UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Laurea Magistrale in Informatica

Curriculum: Sicurezza Informatica

Insegnamento: Compressione Dati

Docenti: Bruno Carpentieri

RELAZIONE PROGETTO

Implementazione "Reversible Data hiding in Encrypted Image"

PARTECIPANTI

Luca Kesler
Giuseppe Scotti
Clemente Vigliotti
Laura Zollo

Sommario

1. Introduzione

Il progetto consiste nell'implementazione dello schema di inserimento di dati nascosti in immagini criptate totalmente ricostruibili, proposto da Ximpeng Zhang, professore dell' Università di Shangai.

2. Paper

2.1 Introduzione

Il paper [1] propone un metodo per inserire dei dati, modificando una piccola parte di un' immagine non compressa ma cifrata con uno stream cipher. Decifrando l'immagine con la chiave di decifratura si ottiene un'immagine simile a quella originale. Decifrando secondo la chiave data-hiding, con l'aiuto della correlazione spaziale nelle immagini naturali, si recuperano dati e immagini.
Questo metodo potrebbe essere utile in ambiti (medico, militare, ecc) in cui è necessario cifrare un'immagine e successivamente aggiungere ad essa dati, a prescindere dal contenuto dell'immagine, da una terza parte. In un momento successivo, un amministratore di database potrebbe dover aggiungere informazioni sul paziente direttamente nell'immagine senza però introdurre differenze all'immagine decifrata con quella originale.

2.2 Schema proposto

L'immagine originale non compressa viene cifrata con una chiave di cifratura, poi vengono aggiunti i dati utilizzando un'altra chiave, indipendentemente dal contenuto dei dati originali. Il receiver decifra l'immagine con la chiave di cifratura ottenendo un'immagine simile all'originale e in seguito estrae i dati aggiunti con la seconda chiave ottenendo anche l'immagine originale.
L'immagine viene cifrata facendo lo XOR esclusivo tra i singoli bit di ogni pixel (con profondità di 8 bit - i valori appartengono all'intervallo [0, 255]) e una stringa di bit pseudorandom ottenuta da uno stream cipher. La sicurezza della cifratura dipenderà dalla sicurezza dello stream cipher.

L'immagine cifrata poi viene suddivisa in blocchi non sovrapposti sxs . Ogni blocco viene utilizzato per portare un bit di informazione addizionale. Quindi, utilizzando la chiave data-hiding, un blocco (con s^2 pixel) viene diviso in maniera pseudorandom in due insiemi: S0 e S1 (non conoscendo la chiave un bit può appartenere con probabilità di circa ½ a S0 o S1). Se il bit da aggiungere è 0, si flippano (si fa il negato) dei 3 least significant bit dei pixel in S0, altrimenti dei pixel di S1. Gli altri bit non vengono modificati.

Quando il receiver riceve l'immagine cifrata con i dati nascosti, potrà innanzitutto decifrare l'immagine utilizzando lo stesso stream cipher (e la stessa password di cifratura) effettuando semplicemente lo XOR bit a bit. In questo modo otterrà un'immagine molto simile all'originale (ma non identica per via dei bit nascosti al suo interno). Infatti calcolando il PSNR tra l'immagine originale e quella decifrata si otterrà un valore di 37.9 dB, che conferma il fatto che ad occhio nudo le differenze non sono percettibili.

A questo punto è possibile estrarre i dati, individuando ,per ogni blocco, i pixel appartenenti a S0 e quelli appartenenti ad S1 i con la seconda chiave, e flippando in H0 i LSB dei pixel del primo insieme e in H1 i LSB dei pixel appartenenti al secondo insieme.
Per sapere quale tra H0 e H1 è il blocco originale, si sfrutta la correlazione spaziale. In particolare viene proposta una definizione di funzione di fluttuazione:

$$f = \sum_{u=2}^{s-1} \sum_{v=2}^{s-1} \left| p_{u,v} - \frac{p_{u-1,v} + p_{u,v-1} + p_{u+1,v} + p_{u,v+1}}{4} \right|$$

Il blocco con fluttuazione minore sarà il blocco originale, l'altro sarà il blocco con i pixel flippati. Quindi se l'originale è H0, il bit che è stato inserito è 0, altrimenti è 1.
Abbiamo così ottenuto il bit inserito in ogni blocco e il blocco originale per ricostruire l'immagine originale.

