Università degli Studi di Salerno

Compressione Dati

Secure JPEG Masking

Professore

Bruno Carpentieri

Studenti

Giuseppe Altobelli
Antonio Citro
Saveriomichele Galiano
Andrea Vitale

Anno Accademico 2019/2020

Indice

1.0 Introduzione

Il numero di immagini condivise dai più disparati dispositivi ha raggiunto proporzioni che erano inimmaginabili già solo 10 anni fa: ogni giorno, infatti, più di 2 miliardi di immagini vengono postate sui social network o scambiate tramite messaggistica istantanea e servizi di condivisione basati su cloud.

I progressi nella condivisione delle immagini hanno sollevato, tuttavia, serie preoccupazioni riguardanti la privacy, poiché le foto possono potenzialmente rivelare un gran numero di informazioni sensibili riguardo le persone ritratte. I social network, dal canto loro, offrono solitamente un grado limitato di protezione della privacy e la soluzione più comune è il semplice accesso condizionato.

I ricercatori nel campo dell'elaborazione digitale delle immagini e della sicurezza dei media hanno proposto vari approcci per garantire la privacy, molti dei quali si concentrano sul criptare o permutare l'intero contenuto dell'immagine. Infatti, dal punto di vista della sicurezza dei dati, un approccio basato sulla crittografia può proteggere la privacy in un modo altamente sicuro e reversibile ma, d'altro canto, criptare semplicemente un'intera immagine può andare ad intaccare significativamente l'usabilità della condivisione di immagini.

In molti casi, infatti, gli utenti cercano soluzioni semplici ed intuitive per condividere le proprie foto online mentre si proteggono parzialmente regioni specifiche di un'immagine applicando, ad esempio, mascheratura, sfocatura o scrambling di aree sensibili.

2.0 Privacy nelle immagini

Al giorno d'oggi l'upload delle foto in rete, specialmente sui social network, cresce in modo esponenziale.
Tuttavia, ci sono numerosi rischi che gli utenti generalmente trascurano:

- la mancanza di **privacy**;
- la possibilità di far vedere le foto solo a chi vogliono;
- l'essere identificati tramite tali elementi e commenti ad essi relativi.

È bene ricordare che, ai sensi del regolamento *UE 679/2016*, è definito **dato personale** qualsiasi informazione riguardante una persona fisica che può essere identificata per esempio dai dati relativi all'ubicazione, un identificativo online, etc.
Mentre per **trattamento** è definito l'uso e la comunicazione mediante trasmissione, diffusione o qualsiasi altra forma di messa a disposizione.
Il trattamento dei dati deve essere *lecito*, *corretto* e *trasparente*. In particolare è considerato lecito solo se l'interessato ha espresso il consenso.

Lo scopo del nostro studio è proprio quello di garantire la protezione della privacy all'interno delle immagini, non solo della foto in sé ma anche dei suoi metadati.

Durante la fase iniziale, la protezione della privacy mira principalmente a consentire la privacy visiva nell'immagine.
I metodi più comuni per fare ciò sono:

- pixelation;
- mascheramento;
- sfocatura;
- scrambling;
- warping;
- morphing.

Con il rapido sviluppo dei social network e dei servizi di condivisione foto, sono nate nuove sfide per la protezione della privacy di foto online.

Una prima soluzione è stata quella di progettare dei meccanismi di controllo dell'accesso, in modo tale che le foto, o un qualche dato condiviso online, possa essere accessibile solo a un gruppo selezionato di persone. Purtroppo, la maggior parte di questi sistemi di controllo non sono adeguati, il che potrebbe portare a gravi perdite di informazioni quando la piattaforma che si sta utilizzando non è in grado di rispettare correttamente le apposite **policy** oppure quando gli utenti non riescono a comprendere le complesse impostazioni per la protezione della privacy.

