuto/Materiale:	**Unità Hounsfield:**
Ossa, calcio, metallo	> 1000
Mezzo di contrasto iodato	100 a 600
Emorragia intracranica	60 a 100
Materia grigia	35
Materia bianca	25
Muscolo, tessuto molle	20 a 40
Acqua	0
Grassi	-30 a -70
Aria	-1000

Tabella 4.1

La parte più critica della tomografia computerizzata consiste nel distinguere e segmentare correttamente i vari tessuti e organi "morbidi" tra loro. È stato detto precedentemente che ciò è possibile mediante l'unione di immagini acquisite da angolazioni diverse, tuttavia questo non

[1] Un acronimo della TC più conosciuto, e ormai in disuso, è "TAC" che indica una tomografia *assiale* computerizzata, ovvero lungo il solo piano assiale.

sempre è sufficiente. Ad esempio, il sangue ha un valore HU di circa +40, mentre il tessuto del fegato ha un valore compreso tra +40 e +60 HU.

Per distinguere meglio degli organi o tessuti, spesso vengono introdotti nel corpo del paziente agenti di contrasto che assorbono i raggi X e modificano le caratteristiche dell'immagine acquisita. In questo modo, il valore HU del sangue può aumentare notevolmente, ad esempio a +120, facilitando così la distinzione tra sangue e fegato. Tuttavia, il sangue con mezzo di contrasto non fluirà soltanto attraverso le arterie, ma dopo un certo tempo raggiungerà anche il fegato. Pertanto, il tempismo diventa un aspetto cruciale per l'imaging di determinate strutture [27].

La Figura (4.2) mostra i valori HU di alcuni tessuti visibili mediante TC al cervello. Le diverse sfumature di grigio rappresentano quindi le differenti quantità di raggi X assorbite e, di conseguenza, evidenziano tessuti tra loro diversi.

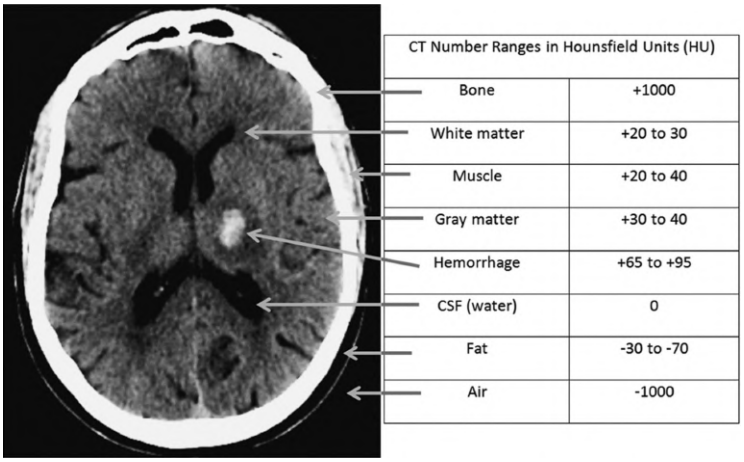


Figura 4.2: Immagine di una TAC con valori tipici di HU [28]

4.1.3 Risonanza Magnetica

È risaputo che la materia è formata da atomi, e ogni atomo ha una struttura interna diversa, caratterizzata dal numero di protoni, neutroni ed elettroni. I protoni e i neutroni costituiscono il nucleo, mentre gli elettroni orbitano intorno ad esso.

Alcune proprietà fisiche non sono facilmente percepibili ma possono essere sfruttate per ottenere informazioni sulla morfologia del corpo umano.

Nell'immaging delle risonanze magnetiche (RM), delle specifiche onde radio vengono utilizzate per sollecitare i nuclei degli atomi di idrogeno, che abbondano sia nell'acqua che nel grasso. Dei magneti rilevano la risposta dell'idrogeno e mappano le posizione dei tessuti dove questo risiede. Alla base del fenomeno di risonanza magnetica, quindi, viene sfruttata la proprietà magnetica dei nuclei, evitando l'utilizzo di radiazioni ionizzanti [24].

A differenza della tomografia computerizzata, dove le unità di Hounsfield indicano le radiazioni assorbite e hanno un significato fisico preciso, la scala utilizzata nelle risonanze magnetiche non ha un corrispondente significato fisico. Il funzionamento di una RM è più simile a quello di una fotografia, dove l'obiettivo principale è distinguere le varie strutture senza la necessità di valori assoluti come quelli presenti nella TC [27].

4.1.4 Ecografia

Le ecografie rientrano nell'imaging ad ultrasuoni, in cui si utilizzano onde sonore ad alta frequenza per visualizzare l'interno del corpo del paziente. Pertanto, anche in questa tipologia di esami non vi è alcuna esposizione a radiazioni ionizzanti. Poiché queste immagini sono acquisite in tempo reale, è possibile monitorare il movimento degli organi interni e il flusso di sangue nei vasi sanguigni.

Durante un'ecografia, si applica uno strato sottile di gel sulla pelle e si posiziona un trasduttore che trasmetterà onde ad ultrasuoni attraverso il gel nel corpo.

Il segnale sonoro viene riflesso dalle diverse strutture interne del corpo e varierà in potenza (ampiezza del segnale) e tempo impiegato. Il trasduttore che riceve le onde sonore riflesse, è in grado di ricavare informazioni sulla composizione e sulla forma degli organi e dei tessuti interni, utilizzandole per creare l'immagine visualizzata sullo schermo dell'ecografo [29].

4.1.5 Confronto tra gli esami diagnostici

Di seguito si riporta, in Tabella (4.2), una sintesi delle caratteristiche degli esami precedentemente illustrati.

Nome	**Tecnologia**	**Descrizione**	**Durata**	**Utilizzata per**
Radiografia	Radiazioni ionizzanti	I raggi X sono test rapidi e indolori che producono immagini delle strutture all'interno del corpo, in particolare delle ossa.	10-15 min	Fratture ossee, artrite, osteoporosi, infezioni, cancro al seno, oggetti ingeriti e per problemi del tratto digestivo.
Tomografia Computerizzata	Radiazioni ionizzanti	Le scansioni TC utilizzano una serie di raggi X per creare sezioni trasversali dell'interno del corpo, inclusi ossa, vasi sanguigni e tessuti molli.	10-15 min	Traumi e lesioni da incidenti, fratture ossee, tumori e cancri, malattie vascolari, malattie cardiache, infezioni e per guidare biopsie.
Risonanza Magnetica	Onde radio	Le risonanze magnetiche utilizzano campi magnetici e onde radio per creare immagini dettagliate degli organi e dei tessuti all'interno del corpo.	45-60 min	Aneurismi, sclerosi multipla, ictus, disturbi del midollo spinale, tumori, problemi ai vasi sanguigni e lesioni articolari o dei tendini.
Ecografia	Ultrasuoni	L'ecografia utilizza onde sonore ad alta frequenza per produrre immagini degli organi e delle strutture all'interno del corpo.	30-60 min	Malattie della cistifellea, noduli al seno, problemi genitali/prostatici, infiammazioni articolari, problemi di flusso sanguigno, monitoraggio della gravidanza e per guidare le biopsie.

Tabella 4.2: Esami diagnostici a confronto [30]

Per concludere il confronto tra gli esami diagnostici, in Figura (4.3) sono presentate una radiografia, una tomografia computerizzata e una risonanza magnetica di un ginocchio, mentre in Figura (4.4) sono mostrati gli stessi esami diagnostici su una colonna vertebrale. Queste immagini illustrano le differenze e la capacità di ciascun esame di fornire informazioni dettagliate su strutture specifiche all'interno del corpo umano.

(a) Radiografia (b) TC (c) RM

Figura 4.3: Esami diagnostici effettuati su un ginocchio [31]

L'utilizzo combinato di diverse tecniche di imaging può essere utile per una diagnosi accurata e un trattamento appropriato dei pazienti.

(a) Radiografia (b) TC (c) RM (d) RM

Figura 4.4: Esami diagnostici effettuati su una colonna vertebrale [32]

Per chiarire meglio la differenza tra una semplice radiografia e una TC è necessario introdurre il concetto di immagini volumetriche. Si noti che, mentre la Figura (4.4a) è il risultato effettivo dell'esame radiologico, lo stesso non vale per la Figura (4.4b). Il risultato di una TC è, in realtà, un'immagine volumetrica e quella rappresentata in Figura (4.4b) consiste in una singola sezione di tale immagine in cui si vede meglio la zona di interesse.

Discorso simile vale anche per le risonanze magnetiche (4.4c) e (4.4d). Queste non sono il risultato di due RM distinte, ma sono due sezioni diverse estratte dalla stessa immagine volumetrica.

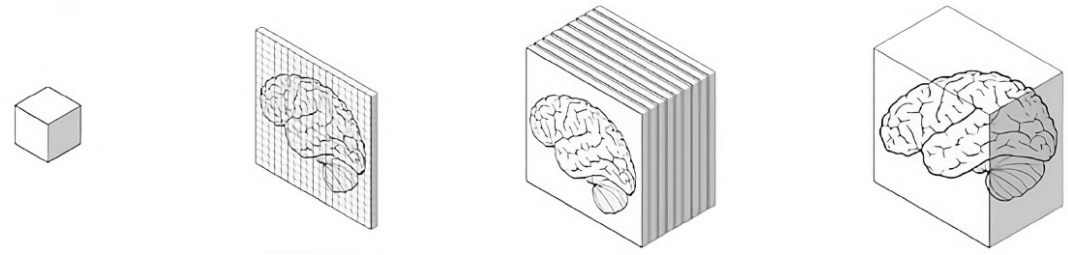
4.2 Immagini Volumetriche e Voxel

Consideriamo l'acquisizione di una risonanza magnetica al cervello anche se, come accennato, i concetti che seguono valgono per l'acquisizione di qualsiasi immagine volumetrica (TC inclusa).

Il cervello occupa spazio, pertanto quando si raccolgono dati di come questo spazio è riempito, si parla di dati volumetrici. Tali dati sono misurati in voxel (*volumetric picture element*)[2] che sono la controparte tridimensionale del pixel (che invece è bidimensionale). Ogni voxel contiene informazioni sulla densità del materiale scansionato.

L'immagine volumetrica, ovvero quella formata da tutti i voxel, viene ottenuta dall'unione di tutte le immagini assiali acquisite dal sensore da posizioni diverse.

Di conseguenza, affermare che una scansione TC o RM genera una sequenza di immagini assiali bidimensionali equivale a dire che queste scansioni creano una rappresentazione in voxel del corpo umano o, nel caso specifico, del cervello.



(a) Voxel (b) Singola sezione (c) Sezioni (d) Volume

Figura 4.5: Schematizzazione dei concetti di voxel e immagini volumetriche [33]

[2]Voxel, per definizione è un'unità di misura del volume.

Ottenuta la rappresentazione in voxel del cervello, è possibile, tramite l'utilizzo di software, ottenere sezioni diverse. La Figura (4.6) mostra un esempio di sezione di un'ipotetica immagine volumetrica del cervello [34].

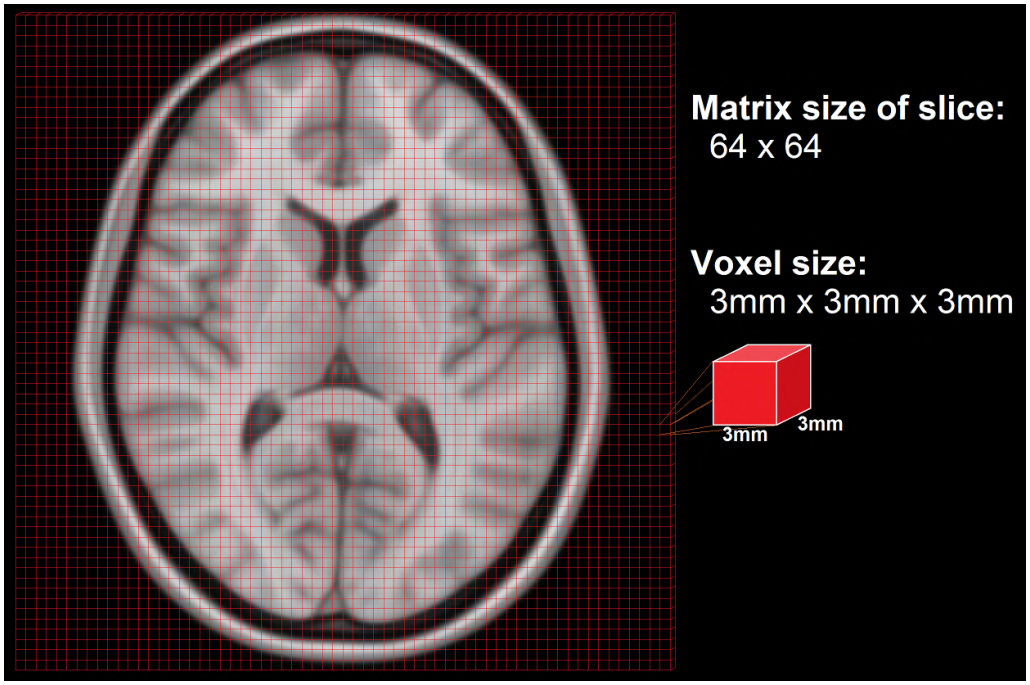


Figura 4.6: Singola sezione di un RM al cervello [34]

4.2.1 Dal 3D al 2D

Per quanto descritto si può affermare che tutte le immagini ottenute da RM e TC sono sicuramente tridimensionali al momento dell'acquisizione, mentre per quanto riguarda l'immagine finale questa potrà essere in 3D o in 2D.

Se si intende mantenere l'immagine finale in 3D, sarà necessario utilizzare appositi software che sfruttano tecniche di *rendering volumetrico* per poterla visualizzare.

D'altra parte, se si vuole ottenere come risultato un'immagine convenzionale in 2D, è necessario applicare procedure che convertono l'immagine tridimensionale in una bidimensionale. Di seguito si riportano i due algoritmi più utilizzati.

Multi Planar Reconstruction (MPR)

MPR, traducibile in italiano come "Ricostruzione multiplanare", consiste nella creazione di immagini 2D scegliendo una sezione dell'immagine volumetrica su un piano anatomico (assiale, coronale, sagittale, o obliquo).

In Figura (4.7) si può vedere un'applicazione di MPR ad una matrice di voxel per selezionare delle sezioni più esterne su alcuni assi [35].

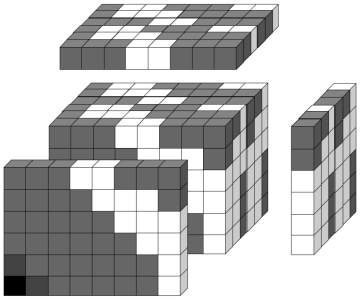


Figura 4.7: MPR [35]

Avendo citato gli assi anatomici, per completezza, si riportano le loro definizioni [36]:

- *Piano sagittale*: si estende lungo il corpo dalla testa ai piedi, suddividendolo in una parte destra e una parte sinistra. Ogni piano parallelo a questo piano è noto come "piano sagittale";
- *Piano coronale (o frontale)*: corre dalla testa ai piedi, dividendo il corpo in due parti: una anteriore e una posteriore;
- *Piano trasversale*: si estende dalla parte posteriore a quella frontale, suddividendo il corpo in due parti: una superiore e una inferiore;
- *Piano obliquo*: qualsiasi piano che attraversi il corpo senza essere parallelo ai tre piani sopra descritti;

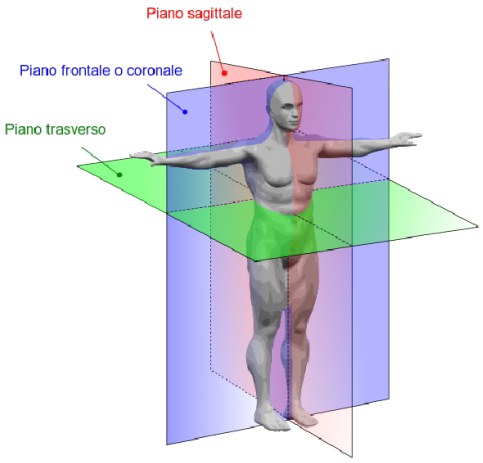


Figura 4.8: Piani anatomici su un corpo [36]

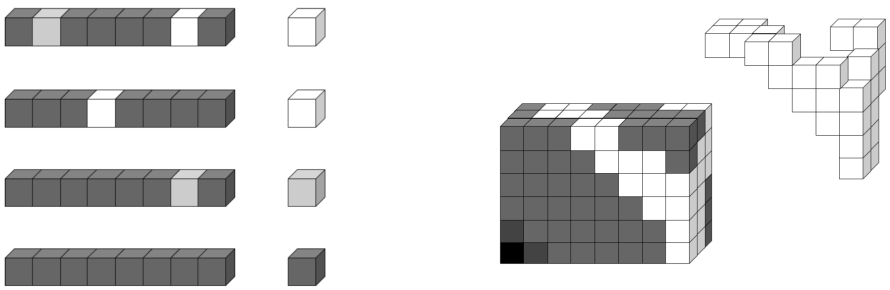
Alla luce di questi concetti, si può comprendere che la sezione del cervello, riportata precedentemente in Figura (4.6), è quindi un immagine 2D ottenuta scegliendo una sezione di voxel sul piano trasversale mediante algoritmo MPR.

Maximum Intensity Projection (MIP)

MIP, traducibile come "Proiezione di Massima Intensità", è un algoritmo a perdita volumetrica ed informativa del tipo 3D-2D tutto-a-uno. Scelto un asse di proiezione sull'immagine volumetrica acquisita, si ottiene un'immagine bidimensionale che presenta le caratteristiche più intense e dominanti dell'oggetto.

Consideriamo la Figura (4.9a) dove una serie di voxel sono disposti su quattro linee, alcuni dei quali presentano un'intensità maggiore rispetto gli altri. Questi voxel con intensità elevate vengono selezionati e filtrati per formare la nuova immagine. La selezione dei voxel più intensi può essere particolarmente utile, nel caso di imaging medico, per evidenziare strutture o dettagli importanti che potrebbero essere altrimenti nascosti in una singola sezione ottenuta con l'algoritmo MPR.

Estendendo tale concetto ad una immagine volumetrica, che rappresenta la scansione di un'arteria, si può immaginare una proiezione dei voxel con intensità più elevata, su più assi, come in Figura (4.9b) [37].



(a) MIP applicato ad un array di voxel

(b) MIP applicato ad una matrice di voxel

Figura 4.9: MIP [35]

Questa tecnica è comunemente utilizzata nelle angiografie al fine di identificare eventuali anomalie, ostruzioni o patologie.

Per angiografia si intende una procedura diagnostica che utilizza l'iniezione di un mezzo di contrasto, per ottenere un'immagine dettagliata dei vasi sanguigni o linfatici del paziente. In Figura (4.10) è mostrata l'immagine di un'angiografia ottenuta applicando la tecnica MIP ad una risonanza magnetica del cervello.

Figura 4.10: Angiografia

Per le angiografie, è evidente che l'utilizzo di MIP è preferibile rispetto ad MPR poiché MIP fornisce un'immagine proiettata che evidenzia chiaramente e completamente i vasi sanguigni in un'unica vista. Al contrario, MPR offre solo sezioni singole di voxel, risultando decisamente meno efficace per mostrare l'intera rete vascolare in un'unica immagine [35].

Tra gli altri algoritmi, di cui non viene fornita una descrizione, vi è un approccio chiamato Minimum Intensity Projection (MinIP), il quale risulta utile per scopi diversi. Questo algoritmo svolge l'operazione opposta rispetto al MIP, estraendo i voxel con l'intensità più bassa invece di quelli con l'intensità più elevata.

4.3 Fasi dell'Elaborazione

Al contrario del processamento di immagini generiche, che ha come obiettivo migliorare l'estetica dell'immagine o creare arte, il solo scopo dell'elaborazione di immagini mediche è quello di migliorare l'interpretabilità dei contenuti rappresentati.

Seguendo l'ordine dell'articolo scientifico [27], si descrivono di seguito le fasi necessarie all'elaborazione di immagini mediche:

- *Image enhancement*: miglioramento generale dell'immagine ad esempio attraverso la rimozione di distorsioni, rumore e disomogeneità nello sfondo;
- *Image registration*: trasformazione spaziale dell'immagine in modo che sia confrontabile con un'immagine di riferimento;
- *Image segmentation*: identificazione dei contorni delle strutture anatomiche di interesse;
- *Quantification*: vengono determinate le proprietà geometriche delle strutture anatomiche, ad esempio volume, diametro e curvatura, o caratteristiche di composizione del tessuto;
- *Visualization*: rendering bidimensionale o tridimensionale dei dati;
- *Computer-aided detection*: rilevazione e ricerca, assistita da computer, di strutture e lesioni patologiche;

4.3.1 Image Enhancement: pre-filtraggio con kernel

Tipicamente, le immagini mediche contengono rumore, rumore che può portare ad un errata segmentazione. Due approcci diffusi per migliorare le immagini sono i filtri di *smoothing* (levigatura) e di *sharpening* (affilatura).

Il metodo di "levigatura" si concentra sulla riduzione del rumore presente nell'immagine. Tuttavia, esso può contaminare anche parti dell'immagine che non contengono rumore, rendendo più complesso il processo successivo di segmentazione dei contorni. Pertanto, viene utilizzato anche il metodo dello "sharpening", il quale mira ad aumentare il contrasto dei bordi, senza considerare l'eventuale rumore presente nell'immagine.

L'applicazione di questi filtri non è limitata solo alle immagini mediche ma riguarda qualsiasi tipo di immagine. Si noti che molti filtri per il miglioramento delle immagini si basano sull'operazione di convoluzione.

La convoluzione è una trasformazione delle immagini che coinvolge l'applicazione di un kernel su ciascun pixel, considerando anche i pixel adiacenti. Il kernel è rappresentato da una matrice, la cui dimensione e valori determinano la natura della trasformazione e, di conseguenza, il filtro specifico utilizzato.

Si considerino i seguenti due kernel:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{4.1}$$

$$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix} \tag{4.2}$$

Filtro di sharpening

La Matrice (4.1) rappresenta un kernel di sharpening.

Durante il processo, viene selezionato un pixel dall'immagine di origine, e mediante una somma ponderata dei valori dei pixel adiacenti, utilizzando come pesi i valori definiti nel kernel, si determina il nuovo valore del corrispondente pixel nell'immagine finale. Questa procedura viene applicata a ciascun pixel dell'immagine.

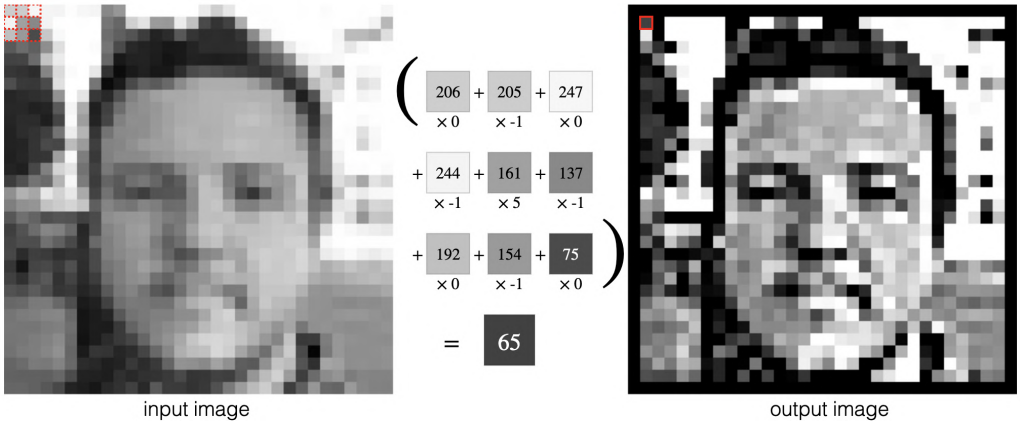


Figura 4.11: Applicazione del kernel 3x3 di sharpening ad un'immagine [38]

Come accennato, in campo medico questo ci aiuta nel segmentare meglio contorni di strutture o organi ignorando, e talvolta introducendo, del rumore (che si potrà equilibrare con lo smoothing).