2.3 Risultati

Le sperimentazioni effettuate dal professor Ximpeng Zhang sono state effettuate sulla famosa immagine di Lena 512x512 pixel. La dimensione di ogni blocco è sxs=32x32. Il PSNR calcolato conferma i dati teorici. Se la dimensione del blocco è minore di 32x32 si introducono errori di estrazione a causa del decremento della fluttuazione.
Più l'immagine presenta smoothing e migliori sono le performance. Con steganalisi sui LSB è possibile rilevare la presenza dei bit nascosti ma non è possibile recuperare i dati nascosti senza la password.

3. Implementazione

L'implementazione da noi proposta è sviluppata in Matlab in quanto abbiamo già affrontato l'elaborazione delle immagini tramite esso e sono a disposizione funzioni per l'accesso ai singoli bit dei pixel e implementazioni di stream cipher.

3.1 Stream Cipher: Trivium

Uno degli algoritmi utilizzati per il processo di cifratura e decifratura è trivium, un cifrario a flusso con chiave simmetrica sviluppato da De Cannière e Preneel nel 2003. L'algoritmo genera fino a 2^64 bit di output da una chiave ed un vettore di inizializzazione (IV) lunghi entrambi 80 bit.Il trivium si basa su uno stato interno a 288 bit, per inizializzare l'algoritmo vengono eseguite due operazioni fondamentali, il **Key setup** e l' **IV setup,** durante le quali gli 80 bit della chiave e gli 80 bit del IV vengono scritti nei primi 160 bit dello stato interno mentre gli altri bit dello stato, fino al 288° vengono impostati a valori di default. La cosa importante di questo algoritmo è che nessun bit viene usato prima di aver subito almeno 64 rotazioni all'interno dei registri del Trivium.

3.2 Stream Cipher: RC4

Per migliorare le prestazioni complessive del lavoro proposto è stato utilizzato RC4, uno tra i più famosi e diffusi algoritmi di cifratura a flusso a chiave simmetrica anche se paga la sua semplicità in termini di sicurezza: l'algoritmo è violabile con relativa semplicità tanto che il suo uso non è più consigliabile.
L'RC4 genera un flusso di bit pseudo casuali (**keystream**): tale flusso viene combinato in un' operazione di XOR con il testo in chiaro per ottenere il testo cifrato. L'operazione di decifratura avviene alla stessa maniera, passando in input il testo cifrato ed ottenendo in output il testo in chiaro. Per la generazione del keystream l'algoritmo utilizza una S-box di 256 byte e 2 indici da 8 bit indicati con i e j. La chiave fornita dall'utente è generalmente compresa tra 40 e 256 bit e viene utilizzata per inizializzare l'S-box mediante una funzione di Key-Scheduling Algorithm. Una volta che questo passaggio è completato, il flusso di bit generato (utilizzando una funzione PRGA) è combinato con il testo in chiaro mediante XOR.