Un’altra soluzione per proteggere la privacy nelle immagini online consiste nel criptare o nel permutare i dati delle immagini, prima di caricarli e condividerli sui Social Network. La crittografia o la permutazione può essere eseguita in diversi domini, come ad esempio nei pixel dell'immagine, nei metadata o nei coefficienti di trasformazione del coseno discreti (DCT).

3.0 Lavoro correlato

Per la realizzazione di questo progetto è stato preso in considerazione il lavoro di un gruppo di nostri colleghi che ha proposto l'implementazione di **Secure JPEG Scrambling**.

Secure JPEG Scrambling è un progetto sviluppato per l'esame di *Compressione Dati* nell'anno accademico 2015-2016 da *Rosario Di Florio, Raffaele Sabato, Steven Rosario Sirchia, Vincenzo Venosi.*

Lo scopo del loro lavoro è la protezione della privacy delle immagini. Partendo da JPEG Baseline come metodo di compressione delle immagini, hanno utilizzato la tecnica dello **scrambling** per effettuare la trasmissione e la ricezione dei dati in modo digitale e criptato (includendo, naturalmente, anche l'operazione di **descrambling** per effettuare l'operazione inversa).

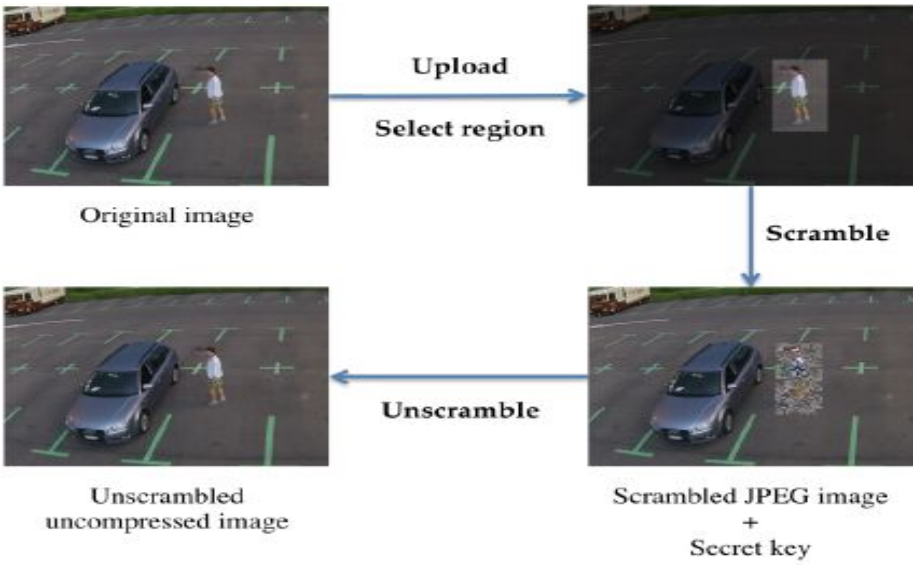

Figura 1. Processo di scrambling e unscrambling dell'immagine.

Nello specifico, sono state implementate due tecniche di *scrambling* in vari livelli di *aggressività* con la possibilità di selezionare *region of interest* (**ROI**) multiple.
Nel *primo caso* veniva semplicemente applicata un'operazione di scrambling alle ROI selezionate, mentre, nel *secondo caso*, si proponeva uno scrambling multi-regione attraverso l'utilizzo di chiavi segrete multiple: ad ogni regione *scramblata* veniva assegnato un identificativo e il descrambler poteva riportare nel formato originale la regione selezionata solamente attraverso la chiave corrispondente.

Per l'implementazione di tale progetto hanno utilizzato **Java** e la libreria **OpenCV** per il riconoscimento dei volti.

Per la fase di testing sono stati analizzati i seguenti parametri:

- **dimensioni dell'immagine** prima e dopo lo scrambling;
- **tempo** impiegato per lo scrambling;
- **numero di volti riconosciuti** nell'immagine originale e in quella modificata.