(a) (b)

Figura 4.12: Radiografia alla colonna (a) ed Applicazione del filtro di sharpening (b)

Filtro di blur

La Matrice (4.2) consiste invece in un kernel di sfocatura. Il processo si basa sempre su una somma pesata, come nel caso dello sharpening.

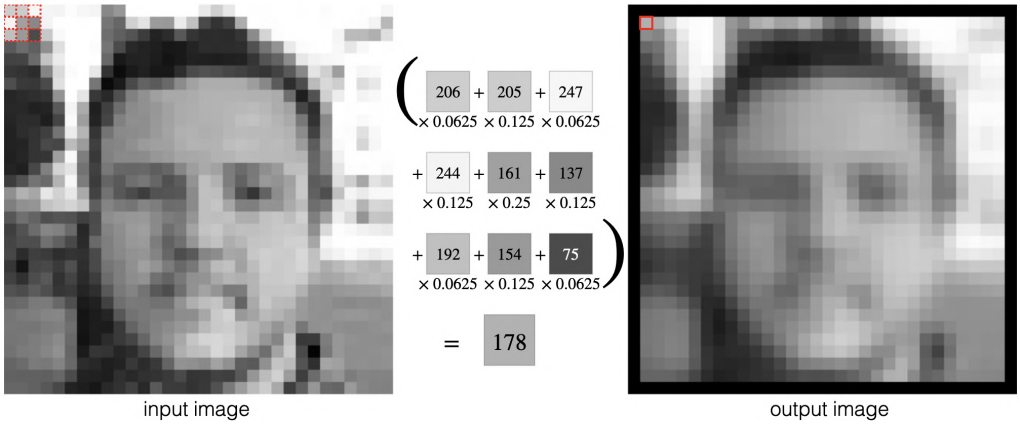


Figura 4.13: Applicazione del kernel 3x3 di sharpening ad un'immagine [38]

In generale, i filtri di blurring non risultano particolarmente utili nell'ambito medico. È stato riportato questo esampio, al solo scopo di evidenziare che l'utilizzo di kernel con valori diversi comporta l'applicazione sull'immagine di effetti tra loro differenti.

Filtro di smoothing

La tecnica più semplice per creare un kernel di smoothing è far si che questo sostituisca ad ogni pixel la media pesata dei diretti vicini. Tuttavia, il modo più comune utilizzato è quello che si basa su una convoluzione binomiale, dove i pesi del kernel vengono calcolati usando una formula di distribuzione gaussiana. Maggiore è la distanza dal centro del kernel, minore sarà il peso di quel pixel su quello preso in considerazione.

Filtri applicati ad immagini volumetriche

La convoluzione è un'operazione matematica generale che può essere applicata ad immagini bidimensionali rappresentate da pixel (come visto finora), o anche ad immagini tridimensionali rappresentati da voxel.

L'idea fondamentale rimane la stessa: un kernel, ovvero una matrice di valori, viene posizionato su ciascun voxel dell'immagine volumetrica, e il risultato della convoluzione determina il nuovo valore del voxel corrispondente nell'immagine filtrata.

4.3.2 Image Registration

L'image registration consiste in una normalizzazione, ovvero mira a standardizzare l'immagine acquisita, dato che questa può provenire da dispositivi diversi. Questo processo comporta attività tra cui l'allineamento, il ridimensionamento, la correzione dell'illuminazione e la compensazione delle distorsioni geometriche introdotte da angoli, distanze ed orientamenti diversi dei sensori. L'esecuzione di tali correzioni risulta estremamente utile in quanto semplifica le operazioni di segmentazione e, potenzialmente, di classificazione delle immagini.

4.3.3 Image Segmentation

La segmentazione costituisce una fase cruciale nell'elaborazione di immagini mediche. La sua finalità è separare le regioni di interesse tra loro e isolare tali regioni dallo sfondo. La segmentazione può essere trattata con pattern di riconoscimento, poiché consiste in una vera e propria classificazione dei singoli pixel (o voxel) dell'immagine.

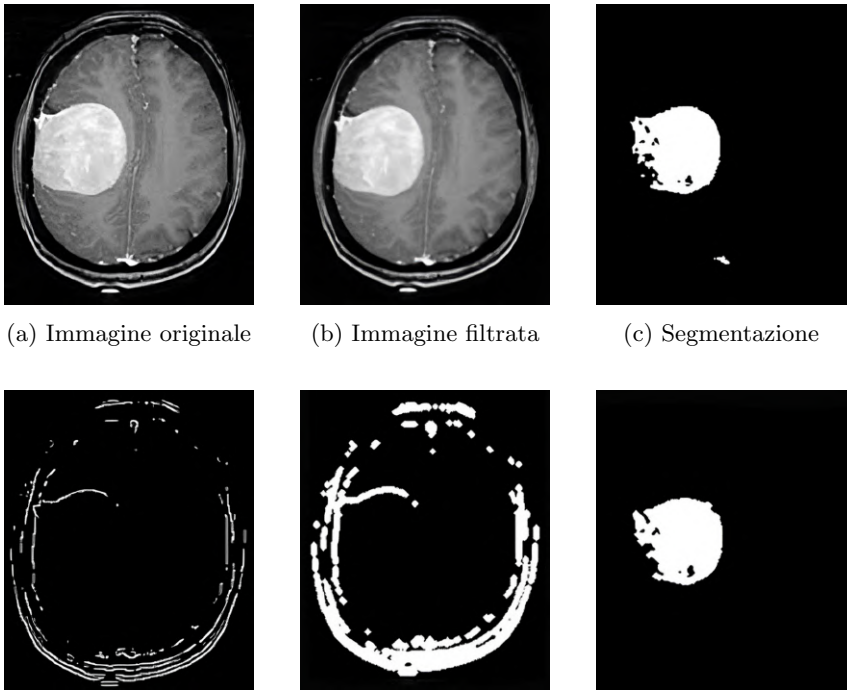
Questa fase, come tutte le altre, può essere effettuata sull'immagine 2D o 3D. Per semplicità di seguito, seguendo l'articolo scientifico di Neha Baraiya e Hardik Modi [39], vengono analizzati due possibili algoritmi di segmentazione rimanendo nel ambito di immagini bidimensionali.

Segmentazione con soglia (thresholding)

Un esempio in cui la segmentazione svolge un ruolo importante è nell'identificazione di tumori.

La segmentazione con soglia consiste in un semplice metodo che si basa sull'intensità dei pixel. Scelta la soglia di intensità, nel caso specifico i valori di intensità tipica di un tumore, si filtra l'immagine per ottenere solo la regione contenente il tumore.

Proprio perché ci si basa sul valore dei singoli pixel è possibile che vengano considerate anche delle regioni estranee. Per rimuovere queste regioni, una volta rilevati e dilatati i bordi vengono eliminate tutte le parti esterne ad essi.



(a) Immagine originale (b) Immagine filtrata (c) Segmentazione

(d) Bordi immagine (e) Bordi dilatati (f) Risultato finale

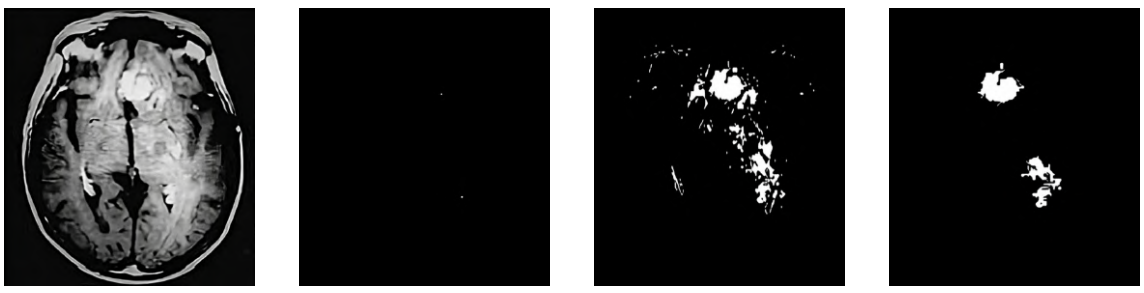
Figura 4.14: Segmentazione con thresholding di un tumore al cervello [39]

La Figura (4.14) mostra i diversi passaggi descritti. Si può vedere che in Figura (4.14c) è evidenziata una regione estranea al tumore, quindi per ottenere la corretta segmentazione finale si sottrae la Figura (4.14e) dalla Figura (4.14c).

Segmentazione con region growing

Il *region growing* è uno dei metodi più diffusi di segmentazione. Questa tecnica parte da uno o più pixel e raggruppa i vicini ad essi omogenei per formare una o più regioni. I criteri che stabiliscono quando due pixel sono considerati omogenei possono essere definiti inizialmente o adattati automaticamente man mano che l'algoritmo si estende ai pixel più lontani.

L'accuratezza di questo metodo dipende molto dalla scelta dei "punti di innesco" (seed points) da cui si inizia il processo di crescita della regione includendo i pixel vicini simili. Se tali punti vengono scelti in modo errato, possono portare a una segmentazione inappropriata.



(a) Immagine originale (b) Seed points (c) Region growing (d) Rumore eliminato

Figura 4.15: Segmentazione con Region growing [39]

La Figura (4.15) illustra i diversi passaggi descritti. Il risultato ottenuto dalla region growing, mostrato in Figura (4.15c), contiene del rumore che può influenzare le decisioni successive. Pertanto, è necessario eliminare tali parti attraverso operazioni morfologiche come l'erosione. Il risultato finale è visibile in Figura (4.15d).

Operazioni morfologiche: eliminazione del rumore e individuazione dei bordi

Secondo l'articolo [40], nell'ambito dell'elaborazione di immagini, gli operatori morfologici matematici effettuano elaborazioni sulla forma di un oggetto e sono tipicamente utilizzati per rimuovere imperfezioni introdotte dalla segmentazione.

Le operazioni morfologiche si basano seguenti termini:

- *Structuring Element*: matrice o un template di piccole dimensioni che viene posizionato in tutte le possibili posizioni dell'immagine e viene confrontato con i pixel che copre;
- *Miss*: succede quando nessun pixel della matrice sopracitata copre alcun pixel dell'oggetto;
- *Hit*: succede quando almeno un pixel della matrice sopracitata copre un pixel dell'oggetto;
- *Fit*: succede quando tutti i pixel della matrice sopracitata coprono solo pixel dell'oggetto;

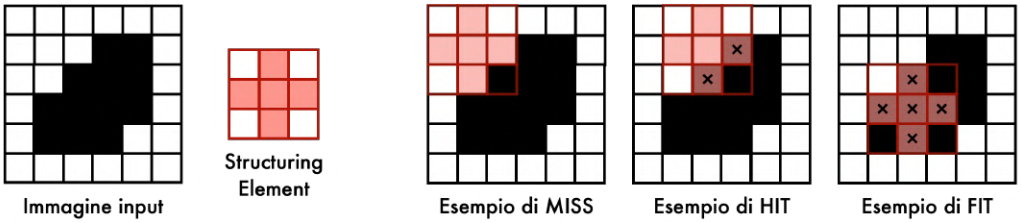


Figura 4.16: Illustrazione delle terminologie

Due operazioni ampiamente utilizzate nell'elaborazione di immagini sono l'erosione e la dilatazione. L'erosione consiste nella rimozione di pixel dai bordi dell'oggetto contenuto nell'immagine, mentre la dilatazione comporta l'aggiunta di pixel ai bordi dell'oggetto. Per entrambe le operazioni viene centrata la matrice su ciascun pixel dell'oggetto, e il valore di output del pixel centrato è calcolato utilizzando una specifica equazione.

L'erosione, illustrata in Figura (4.17), segue l'equazione:

$$Pixel(output) = \begin{cases} 1, & \text{se si verifica un "Fit"} \\ 0, & \text{altrimenti} \end{cases} \tag{4.3}$$

Questo implica che se la matrice è centrata su un pixel interno all'oggetto e si verifica il caso di "fit", tale pixel sarà mantenuto anche nell'immagine di output. Invece, nel caso in cui sia centrata su un pixel posizionato sul bordo dell'oggetto, sicuramente non ci sarà un "fit" completo e quindi quel pixel verrà "eroso" nell'immagine finale, risultando nella sua rimozione o riduzione.

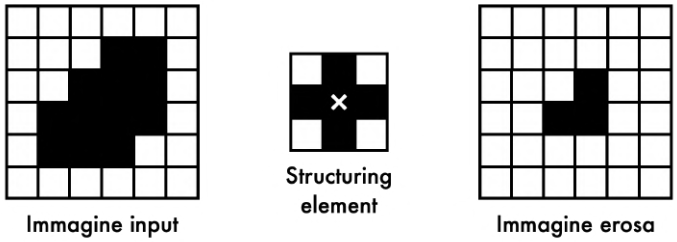


Figura 4.17: Esempio di erosione

L'erosione può essere utilizzata per separare oggetti connessi e per rimuovere prolungamenti sottili o piccole estensioni presenti nell'oggetto, aiutando ad ottenere una segmentazione più accurata e riducendo il rumore o dettagli indesiderati.

La dilatazione invece, illustrata in Figura (4.18), segue l'equazione:

$$Pixel(output) = \begin{cases} 1, & \text{se si verifica un "Hit"} \\ 0, & \text{altrimenti} \end{cases} \tag{4.4}$$

Questo significa che, indipendentemente dalla posizione del centro della matrice, è sufficiente che anche un solo pixel dell'oggetto sia compreso nello structuring element per poter espandere i bordi dell'oggetto.

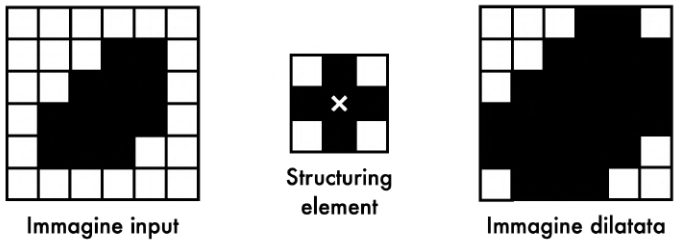


Figura 4.18: Esempio di dilatazione

Al contrario dell'erosione, la dilatazione può essere utilizzata per "riparare" interruzioni o piccoli vuoti all'interno di un oggetto, consentendo di riunire parti separate.

L'importanza di queste due tecniche è data dal fatto che la maggior parte delle operazioni morfologiche si basano su l'utilizzo di entrambe. Ad esempio si può effettuare un'estrazione dei contorni di un oggetto semplicemente calcolando l'immagine erosa e sottraendo questa all'immagine originale.

Riprendendo l'esempio utilizzato nelle Figure (4.17) e (4.18), in Figura (4.19) si può vedere il risultato della sottrazione di immagine di origine e immagine erosa.

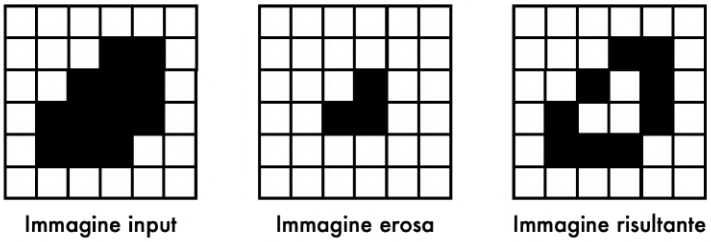


Figura 4.19: Esempio di estrazione dei bordi

4.3.4 Quantification e Visualization

Tutte le fasi analizzate riguardanti l'elaborazione delle immagini mediche sono eseguite in modo automatizzato dai software dedicati all'acquisizione e alla visualizzazione degli esami diagnostici.

Consideriamo, ad esempio, l'analisi delle strutture vascolari per la diagnosi di malattie cardiovascolari e altre patologie correlate. Attraverso tecniche di quantificazione, è possibile ottenere informazioni dettagliate sul diametro dei vasi, la presenza di eventuali stenosi (restringimenti), lo spessore delle pareti e le caratteristiche dei tessuti circostanti.

Una volta effettuate le misurazioni nella fase di quantificazione, la fase di visualizzazione si occupa di effettuare il rendering (in 2D o 3D) per fornire una rappresentazione visuale sia dell'immagine acquisita, sia delle misurazioni calcolate [27].

4.4 Standard DICOM

Negli anni '90 viene creato un consorzio formato da NEMA (National Electrical Manufactures Association) e ACR (American College of Radiology) con l'obiettivo di creare un formato standard di comunicazione in ambito medico. Nasce così lo standard non proprietario DICOM (Digital Imaging and Communication in Medicine) per la memorizzazione, visualizzazione e lo scambio di immagini mediche [41].

DICOM consiste in un insieme di layer e protocolli che mirano a standardizzare:

- La trasmissione e memorizzazione di informazioni ed immagini mediche;
- L'interrogazione e recupero di tali oggetti;
- L'esecuzione di azioni specifiche su tali oggetti;
- La gestione del flusso di lavoro;

4.4.1 Formato del file

Uno dei principali errori commessi nell'interpretazione del termine "Immagine DICOM" è considerarlo come un formato di compressione dell'immagine. In realtà, lo standard dati DICOM non introduce nuovi algoritmi di compressione, ma serve come tecnica per incapsulare e definire la codifica dei dati.

Oltre all'immagine effettiva, un file DICOM include varie informazioni di supporto, chiamate metadati. Questi forniscono informazioni legate al paziente, allo staff clinico, all'ospedale di riferimento, al dispositivo utilizzato per l'acquisizione, alle condizioni in cui è stato effettuato l'esame e altre informazioni cliniche rilevanti.

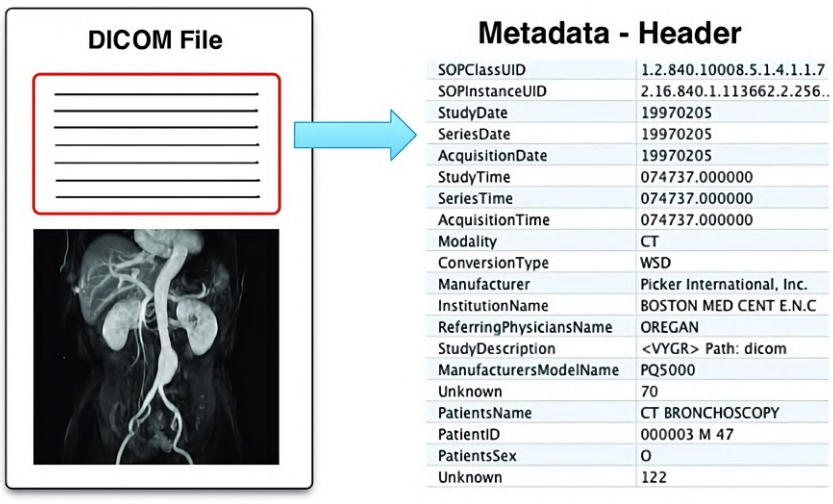


Figura 4.20: Struttura file DICOM semplificata con alcuni metadati [42]

Un file può contenere un numero variabile di elementi a seconda della tipologia di esame effettuato.

DICOM Information Model

Il DICOM Information Model si occupa di rappresentare elementi del mondo reale con degli oggetti che sono tra loro correlati.

La gerarchia Patient-Study-Series-Image, illustrata in Figura (4.21), riproduce esattamente ciò che accade nella realtà: il paziente (patient) può effettuare uno o più esami (study), e ciascun esame può avvenire con modalità diverse.
In base alla modalità scelta verrà prodotto uno o più insiemi di immagini (series) acquisite all'interno di un unico esame (ad esempio, diverse proiezioni o tagli di una scansione). Infine, "image" rappresenta l'immagine singola all'interno di una serie [42].

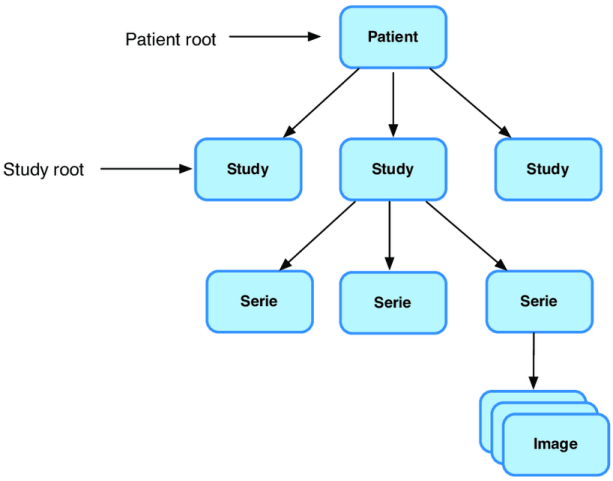


Figura 4.21: DICOM Information Model [42]

Visione ad alto livello: IOD

L'*Information Object Definitions* (IOD) consiste nella definizione di un oggetto, dove vengono specificati gli attributi (metadati) che devono essere contenuti in esso. Tra gli oggetti definiti tramite IOD ci sono quelli previsti dal DICOM Information Model, oltre ad altri oggetti di supporto.

Da una prospettiva di programmazione ad oggetti, l'IOD può essere paragonato ad un template o ad una classe. Essenzialmente, rappresenta una struttura astratta che delinea le caratteristiche e le proprietà di un oggetto DICOM, senza rappresentarne una specifica istanza.

Quando si crea un'immagine DICOM, ciò corrisponde all'istanziazione di una serie di IOD con valori specifici impostati per i loro attributi.

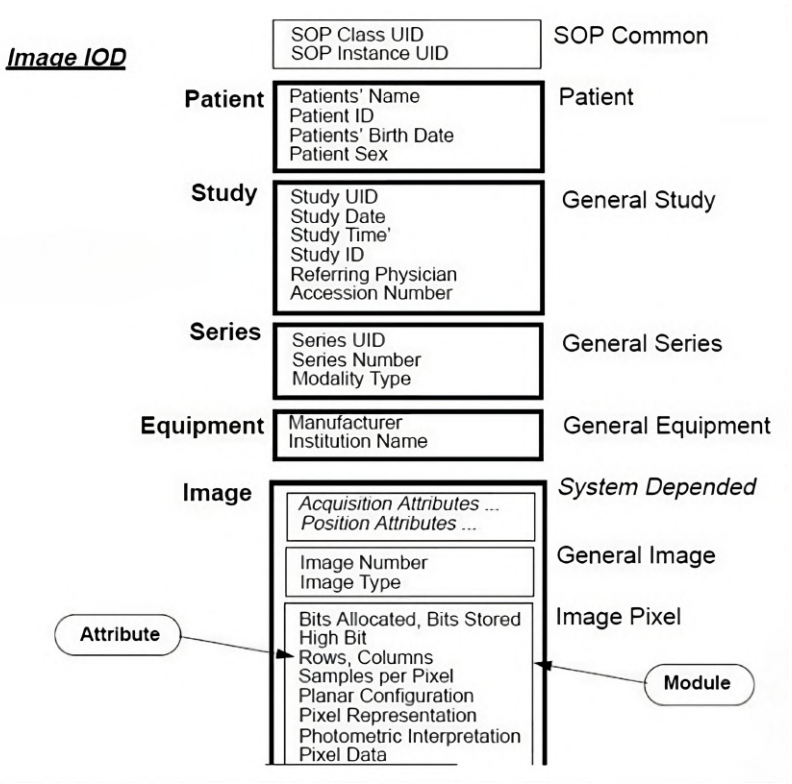


Figura 4.22: Struttura file DICOM ad alto livello [43]

Gli IOD trovano ampio utilizzo quando si accede a un file DICOM tramite software o librerie specializzate. Durante questo processo, i dati del file vengono letti e interpretati in base agli IOD utilizzati. Questo meccanismo permette di estrarre ed organizzare i dati in modo strutturato, semplificando notevolmente l'accesso alle diverse informazioni contenute nel file DICOM [41].