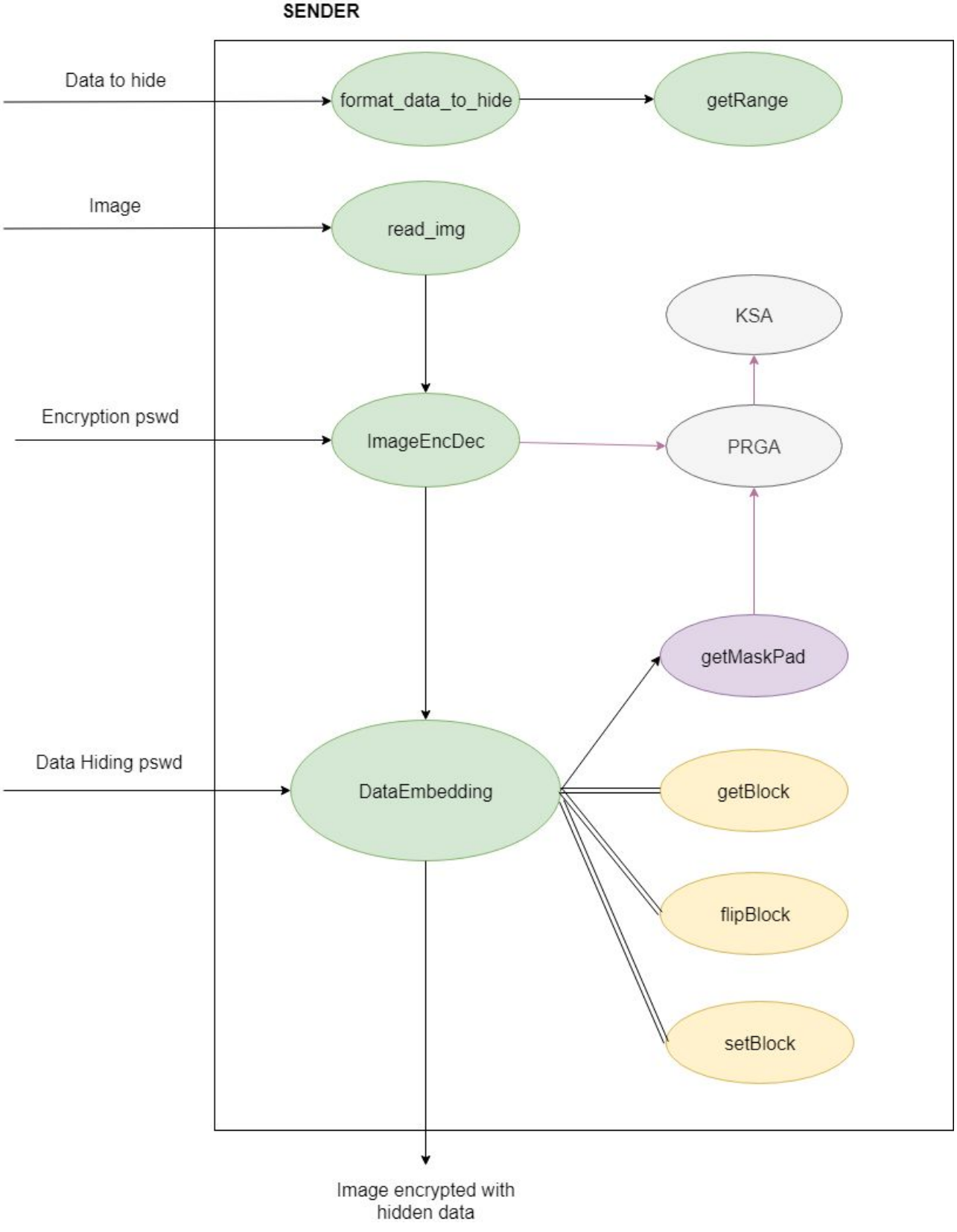
3.3 Funzionalità implementate

Le funzionalità implementate sono:

1. Caricamento immagine e conversione in scala di grigi (immagine a due dimensioni con profondità di 8 bit);
2. Conversione dei dati da nascondere in stringa di bit;
3. Calcolo delle possibili lunghezze della stringa di bit da inserire;
4. Cifratura (e decifratura) dell'immagine con stream cipher RC4[2];
5. Parallelizzazione di RC4;
6. Cifratura (e decifratura) dell'immagine con Trivium (sconsigliata)[3];
7. Parallelizzazione di Trivium(sconsigliata);
8. Inserimento dei dati da nascondere:
    a. estrazione del blocco (i,j) di dimensione s;
    b. flip del blocco in base al bit da inserire e in base alla divisione pseudocasuale definita dallo stream cipher (RC4 o Trivium);

9. Estrazione dei dati e dell'immagine originale;
10. Conversione della stringa di bit inseriti in stringa di caratteri.

Ecco uno schema generale del funzionamento di SENDER e RECEIVER:



FUNZIONE:

format_data_to_hide

INPUT:

stringa di caratteri

OUTPUT:

stringa di bit

DESCRIZIONE:

Se il segreto da nascondere è una stringa di caratteri, questa funzione la converte in una stringa di bit.