Ciò che hanno ottenuto è che il secondo approccio di *scrambling* riesce a far decadere il numero di volti rilevati da *OpenCV* nei dataset già dal livello 2 con un livello di compressione delle immagini molto vicino all'originale.

Hanno constatato, inoltre, che un occhio umano, al contrario di *OpenCV*, non farebbe alcuno sforzo nel riconoscere la presenza di un volto su un'immagine scramblata, pur avendo parecchie difficoltà nel riconoscere l'identità delle persone ritratte.
Dato che l'obiettivo principale dello *scrambling* non è quello di nascondere totalmente la presenza di una persona da una foto, bensì di fare in modo che il suo volto non sia riconoscibile, hanno concluso che, nonostante il numero di volti rilevati da *OpenCV* possa risultare in alcuni casi alto, il numero di persone effettivamente riconoscibile è molto più basso.

4.0 Argomenti trattati

4.1 JPEG

JPEG è uno degli standard più conosciuti per la compressione di immagini a tono continuo. Lo scopo di JPEG è la creazione di uno standard ottimale per comprimere immagini di qualsiasi qualità che sia implementabile su qualsiasi architettura.

Tale standard definisce due metodi di compressione: **lossless** e **lossy**.
La compressione lossless non prevede perdita di informazione, mentre la compressione lossy, al contrario, prevede perdita di informazioni.

L'algoritmo base per il JPEG, che è di tipo lossy, è chiamato *algoritmo baseline*.

JPEG Baseline è anche conosciuto come *sequential encoding* perché ogni componente dell'immagine è codificata singolarmente dall'alto verso il basso e da sinistra a destra.
Tale algoritmo, in fase di compressione, effettua un taglio di tutte quelle frequenze non visibili ad occhio nudo. causando un calo di qualità dell'immagine. Il decompressore, preso in input un'immagine compressa tramite JPEG, non è in grado di ricostruire tutte le frequenze andate perdute, ma cerca di essere il più accurato possibile.