Visione a basso livello: DataSet

Dalla visione ad alto livello con l'astrazione del paradigma ad oggetti, si passa ora ai dettagli più tecnici della struttura dati. A basso livello, i metadati vengono organizzati in gruppi, ognuno dei quali rappresenta informazioni relative ad aspetti specifici.

Ciascun gruppo è identificato da un numero esadecimale univoco di quattro cifre. All'interno di ciascun gruppo, ciascun metadato è identificato da un altro numero esadecimale sempre di quattro cifre. Questi due numeri insieme formano il *DICOM Tag*.

La Tabella (4.3) riporta alcuni tag di metadati con i rispettivi nomi.

Tag	**Nome tag**
(0008, 0020)	Data dell'esame
(0008, 0030)	Ora dell'esame
(0010, 0010)	Nome del paziente
(0010, 0020)	ID del paziente
(0010, 0030)	Data di nascita del paziente
(0010, 0040)	Sesso del paziente

Tabella 4.3: Alcuni tag DICOM in formato (Gruppo, Elemento)

Ogni metadato (o attributo, o data element) DICOM, all'interno di un file, è organizzato in un formato: tag, VR, length e value. Di seguito si riporta una descrizione di ciascun campo:

- *Tag*: identificativo univoco all'interno di un file, è formato da due valori di 16 bit che rappresentano il gruppo e il numero dell'elemento;
- *Value Representation*: specifica l'encoding, ovvero il tipo, dei valori;
- *Length*: definisce la lunghezza del campo value per l'attributo;
- *Value*: contiene il valore memorizzato per l'attributo;

Ignorando il campo VR e length, si ha che un file DICOM comprende al suo interno un dataset composto da una serie di coppie chiave-valore, dove la chiave è rappresentata dal tag e il valore è ciò che è nel campo value [42].

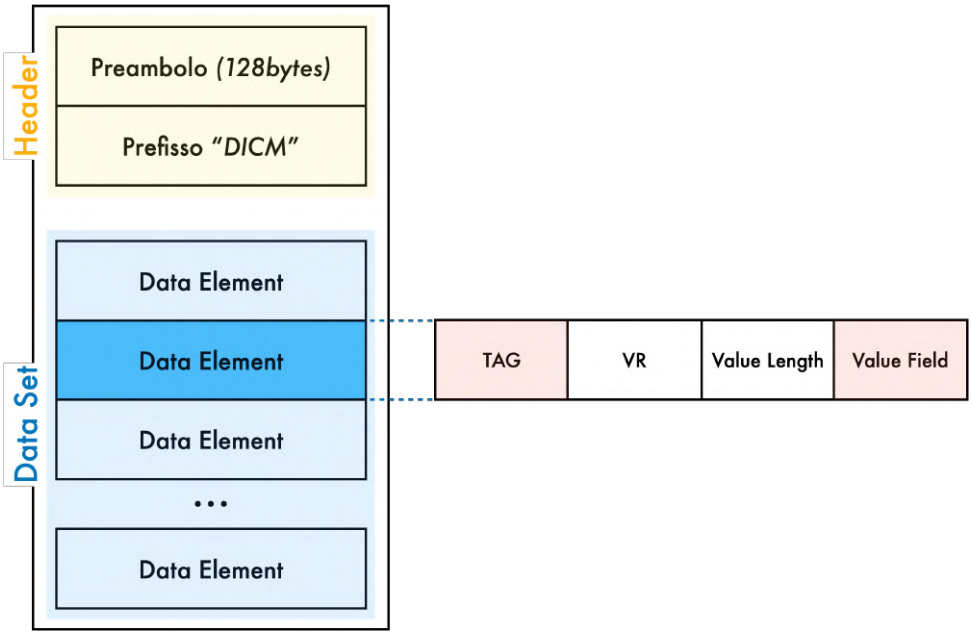


Figura 4.23: Struttura di un file DICOM

Si può osservare in Figura (4.23) che oltre al dataset contenente tutti gli attributi, ordinati per gruppo in ordine crescente, vi è un header composto da 128 bytes nulli di preambolo e un prefisso contenente i caratteri 'D', 'I', 'C' ed 'M'.

Dati dell'immagine

Non è stato ancora descritto dove si trova l'immagine vera e propria all'interno del file DICOM. Le informazioni riguardanti la rappresentazione dell'immagine e i valori effettivi dei pixel sono contenute in gruppi specifici di data element, ovvero nel gruppo 0028 e nel gruppo 7FE0.

Il gruppo 0028 contiene attributi legati alla presentazione dell'immagine, tra cui si trovano:

- (0028, 0010) Rows: numero di righe nell'immagine;
- (0028, 0011) Columns: numero di colonne nell'immagine;
- (0028, 0100) Bits Allocated: numero di bit allocati per ogni singolo pixel;

Il gruppo 7FE0 contiene i valori dei pixel, esistono elementi differenti per formati diversi:

- (7FE0,0008) Float Pixel Data;
- (7FE0,0009) Double Float Pixel Data;
- (7FE0, 0010) Pixel Data: contiene un array formato dai valori dei pixel dell'immagine espressi in binario. L'ordine dei pixel segue la sequenza sinistra-destra e alto-basso;

Nonostante le informazioni inerenti alla rappresentazione e ai valori dell'immagine siano organizzate in due gruppi distinti, dal punto di vista concettuale, ci si aspetta che un "*Image IOD*" contenga le definizioni degli attributi sia del gruppo 0028 che del 7FE0. Questa deduzione è esatta e confermata nella Figura (4.22). Anche se a livello implementativo fanno parte di gruppi diversi, a livello logico sono contenuti in un unico oggetto.

Di solito, le immagini vengono incluse senza compressione, mantenendo la loro forma originale. Questo perché spesso si richiede una risoluzione molto elevata, ragione per cui i file tendono ad essere di grandi dimensioni nonostante le immagini siano in scala di grigi.

Nell'Appendice (D) è mostrato uno script Python in grado di leggere e stampare tutti i metadati di un file DICOM, e ricostruirne l'immagine.

4.4.2 Protocollo di Rete e Servizi

Nelle pagine precedenti è stato descritto il formato standard dei file DICOM. In questa sezione si vuole dare una semplice introduzione allo standard di comunicazione per lo scambio di tali file [42].

DICOM stabilisce diversi servizi che seguono un'architettura client-server. In questo scenario si definiscono SCP (*Service Class Provider*) e SCU (*Service Class User*).

Ad esempio l'attrezzatura che si occupa di produrre l'immagine e di farla memorizzare in un archivio PACS (*Picture archiving and communication system*) si comporterà da SCU, in quanto utilizza un servizio. Al contrario, l'archivio che si occupa di ricevere e salvare le immagini, sta offrendo un servizio e quindi agisce in modalità SCP.

Per stabilire una comunicazione tra dispositivi DICOM, il primo passo è la procedura di *DICOM association*, in cui si negoziano alcuni parametri come il tipo di informazione da trasferire, la codifica e la durata dell'associazione.

Servizi più diffusi

Successivamente alla negoziazione, possono essere utilizzati i servizi tra SCU ed SCP, ognuno dei quali ha un insieme di comandi associati. I più importanti sono riportati in Tabella (4.4).

Servizio:	**Descrizione:**	**Comando DICOM:**
Verification	Controlla lo stato di un device DICOM	C-ECHO
Storage	Invia un'immagine all'archivio PACS	C-STORE
Query/Retrieve	Ricerca su database e ottiene immagine	C-FIND e C-MOVE

Tabella 4.4: Servizi DICOM più comuni e rispettivi comandi

Il servizio di verifica, mediante il comando C-ECHO, permette all'SCU di controllare la comunicazione e di verificare se l'SCP sta lavorando in modo corretto .

Lo Storage, in Figura (4.24), è un servizio che permette all'SCU di far memorizzare una o più immagini in un archivio. Per ogni singola immagine, è necessario effettuare una richiesta contenente l'intero file DICOM tramite il comando C-STORE. L'archivio, che svolge il ruolo di SCP del servizio, invierà una risposta per ogni richiesta ricevuta.

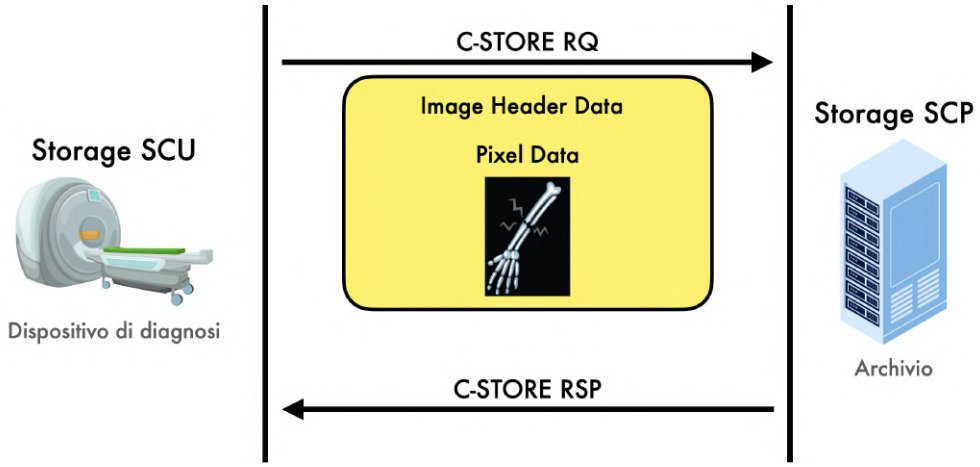


Figura 4.24: Servizio di Storage

La combinazione dei servizi Query e Retrieve permette di cercare ed ottenere un'immagine dall'archivio. Mediante il servizio Query, l'SCU può effettuare una ricerca con il comando C-FIND utilizzando metadati come filtri di ricerca, ad esempio nome del paziente, data dell'esame e tipo di esame. La Figura (4.25) illustra un esempio di query in cui si cercano esami effettuati nella data odierna su pazienti con nomi che iniziano con la lettera "A". L'SCP si occupa di fornire una o più risposte in base alla query effettuata.

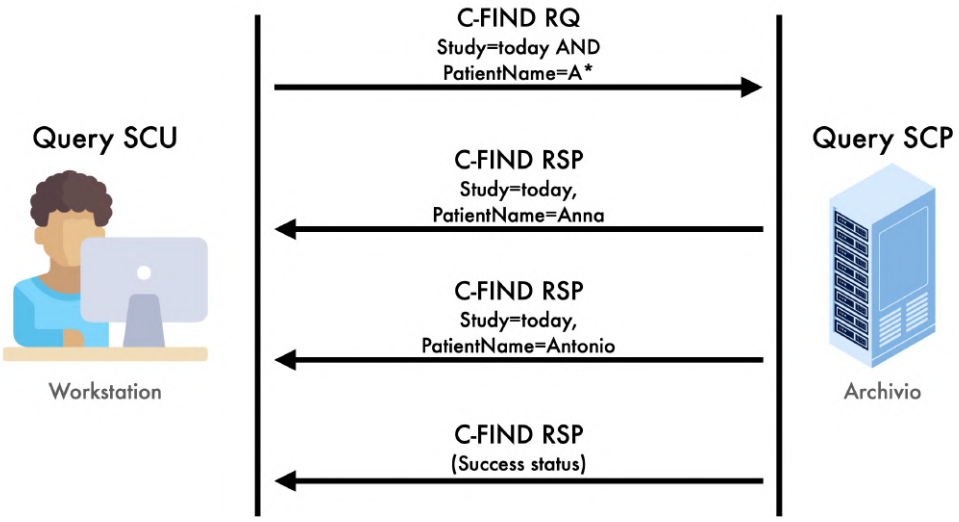


Figura 4.25: Servizio di Query

Il metodo Retrieve, invece, permette all'SCU di ottenere l'immagine da un SCP. L'operazione di recupero può essere eseguita utilizzando il comando C-MOVE o C-GET.

Nel caso di C-MOVE, viene sfruttato il comando C-STORE per effettuare un trasferimento inverso rispetto a quello descritto precedentemente.

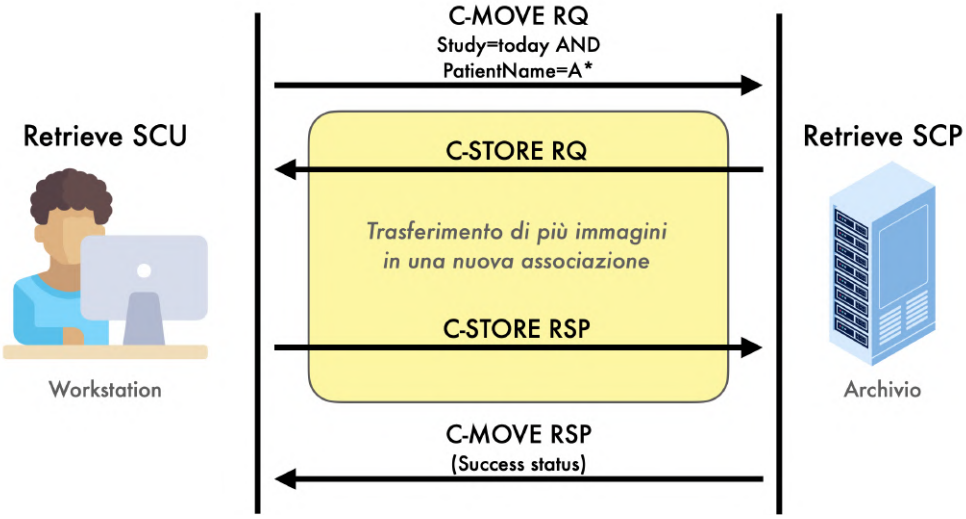


Figura 4.26: Servizio di Retrieve

WADO

WADO (*Web Access to DICOM persistent Objects*) è un meccanismo che permette di accedere e visualizzare oggetti DICOM direttamente da pagine HTML o documenti XML, attraverso il protocollo HTTP/HTTPS.

Un servizio web su Internet può sfruttare questa estensione del protocollo per creare accessi a dati tra strutture diverse. Essendo un servizio dedicato al processo di recupero il suo funzionamento è paragonabile a quello del comando C-MOVE, ma in questo caso, essendo online, vi sono dei formati di query e MIME type standardizzati.

File DICOM su CD

Esistono diverse situazioni in cui il file DICOM deve essere esportato su un dispositivo offline, come nel caso di consegna del referto su disco.

Anche se la struttura delle cartelle all'interno di un CD può variare, generalmente tra i file DICOM contenuti, è incluso un file strutturato chiamato DICOMDIR che indicizza tutto il contenuto in una struttura ad albero. Tale albero è formato da quattro livelli di gerarchia corrispondenti a quelli del DICOM Information Model (PATIENT - STUDY - SERIES - IMAGE).

Purtroppo, a differenza di altri formati di immagini, i file DICOM non sono riconosciuti come tipo immagine da sistemi operativi come Windows. Per questo motivo, spesso si deve ricorrere a strumenti terzi che interpretano il file DICOM e ne consentono la visualizzazione.

4.4.3 Conversione di formato

Le immagini DICOM sono ampiamente utilizzate, ma presentano due svantaggi significativi: dimensioni dei file elevate e la necessità di un software specifico per visualizzarle.

È possibile convertire queste immagini in altri formati più popolari, come JPEG, JPEG 2000, TIFF, GIF e PNG. Tale conversione, spesso può essere eseguita da una workstation di diagnostica o, al di fuori dell'ambiente di radiologia, da un client Web.

Se da una parte i formati più diffusi richiedono meno spazio per la loro memorizzazione, dall'altra la trasformazione presenta alcuni svantaggi significativi. In primo luogo, si verifica una perdita di informazioni diagnostiche poiché tutti i metadati vengono eliminati durante il processo di trasformazione. Inoltre, vi è una limitazione sulla post-elaborazione che può essere eseguita sull'immagine convertita a causa della degradazione di qualità dell'immagine. Un ultimo problema riguarda la compressione lossy utilizzata in formati come JPEG, che può generare artefatti nell'immagine, compromettendo ulteriormente la sua qualità e precisione [44].

4.5 Problemi di Sicurezza e Privacy

Come l'imaging medico ricopre un ruolo fondamentale nei dati sanitari, la stessa importanza deve essere attribuita alla privacy e alla confidenzialità di tali informazioni.

La privacy dei dati di un paziente è necessaria a mantenere una relazione di fiducia tra paziente e medico. Un paziente che si fida del medico può aiutare il medico a raccogliere dati più accurati, i quali a loro volta si traducono in diagnosi più precise e conclusioni mediche più accurate.

Questa sezione trae ispirazione dal Capitolo 4 del libro "*Data Protection and Privacy in Healthcare*" [45], il quale offre una descrizione delle minacce e delle misure di protezione necessarie per affrontare i problemi di privacy nelle immagini mediche. Nella seconda parte, sarà invece presentato un sommario delle norme europee che regolano la privacy nel campo medico.

4.5.1 Minacce alla privacy

Come evidenziato precedentemente, il formato di file medici più diffuso è il DICOM, il quale include diverse informazioni, tra cui alcuni campi di *protected health information* (PHI).

Tra le PHI che possono violare la privacy del paziente abbiamo: nomi, posizioni geografiche, date legate al soggetto, numeri di telefono, email, indirizzi IP, numeri di licenza, identificatori biometrici, foto ed identificatori univoci. Un esempio di metadati configurabili come PHI sono i tag dell'intero gruppo 0010 che contengono informazioni sul paziente. È fondamentale che questi dati siano trattati con la massima riservatezza e condivisi solo tra coloro che hanno effettiva necessità di conoscerli.

Le minacce alla privacy dei pazienti, a seguito di una diagnosi per immagini, possono essere suddivise per tipologia e sono riportate nella Tabella (4.5).

Tipo di minaccia:	**Descrizione:**
Diretta	Rivela una condizione o delle informazioni private
Ricollegabile	Metadati permettono di risalire al paziente
Inferenza esistenziale	L'esistenza dell'immagine suggerisce la presenza di una certa condizione o di un'informazione privata
Identificazione	Un'immagine ad alta risoluzione rivela alcuni dettagli del soggetto che permettono il suo riconoscimento

Tabella 4.5: Tipi di minacce alla privacy nelle immagini mediche

4.5.2 Protezioni per la privacy

Il principale obiettivo delle protezioni, come detto, è limitare l'accesso alle Informazioni di Salute Personali (PHI) del paziente, consentendo l'accesso solo a medici e professionisti sanitari coinvolti nella diagnosi e/o nel trattamento. Di seguito vengono elencati alcuni meccanismi che contribuiscono a raggiungere tale obiettivo.

Protezione tramite Cifratura

Durante la trasmissione delle immagini mediche, è fondamentale garantire la confidenzialità e l'integrità delle informazioni. A tal fine, è necessario cifrare le PHI del paziente. Il processo di cifratura protegge i dati sensibili rendendoli inaccessibili e non modificabili da terze parti non autorizzate, garantendo così la sicurezza delle informazioni durante la trasmissione.

Protezione tramite Anonimizzazione

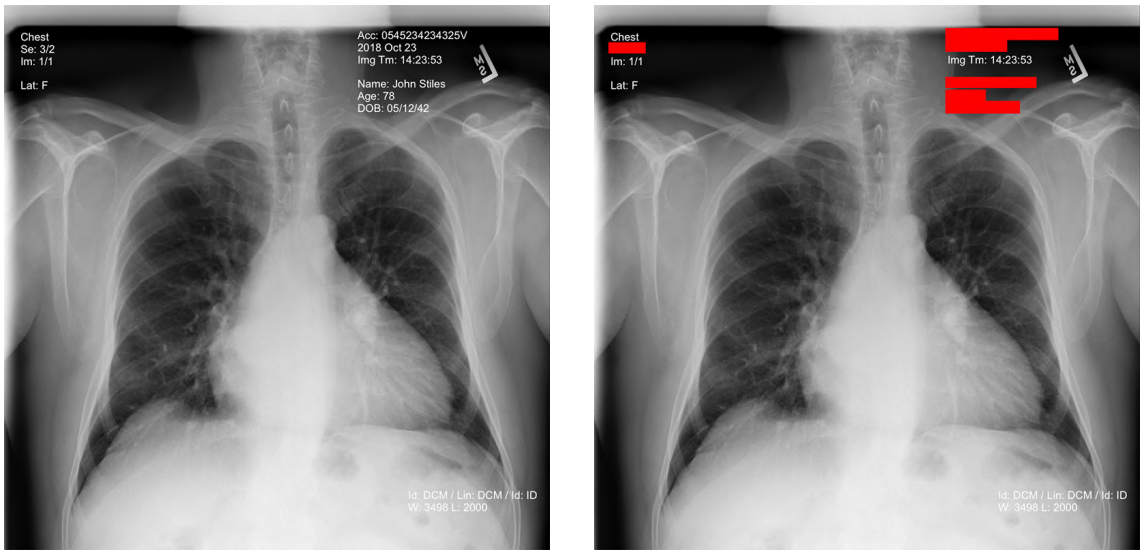
L'anonimizzazione è il processo mediante il quale vengono rimosse o modificate le informazioni identificative di un soggetto, al fine di preservarne l'anonimato e proteggerne la privacy.

Un caso concreto in cui si ricorre all'anonimizzazione riguarda le immagini messe a disposizione per scopi di ricerca e/o studio, dove è necessario rimuovere i dati sensibili contenuti.

Una soluzione per la rimozione può essere l'esportazione dell'immagine DICOM in formati generici come JPEG, TIFF o PNG, ma questo comporta le altre problematiche precedentemente descritte, come la perdita degli altri metadati, la compressione e la limitazione sulla post-elaborazione. Un'alternativa meno estrema può essere quella di cancellare semplicemente i valori contenuti in dei tag specifici.

Protezione dei dati visibili nell'immagine

Un altro aspetto da considerare riguarda certe tipologie di esami, dove alcune PHI possono essere visibili direttamente nell'immagine. In tali casi, è importante eliminare dai pixel solo i dati sensibili, mantenendo le altre annotazioni per la diagnostica inalterate.



(a) Immagine non anonimizzata

(b) Immagine anonimizzata

Figura 4.27: Esempio di anonimizzazione con servizi AWS [46]

Questa operazione può essere fatta in modo manuale oppure esistono strumenti che sfruttano il machine learning per riconoscere ed oscurare automaticamente il testo sensibile.