FUNZIONE:

read_img

INPUT:

path dell'immagine

OUTPUT:

matrice di 2 dimensioni con valori compresi tra 0 e 255

DESCRIZIONE:

Legge l'immagine e se il formato è RGB o non in scala di grigi la converte.

FUNZIONE:

getRange

INPUT:

matrice

OUTPUT:

array di interi

DESCRIZIONE:

calcola tutte le possibili lunghezze dei messaggi in bit da inserire (numero di blocchi in cui potrà essere divisa l'immagine).

FUNZIONE:

imageEncDec

INPUT:

matrice, stringa (chiave di cifratura)

OUTPUT:

matrice

DESCRIZIONE:

calcola una matrice di bit pseudo-random delle stesse dimensioni dell'immagine in input utilizzando PRGA (che a sua volta utilizza la funzione KSA) ed effettua lo XOR tra le due matrici

FUNZIONE:

DataEmbedding

INPUT:

stringa di bit (dati), matrice, stringa (chiave data hiding)

OUTPUT:

matrice

DESCRIZIONE:

calcola le dimensioni di un blocco sxs in base alle dimensioni della matrice e dei dati da inserire
calcola una matrice pad delle stesse dimensioni di un blocco con bit pseudo-random (chiama la funzione getMaskPad)
per ogni blocco z (getBlock) flippa il blocco (FlipBlock) in base al pad e allo z-esimo bit da inserire
ricostruisce l'immagine con i blocchi flippati (setBlock)

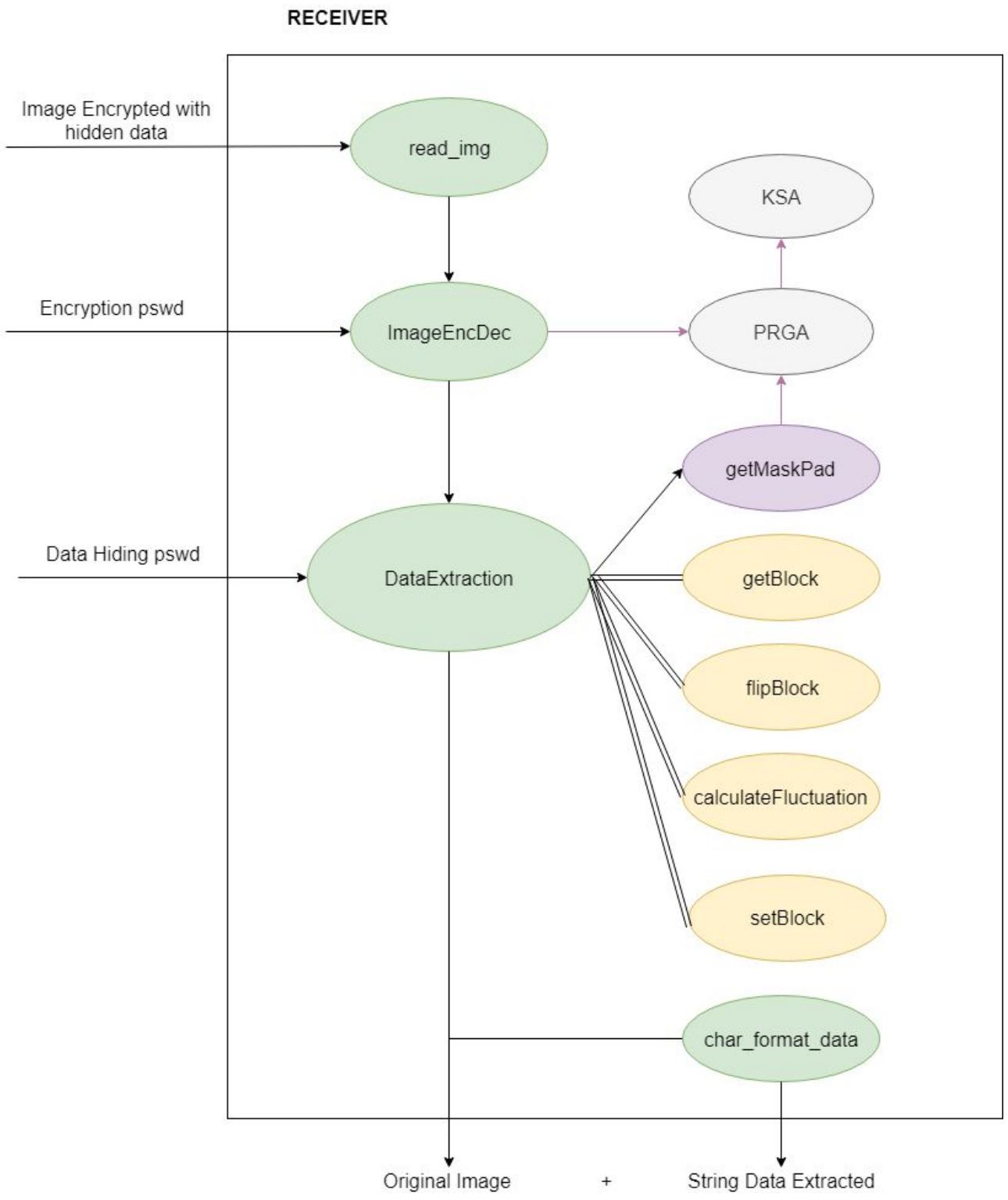
FUNZIONE:
getMaskPad
INPUT:
intero (dimensione s del blocco), stringa (chiave)
OUTPUT:
matrice sxs
DESCRIZIONE:
calcola una matrice pad delle stesse dimensioni di un blocco con bit pseudo-random calcolati con PRGA (poichè PRGA restituisce char, effettua dei calcoli per gestire la discrepanza tra s e la stringa restituita)