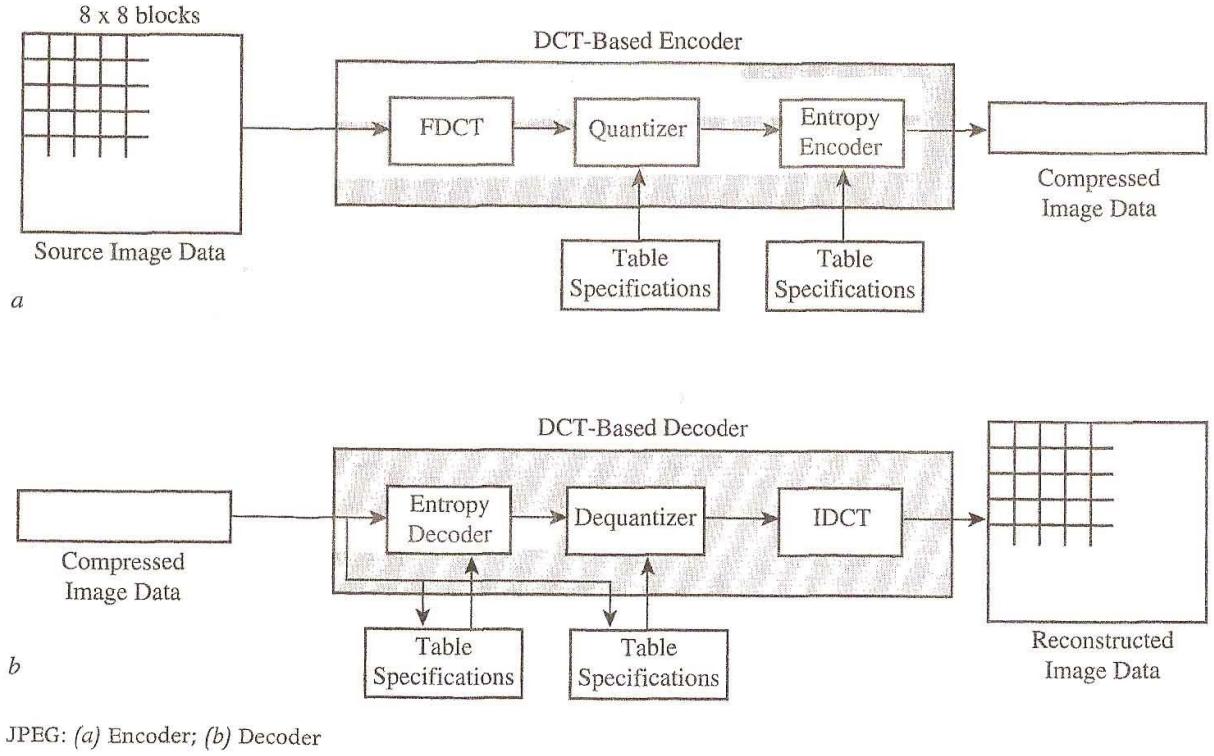

Figura 2. Algoritmo di encoding e decoding JPEG baseline.

L'*algoritmo di codifica* è strutturato in quattro fasi: **partizionamento**, **applicazione della DCT**, **quantizzazione** e **codifica entropica**.

Nella *prima fase* l'immagine viene divisa in blocchi di 8x8 pixels i quali contengono informazioni sulla luminosità codificate ciascuna in 8 bit.

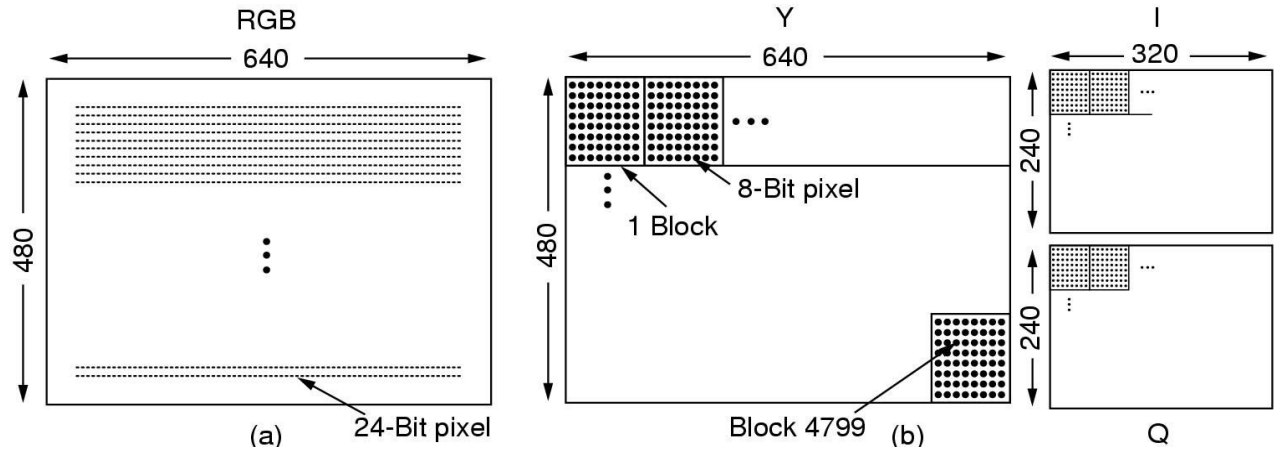

Figura 3. Partizionamento dell'immagine nella prima fase dell'algoritmo di codifica JPEG.

Nella *seconda fase* viene calcolata la trasformata discreta del coseno (DCT) per ogni pixel ottenendo, così, 64 coefficienti DCT per blocco disposti in matrici 8x8.