A questo scopo, *Amazon AWS* offre dei servizi come *Amazon Comprehend Medical* e *Amazon Rekognition*. Il primo è un servizio di elaborazione del linguaggio naturale che analizza testi medici per identificare le informazioni personali, mentre il secondo è un servizio in grado di estrarre testo da immagini e video.

La Figura (4.27) mostra il risultato dell'integrazione di questi servizi, che permettono di automatizzare il processo di anonimizzazione delle immagini mediche, riducendo tempo e sforzi richiesti per la revisione e l'editing manuale delle immagini [46].

Protezione tramite Cifratura e Steganografia

Il metodo proposto da Ming Yang et al. utilizza una combinazione di cifratura e steganografia.

Le informazioni sensibili vengono cifrate con l'algoritmo RSA e successivamente nascoste all'interno dell'immagine.

Questo approccio permette di ottenere due versioni distinte dell'immagine medica: una versione senza le PHI visibili, per persone non autorizzate, e una versione con le informazioni sensibili visibili per coloro che sono autorizzati a vederle. Infatti, solo chi è in possesso della chiave di decifratura può estrarre il testo nascosto dall'immagine, decifrarlo e visualizzare la versione dell'immagine contenente le informazioni sensibili.

4.5.3 Sicurezza e vulnerabilità nelle immagini DICOM

Attualmente, sebbene lo standard DICOM offra facilitazioni per l'utilizzo della cifratura, questa non è obbligatoria. La decisione di implementarla o meno è una politica scelta dalle singole istituzioni sanitarie e dai fornitori dei dispositivi medici.

È importante sottolineare che la sicurezza delle reti e degli archivi degli istituti sanitari, gioca un ruolo fondamentale nella protezione delle immagini stesse. Ad oggi, si sono osservate due tipologie di attacchi: di accesso e di injection.

Gli attacchi di accesso si verificano quando persone non autorizzate riescono ad accedere a server DICOM non protetti, ottenendo così accesso alle immagini e alle informazioni contenute. Questi attacchi possono anche coinvolgere utenti legittimi che concedono involontariamente accesso a terze parti non autorizzate.

Gli attacchi di injection, d'altra parte, si verificano a seguito di un attacco di accesso e mirano a modificare le immagini stesse, introducendo o rimuovendo anomalie direttamente nell'immagine tramite elaborazioni con machine learning o nascondendo del malware all'interno dei file.

Il Gruppo di Lavoro 14 (WG-14) ha la responsabilità di gestire tutti gli aspetti legati alla sicurezza dello standard DICOM. Per prevenire futuri attacchi e vulnerabilità, il WG-14 si sta concentrando sullo sviluppo di un sistema di gestione di chiavi e certificati, un sistema di firma digitale fatta dal creatore dell'immagine e la rimozione di preamboli non desiderati per evitare la diffusione di malware.

4.5.4 Regolamentazioni europee in ambito sanitario

Negli Stati Uniti è vigente l'Health Insurance Portability and Accountability Act (HIPAA) del 1996, una legge federale che stabilisce i requisiti riguardanti la privacy e la sicurezza dei dati per le organizzazioni incaricate di proteggere i dati sanitari protetti dei privati.

Nell'Unione Europea, invece, è in vigore il GDPR (Regolamento generale sulla protezione dei dati), un regolamento entrato in vigore il 24 maggio 2016, che disciplina il trattamento dei dati personali e la tutela della privacy in generale (non è specifico per i dati sanitari).

Di seguito vengono riportati in forma parziale alcuni articoli rilevanti per il trattamento di immagini mediche, del suddetto Regolamento. Per informazioni più dettagliare si rimanda al provvedimento del Garante n. 55 del 7 marzo 2019 [9091942].

Definizioni

L'**art. 4 GDPR** fornisce una serie di definizioni e riporta quanto segue:

> "Dati relativi alla salute: i *dati personali* attinenti alla salute fisica o mentale di una persona fisica, compresa la prestazione di servizi di assistenza sanitaria, che rivelano informazioni relative al suo stato di salute"

Dove per "dati personali" si intende:

> "qualsiasi informazione riguardante una persona fisica identificata o identificabile; [...]"

Per completezza di informazione si riporta, dal medesimo articolo, anche la definizione di:

> "Trattamento: qualsiasi operazione o insieme di operazioni, compiute con o senza l'ausilio di processi automatizzati e applicate a dati personali o insiemi di dati personali, come la raccolta, la registrazione, l'organizzazione, la strutturazione, la conservazione, l'adattamento o la modifica, l'estrazione, la consultazione, l'uso, la comunicazione mediante trasmissione, diffusione o qualsiasi altra forma di messa a disposizione, il raffronto o l'interconnessione, la limitazione, la cancellazione o la distruzione"

Il **Motivo 35 GDPR**, specifica che nei "dati relativi alla salute" rientrano anche le immagini:

> "Nei dati personali relativi alla salute dovrebbero rientrare tutti i dati riguardanti lo stato di salute dell'interessato che rivelino informazioni connesse allo stato di salute fisica o mentale passata, presente o futura dello stesso. Questi comprendono [...] le *informazioni risultanti da esami e controlli effettuati su una parte del corpo* [...]".

Trattamento e consenso

I "dati rientranti in particolari categorie", definiti comunemente "dati sensibili", tra cui rientrano quelli relativi alla salute, rivelano dettagli intimi della persona, pertanto sono soggetti a tutela rafforzata ai sensi dell'(**art. 9 GDPR**).

L'articolo in questione vieta in generale il trattamento dei dati relativi alla salute, ma consente l'elaborazione di tali dati per finalità di: cura (art. 9, par. 2, lett. h), motivi di interesse pubblico o finalità di governo (art. 9, par. 2, lett. i) e ricerca nel pubblico interesse.

Sono autorizzati a trattare dati sanitari gli esercenti di una professione sanitaria e gli organismi sanitari pubblici. Le professioni sanitarie riconosciute dal Ministero della Salute comprendono ad esempio medici, odontoiatri, infermieri, psicologi, etc...

Consenso

I trattamenti che: sono essenziali per il raggiungimento di una o più finalità determinate ed esplicitamente connesse alla cura della salute; e sono effettuati da (o sotto la responsabilità di) un professionista sanitario soggetto al segreto professionale o da altra persona anch'essa soggetta all'obbligo di segretezza NON richiedono il consenso al trattamento dei dati da parte dell'interessato.

Esempi di trattamenti in ambito sanitario che non rientrano nelle ipotesi sopra descritte e, che quindi richiedono il consenso esplicito dell'interessato (art. 9, par. 2, lett. a) sono:

- Utilizzo di app attraverso cui vengono raccolti dati anche sanitari dell'interessato, per finalità diverse dalla telemedicina, oppure quando ai dati dell'interessato possono avere accesso soggetti diversi da professionisti sanitari o altri soggetti tenuti al segreto professionale;
- Fidelizzazione della clientela effettuata dalle farmacie attraverso programmi di accumulo punti, al fine di fruire di servizi o prestazioni accessorie;
- Finalità promozionali o commerciali (es. promozioni su programmi di screening)
- Consegna di un referto online

Informativa

Prima di ottenere il consenso, il titolare deve fornire all'individuo un'adeguata informativa sulla privacy, in modo che quest'ultimo possa prendere una decisione informata riguardo all'utilizzo dei propri dati personali.

Il principio di trasparenza previsto dall'**art. 5 GDPR**, impone ai titolari di informare l'interessato sui principali elementi del trattamento, al fine di renderli consapevoli sulle caratteristiche dello stesso.

L'informativa deve essere concisa, trasparente, intelligibile, facilmente accessibile e scritta con linguaggio semplice e chiaro. Deve indicare il soggetto titolare del trattamento, le finalità, le modalità, la natura obbligatoria o facoltativa del conferimento dei dati, i soggetti a cui possono essere comunicati e i diritti degli interessati (**art. 13 e 14 GDPR**).

Conservazione

I documenti contenenti dati sanitari devono essere conservati in archivi ad accesso controllato con un "*livello di sicurezza adeguato al rischio*" (**art. 32 GDPR**).

I tempi di conservazione, qualora non siano fissati da specifiche norme, spetta al titolare definirli in base alla finalità del trattamento e devono essere indicati nell'informativa.

Altri obblighi

La nomina di un Responsabile della Protezione dei Dati (RPD), prevista dall'**art. 37 GDPR**, è obbligatoria per gli organismi pubblici (es: struttura appartenente al SSN) e nel caso di trattamenti su larga scala (come può avvenire per ospedali e case di cura).

Anche il mantenimento di un registro delle attività di trattamento, ovvero di un documento contenente le informazioni relative alle operazioni di trattamento svolte, è obbligatorio ai sensi dell'**art. 30 GDPR**.

Capitolo 5

Sviluppo di un framework per la condivisione sicura di file DICOM

In questo capitolo, viene presentato il progetto di tesi implementato, il quale si basa su concetti precedentemente esaminati nei vari capitoli.

5.1 Introduzione

Come già evidenziato, i file DICOM non sono soggetti a obblighi di cifratura, pertanto la sicurezza di tali file dipende dalle misure di sicurezza implementate nelle reti e nei database degli istituti sanitari. Tuttavia, spesso, queste misure si rivelano insufficienti per proteggere gli archivi da potenziali attacchi di accesso non autorizzato e di injection di dati malevoli o contraffatti.

5.1.1 Obiettivo

L'obiettivo del presente progetto è la creazione di uno strumento che consenta la condivisione e la memorizzazione sicura dei risultati degli esami diagnostici, garantendo integrità e riservatezza delle informazioni e preservando la privacy dei pazienti. Il tool in questione opererà direttamente sui file DICOM, utilizzando la steganografia per proteggere i metadati sensibili (PHI), e la crittografia visuale per preservare la sicurezza dei pixel data.

5.1.2 Architettura

Architettura attuale

L'attuale architettura impiegata, che copre l'intero processo di trattamento del referto, dall'acquisizione alla sua consegna, è illustrata schematicamente in Figura (5.1).

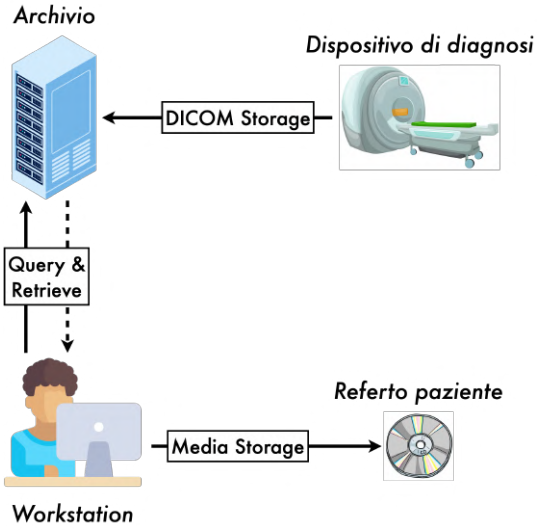


Figura 5.1: Architettura attuale

Inizialmente, il dispositivo diagnostico effettua l'acquisizione e genera uno o più file DICOM, che vengono memorizzati in un archivio dedicato. In una fase successiva, un operatore designato è incaricato del recupero di tali file e della loro copia su un supporto destinato al paziente. Il supporto più largamente utilizzato è il CD, anche se è possibile consegnare i referti attraverso mezzi alternativi come chiavette USB o tramite email.

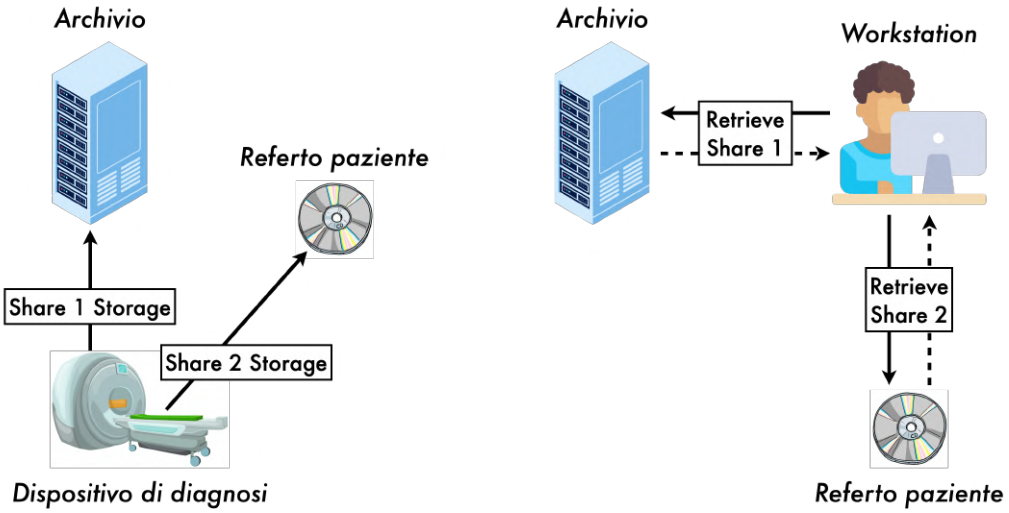
Si noti che lo scambio di referti tramite email è una pratica che può verificarsi non solo tra medico e paziente, ma anche tra istituti sanitari diversi o tra medici che necessitano l'accesso ai risultati degli esami. Pertanto, è essenziale garantire la sicurezza non solo per i file DICOM archiviati o forniti direttamente al paziente, ma anche per quelli scambiati attraverso canali alternativi.

Architettura proposta

La architettura sicura proposta prevede la suddivisione del referto in due parti distinte: una che viene memorizzata nell'archivio della struttura e un'altra che viene consegnata direttamente al paziente.

Quando un medico desidera consultare il risultato dell'esame, non può semplicemente effettuare una richiesta all'archivio, poiché quest'ultimo contiene solo una parte della diagnosi. È invece necessario che ottenga l'approvazione del paziente, che è rappresentata dalla richiesta ed ottenimento dalla sua parte di referto. Avendo entrambi gli share a disposizione, il medico sarà in grado di visualizzare l'esito.

Entrambe queste fasi sono illustrate nella Figura (5.16).



(a) Acquisizione e memorizzazione

(b) Accesso e ricostruzione

Figura 5.2: Architettura proposta

Tale autorizzazione sarà necessaria anche se avviene la condivisione di metà referto tra due medici o istituti sanitari differenti. In questa situazione, il primo attore potrà condividere solamente lo share in suo possesso. Il secondo attore, invece, per accedere al risultato completo dell'esame, dovrà richiedere al paziente l'altra metà.

Se da una parte, questo meccanismo, elimina la possibilità di consultazioni e/o manipolazioni non autorizzate ai referti, dall'altra viene affidata una responsabilità ad entrambe le parti coinvolte. Tale responsabilità consiste nel conservare con cura il proprio share, poiché, in caso di smarrimento di una delle due parti, non sarà possibile ricostruire il referto.

5.2 Cifratura

L'intero processo di cifratura può avvenire in due modalità distinte. L'unica differenza tra le due è data dall'insieme di metadati estratti ed utilizzati nella fase di steganografia. In entrambi i casi, il punto di partenza è costituito dal file DICOM che si intende trasmettere o memorizzare in modo cifrato.

Nella prima modalità, denominata nel codice '*partial metadata mode*', vengono estratti dal file DICOM i valori dei *pixel data*, contenuti nel campo con tag (7FE0,0010), insieme a una serie di metadati ritenuti sensibili (PHI).

Successivamente, l'insieme dei valori dei pixel viene impiegato per la creazione di un'immagine in formato PNG. Tale procedura consiste nella conversione dell'immagine DICOM in un formato standard, portando con sé le problematiche precedentemente descritte, quali la perdita dei metadati DICOM.

Dopo la fase di estrazione dei dati, attraverso un processo di steganografia, i PHI salvati inizialmente vengono nascosti all'interno dell'immagine PNG, dando origine ad uno "*stego object*".

Infine, l'oggetto ottenuto è sottoposto a un processo di visual secret sharing, il quale produce due share distinti: uno destinato all'archiviazione presso la struttura sanitaria e l'altro consegnato al paziente.

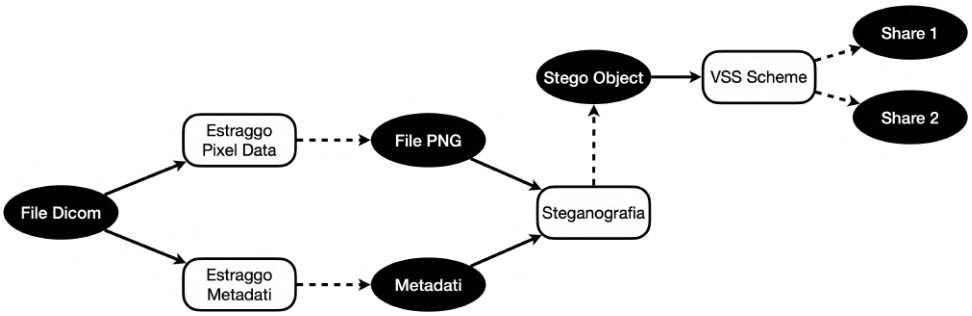


Figura 5.3: Processo di cifratura

La seconda modalità, denominata '*all metadata mode*', segue il medesimo iter descritto per la modalità parziale. Tuttavia, anziché occultare esclusivamente i metadati sensibili tramite steganografia, questa modalità nasconde tutti i metadati. In pratica, questa modalità, permette di salvare tutte le informazioni contenute in un file DICOM incorporandole nei pixel dell'immagine PNG ottenuta.

5.3 Decifratura

Il processo di decifratura consiste nella sequenza di operazioni inverse rispetto alla cifratura: si parte da due share e si cerca di ricostruire il file DICOM di partenza o alcune delle informazioni in esso contenute.

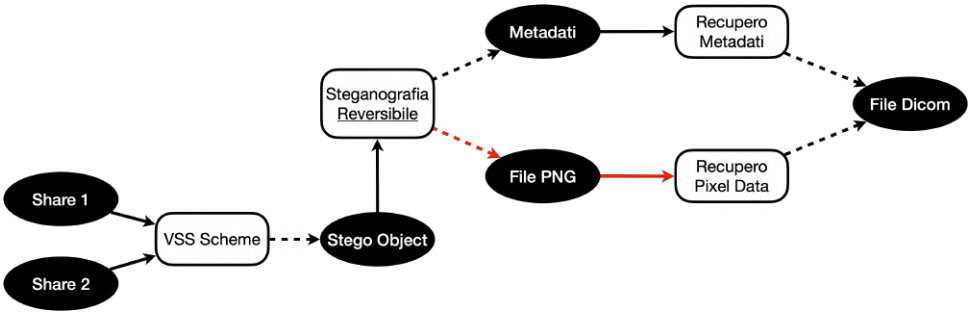


Figura 5.4: Processo di decifratura

Nella decifratura la differenza sostanziale tra le due modalità sta nel fatto che nella '*partial metadata mode*' è possibile ricostruire l'immagine e recuperare i soli metadati sensibili, mentre nella '*all metadata mode*' si è in grado di ricostruire l'intero file DICOM.

5.3.1 Problematiche

Algoritmo di steganografia

In entrambe le modalità, emerge una problematica relativa al recupero dei valori originali dei pixel.

Nel processo di cifratura, i due share finali sono derivati dallo stego-object, il quale, a sua volta, viene ottenuto dall'unione dell'immagine PNG (cover-object) e metadati (secret). Per questo, unendo i due share nella decifratura, si ricostruirà lo stego-object e non l'immagine PNG.

Mentre l'estrazione dei metadati è relativamente semplice, la ricostruzione dei pixel rappresenta una sfida più complessa, che può essere risolta solamente attraverso l'impiego di un algoritmo di steganografia reversibile.

Utilizzare algoritmi più semplici, per esempio LSB, non permetterebbe il recupero dei valori di pixel originariamente contenuti nel file DICOM su cui è stata fatta la cifratura. Ciò comporterebbe la creazione di un file DICOM, nel processo di decifratura, diverso da quello inizialmente cifrato, anche se con variazioni di pixel minime.

Algoritmi diversi, modalità diverse

Dalla necessità di dover utilizzare la steganografia reversibile, nasce l'idea di consentire al tool di operare nelle due modalità distinte.

La modalità parziale può risultare utile quando la decifratura è finalizzata esclusivamente alla valutazione visuale del referto. In questo contesto, un processo di ricostruzione completo dei pixel non risulta essenziale, poiché eventuali piccole discrepanze tra l'oggetto steganografico e l'oggetto di copertura non sarebbero percepibili dall'occhio umano del medico che esamina il referto. Al contempo, in questa modalità, sarà possibile avere a disposizione le informazioni sensibili legate al referto, per avere un minimo di contesto.

Al contrario, la modalità totale impiega un algoritmo di steganografia reversibile, consentendo quindi la ricostruzione del file DICOM. Questo è possibile grazie fatto che, al termine del processo di decifratura, saranno disponibili tutte le informazioni necessarie: i valori dei pixel, ricostruiti mediante l'utilizzo della steganografia reversibile, e tutti i metadati, nascosti anch'essi attraverso la steganografia.

Ricostruzione completa del file DICOM

Nonostante il procedimento di ricostruzione del file DICOM possa apparire concettualmente semplice, nell'ambito pratico sono stati incontrati diversi ostacoli dovuti alla non semplicità della standardizzazione DICOM e ad elementi appartenenti al *DICOM File Meta Information*. Questa problematica sarà esaminata più in dettaglio nella prossima sezione, dedicata all'implementazione del tool.

5.4 Implementazione

In questa sezione si descriveranno le parti salienti del tool sviluppato. Tutti gli script sono riportati per intero nelle appendici a fine tesi.

5.4.1 Script principale

Lo script principale del tool, in Appendice (E.1), utilizza la libreria '*argparse*' per gestire gli argomenti forniti da riga di comando. In base a tali parametri, il programma determina se effettuare cifratura o decifratura, in quale modalità e con quali caratteristiche.

Argomenti

La prima scelta di argomenti è tra '*-e*' per la cifratura (encryption) e '*-d*' per la decifratura (decryption). In entrambi i casi, è richiesto che venga specificata l'opzione '*-m*' per indicate se si vogliono nascondere nello stego object tutti i metadati (*all*) o solo quelli sensibili (*partial*). È necessario inoltre specificare anche l'encoding format con '*-e*', che indica se si vuole nascondere e prelevare i metadati in formato *json* o *txt*.

Se viene scelta l'encryption ('*-e*'), viene anche richiesto di specificare il parametro '*-f*' che indicherà il percorso da cui recuperare il file DICOM da cifrare, e '*-o*', ovvero il percorso della cartella in cui salvare tutti i file generati dal processo di cifratura.

Al contrario, nel caso di decryption ('*-d*') è necessario specificare, oltre ai parametri comuni, l'opzione '*-i*' che indica il percorso da cui prelevare i file da decifrare.

Determinazione operazione

All'interno dello stesso script, avviene la gestione dei parametri forniti, con un percorso decisionale che porta all'esecuzione della funzione corretta, che può essere cifratura, decifratura, ricostruzione o la segnalazione di un errore in caso di parametri non validi.

5.4.2 Cifratura

Dopo aver verificato tutti i parametri forniti, la chiamata alla funzione di cifratura è la seguente:

```
64 encrypt(args.file, args.metadata, args.encoding_format, args.output_folder)
```

Per consultare il codice relativo all'operazione di cifratura, fare riferimento all'Appendice (E.2). A seguire, sono illustrati una serie di diagrammi di sequenza commentati e semplificati che delineano il flusso di esecuzione della funzione.