FUNZIONE:
getBlock
INPUT:
matrice, intero (dimensione s del blocco), intero (coordinata i del blocco), intero (coordinata j del blocco)
OUTPUT:
matrice (blocco i, j)
DESCRIZIONE:
controlla che esista il blocco (i,j) di dimensione s (e che le dimensioni della matrice siano divisibili per s) e lo restituisce

FUNZIONE:
FlipBlock
INPUT:
matrice (blocco), matrice (pad), intero (0 o 1)
OUTPUT:
matrice
DESCRIZIONE:
se l'intero in input (0 o 1) è proprio uguale a zero il pad viene negato
moltiplica il pad (o il pad negato) per 7(che nello XOR bit a bit sarà 00000111 sicché verranno flippati solo gli ultimi 3 bit) in modo da avere 7 lì dove c'era 1 (pixel dell'insieme S1) e 0 lì dove c'era 0 (pixel dell'insieme S0)
effettua uno XOR bit a bit tra il blocco e il pad modificato

FUNZIONE:
setBlock
INPUT:
matrice (immagine semi-ricostruita), matrice (blocco), intero (i,j), intero (i,j)
OUTPUT:
matrice
DESCRIZIONE:

controlla che esista il blocco (i,j) di dimensione s (e che le dimensioni della matrice siano divisibili per s) e lo inserisce nell'immagine.



FUNZIONE:

DataExtraction

INPUT:

matrice (immagine), stringa di caratteri(chiave data hiding), intero (numero di bit nascosti)

OUTPUT:

matrice (immagine originale), stringa di bit (dati estratti)

DESCRIZIONE:

calcola la dimensione s del blocco in base alle dimensioni della matrice e dei dati nascosti
calcola il pad con la funzione getMaskPad
per ogni blocco (i,j) (getBlock) flippa sia i LSB dei pixel appartenenti all'insieme S0 che a S1 in H0 in e in H1 (FlipBlock)
calcola la fluttuazione di H0 e H1 e la confronta per scegliere quale blocco corrisponde all'originale ed estrarre il bit nascosto
ricostruisce l'immagine originale e la stringa di bit nascosti

FUNZIONE:

char_format_data

INPUT:

stringa di bit

OUTPUT:

stringa di caratteri

DESCRIZIONE:

converte la stringa di bit in stringa di caratteri per recuperare il segreto

FUNZIONE:

parallelPRGA

INPUT:

stringa di caratteri (chiave), intero (numero di bit da generare), intero (numero di core)

OUTPUT:

vettore di interi (pad)

DESCRIZIONE:

Sia p il numero di processori, L la lunghezza della chiave e 'n' la lunghezza del pad che voglio ottenere. Eseguo PRGA per ottenere una stringa di L*p caratteri che verrà divisa in p chiavi di lunghezza L. Eseguo in parallelo p run di PRGA, ognuno con la sua chiave. Da ogni esecuzione si ottengono n/p caratteri che concatenati rappresenteranno l'output.