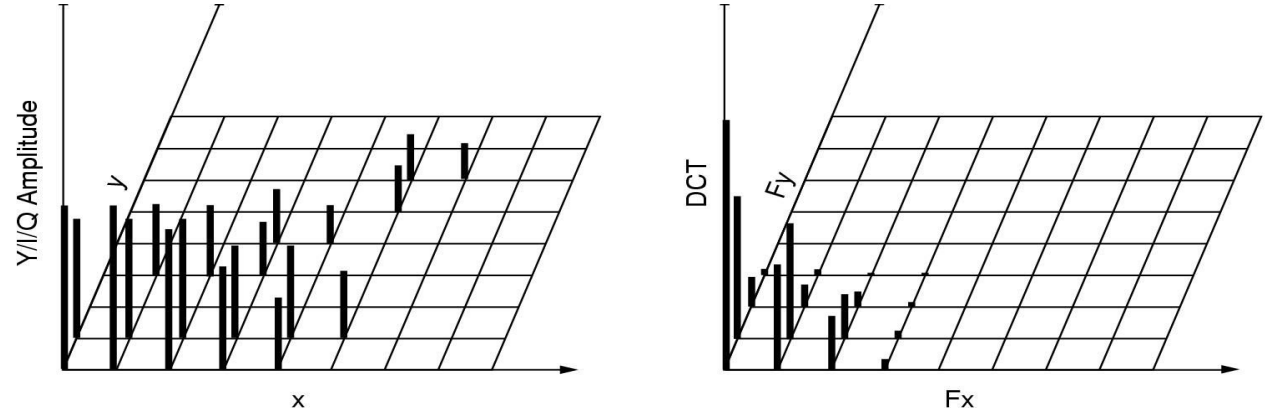

Figura 4. Applicazione della DCT nella seconda fase dell'algoritmo di codifica JPEG.

Nella *terza fase* ogni coefficiente DCT viene diviso per la sua tabella di quantizzazione, scartando tutte le informazioni meno significative a livello visuale e ottenendo una matrice che tende ad avere i valori più alti nell'angolo superiore a sinistra, mentre nelle parti restanti tende a 0.

DCT Coefficients

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantized coefficients

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantization table

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

Figura 5. Quantizzazione dei coefficienti DCT nella terza fase dell'algoritmo di codifica JPEG.

Nella *quarta fase* viene applicata la codifica entropica per la linearizzazione della matrice.
La linearizzazione della matrice può essere fatta o tramite **zig-zag scanning** oppure tramite **codifica di Huffman** o **aritmetica**.

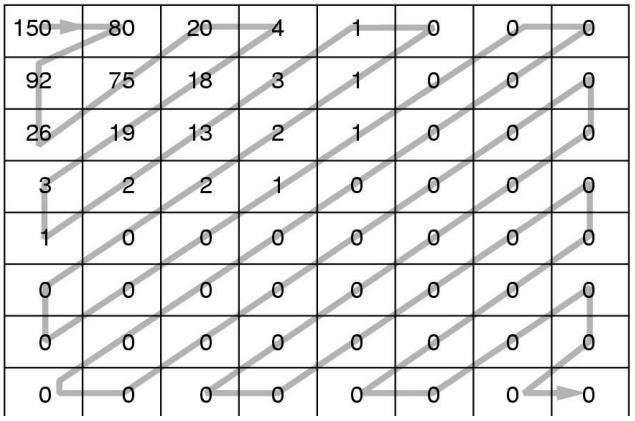

Figura 6. Zig-zag scanning nella quarta fase dell'algoritmo di codifica JPEG.

4.1.1 Metadata JPEG

I **metadata** sono informazioni aggiuntive, contenute all'interno di immagini digitali, che racchiudono dettagli tecnici ad esse relative.

Per la realizzazione del nostro lavoro sono stati sfruttati due segmenti di metadata contenuti in un'immagine JPEG: **EXIF** e **IPTC**.