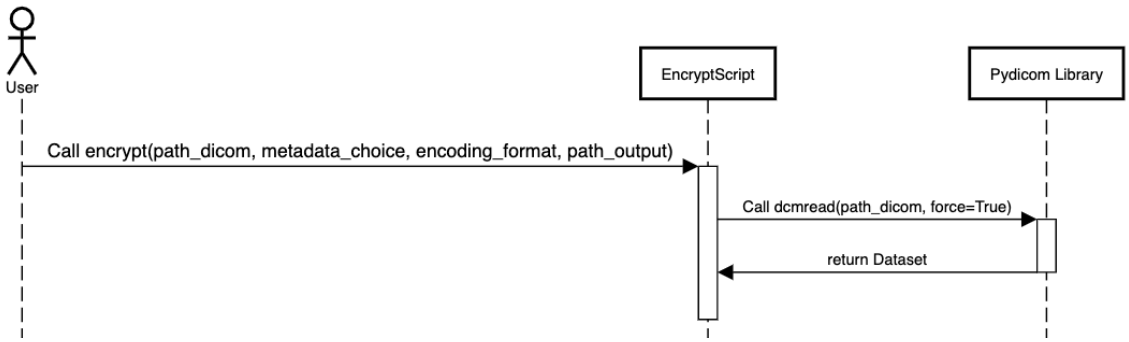


Figura 5.5: Ottenimento Dataset

Nella fase iniziale del processo, come mostrato nella Figura (5.5), si fa uso della libreria Pydicom per effettuare la lettura del file DICOM e recuperare il dataset che racchiude tutti i dati necessari per le fasi successive dell'elaborazione.

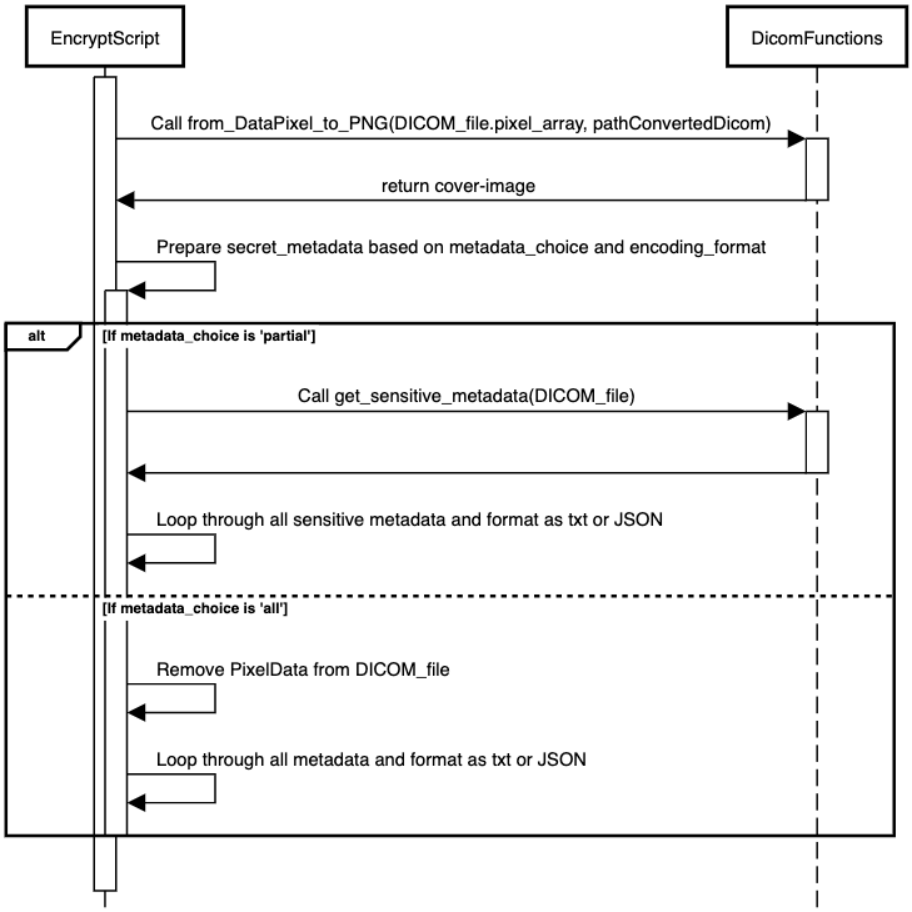


Figura 5.6: Preparazione segreto e cover-object

Successivamente, lo script effettua chiamate alla pseudo-libreria di interazione con i file DICOM, sviluppata dall'autore e disponibile in Appendice.

Le funzioni utilizzate prevedono l'estrazione del campo PixelData dal file DICOM, necessaria per la creazione dell'immagine di copertura utilizzata nella steganografia. In seguito, in base alla selezione effettuata, vengono recuperati tutti i metadati, o solo quelli sensibili, che verranno formattati come file di testo o JSON.

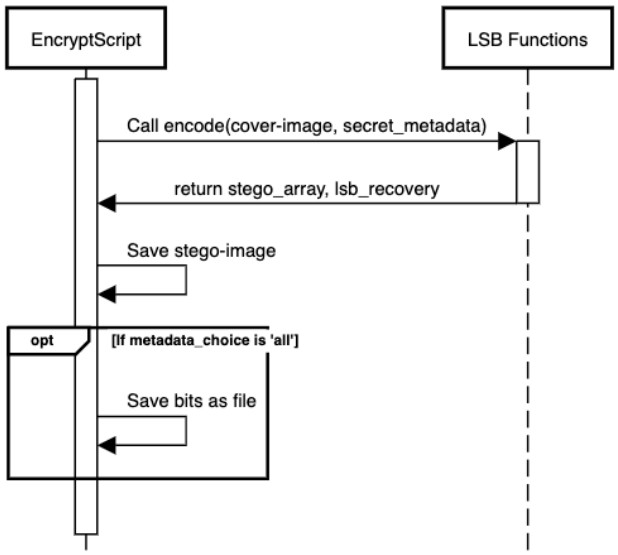


Figura 5.7: Applicazione della steganografia ed ottenimento stego-object

Ora, avendo a disposizione tutti gli elementi necessari per eseguire la steganografia, si procede con l'encoding dei metadati nell'immagine precedentemente creata.

La steganografia applicata è quella LSB. Tale algoritmo, come verrà spiegato nelle prossime sezioni, è stato reso reversibile grazie ad una piccola modifica. Tale variazione è contenuta nell'OPT di Figura (5.7). I bit salvati corrispondono ai bit meno significativi dell'immagine di copertura, che sono stati utilizzati per occultare il segreto. In questo modo, nel processo di decifratura, una volta estratto il segreto, sarà possibile ricostruire la cover-image sostituendo i bit meno significativi con quelli salvati.

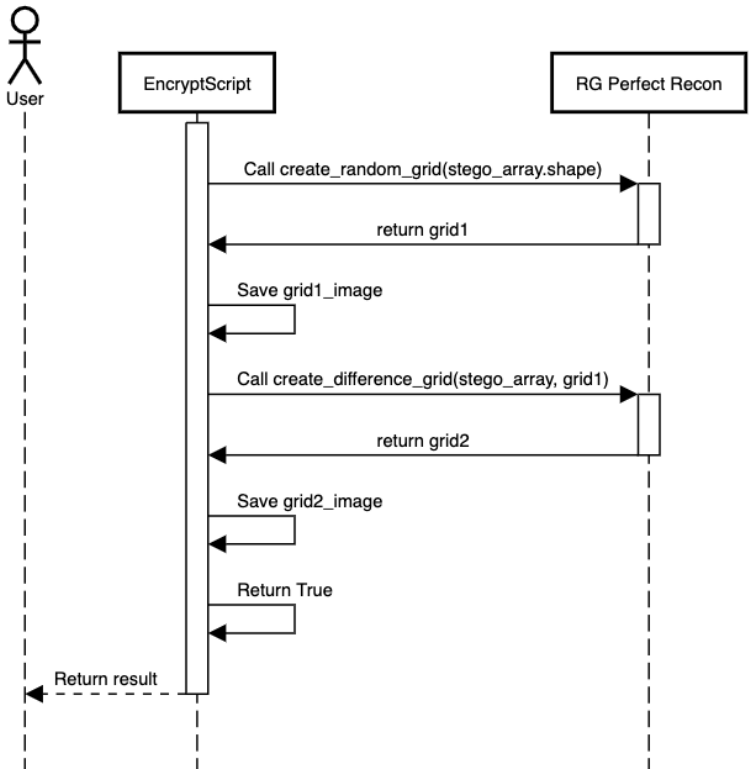


Figura 5.8: Creazione delle due random grid

La fase conclusiva per portare a termine la cifratura consiste nell'applicazione dell'algoritmo di Random Grid con ricostruzione perfetta allo stego-object creato. Tale operazione porterà alla creazione delle due random grid contenenti i pixel ed i metadati (parziali o totali) del file DICOM di partenza.

Composizione del segreto in modalità totale e formato JSON

Se la modalità di cifratura è impostata su "totale", come precedentemente spiegato, vengono preservati dei bit specifici nell'immagine di copertura al fine di consentirne la successiva ricostruzione durante il processo di decifratura.

In aggiunta a ciò, se il formato richiesto è txt il segreto verrà strutturato come stringa composta da tutti i campi del dataset. Al contrario, se si sceglie l'encoding JSON, oltre a tutto il dataset, vengono memorizzati, come segreto, anche dei campi definiti nella libreria Pydicom come '*metafile*'. Tali metafile, tutti contenuti in quello che si può chiamare header del file DICOM, consistono nel preambolo e prefisso del file, più una serie di elementi appartenenti al gruppo $0x0002$.

Tra gli elementi dell'header, vi è un campo denominato *Transfer Syntax*, identificato dal tag (0002, 0010), che ha il compito di definire le regole di codifica utilizzate per rappresentare in modo univoco i dati contenuti nei file DICOM. In altre parole, stabilisce il metodo con cui i dati sono codificati e organizzati all'interno del file, garantendo che possano essere decodificati e interpretati correttamente da qualsiasi sistema compatibile con il formato DICOM.

Considerata la necessità di includere anche i metafile per la successiva ricostruzione di un file DICOM valido, l'oggetto JSON che viene creato e utilizzato come segreto, quando le opzioni *-ef json* e *-m all* sono attivate, viene generato dalla funzione presente nell'Appendice (D). Di seguito, viene richiamata tale funzione:

```
31 def get_all_metadata_json(ds):
32     del ds.PixelData
33 
34     json_data = {
35         "dataset_json": ds.to_json(),
36         "metafile_json": ds.file_meta.to_json()
37     }
38 
39     return json_data
```

5.4.3 Decifratura

Verificati tutti i parametri forniti, la chiamata alla funzione di cifratura è la seguente:

```
82 decrypt(args.input_folder, args.metadata, args.encoding_format)
```

Il codice completo di decifratura è consultabile nell'Appendice (E.3). Ora è possibile esaminare i diagrammi di sequenza che rappresentano questa funzione.

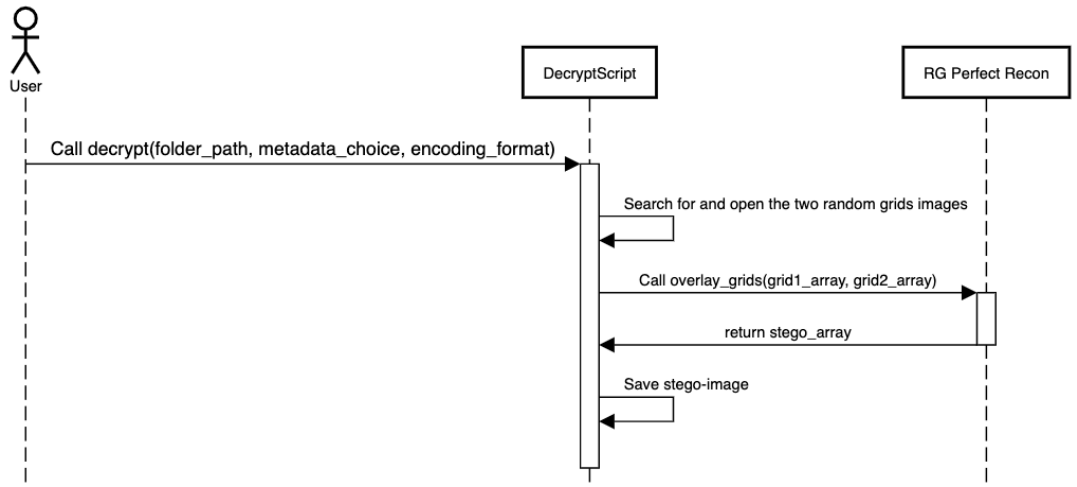


Figura 5.9: Ricostruzione dello stego-object partendo dalle due random grid

Il processo di decifratura ha inizio partendo esclusivamente dalle due griglie casuali, e la sua prima operazione consiste nell'utilizzare queste griglie per la ricostruzione della stego-image, la quale viene poi salvata.

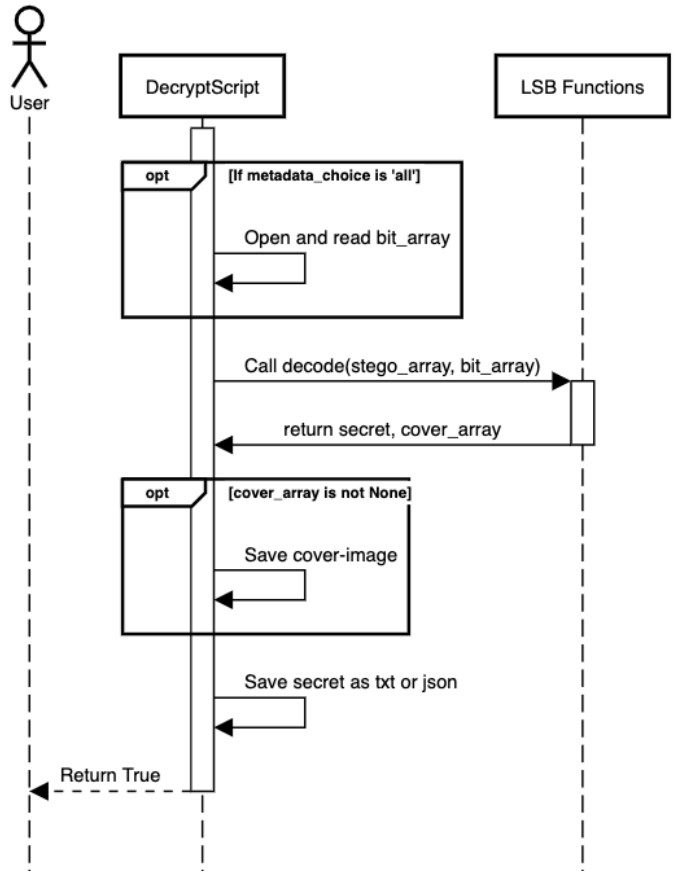


Figura 5.10: Decoding steganografico e salvataggio risultati

Nel caso in cui la modalità di decifratura sia impostata su "totale", vengono recuperati i bit che erano stati salvati durante la cifratura e che permettono al processo di decodifica steganografica di ricostruire completamente l'immagine di copertura. Al contrario, se la modalità è impostata su "parziale", il processo di decodifica restituirà solo il segreto e non la cover-image ricostruita.

A questo punto, tutto ciò che la funzione di decodifica LSB restituisce viene salvato, che si tratti del segreto o, eventualmente, anche dell'immagine ricostruita.

5.4.4 Ricostruzione del file DICOM

A seguito della decifratura, nel caso in cui tra le opzioni vi sia *-ef json* e *-m all*, avverrà la ricostruzione del file DICOM.

```
91 reconstruction(args.input_folder)
```

Il codice che permette la ricostruzione del file DICOM è riportato nell'Appendice (E.5).

La procedura di ricostruzione, illustrata in Figura (5.11), richiede solamente il percorso di una cartella. Da essa verrà recuperato il segreto in formato JSON e verrà utilizzata come destinazione in cui salvare il file DICOM ricostruito.

Una volta recuperato l'oggetto, viene eseguito un parsing per distinguere i campi del dataset dai metafile. Successivamente, sfruttando una funzione dalla libreria Pydicom, viene creato un dataset utilizzando il contenuto del primo oggetto, e successivamente i metafile vengono inseriti nel dataset creato.

L'ultimo passaggio implica l'uso di una delle funzioni presenti nel file Dicom Function per recuperare i data pixel dall'immagine di copertura, convertiti in formato binario. Questi vengono caricati nel campo Pixel Data del file DICOM. A questo punto, il processo di ricostruzione è completato, e l'unico passo rimanente è il salvataggio del file DICOM ricostruito.

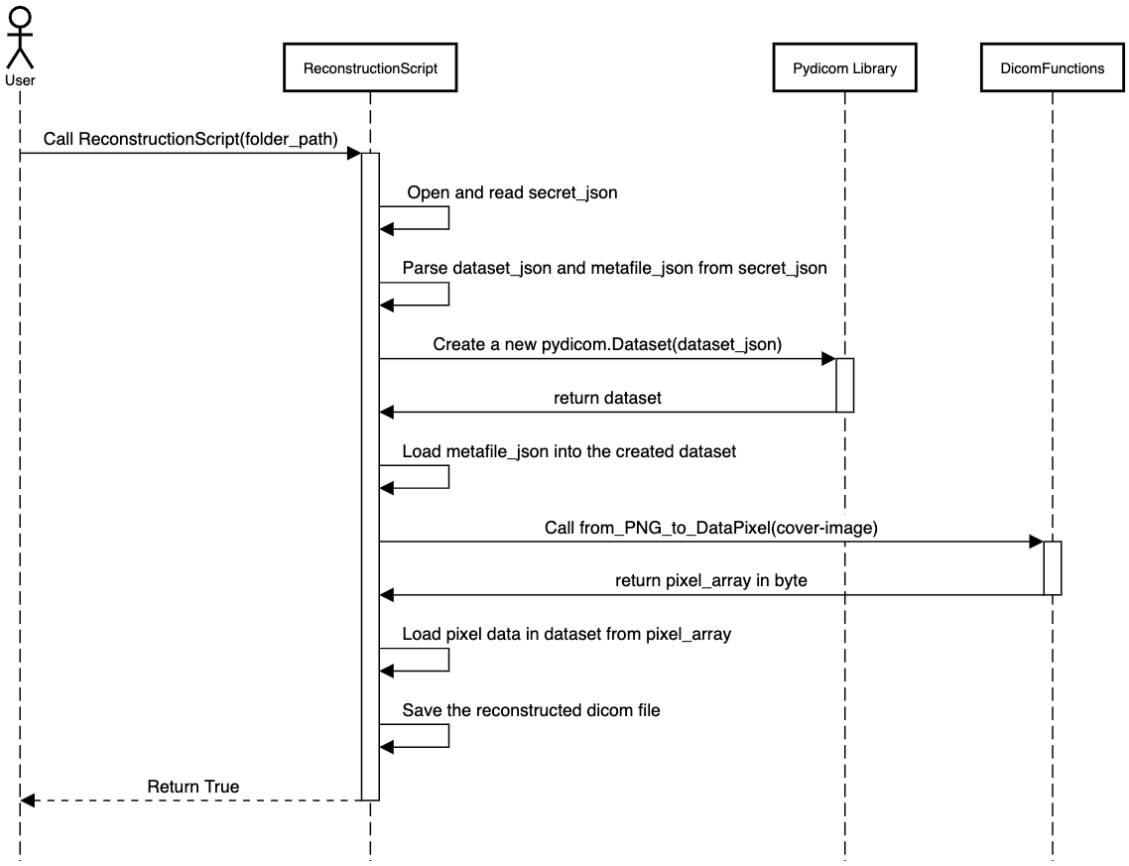


Figura 5.11: Diagramma di sequenza della funzione di ricostruzione

I diagrammi presentati in questa sezione offrono una rappresentazione semplificata delle procedure svolte negli script, concentrandosi sugli aspetti chiave della cifratura e della decifratura. Non riportano dettagli come la conversione da immagine PIL a np.array, l'apertura di file o la gestione degli errori, ma mirano a fornire una panoramica generale delle fasi principali di ciascuna procedura.

Gli attori considerati nei diagrammi sono le librerie create per la gestione del progetto, mentre, tranne Pydicom, eventuali librerie di terze parti non sono state incluse come attori nei diagrammi.

Per una comprensione più dettagliata delle procedure e delle funzionalità, si consiglia nuovamente di fare riferimento agli script completi forniti in Appendice, dove saranno disponibili tutti i dettagli implementativi e gli aspetti più specifici del progetto.

5.5 Caso di studio

In questa sezione, viene esaminato un caso di studio con l'obiettivo di dimostrare l'utilizzo del software nelle due diverse modalità.

5.5.1 Helper

Il primo comando che si può utilizzare è l'helper, grazie ad esso è possibile ottenere informazioni sul funzionamento e sulle opzioni offerte dal programma.

```
$ ./dcm-sharing.py -h
DICOM Image Encryptor/Decryptor

Syntax for encryption:
    ./dcm-sharing.py -e -m <partial | all> -ef <txt | json> -f <DICOM FILE> -o <FOLDER>

Syntax for decription:
    ./dcm-sharing -d -m <partial | all> -ef <txt | json> -i <FOLDER>
```

```
If you want to ensure that the DICOM file is reconstructable during decryption, you must
↪  set the options '-m all' and '-ef json' both when encrypting and decrypting.

options:
  -h, --help            show this help message and exit
  -e, --encrypt         Encrypt a DICOM file
  -d, --decrypt         Decrypt a DICOM file
  -m {all,partial}, --metadata {all,partial}
                        When set to 'partial', only pixel data and PHI reconstruction is
                        ↪  possible. When set to 'all', full DICOM file reconstruction
                        ↪  is possible (required).
  -ef {json,txt}, --encoding-format {json,txt}
                        Desired encoding format for metadata (required).
  -f FILE, --file FILE  Path to the DICOM file (required for encryption)
  -o OUTPUT_FOLDER, --output-folder OUTPUT_FOLDER
                        Path to the output folder for results (required for encryption)
  -i INPUT_FOLDER, --input-folder INPUT_FOLDER
                        Path to the folder containing random grids (required for
                        ↪  decryption)
```

5.5.2 Modalità parziale

Si consideri il file 'referto.dcm', contenuto nella cartella '/dcm'. Si vuole effettuare una cifratura per una futura consultazione visiva dell'esame e dei dati sensibili in formato testuale. La modalità di cifratura più adatta a tale scopo è quella parziale.