FUNZIONE:

PSNR

INPUT:

matrice (immagine originale), matrice(immagine decifrata)

OUTPUT:

double

DESCRIZIONE:

Il PSNR o peak signal-to-noise-ratio è una misura adottata per valutare la qualità di un'immagine compressa rispetto all'originale. L'output è ottenuto dal rapporto tra la massima potenza di un segnale (255*255) e la potenza di rumore(ottenuta con la funzione ErrorRate) che può invalidare la fedeltà della rappresentazione compressa. Il tutto espresso in scala logaritmica di decibel. Un PSNR di 37.9 db indica un cambiamento impercettibile.

FUNZIONE:

errorRate

INPUT:

matrice(immagine originale),matrice(immagine decifrata)

OUTPUT:

double(valore dell'energia media dell'errore)

DESCRIZIONE:

Date le due immagini in input, originale e decifrata, l'errore medio viene calcolato solo sui 3 LSB di ogni pixel, in decimale. L'output è dato dalla la sommatoria della differenza al quadrato degli ultimi 3 bit di ogni pixel, diviso il numero di pixel.

FUNZIONE:

calculateFluctuation

INPUT:

matrice(immagine originale)

OUTPUT:

intero(valore di fluttuazione)

DESCRIZIONE:

Data in input un'immagine, la funzione calcola f, ovvero la fluttuazione calcolata come la sommatoria della differenza, per ogni pixel, tra il valore dell'i-esimo pixel e la media del suo 4-intorno.

4. Risultati

Sono stati effettuati vari test su macchina virtuale con 4 core e 8GB di memoria RAM. I nostri risultati confermano i risultati sperimentali ottenuti dalla ricerca originale (i valori di Error rate e PSNR coincidono). Le immagini usate nei nostri test ci hanno sempre permesso di effettuare una ricostruzione fedele al 100% dei dati, tranne in un caso.
Infatti sull'immagine *baboon_color.bmp* si commette un errore sull'estrazione del bit dal 71-esimo blocco, questo poiché la fluttuazione del blocco originale H1 ottenuto flippando i bit del blocco decriptato è maggiore di quella del blocco H2 (blocco sbagliato).
Questo comportamento è probabilmente generato dal fatto che l'immagine risulta alterata da qualche algoritmo di compressione che ha introdotto rumore tra i pixel che va ad alterare il valore della funzione di fluttuazione diversamente da come ci si aspetterebbe da un immagine naturale.
Sono presenti di seguito i risultati dei vari esperimenti:

Image	size image	block size (pixel)	size secret (bit)	number of bits extracted correctly	Avg time to encryption (and hiding)	Avg time to decryption (and extraction)
Lena512.bmp	512x512	32x32	256	256/256	5.6846s	2.24425s
baboon900.bmp	900x900	45x45	400	400/400	9.7229s	5.60120s
baboon_color.bmp	400x500	50x50	80	79/80	11.3471s	2.91887s
barbara.bmp	576x720	36x36	320	320/320	15.9956s	7.08211s
bone_scint.pgm	1800x1100	50x50	792	792/792	35.0979s	23.3922s