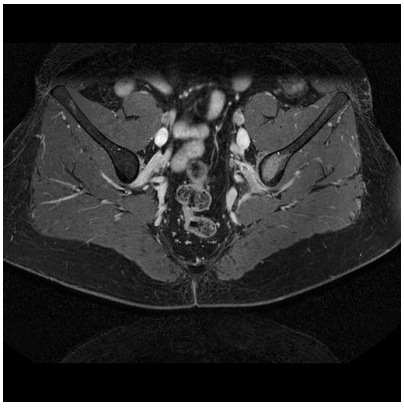
Cifratura

Di seguito vengono riportati i comandi e le risposte date dal tool per effettuare tale cifratura:

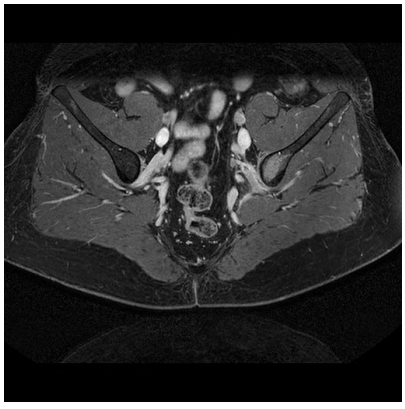
```
$ ls dcm
referto.dcm
$ ./dcm-sharing.py -e -m partial -ef txt -f dcm/referto.dcm -o dcm/
Encrypting DICOM file...
Encryption completed.
$ ls dcm
grid1.png       grid2.png       referto.dcm     tmp
$ ls dcm/tmp
converted_DICOM.png     stego_object.png
```

I primi due file generati sono gli elementi presenti nella cartella 'tmp':

- *converted_DICOM.png*: è l'immagine PNG generata dall'estrazione dei pixel data;
- *stego_object.png*: è il risultato del processo di steganografia applicato all'immagine precedente, con l'obiettivo di occultare i metadati sensibili estratti.



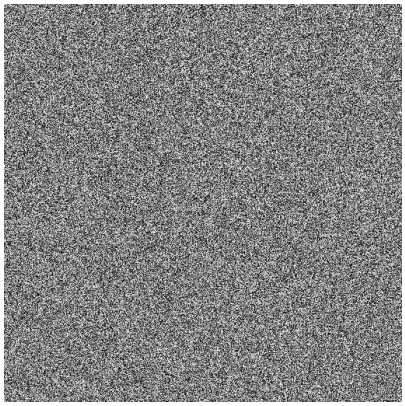
(a) converted_DICOM.png



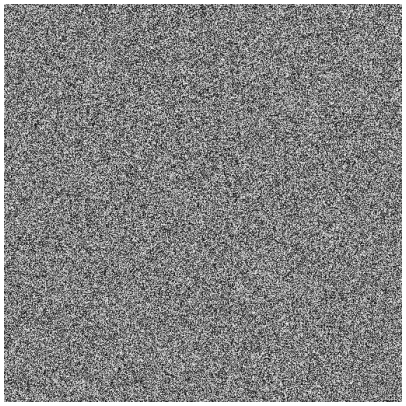
(b) stego_object.png

Figura 5.12: File temporanei generati dalla cifratura

Le due immagini rimanenti sono le random grid, che costituiscono gli unici elementi richiesti dal processo di decifratura.



(a) grid1.png



(b) grid2.png

Figura 5.13: Random grid generate da '*stego_object.png*'

Decifratura

Per sottolineare il fatto che il processo di decifratura si basa esclusivamente sulle random grid, vengono spostati tutti i file temporanei e il file "referto.dcm" in un'altra directory. Una volta completato questo passaggio, si procede con il processo di decifratura.

```
$ mv dcm/tmp/* dcm-backup
$ mv dcm/referto.dcm dcm-backup
$ ls dcm
grid1.png       grid2.png       tmp
$ ls dcm/tmp
$ ./dcm-sharing.py -d -m partial -ef txt -i dcm/
Decrypting DICOM file...
Decryption completed.
$ ls dcm
grid1.png       grid2.png       secret.txt      tmp
$ ls dcm/tmp
stego_object.png
```

Come ci si aspetterebbe il contenuto di '*secret.txt*', di seguito riportato, è un insieme di PHI estratti dal file DICOM originale.

```
$ cat dcm/secret.txt
Tag: (0008, 0018) | Descrizione: SOP Instance UID | Valore:
↪  1.2.276.0.50.192168001099.7810872.14547392.469;
Tag: (0008, 0020) | Descrizione: Study Date | Valore: 20010101;
Tag: (0008, 0030) | Descrizione: Study Time | Valore: 113323;
Tag: (0008, 0050) | Descrizione: Accession Number | Valore: 11788759296818;
Tag: (0008, 0080) | Descrizione: Institution Name | Valore: Anonymized Hospital;
Tag: (0008, 0090) | Descrizione: Referring Physician Name | Valore: Dr. Anonymous;
Tag: (0008, 1030) | Descrizione: Study Description | Valore: MRT Sakroiliakalgelenke;
Tag: (0010, 0010) | Descrizione: Patient Name | Valore: Fall 1;
Tag: (0010, 0020) | Descrizione: Patient ID | Valore: 11788759296811;
Tag: (0010, 0030) | Descrizione: Patient Birth Date | Valore: 19000101;
Tag: (0010, 0040) | Descrizione: Patient Sex | Valore: O;
Tag: (0020, 0010) | Descrizione: Study ID | Valore: 11788759296812;
```

Invece, l'immagine '*stego_object.png*' generata durante il processo di decifratura risulta identica a quella prodotta nella fase di cifratura. Tale uguaglianza può essere verificata attraverso l'uso del comando '*diff*':

```
$ diff dcm/tmp/stego_object.png dcm-backup/stego_object.png
```

Nella pratica, l'immagine presenterà differenze impercettibili rispetto a quella originariamente estratta (*converted_DICOM.png*) dal file DICOM. Pertanto potrà essere utilizzata da un medico per condurre le proprie analisi e verifiche, a meno che tali attività richiedano specificamente un file nel formato DICOM.

5.5.3 Modalità totale

Ora, si vuole eseguire una cifratura completa dello stesso file DICOM utilizzato nel test precedente.

L'obiettivo è quello di cifrare un file DICOM, ed essere in grado di ricostruirlo identico in seguito alla decifratura. Per tale motivo, è necessario usare come *encoding format* '*json*'.

Cifratura

```
$ ls dcm
referto.dcm
$ ./dcm-sharing.py -e -m all -ef json -f dcm/referto.dcm -o dcm/
Encrypting DICOM file...
Encryption completed.
$ ls dcm
grid1.png       grid2.png       reconstruction_bits.txt     referto.dcm     tmp
$ ls dcm/tmp
converted_DICOM.png     stego_object.png
```

Nella directory '*dcm/tmp*' vengono creati gli stessi file visti nella della modalità parziale.

All'interno della directory '*dcm*', invece, sono state generate le due random grid, e in aggiunta è stato creato un nuovo file che non era presente in precedenza: '*reconstruction_bits.txt*'.

Tale file, come suggerisce il suo nome, è essenziale per la ricostruzione dei pixel da inserire nel file DICOM che si vuole ricostruire. Questo elemento salva tutti i bit meno significativi della cover-image dove è avvenuta una modifica per nascondere il segreto, rendendo di fatto l'algoritmo di steganografia LSB reversibile.

```
$ head -c50 dcm/reconstruction_bits.txt
00000000000000000000000000000000000000000000000000%
$ tail -c50 dcm/reconstruction_bits.txt
10101111010111010001110111100101101011011011110110%
```

Le potenziali minacce alla sicurezza, introdotte da questo, saranno esaminate nella prossima sezione.

Decifratura

Come effettuato nella modalità precedente, i file non necessari vengono spostati in una cartella apposita al fine di consentire confronti successivi. Successivamente, si procede con l'esecuzione della decifratura.

```
$ mv dcm/tmp/* dcm-backup
$ mv dcm/referto.dcm dcm-backup
$ ls dcm
grid1.png       grid2.png       reconstruction_bits.txt     tmp
$ ls dcm/tmp
$ ./dcm-sharing.py -d -m all -ef json -i dcm/
Decrypting DICOM file...
Decryption completed.
Reconstructing DICOM file...
Reconstruction completed.
$ ls dcm
grid1.png       grid2.png       reconstructed.dcm       reconstruction_bits.txt     tmp
$ ls dcm/tmp
converted_DICOM.png     secret.json     stego_object.png
```

Utilizzando esclusivamente le due random grid e il file '*reconstruction_bits*', la procedura di decifratura è stata in grado di ricostruire e recuperare con successo tutti gli elementi necessari per la ricreazione del file DICOM precedentemente cifrato.

Di seguito sono riportati i comandi utilizzati per verificare l'uguaglianza tra i file DICOM e tra i file temporanei generati durante il processo di cifratura e quelli ottenuti durante la fase di decifratura.

```
$ diff dcm/tmp/stego_object.png dcm-backup/stego_object.png
$ diff dcm/tmp/converted_DICOM.png dcm-backup/converted_DICOM.png
$ diff dcm/reconstructed.dcm dcm-backup/referto.dcm
```

Poiché non è stata prodotta alcuna risposta dal terminale, possiamo concludere che il processo di decifratura ha perfettamente ricreato ciò che aveva generato il processo di cifratura, incluso il file DICOM di partenza.

5.6 Sicurezza e privacy

La sicurezza dell'intero schema è garantita dall'utilizzo degli algoritmi delle random grid. È stato sottolineato più volte che gli schemi di visual secret sharing forniscono una sicurezza perfetta, e poiché le random grid rappresentano un'implementazione di tali schemi, anch'esse garantiscono una sicurezza perfetta.

Confidenzialità

Il possesso di una delle due grid non consente in alcun modo di ricostruire l'altra. Pertanto, la confidenzialità è garantita; senza entrambe le grid, non è possibile recuperare né l'immagine medica originale né i metadati nascosti al loro interno.

Integrità

L'integrità può essere considerata come parzialmente garantita. Non si può affermare di avere un'integrità totale, poiché modifiche minime ai pixel di una delle due griglie sarebbero difficilmente rilevabili anche dopo l'unione di esse. Tuttavia, stravolgimenti significativi dei valori dei pixel sarebbero evidenti nei file costruiti durante la decifratura.

Inoltre, è importante tenere presente che i bit meno significativi potrebbero contenere metadati, pertanto qualsiasi modifica a tali bit sarebbe più facilmente rilevabile.

5.6.1 Protezione da attacchi e da minacce alla privacy

Grazie a questo meccanismo, vengono efficacemente contrastati eventuali tentativi di accesso non autorizzato agli archivi delle strutture ospedaliere. In caso di successo, tali attacchi otterrebbero solamente una lista di share, eventualmente associati a nomi e cognomi, che non rivelano alcuna informazione sull'esame medico.

Inoltre, questa tecnica garantisce la privacy di tutti i dati presenti nel file DICOM, compresi i metadati sensibili, quelli non sensibili e i dati pixel che costituiscono l'immagine medica. Offre anche una protezione per i dati visibili direttamente nell'immagine stessa.

Tutte le potenziali minacce alla privacy sono state neutralizzate, inclusi gli attacchi diretti che potrebbero rivelare condizioni o informazioni private, quelli ricollegabili che permettono di risalire al paziente e quelli di inferenza esistenziale e di identificazione.

Possibile attacco: injection

Gli attacchi di injection, ad esempio contro archivi ospedalieri, sono possibili in due modalità.

La prima modalità di attacco mira a modificare un referto già presente nell'archivio. Tuttavia, questo tipo di attacco che minaccia l'integrità dei dati, sarebbe complesso da realizzare, poiché qualsiasi modifica dovrebbe essere apportata su una griglia di valori casuali che contiene anche metadati.

La seconda modalità di attacco di injection, invece, coinvolge referti creati appositamente. La fattibilità di questo tipo di attacco dipenderà principalmente dalla sicurezza del server che ospita i dati. In questo scenario, gli attaccanti dovrebbero avere accesso a un DICOM falso con metadati manipolati, conoscere perfettamente l'algoritmo utilizzato per la cifratura e trovare un modo per iniettare lo share falso creato.

Inoltre, in questo scenario, gli attaccanti dovrebbero trovare un metodo per sostituire anche lo share in possesso del paziente. Altrimenti, a seguito della sovrapposizione degli share, si comprenderebbe subito che c'è stata una manipolazione.

Nonostante la complessità di questo tipo di attacco, che mette a rischio sia l'integrità che l'autenticazione dei dati, va notato che rimane ancora una possibilità realizzabile, se confrontata con le altre minacce, completamente neutralizzate, precedentemente descritte.

5.6.2 Steganografia LSB è un punto debole?

Nell'implementazione del software, è stata fatta la scelta di utilizzare l'algoritmo steganografico LSB, configurato in modo reversibile. La reversibilità è stata raggiunta mediante la scrittura di un file contenente i bit meno significativi dell'immagine di copertura, che sono stati sovrascritti.

Dal momento che l'intero schema si fonda sulla sicurezza perfetta delle random grid, l'utilizzo di un algoritmo di steganografia semplice come LSB non mette in pericolo la sicurezza del processo. Questo perché, nonostante LSB sia una forma di steganografia semplice, i suoi risultati vengono sempre cifrati, garantendo così la sicurezza complessiva.

File *reconstruction_bits.txt*

Escludendo le random grid dai file finali generati durante il processo di cifratura, che abbiamo già confermato essere sicure, rimane da valutare se il file '*reconstruction_bits.txt*' possa costituire un possibile punto debole. Di seguito, si riporta il frammento di lettura parziale di tale file, che è stato precedentemente esaminato.

```
$ head -c50 dcm/reconstruction_bits.txt
00000000000000000000000000000000000000000000000000%
$ tail -c50 dcm/reconstruction_bits.txt
10101111010111010001110111100101101011011011110110%
```

Nel esempio in questione, una considerevole quantità di bit contenuti nel file è composta da zeri consecutivi. Questo fenomeno è dovuto al fatto che, come si può vedere nel caso di studio specifico in Figura (5.12a), la porzione superiore dell'immagine è costituita da pixel neri. Essendo un pixel nero rappresentato come '00000000', gli lsb salvati nella parte iniziale saranno zeri, come osservato nel comando head.

Il comando "tail", al contrario, presenta una sequenza diversa che non è composta solo da zeri. È importante sottolineare che questi bit non rappresentano gli lsb dell'intera immagine; altrimenti, anche il comando "tail" avrebbe restituito un insieme di zeri, data la presenza dei pixel neri anche nella parte inferiore dell'immagine PNG. Questi bit rappresentano gli lsb dell'immagine nella parte in cui è stato incorporato il segreto.

Di conseguenza, l'unica informazione che può essere dedotta da questo file è la lunghezza in bit dell'oggetto JSON nascosto nell'immagine.

Possibile attacco: ricostruzione bitplane meno significativa

Un possibile metodo di attacco che potrebbe essere considerato, basato su tale file, sarebbe quello di utilizzare i bit contenuti per tentare di ricostruire la bitplane meno significativa dell'immagine medica. Tuttavia, questa tecnica non sarebbe molto efficace, poiché, come già illustrato nei capitoli precedenti, la bitplane meno significativa è assimilabile a rumore e, inoltre, con alta probabilità non sarebbe nemmeno completa.

Il numero di bit presenti nel file, infatti, dipende dalla lunghezza della stringa o del JSON di metadati nascosti. Tale numero sarà sempre minore, o al massimo uguale, al numero di pixel dell'immagine.

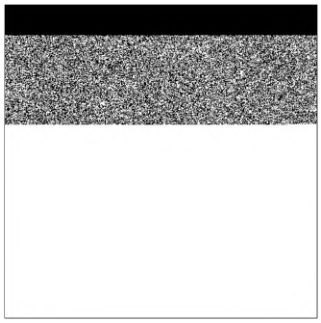


Figura 5.14: Immagine costruita dal file *reconstruction_bits.txt*

Questo ragionamento sulla impraticabilità di un attacco di questo tipo è confermato in Figura (5.14). Questa immagine è stata ottenuta dal file *reconstruction_bits.txt* del caso di studio precedentemente descritto. L'unica osservazione possibile è la presenza di un padding all'inizio dell'immagine; tutto il resto sembra costituito da semplice rumore e non offre alcuna possibilità di deduzione.

5.7 Versione precedente: text-to-image

La prima versione sviluppata del software era una combinazione della modalità parziale e totale.

In questa prima implementazione, il processo di cifratura prevedeva l'utilizzo della steganografia per nascondere solo i dati sensibili all'interno dell'immagine utilizzata per generare i due share. In un secondo momento, i dati non sensibili venivano convertiti da testo in un'immagine in scala di grigi, dove ogni pixel rappresentava un singolo carattere. In seguito alla conversione "text-to-image," l'immagine risultante poteva essere utilizzata per la crittografia visuale.

Una schematizzazione di questo processo è mostrata in Figura (5.15).

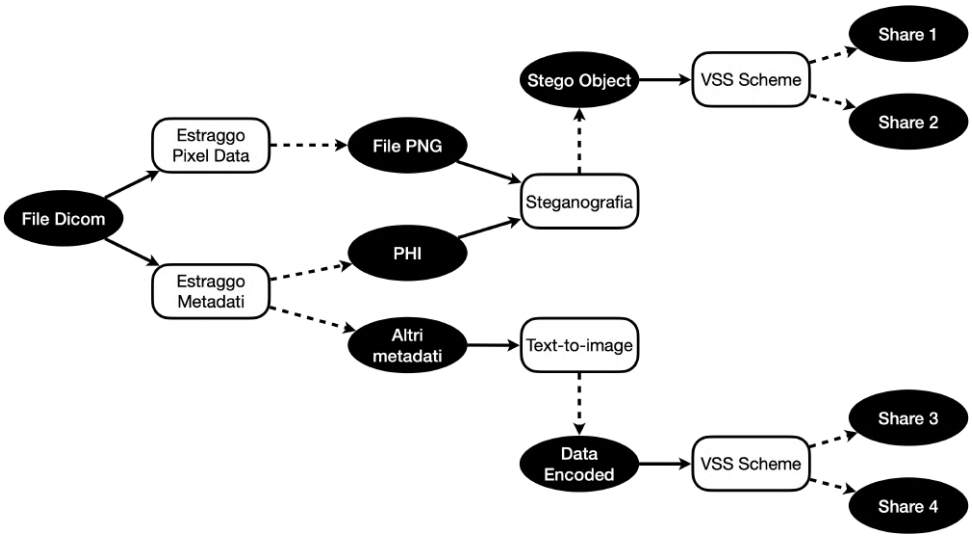
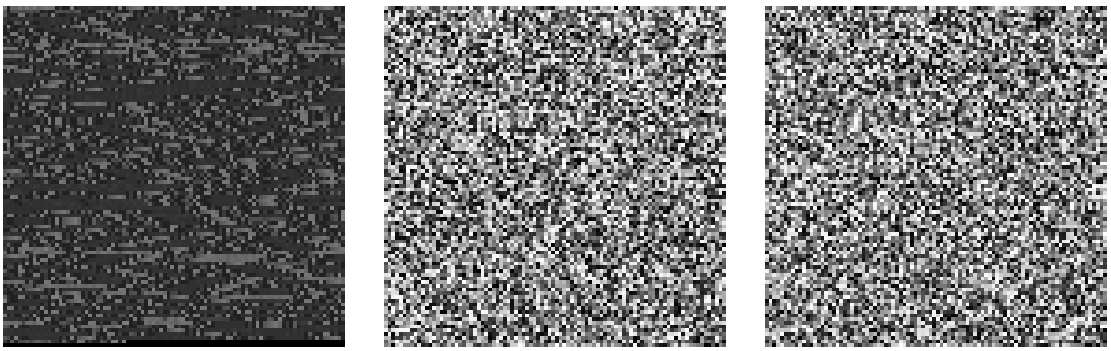


Figura 5.15: Processo di cifratura nella prima versione

In questa versione del programma, il processo di cifratura generava quindi quattro share, il che significa che sia il paziente che la struttura ospedaliera avrebbero dovuto conservare due share ciascuno.

Questa soluzione è stata successivamente abbandonata quando è emerso che la maggior parte dei file DICOM disponeva di spazio sufficiente nei Pixel Data per nascondere tutti i metadati tramite steganografia. Nonostante ciò, l'idea di utilizzare la tecnica "text-to-image" potrebbe essere riesaminata nei casi in cui lo spazio disponibile non sia sufficiente per nascondere un eventuale alto numero di metadati presenti nel file DICOM.

Riprendendo il file '*secret.json*', generato dalla decifratura in modalità totale nella Sezione (5.5.3), viene riportata di seguito l'immagine creata dall'algoritmo Text-to-Image a partire da tale file, seguita da una cifratura dell'immagine risultante con random grid.



(a) text_encoded.png (b) text_encoded_grid1.png (c) text_encoded_grid2.png

Figura 5.16: Rappresentazione dei metadati come immagine e applicazione di Random Grid

Lo script per ottenere l'encoding è stato sviluppato dall'autore per la prima versione del software e, anche se non fa parte della versione definitiva del progetto, è comunque consultabile nell'Appendice per ulteriori dettagli.

Capitolo 6

Conclusione

Come più volte sottolineato, l'utilizzo di questo schema di cifratura offre un approccio innovativo per proteggere i dati che formano un file DICOM, combinando tecniche di steganografia e di crittografia visuale. Le due modalità, parziale e totale, forniscono un alto livello di flessibilità, consentendo agli utenti di bilanciare le proprie esigenze con la sicurezza garantita.

Gli obiettivi iniziali del progetto erano mirati a creare un meccanismo di condivisione e memorizzazione dei file DICOM, che garantisse la sicurezza dell'immagine e dei dati sensibili. I risultati ottenuti confermano che tali obiettivi sono stati raggiunti con successo. Inoltre, è stato illustrato come la sicurezza garantita dall'utilizzo delle random grid renda lo schema estremamente robusto ad attacchi di varia natura. La vera difficoltà nell'impiego di un meccanismo di questo tipo riguarda principalmente la gestione e la conservazione degli share.

Le strutture ospedaliere sono già in grado di conservare referti medici in modo relativamente affidabile. Il passaggio dalla conservazione di un file DICOM tradizionale a quella di uno share è minimo e non comporta significative modifiche alla loro infrastruttura.

La vera sfida riguarda invece il paziente. Mentre con i file DICOM standard, se il paziente perde il dispositivo contenente il referto, è molto probabile che l'ospedale ne abbia una copia di backup. Con una tecnologia di questo tipo, il paziente deve essere in grado di conservare correttamente e non perdere il proprio share.

La perdita di uno dei due share comporterebbe la perdita definitiva del referto, poiché la ricostruzione sarebbe impossibile. Pertanto, la nuova architettura ideata deve fare affidamento su una gestione accurata dei server di memorizzazione ospedalieri e sulla massima affidabilità da parte dei pazienti.

Possibili Sviluppi Futuri

Nel corso dello sviluppo del progetto sono emerse diverse aree che potrebbero essere migliorate.

Tra i possibili miglioramenti più immediati vi è l'aggiunta di una nuova opzione, "*–notmp*" o "*–complete*", che consentirebbe agli utenti di scegliere se eliminare o mantenere i file temporanei generati durante il processo di cifratura e decifratura.

Un altro miglioramento potrebbe riguardare la gestione del file *reconstruction_bits.txt*. Anche se non costituisce una minaccia per la sicurezza, potrebbe essere desiderabile trovare un metodo più efficiente per gestire queste informazioni necessarie, eliminando la necessità di conservare un file in più oltre alle due random grid. In alternativa potrebbe essere sfruttato un algoritmo di steganografia reversibile più complesso di quello utilizzato nel progetto.

Un ulteriore sviluppo potrebbe consentire la cifratura di più file DICOM con una singola esecuzione del comando *dcm-sharing*. Questa funzionalità sarebbe particolarmente utile quando si desidera cifrare più file provenienti dalla stessa serie di esami. Un semplice script di shell potrebbe essere implementato per raccogliere tutti i file in una cartella e successivamente eseguire il comando di cifratura su ciascuno di essi. Naturalmente, un processo simile sarebbe necessario anche per la decifratura.

Altra idea implementabile è quella di permettere a chi utilizza il tool di specificare i metadati considerati sensibili e, quindi, che verranno nascosti in modalità parziale.

Oltre ai possibili miglioramenti presentati, ci sono due aspetti che andrebbero gestiti in modo più efficiente. Il primo problema riguarda la gestione dei file DICOM che contengono una quantità elevata di metadati. In questi casi può accadere che i data pixel non riescano a nascondere tutti i metadati mediante l'uso della steganografia. Una possibile soluzione a questa sfida potrebbe consistere nell'applicare l'approccio "text-to-image", al fine di convertire i metadati eccedenti in

un'altra immagine. In questo modo, come nella prima versione del software, si creerebbero quattro random grid da gestire. In realtà, sarebbe desiderabile sviluppare un metodo che non comporti l'aumento del numero di file generati. Questo obiettivo potrebbe essere raggiunto attraverso la concatenazione dei data pixel dell'immagine principale e quelli dell'immagine generata tramite la conversione dei metadati, che non si è riusciti a nascondere, in formato text-to-image.

Il secondo problema riguarda i file DICOM in cui la codifica dei pixel subisce variazioni durante il processo di conversione nel formato PNG. La codifica dei pixel, all'interno dei file DICOM, è influenzata da diversi fattori, tra cui il tipo di esame e l'attrezzatura utilizzata. Tale variabilità andrebbe gestita caso per caso al fine di garantire sempre la possibilità di cifratura e decifratura senza errori di alcun genere.

In sintesi, il progetto attuale costituisce una solida base su cui costruire implementazioni più avanzate per migliorarne l'efficienza e la sua possibilità di utilizzo.

Appendice A

Script di Crittografia Visuale

In questa appendice vengono riportati gli script Python creati per generare alcune delle immagini nel capitolo sulla Crittografia Visuale.

A.1 Immagini binarie e grayscale

In questa sezione viene fornito l'intero codice sorgente relativo all'implementazione della Crittografia Visuale base (che lavora su immagini bitmap in bianco e nero) e in scala di grigi.

A.1.1 Funzioni

```
import random
import copy


# Funzione che mappa il valore: dalla codifica RGB allo standard VC
def map_rgb_to_bit(rgb_value):
    if rgb_value == 0:
        return 1
    if rgb_value == 255:
        return 0


# Funzione che mappa il valore: dallo standard VC alla codifica RGB
def map_bit_to_rgb(vc_value):
    if vc_value == 1:
        return 0
    if vc_value == 0:
        return 255


# Funzione che permuta sulla colonna, in modo casuale, la matrice fornita
def random_col_permutation(matrix):
    n_rows = len(matrix)
    n_cols = len(matrix[0])
    new_matrix = copy.deepcopy(matrix)

    # Lista contenente gli indici delle colonne della matrice originale
    col_indices = list(range(n_cols))

    # Mischio in modo casuale gli elementi della lista
    random.shuffle(col_indices)

    # Creo una matrice con le colonne mischiate
    for i in range(n_rows):
        for j in range(n_cols):
            new_matrix[i][j] = matrix[i][col_indices[j]]

```

```
    return new_matrix


# Funzione che popola i subpixel degli share in base alla matrice ricevuta
def populate_subpixels(share1, share2, i, j, matrix):
    # Popolo subpixel Share1
    share1.putpixel((i * 2, j * 2), matrix[0][0])
    share1.putpixel((i * 2, j * 2 + 1), matrix[0][1])
    share1.putpixel((i * 2 + 1, j * 2), matrix[0][2])
    share1.putpixel((i * 2 + 1, j * 2 + 1), matrix[0][3])

    # Popolo subpixel Share2
    share2.putpixel((i * 2, j * 2), matrix[1][0])
    share2.putpixel((i * 2, j * 2 + 1), matrix[1][1])
    share2.putpixel((i * 2 + 1, j * 2), matrix[1][2])
    share2.putpixel((i * 2 + 1, j * 2 + 1), matrix[1][3])
```

Le funzioni *map_rgb_to_bit* e *map_bit_to_rgb* permettono di passare dalla codifica RGB allo standard della Crittografia Visuale, e viceversa. Si noti che RGB prevede il valore 0 per il nero e 255 per il bianco, mentre lo standard VC prevede il valore 1 per il nero e 0 per il bianco

A.1.2 Script principale

```
from PIL import Image
from src.vc.basic.functions import *

# Immagine selezionata e convertita in bianco e nero
image = Image.open("../../../resources/secret.jpg")
image = image.convert('1')

# Matrici per determinare subpixel
w = [[1, 1, 0, 0], [1, 1, 0, 0]]
b = [[1, 1, 0, 0], [0, 0, 1, 1]]

# Pixel expansion
share1 = Image.new("1", (image.size[0] * 2, image.size[1] * 2))
share2 = Image.new("1", (image.size[0] * 2, image.size[1] * 2))
out = Image.new("1", (image.size[0] * 2, image.size[1] * 2))


# Generazione degli share
for i in range(0, image.size[0]):
    for j in range(0, image.size[1]):
        # Ottengo valore pixel secondo standard VC
        source_pixel_rgb = image.getpixel((i, j))
        source_pixel_vc = map_rgb_to_bit(source_pixel_rgb)

        # Bianco
        if source_pixel_vc == 0:
            random_white_matrix = random_col_permutation(w)
            populate_subpixels(share1, share2, i, j, random_white_matrix)

        # Nero
        else:
            random_black_matrix = random_col_permutation(b)
            populate_subpixels(share1, share2, i, j, random_black_matrix)


# Simulo sovrapposizione degli share e rimappo ad RGB
for i in range(0, image.size[0] * 2):
    for j in range(0, image.size[1] * 2):
        # Sovrapposizione degli share (OR)
        share1_vc_value = share1.getpixel((i, j))
        share2_vc_value = share2.getpixel((i, j))
```

```
        out_vc_value = share1_vc_value | share2_vc_value

        # Re-mapping dei pixel da standard VC a codifica RGB
        share1.putpixel((i, j), map_bit_to_rgb(share1_vc_value))
        share2.putpixel((i, j), map_bit_to_rgb(share2_vc_value))
        out.putpixel((i, j), map_bit_to_rgb(out_vc_value))

# Salvo immagini
share1.save("../../../resources/output/vc/basic/share1.png")
share2.save("../../../resources/output/vc/basic/share2.png")
out.save("../../../resources/output/vc/basic/overlap.png")
```

È interessante notare che la funzione *image.convert()* della libreria Python *Pillow* riporta nella propria descrizione quanto segue:

> "The default method of converting a greyscale ("L") or "RGB" image into a bilevel (mode "1") image uses Floyd-Steinberg dither to approximate the original image luminosity levels."

Questo a sottolineare quanto sia diffuso l'utilizzo dell'algoritmo di Floyd-Steinberg presentato nel capitolo sulla VC. Per questo motivo il codice presentato può essere utilizzato anche per le immagini in scala di grigio, in quanto la funzione *image.convert()* applica l'algoritmo di dithering per convertire l'immagine in bianco e nero.

Nel caso in cui l'input dovesse essere un'immagine a colori, il codice utilizza sempre l'algoritmo di Floyd-Steinberg per convertirla in una bitmap in bianco e nero.

A.2 Immagini a colori

Di seguito, viene presentato il codice che, a partire da un'immagine segreta, consente di estrarre le immagini monocromatiche CMY (Ciano, Magenta, Giallo) e di applicare il processo di dithering. Tale procedura riproduce il processo di stampa di immagini a colori illustrato nel capitolo sulla Crittografia Visuale in Figura (1.11).

Il codice presentato rappresenta anche i passaggi iniziali necessari per la creazione di un algoritmo di crittografia visuale su immagini a colori, come quelli descritti.

A.2.1 Funzioni

```
from PIL import Image


# Funzione che restituisce l'immagine monocromatica sul canale indicato
def zero_other_channels(image, channel):
    zeroed_image = Image.new("CMYK", image.size)

    for y in range(image.size[1]):
        for x in range(image.size[0]):
            pixel = list(image.getpixel((x, y)))

            for c in range(3):
                if c != channel:
                    pixel[c] = 0

            zeroed_image.putpixel((x, y), tuple(pixel))

    return zeroed_image


# Funzione che implementa l'algoritmo dithering di Floyd-Steinberg
def floyd_steinberg_dithering(img, channel):
    dithered_image = Image.new("CMYK", img.size)

    for y in range(img.size[1]):  # altezza
```

```
        for x in range(img.size[0]):  # larghezza
            pixel = list(img.getpixel((x, y)))
            old_value = pixel[channel]

            pixel[channel] = 255 if pixel[channel] > 128 else 0
            new_value = pixel[channel]

            dithered_image.putpixel((x, y), tuple(pixel))

            quant_error = old_value - new_value

            # Parte di Error Diffusion (+ controlli sui pixel ai bordi)
            if x < img.size[0] - 1:
                pixel = list(img.getpixel((x + 1, y)))
                pixel[channel] += quant_error * 7 // 16
                img.putpixel((x + 1, y), tuple(pixel))

                if y < img.size[1] - 1:
                    pixel = list(img.getpixel((x + 1, y + 1)))
                    pixel[channel] += quant_error * 1 // 16
                    img.putpixel((x + 1, y + 1), tuple(pixel))

            if y < img.size[1] - 1:
                pixel = list(img.getpixel((x, y + 1)))
                pixel[channel] += quant_error * 5 // 16
                img.putpixel((x, y + 1), tuple(pixel))

                if x > 0:
                    pixel = list(img.getpixel((x - 1, y + 1)))
                    pixel[channel] += quant_error * 3 // 16
                    img.putpixel((x - 1, y + 1), tuple(pixel))

    return dithered_image


# Funzione che prende tutte le immagini monocromatiche halftoned e le unisce
def merge_dithered_images(cyan_image, magenta_image, yellow_image):
    assert cyan_image.size == magenta_image.size == yellow_image.size, "Le dimensioni
    ↪  delle immagini dithered non corrispondono"
    combined_image = Image.new("CMYK", cyan_image.size)

    for y in range(cyan_image.size[1]):
        for x in range(cyan_image.size[0]):
            cyan_value = cyan_image.getpixel((x, y))[0]
            magenta_value = magenta_image.getpixel((x, y))[1]
            yellow_value = yellow_image.getpixel((x, y))[2]

            pixel = (cyan_value, magenta_value, yellow_value, 0)
            combined_image.putpixel((x, y), pixel)

    return combined_image
```

In questo script, è stata specificamente implementata la funzione di dithering al solo scopo di illustrarne la forma, altrimenti si sarebbe potuto utilizzare funzioni di librerie già esistenti.

A.2.2 Script principale

```
from src.vc.color.functions import *


image = Image.open("../../../resources/cinquecento_color.png")

# Decomposizione di colori
cyan_channel = 0
```

```
magenta_channel = 1
yellow_channel = 2

image = image.convert("CMYK")

# Azzero altri canali => ottengo immagini monocromatiche e le salvo
cyan_monochrome = zero_other_channels(image, cyan_channel)
magenta_monochrome = zero_other_channels(image, magenta_channel)
yellow_monochrome = zero_other_channels(image, yellow_channel)

cyan_monochrome.save("../../../resources/output/vc/color/monochrome_cyan.tiff")
magenta_monochrome.save("../../../resources/output/vc/color/monochrome_magenta.tiff")
yellow_monochrome.save("../../../resources/output/vc/color/monochrome_yellow.tiff")

# Effettuo dithering su ciascuna immagine monocromatica e le salvo
dthrd_cyan = floyd_steinberg_dithering(cyan_monochrome, cyan_channel)
dthrd_magenta = floyd_steinberg_dithering(magenta_monochrome, magenta_channel)
dthrd_yellow = floyd_steinberg_dithering(yellow_monochrome, yellow_channel)

cyan_monochrome.save("../../../resources/output/vc/color/dithered_cyan.tiff")
magenta_monochrome.save("../../../resources/output/vc/color/dithered_magenta.tiff")
yellow_monochrome.save("../../../resources/output/vc/color/dithered_yellow.tiff")

# Combino le immagini tra loro e salvo il risultato
combined_image = merge_dithered_images(dthrd_cyan, dthrd_magenta, dthrd_yellow)
combined_image.save("../../../resources/output/vc/color/combined_image.tiff")
```

Le immagini vengono salvate con l'estensione "*.tiff*" perché questo formato consente di salvare le immagini con valori CMYK anziché i tradizionali valori RGB.

Appendice B

Script su Random Grid

In questa appendice vengono riportati gli script Python creati per generare alcune delle immagini nel capitolo su Random Grid.

B.1 Random Grid con Halftoning

In questa sezione è presentato il codice sorgente completo relativo all'implementazione delle Random Grid basate su halftoning applicate alle immagini in scala di grigi.

Questa variante è stata applicata sia a immagini in scala di grigi, come descritto nel capitolo, sia a immagini binarie, poiché il processo di halftoning su queste ultime non genera alcun risultato significativo.

B.1.1 Funzioni

```python
import numpy as np


# Funzione che crea una random grid della dimensione specificata
def create_first_random_grid(size):
    grid = np.random.randint(2, size=size)
    return grid


# Funzione che crea la seconda random grid secondo la prima equazione di Kafri e Keren
def create_second_random_grid(image, grid):
    transformed_grid = np.where(image == 0, grid, 1 - grid)
    return transformed_grid


# Funzione che sovrappone le due grid con XOR
def overlay_images_XOR(image1, image2):
    overlaid_image = np.bitwise_xor(image1, image2)
    overlaid_image = 1 - overlaid_image
    return overlaid_image


# Funzione che sovrappone le due grid con OR
def overlay_images_OR(image1, image2):
    overlaid_image = np.bitwise_or(image1, image2)
    overlaid_image = 1 - overlaid_image
    return overlaid_image
```

B.1.2 Script principale

```python
from PIL import Image
from src.rg.halftoning.functions import *
```

```
3
4  # Carico l'immagine in scala di grigi e faccio halftoning
5  image = Image.open("../../../resources/secret.jpg").convert('1')
6  image.save("../../../resources/output/rg/halftoning/dithered.png")
7
8  # Trasformo immagine in un array di 0 (bianco) e 1 (nero)
9  image_array = 1 - np.array(image).astype(int)
10
11 # Creo le random grid
12 rg1 = create_first_random_grid(image_array.shape)
13 rg2 = create_second_random_grid(image_array, rg1)
14
15 # Sovrappongo le immagini
16 overlap_OR = overlay_images_OR(rg1, rg2)
17 overlap_XOR = overlay_images_XOR(rg1, rg2)
18
19 # Salvo le random grid e il risultato delle sovrapposizioni
20 image_rg1 = Image.fromarray(rg1.astype(np.uint8) * 255)
21 image_rg1.save("../../../resources/output/rg/halftoning/RG1.png")
22 image_rg2 = Image.fromarray(rg2.astype(np.uint8) * 255)
23 image_rg2.save("../../../resources/output/rg/halftoning/RG2.png")
24
25 XOR_image = Image.fromarray(overlap_XOR.astype(np.uint8) * 255)
26 XOR_image.save("../../../resources/output/rg/halftoning/overlap_XOR.png")
27 OR_image = Image.fromarray(overlap_OR.astype(np.uint8) * 255)
28 OR_image.save("../../../resources/output/rg/halftoning/overlapOR.png")
```

B.2 Random Grid con Bitplane

Di seguito è riportato il codice per l'estrazione dei diversi bitplane da un'immagine. Questo script rappresenta la fase iniziale di un potenziale programma che implementa la cifratura tramite random grid basata proprio sulle bitplane.

La prossima fase prevedrebbe l'uso dell'algoritmo di Kafri e Keren, le cui funzioni sono state implementate nella sezione precedente, applicate alle bitplane più significative che verrebbero successivamente combinate tra loro.

B.2.1 Funzioni

```
1  import numpy as np
2
3
4  # Funzione che restituisce un array di bitplane
5  def slice_bit_plane(img, num_planes):
6      bit_planes = []
7      img_array = np.array(img)
8
9      for i in range(num_planes - 1, -1, -1):
10         # Estraggo il bit i-esimo e lo aggiungo alla bitplane
11         bit_i = (img_array >> i) & 1
12         bit_planes.append(bit_i)
13
14     return bit_planes
```

B.2.2 Script principale

```
1  from PIL import Image
2  from src.rg.bitplane.functions import *
3
4  # Decompongo l'immagine in bitplane
5  img = Image.open('../../../resources/secret.jpg').convert('L')
```

```
bit_planes = slice_bit_plane(img, 8)

# Salvo tutti i bitplane
for i, bit_plane in enumerate(bit_planes):
    bit_plane_image = Image.fromarray((bit_plane * 255).astype(np.uint8))

    path = f'../../../resources/output/rg/bitplane/bitplane{i}.png'
    bit_plane_image.save(path)
```

B.3 Perfect Reconstruction

In questa sezione, viene illustrato il codice che permette di suddividere le immagini in random grid e successivamente di eseguire una ricostruzione perfetta, ovvero fedele all'originale, unendo le griglie.

B.3.1 Funzioni

```
import numpy as np


# Funzione che crea la prima random grid con valori casuali nel range (0-255)
def create_random_grid(size):
    grid = np.random.randint(0, 256, size=size)
    return grid


# Funzione che crea la seconda random grid
def create_difference_grid(image, grid):
    transformed_grid = grid - image
    return transformed_grid


# Funzione che "sovrappone" le due random grid
def overlay_grids(image1, image2):
    overlaid_image = image1 - image2
    return overlaid_image
```

B.3.2 Script principale

```
from PIL import Image
from src.rg.perfectrecon.functions import *

# Carico l'immagine in scala di grigi e costruisco array np
image = Image.open("../../../resources/secret.jpg").convert('L')
image_array = np.array(image).astype(int)

# Creo e salvo la prima random grid
grid1 = create_random_grid(image_array.shape)
grid1_image = Image.fromarray(grid1.astype(np.uint8))
grid1_image.save("../../../resources/output/rg/perfectrecon/grid1.png")

# Creo e salvo la seconda random grid
grid2 = create_difference_grid(image_array, grid1)
grid2_image = Image.fromarray(grid2.astype(np.uint8))
grid2_image.save("../../../resources/output/rg/perfectrecon/grid2.png")

# Effettuo una "sovrapposizione" delle due grid
over = overlay_grids(grid1, grid2)
over_image = Image.fromarray(over.astype(np.uint8))
over_image.save("../../../resources/output/rg/perfectrecon/over.png")
```

Appendice C

Script di Steganografia LSB

In questa appendice è fornito uno script che consente di realizzare la steganografia LSB.

C.1 Funzione di codifica

```
# Terminatore del messagio segreto
DELIMITER = '$$$'


# Funzione che converte una stringa di caratteri in una stringa binaria
def from_string_to_bin(string):
    return ''.join([format(ord(c), "08b") for c in string])


# Funzione che dato un messaggio e un oggetto di copertura applica steganografia LSB
def encode(img_array, message):
    # Aggiungo terminatore alla fine del messaggio segreto e converto in binario
    message += DELIMITER
    binary_message = from_string_to_bin(message)

    # Variabile che permette di ricostruire la cover-image nella fase di decodifica
    lsb_recovery = []

    # Calcolo dimensioni
    required_pixels = len(binary_message)
    # print("Pixel richiesti: ", required_pixels, "\t Pixel disponibili: ",
    ↪  img_array.size)
    if required_pixels > img_array.size:
        return None, None

    # Applico steganografia
    else:
        index = 0

        for i in range(img_array.shape[0]):
            for j in range(img_array.shape[1]):

                if index < required_pixels:
                    # Salvo bit che verrà sostituito per eventuale recovery
                    lsb = img_array[i][j] & 1
                    lsb_recovery.append(lsb)

                    # Sostituisco l'ultimo bit del pixel con un bit di segreto
                    img_array[i][j] = (int(img_array[i][j]) & 0xFE) |
                    ↪  int(binary_message[index])
                    index += 1

                else:
                    break
```

```
43
44          return img_array, lsb_recovery
```

Con l'istruzione '*img_array[i][j] = (int(img_array[i][j]) & 0xFE) | int(binary_message[index])*' sostituisco l'ultimo bit meno significativo di *img_array[i][j]* con il valore del bit corrispondente in *binary_message[index]*.

C.1.1 Funzione di decodifica

```
1  # Funzione che converte una stringa binaria in una stringa di caratteri
2  def from_bin_to_string(bin):
3      return ''.join([chr(int(bin[k:k + 8], 2)) for k in range(0, len(bin), 8)])
4
5
6  # Funzione che riceve uno stego-object e restituisce il segreto.
7  # Se gli viene fornito l'array lsb_recovery restituisce anche il cover-obj ricostruito
8  def decode(img_array, lsb_recovery=None):
9      hidden_bits = ""
10     complete_secret = False
11     index = 0
12
13     delimiter_binary = ''.join(format(ord(c), '08b') for c in DELIMITER)
14     delimiter_length = len(delimiter_binary)
15
16     for i in range(img_array.shape[0]):
17         for j in range(img_array.shape[1]):
18             # Recupero bit di segreto
19             pixel_lsb = img_array[i][j] & 1
20             hidden_bits += str(pixel_lsb)
21
22             # Se c'è a disposizione lsb_recovery => steganografia reversibile
23             if lsb_recovery is not None:
24                 img_array[i][j] = (int(img_array[i][j]) & 0xFE) | lsb_recovery[index]
25                 index += 1
26
27             # Se ho letto la stringa di terminazione completa
28             if hidden_bits[-delimiter_length:] == delimiter_binary:
29                 # Rimuovo tale stringa dal segreto ed esco dai due for
30                 hidden_bits = hidden_bits[:-delimiter_length]
31                 complete_secret = True
32                 break
33
34         if complete_secret:
35             break
36
37     # Restituisco la stringa di segreto e se presente anche la cover image
38     if complete_secret:
39         if lsb_recovery is not None:
40             return from_bin_to_string(hidden_bits), img_array
41         else:
42             return from_bin_to_string(hidden_bits), None
43     else:
44         return None, None
```

C.2 Script principale

```
1  import numpy as np
2  from PIL import Image
3  from src.steganography.lsb.functions import *
4
5  # Apro e converto cover image in scala di grigi e poi in np.array
```

```
6  cover_image = Image.open("../../../resources/secret.jpg").convert('L')
7  cover_array = np.array(cover_image)
8
9
10 # Faccio encoding e salvo stego-image
11 message = 'Messaggio segreto di Christian'
12 stego_array, lsb_array = encode(cover_array, message)
13
14 if stego_array is not None:
15     stego_image = Image.fromarray(stego_array.astype('uint8'))
16     stego_image.save("../../../resources/output/lsb/stego_image.png")
17
18
19 # Faccio decoding e stampo messaggio segreto
20 secret, _ = decode(stego_array)
21
22 if secret is not None:
23     print(secret)
24
25 """
26 Se volessi ottenere una steganografia reversibile, è sufficiente passare lsb_array alla
   ↪  funzione decode(). In tale caso mi verrebbe restiuito, come secondo parametro un
   ↪  recovered_array che sarà identico al cover_array
27 """
```

Appendice D

Script di Elaborazione file DICOM

In questa appendice è fornito uno script che consente di visualizzare i metadati di un'immagine DICOM e di eseguirne la conversione e il salvataggio in formato PNG.