Immagine criptata con dentro i dati nascosti

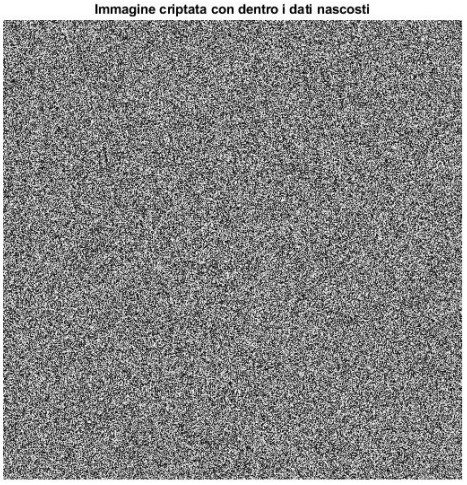


Immagine Decifrata

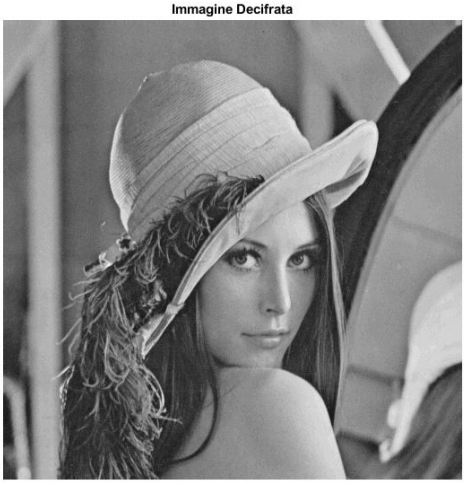


Immagine criptata con dentro i dati nascosti

Immagine Decifrata

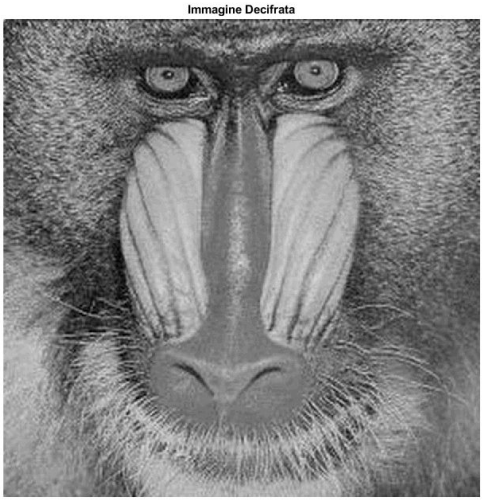
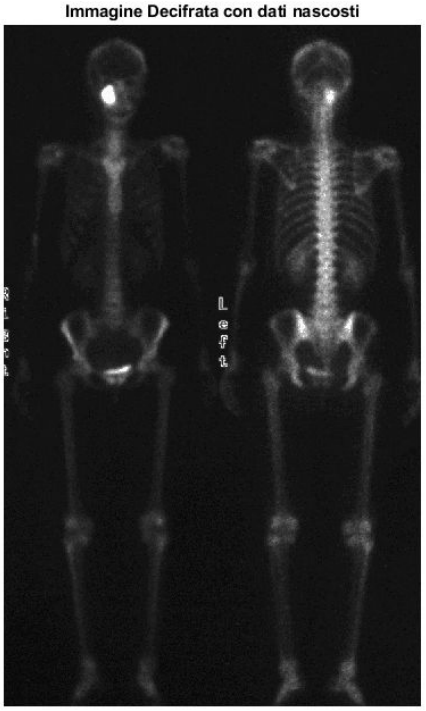


Immagine Decifrata con dati nascosti	Immagine Decifrata con dati nascosti ESTRATTI
	

5. Conclusioni e sviluppi futuri

Una alternativa a limitare inferiormente la size del blocco a 32x32 è implementare un meccanismo di correzione degli errori, con il draw-back di avere meno bit a disposizione per inserire i dati.

Uno dei possibili sviluppi futuri potrebbe essere l'implementazione di tecniche alternative di cifratura per garantire maggior sicurezza a tutto il protocollo. Un ulteriore sviluppo futuro potrebbe essere rendere tutto l'algoritmo parallelo (dare una parte di blocchi ad ogni core) per migliorare prestazioni e performance.

6. Riferimenti

[1] Paper: Zhang, Xinpeng. (2011). Reversible Data Hiding in Encrypted Image. Signal Processing Letters, IEEE. 18. 255 - 258. 10.1109/LSP.2011.2114651.

[2] PRGA e KSA: https://it.mathworks.com/matlabcentral/fileexchange/67242-rc4

[3] Trivium: https://github.com/ncarusso/Trivium