D.1 Funzioni

```
import numpy as np
from PIL import Image
from pydicom import Dataset

# Lista di tag considerati sensibili
sensitive_tags = [
    '(0010, 0010)', '(0010, 0030)', '(0010, 0040)', '(0010, 0020)',
    '(0008, 0050)', '(0008, 0090)', '(0010, 1040)', '(0010, 1010)',
    '(0008, 0080)', '(0008, 1030)', '(0008, 103E)', '(0020, 0010)',
    '(0008, 0020)', '(0008, 0030)', '(0020, 000D)', '(0008, 0018)'
]


# Funzione che dato un tag dice se è considerato sensibile o no
def is_sensitive_metadata(tag):
    return str(tag) in sensitive_tags


# Funzione che restituisce un Dataset contenente i metadati considerati sensibili
def get_sensitive_metadata(ds):
    sensitive_metadata = Dataset()

    for elem in ds:
        if is_sensitive_metadata(elem.tag):
            sensitive_metadata.add(elem)

    return sensitive_metadata


# Funzione che restituisce tutti i metadati del Dataset e i Metafile
def get_all_metadata_json(ds):
    del ds.PixelData

    json_data = {
        "dataset_json": ds.to_json(),
        "metafile_json": ds.file_meta.to_json()
    }

    return json_data


# Funzione che prende i pixel data, salva immagine convertita nel path indicato e
↪  restituisce l'immagine in forma array
```

```python
def from_DataPixel_to_PNG(pixel_data_array, destination_path):
    image_data = pixel_data_array.astype(float)
    rescaled_image = (np.maximum(image_data, 0) / image_data.max()) * 255
    final_image = np.uint8(rescaled_image)

    imagePIL = Image.fromarray(final_image)
    imagePIL.save(destination_path)

    return final_image


# Funzione che prende il percorso dell'immagine PNG, e ne ricava i pixel data per ricostruire il file DICOM
def from_PNG_to_DataPixel(png_image_path):
    img = Image.open(png_image_path)
    img_array = np.array(img.getdata(), dtype=np.uint8)

    return img_array.tobytes()
```

I "*sensitive_tags*" sono una lista di tag che dovrebbe poter essere personalizzata in base alle esigenze.

Poiché i file DICOM possono variare notevolmente in base al tipo di esame medico, all'apparecchiatura utilizzata e all'istituzione medica, è importante adattare la lista in modo da identificare e gestire in modo appropriato tutti i metadati rilevanti per lo specifico caso d'uso.

D.2 Script principale

```python
from pydicom import dcmread
from src.dicom.functions import *

# dcmread: legge file DICOM e restituisce un Dataset
pathDICOM = '../../resources/file.dcm'
DICOM_file = dcmread(pathDICOM)

# Effettuo una conversione dell'immagine DICOM in PNG e la salvo
from_DataPixel_to_PNG(DICOM_file.pixel_array, '../../resources/output/dicom/converted.png')

# Stampo i metadati considerati sensibili
sensitive_metadata = get_sensitive_metadata(DICOM_file)
print(sensitive_metadata)
```

Nell'elaborazione in Python di questi file è comune l'utilizzo della libreria *matplotlib.pyplot*. Nonostante ciò, nel codice riportato, si è optato per l'uso di Pillow, in quanto, tale libreria, è stata utilizzata in tutti i precedenti script.

Appendice E

Progetto dcm-sharing

E.1 Script principale

```
#!/usr/bin/env python3

import argparse
import sys
from argparse import RawTextHelpFormatter

from Encrypt import *
from Decrypt import *
from Reconstruct import *


# Funzione che verifica l'esistenza delle cartelle indicate
def check_folder_existence(path):
    return os.path.exists(path) and os.path.isdir(path)


# Creo parser
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS,
↪  formatter_class=RawTextHelpFormatter)

# Aggiungo descrizione personalizzata all'helper
custom_help = """
DICOM Image Encryptor/Decryptor

Syntax for encryption:
    ./dcm-sharing.py -e -m <partial | all> -ef <txt | json> -f <DICOM FILE> -o <FOLDER>

Syntax for decription:
    ./dcm-sharing -d -m <partial | all> -ef <txt | json> -i <FOLDER>

If you want to ensure that the DICOM file is reconstructable during decryption, you must
↪  set the options '-m all' and '-ef json' both when encrypting and decrypting.
"""

parser.description = custom_help

# Modalità
parser.add_argument("-e", "--encrypt", help="Encrypt a DICOM file", action="store_true")
parser.add_argument("-d", "--decrypt", help="Decrypt a DICOM file", action="store_true")

# Argomenti comuni
parser.add_argument("-m", "--metadata", choices=["all", "partial"], required=True,
↪  help="When set to 'partial', only pixel data and PHI reconstruction is possible. When
↪  set to 'all', full DICOM file reconstruction is possible (required).")
parser.add_argument("-ef", "--encoding-format", choices=["json", "txt"], required=True,
↪  help="Desired encoding format for metadata (required).")
```

```
42
43  # Argomenti per la cifratura
44  parser.add_argument("-f", "--file", help="Path to the DICOM file (required for
    ↪  encryption)")
45  parser.add_argument("-o", "--output-folder", help="Path to the output folder for results
    ↪  (required for encryption)")
46
47  # Argomenti per la decifratura
48  parser.add_argument("-i", "--input-folder", help="Path to the folder containing random
    ↪  grids (required for decryption)")
49
50  # Parsing argomenti
51  args = parser.parse_args()
52
53
54  # Controllo gli argomenti
55  if args.encrypt and args.decrypt:
56      print("Error: Specify a unique action to perform (-e for encryption, -d for
        ↪  decryption).")
57
58  elif args.encrypt:
59      if args.file and args.output_folder and args.metadata and args.encoding_format:
60
61          if check_folder_existence(args.output_folder) is True:
62              print("Encrypting DICOM file...")
63
64              if encrypt(args.file, args.metadata, args.encoding_format,
                ↪  args.output_folder) is None:
65                  sys.exit()
66              else:
67                  print("Encryption completed.")
68
69          else:
70              print(f"Error: The path '{args.output_folder}' does not exist or is not a
                ↪  valid folder.")
71
72      else:
73          print("Error: For encryption, you must specify -m (metadata), "
74                "-ef (encoding format), -f (file) and -o (output folder) options.")
75
76  elif args.decrypt:
77      if args.input_folder and args.metadata and args.encoding_format:
78
79          if check_folder_existence(args.input_folder) is True:
80              print("Decrypting DICOM file...")
81
82              if decrypt(args.input_folder, args.metadata, args.encoding_format) is None:
83                  sys.exit()
84              else:
85                  print("Decryption completed.")
86
87              # Se modalità 'all metadata' con encoding 'json' ho a disposizione tutti i
                ↪  file per ricostruire
88              if args.metadata == 'all' and args.encoding_format == 'json':
89                  print("Reconstructing DICOM file...")
90
91                  if reconstruction(args.input_folder) is None:
92                      sys.exit()
93                  else:
94                      print("Reconstruction completed.")
95
96          else:
97              print(f"Error: The path '{args.output_folder}' does not exist or is not a
                ↪  valid folder.")
98
```

```
    else:
        print("Error: For decryption, you must specify -m (metadata), -ef (encoding format) "
              "and -i (input folder) options.")

else:
    print("Error: Specify a valid action to perform (-e for encryption, -d for decryption).")
```

E.2 Encrypt

```
import json
import os
import pydicom
from pydicom import dcmread

from dicom.functions import *
from steganography.lsb.functions import *
from rg.perfectrecon.functions import *


# Funzione che cifra un file DICOM con le varie opzioni settate
def encrypt(path_dicom, metadata_choice, encoding_format, path_output):
    pathConvertedDicom = path_output + "tmp/converted_DICOM.png"
    pathStegoImage = path_output + "tmp/stego_object.png"
    pathReconBits = path_output + "reconstruction_bits.txt"

    pathRandomGrid1 = path_output + "grid1.png"
    pathRandomGrid2 = path_output + "grid2.png"

    # Apro DICOM file e lo verifico
    try:
        DICOM_file = dcmread(path_dicom, force=True)

    except pydicom.errors.InvalidDicomError:
        print("Error: The file is not a valid DICOM file.")
        return None

    except FileNotFoundError:
        print(f"Error: The file {path_dicom} does not exist.")
        return None

    except Exception as e:
        print(f"An error occurred: {str(e)}")
        return None

    # Verifico se esiste cartella tmp, altrimenti la creo
    if not os.path.exists(path_output + "/tmp"):
        os.makedirs(path_output + "/tmp")

    # LSB: Ottengo cover-image dal file DICOM
    cover_array = from_DataPixel_to_PNG(DICOM_file.pixel_array, pathConvertedDicom)

    # LSB: Preparo il segreto con i metadati da nascondere
    secret_metadata = ""

    if metadata_choice == 'partial':
        sensitive_dataset = get_sensitive_metadata(DICOM_file)

        if encoding_format == 'txt':
            for md in sensitive_dataset:
                secret_metadata += (f"Tag: {md.tag} | "
                                    f"Descrizione: {md.name} | "
```

```
                                    f"Valore: {md.value}; \n")

        else:  # encoding_format == 'json'
            secret_metadata = sensitive_dataset.to_json()

    else:  # metadata_choice == 'all':
        if encoding_format == 'txt':
            # Cancello i PixelData, li ricostruirò dall'immagine PNG
            del DICOM_file.PixelData

            for md in DICOM_file:
                secret_metadata += (f"Tag: {md.tag} | "
                                    f"Descrizione: {md.name} | "
                                    f"Valore: {md.value}; \n")

        else:  # encoding_format == 'json'
            json_metadata = get_all_metadata_json(DICOM_file)
            secret_metadata = json.dumps(json_metadata)

    # LSB: Costruisco e salvo stego-object
    stego_array, lsb_array = encode(cover_array, secret_metadata)

    if stego_array is None:
        print("Error: Message too long")
        return None

    stego_image = Image.fromarray(stego_array.astype('uint8'))
    stego_image.save(pathStegoImage)

    # LSB: Se richiesto salvo i bit che consentiranno la ricostruzione
    if metadata_choice == 'all' and (lsb_array is not None):
        with open(pathReconBits, 'w') as file:
            for bit in lsb_array:
                file.write(str(bit))

    # RG Perfect Reconstruction: Creo e salvo la prima random grid
    grid1 = create_random_grid(stego_array.shape)
    grid1_image = Image.fromarray(grid1.astype(np.uint8))
    grid1_image.save(pathRandomGrid1)

    # RG Perfect Reconstruction: Creo e salvo la seconda random grid
    grid2 = create_difference_grid(stego_array, grid1)
    grid2_image = Image.fromarray(grid2.astype(np.uint8))
    grid2_image.save(pathRandomGrid2)

    return True
```

E.3 Decrypt

```
import os
from PIL import Image

from steganography.lsb.functions import *
from rg.perfectrecon.functions import *


# Funzione che decifra le random grid ottenute da un file DICOM con le varie opzioni
↪  settate
def decrypt(folder_path, metadata_choice, encoding_format):
    pathRandomGrid1 = folder_path + "grid1.png"
    pathRandomGrid2 = folder_path + "grid2.png"

    pathReconBits = folder_path + "reconstruction_bits.txt"
```

```
    pathStegoImage = folder_path + "tmp/stego_object.png"
    pathConvertedDicom = folder_path + "tmp/converted_DICOM.png"

    pathSecretTxt = folder_path + "secret.txt"
    pathSecretJson = folder_path + "tmp/secret.json"

    # Apro le due immagini contenenti le random grid
    try:
        grid1_img = Image.open(pathRandomGrid1)
        grid2_img = Image.open(pathRandomGrid2)

    except FileNotFoundError:
        print(f"Error: At least one of the files, '{pathRandomGrid1}' or
        ↪  '{pathRandomGrid2}' does not exist.")
        return None

    except Exception as e:
        print(f"An error occurred: {str(e)}")
        return None

    # Verifico se esiste cartella tmp, altrimenti la creo
    if not os.path.exists(folder_path + "/tmp"):
        os.makedirs(folder_path + "/tmp")

    # RG Perfect Reconstruction: ricostruisco e salvo lo stego-object
    grid1_array = np.array(grid1_img).astype(int)
    grid2_array = np.array(grid2_img).astype(int)
    stego_array = overlay_grids(grid1_array, grid2_array)

    stego_image = Image.fromarray(stego_array.astype(np.uint8))
    stego_image.save(pathStegoImage)

    # LSB: recupero segreto come stringa
    bit_array = None

    if metadata_choice == 'all':
        try:
            with open(pathReconBits, 'r') as file:
                bit_string = file.read()
                bit_array = [int(bit) for bit in bit_string]

        except FileNotFoundError:
            print(f"Error: The file '{pathReconBits}' required for decryption in 'all
            ↪  metadata' mode does not exist.")
            return None

        except Exception as e:
            print(f"An error occurred: {str(e)}")
            return None

    secret, cover_array = decode(stego_array, bit_array)

    if secret is None:
        print("Error: Unable to reconstruct the secret.")
        return None

    # LSB: se è avvenuta ricostruzione del cover object la salvo
    if cover_array is not None:
        cover_image = Image.fromarray(cover_array.astype(np.uint8))
        cover_image.save(pathConvertedDicom)

    # LSB: salvo segreto in formato txt o json
    if encoding_format == 'txt':
        with open(pathSecretTxt, 'w') as file:
            file.write(secret)
```

```
77
78      else:  # encoding_format == 'json':
79          with open(pathSecretJson, 'w') as file:
80              file.write(secret)
81
82      return True
```

E.4 Reconstruction

```
1   import json
2   import pydicom
3   from dicom.functions import from_PNG_to_DataPixel
4
5   # Funzione che ricostruisce il file DICOM recuperando i metadati dal segreto decifrato ed
    ↪  i valori dei pixel dell'immagine convertita
6   def reconstruction(folder_path):
7       pathSecretJson = folder_path + "tmp/secret.json"
8       pathReconstructedDicom = folder_path + "reconstructed.dcm"
9       pathConvertedDicom = folder_path + "tmp/converted_DICOM.png"
10
11      # Apro file JSON
12      try:
13          with open(pathSecretJson, 'r') as file:
14              data_json = json.load(file)
15
16      except FileNotFoundError:
17          print(f"Error: The file '{pathSecretJson}' required for reconstructioning does
            ↪  not exist.")
18          return None
19
20      except Exception as e:
21          print(f"An error occurred: {str(e)}. \n"
22                f"Please check if you are trying to decrypt a file encrypted with '-ef txt'
                  ↪  using '-ef json'.")
23          return None
24
25      # Estraggo i due oggetti
26      dataset_json = json.dumps(data_json["dataset_json"])
27      metafile_json = json.dumps(data_json["metafile_json"])
28
29      # Creo dataset
30      ds = pydicom.Dataset.from_json(json.loads(dataset_json))
31      ds.file_meta = pydicom.Dataset.from_json(json.loads(metafile_json))
32      ds.BitsAllocated = 8
33
34      # Recupero valori dei pixel dall'immagine PNG e li inserisco nel Dataset
35      ds.PixelData = from_PNG_to_DataPixel(pathConvertedDicom)
36      ds.save_as(pathReconstructedDicom)
37
38      return True
```

E.5 Text-to-image alghoritm

```
1   import math
2   import numpy as np
3   from PIL import Image
4
5
6   # Funzione di encoding: trasforma una stringa in immagine in scala di grigi
```

```python
def string_to_grayscale_image(input_string, output_file):
    # Converte ogni carattere della stringa nel suo valore ASCII
    pixel_values = [ord(char) for char in input_string]
    input_length = len(input_string)

    # Crea un'immagine quadrata vuota
    image_size = int(math.ceil(math.sqrt(input_length)))
    grayscale_image = Image.new("L", (image_size, image_size))

    # Carica i pixel e salva l'immagine
    grayscale_image.putdata(pixel_values)
    grayscale_image.save(output_file)

    # Converte l'immagine in un array np e lo restituisce
    grayscale_array = np.array(grayscale_image).astype(int)

    return grayscale_array


# Funzione di decoding: trasforma l'immagine in scala di grigi nella corrispondente stringa
def grayscale_image_to_string(input_file):
    # Recupero pixel con valori ASCII "significativi"
    grayscale_image = Image.open(input_file)
    pixel_values = list(grayscale_image.getdata())
    meaningful_pixels = [pixel for pixel in pixel_values if pixel >= 32]

    # Converto valori in caratteri e creo stringa
    char_values = [chr(pixel) for pixel in meaningful_pixels]
    decoded_string = ''.join(char_values)

    return decoded_string
```

Bibliografia

[1] Hannah T. Neprash, Claire C. McGlave, Dori A. Cross, Beth A. Virnig, Michael A. Puskarich, Jared D. Huling, Alan Z. Rozenshtein, and Sayeh S. Nikpay. Trends in Ransomware Attacks on US Hospitals, Clinics, and Other Health Care Delivery Organizations, 2016-2021. *JAMA Health Forum*, 3(12):e224873–e224873, 12 2022.

[2] Jonathan Weir and WeiQi Yan. A comprehensive study of visual cryptography. *Trans. Data Hiding Multim. Secur.*, 5:70–105, 2010.

[3] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[4] Akshay G. Bhosale and Vikram S. Patil. A (2, 2) visual cryptography technique to share two secrets. In *2020 International Conference on Inventive Computation Technologies (ICICT)*, pages 563–569, 2020.

[5] Giuseppe Ateniese, Carlo Blundo, Alfredo De Santis, and Douglas R Stinson. Extended capabilities for visual cryptography. *Theoretical Computer Science*, 250(1-2):143–161, 2001.

[6] Giuseppe Ateniese, Carlo Blundo, Alfredo De Santis, and Douglas R Stinson. Visual cryptography for general access structures. *Information and computation*, 129(2):86–106, 1996.

[7] Wei Qiao, Hongdong Yin, and Huaqing Liang. A kind of visual cryptography scheme for color images based on halftone technique. In *2009 International Conference on Measuring Technology and Mechatronics Automation*, volume 1, pages 393–395, 2009.

[8] Young-Chang Hou. Visual cryptography for color images. *Pattern Recognition*, 36(7):1619–1629, 2003.

[9] Jonathan Weir and WeiQi Yan. Sharing multiple secrets using visual cryptography. In *2009 IEEE International Symposium on Circuits and Systems*, pages 509–512, 2009.

[10] Jonathan Patrick Weir. *Visual cryptography and its applications*. Bookboon, 2011.

[11] Anjney Pandey and Subhranil Som. Applications and usage of visual cryptography: A review. In *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pages 375–381, 2016.

[12] Oded Kafri and E Keren. Keren, e.: Encryption of pictures and shapes by random grids. opt. lett. 12, 377-379. *Optics letters*, 12:377–9, 07 1987.

[13] J Vahidi, M Riyahi, and R Motevalli. A new approach for gray scale image encryption by random grids. *International journal of mechatronics, Electrical and Computer Technology*, 5(16):2169–2174, 2015.

[14] Shuliang Sun. A new information hiding method based on improved bpcs steganography. *Adv. Multim.*, 2015:698492:1–698492:7, 2015.

[15] Wikipedia. Bitplane.

[16] Roberto De Prisco and Alfredo De Santis. On the relation of random grid and deterministic visual cryptography. *IEEE transactions on information forensics and security*, 9(4):653–665, 2014.

[17] Harpreet Kaur and Jyoti Rani. A survey on different techniques of steganography. *MATEC Web of Conferences*, 57:02003, 05 2016.

[18] Neil F Johnson and Sushil Jajodia. Exploring steganography: Seeing the unseen. *Computer*, 31(2):26–34, 1998.

[19] SHRIKANT KHAIRE and Sanjay Nalbalwar. Review: Steganography – bit plane complexity segmentation (bpcs) technique. *International Journal of Engineering Science and Technology*, 2, 09 2010.

[20] Ki-Hyun Jung and Yoo Kee-Young. Three-directional data hiding method for digital images. *Cryptologia*, 38, 04 2014.

[21] Tzu-Chuen Lu and Thanh Nhan Vo. 10 - reversible steganography techniques: A survey. In Mahmoud Hassaballah, editor, *Digital Media Steganography*, pages 189–213. Academic Press, 2020.

[22] Jim Bartel. Steganalysis overview, 2007. Accessed: August 2, 2023.

[23] LibreTexts K12 Education. Study of space by the electromagnetic spectrum.

[24] Los Alamos National Laboratory. What is the difference between x-rays, ct scans, and mris?

[25] Richard Bibb, Dominic Eggbeer, and Abby Paterson. 2-medical imaging. *Medical Modelling (Second Edition) Woodhead Publishing*, pages 7–34, 2015.

[26] M.H. Lev and R.G. Gonzalez. 17 - ct angiography and ct perfusion imaging. In Arthur W. Toga and John C. Mazziotta, editors, *Brain Mapping: The Methods (Second Edition)*, pages 427–484. Academic Press, San Diego, second edition edition, 2002.

[27] Felix Ritter, Tobias Boskamp, A. Homeyer, Hendrik Laue, Michael Schwier, Florian Link, and H.-O. Peitgen. Medical image analysis. *IEEE Pulse*, 2(6):60–70, 2011.

[28] Shervin Kamalian, Michael H. Lev, and Rajiv Gupta. Chapter 1 - computed tomography imaging and angiography – principles. In Joseph C. Masdeu and R. Gilberto González, editors, *Neuroimaging Part I*, volume 135 of *Handbook of Clinical Neurology*, pages 3–20. Elsevier, 2016.

[29] U.S. Food and Drug Administration. Ultrasound Imaging.

[30] UVA Health. Different imaging tests.

[31] MD Cory Calendine. Bone and joint imaging comparison: Xray, ct, mri.

[32] Roseville California Clinic. Mri vs. x-ray in treatment of disc bulges.

[33] Natasha Morales Drissi. *Brain Networks and Dynamics in Narcolepsy*. Linköping University Electronic Press, 01 2019.

[34] Michael P. Notter. Introduction to neuroimaging.

[35] rmnonline. Principi informatici RM.

[36] Wikipedia. Piano anatomico.

[37] Neal C Dalrymple, Srinivasa R Prasad, Michael W Freckleton, and Kedar N Chintapalli. Introduction to the language of three-dimensional imaging with multidetector ct. *Radiographics*, 25(5):1409–1428, 2005.

[38] Victor Powell and Lewis Lehe. Image kernels.

[39] Neha Baraiya and Hardik Modi. Comparative study of different methods for brain tumor extraction from mri images using image processing. *Indian Journal of Science and Technology*, 9, 01 2016.

[40] Prateek Chhikara. Understanding morphological image processing and its operations, 03 2022.

[41] Mario Mustra, Kresimir Delac, and Mislav Grgic. Overview of the dicom standard. In *2008 50th International Symposium ELMAR*, volume 1, pages 39–44, 2008.

[42] Luís Bastião Silva. *Medical imaging services supported on cloud*. PhD thesis, Universidade de Aveiro, 02 2014.

[43] Medical Imaging e Technology Alliance (a division of NEMA). Dicom lookup.

[44] Dandu Ravi Varma. Managing DICOM images: Tips and tricks for the radiologist. *Indian J Radiol Imaging*, 22(1):4–13, 2012.

[45] Ahmed Elngar, Ambika Pawar, and Prathamesh Churi. *Data protection and privacy in healthcare: research and innovations*. CRC Press, 2021.

[46] James Wiggins. De-identify medical images with the help of amazon comprehend medical and amazon rekognition. Amazon Web Services Blog, 2019.