Gli *Exchangeable Image File Format* (**EXIF**) sono strutture di metadati contenute all'interno di specifici formati di file JPEG, TIFF e RIFF.
Exif è stato creato dalla *JEIDA* e aggiornato nel corso degli anni, anche se è stato quasi del tutto abbandonato come standard poiché ha numerosi problemi relativi all'utilizzo della struttura legacy di partenza. Nonostante tali problemi, i produttori di fotocamere lo

utilizzano come standard di riferimento per la memorizzazione di informazioni approfondite su alcune proprietà delle foto (data e ora, info sulla fotocamera, impostazioni di scatto, miniature delle foto, info sul copyright).

L'*Information Interchange Model* (**IPTC**) è un struttura di file e un insieme di attributi di metadati che può essere applicato in file di immagini.
Tale standard è ampiamente utilizzato da fotografi e notiziari, poichè essi contengono informazioni quali il nome del fotografo, il copyright ed altre informazioni simili.
I metadati *IPTC* possono essere incorporati nei formati file di immagine JPEG, TIFF e altri formati come GIF o JPEG2000.

4.2 Algoritmi di cifratura

Per la realizzazione del nostro lavoro sono stati sfruttati due algoritmi di cifratura: **DES** e **AES**.

4.2.1 DES

Il *Data Encryption Standard* (**DES**) è un algoritmo di cifratura sviluppato negli anni 70 da IBM.
Tale standard si basa su una *Rete di Feistel* a 16 round il cui singolo blocco è lungo 64 bit mentre la chiave 56 bit.

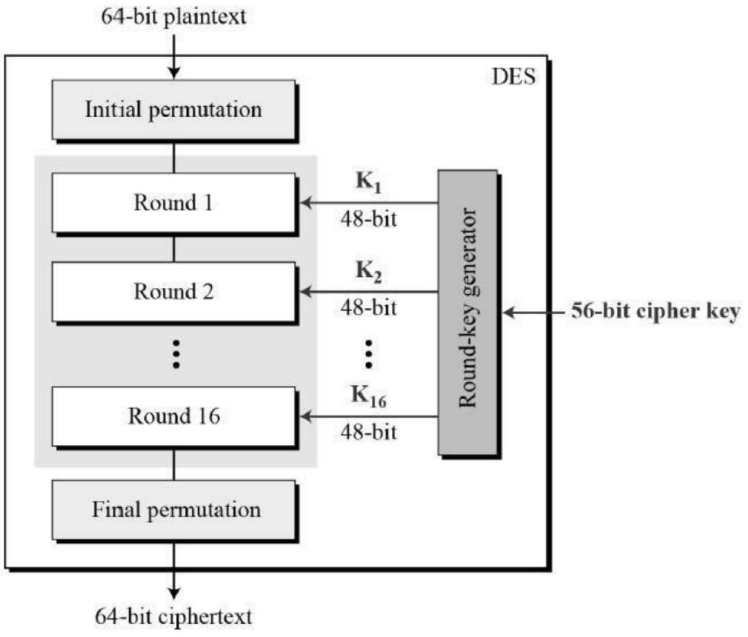

Figura 7. Algoritmo di cifratura del DES.

Ogni round prende in input una stringa di 32 bit ed un sottochiave di 48 bit.
La sottochiave viene generata da un *algoritmo di scheduling* delle chiavi che prende in input la chiave di 56 bit.

La funzione di round del DES è costruita usando il paradigma della *Reti di Sostituzione e Permutazione* (**SPN**) ed è descritta nel modo seguente:

$$\hat{f}_i(K_i, R), \qquad K_i \in \{0,1\}^{48}, \qquad R \in \{0,1\}^{32}$$

$\boldsymbol{f_i}$ definisce la i-esima funzione del round;
$\boldsymbol{K_i}$ definisce la i-esima sottochiave del round;
$\boldsymbol{R}$ definisce l'input del round.

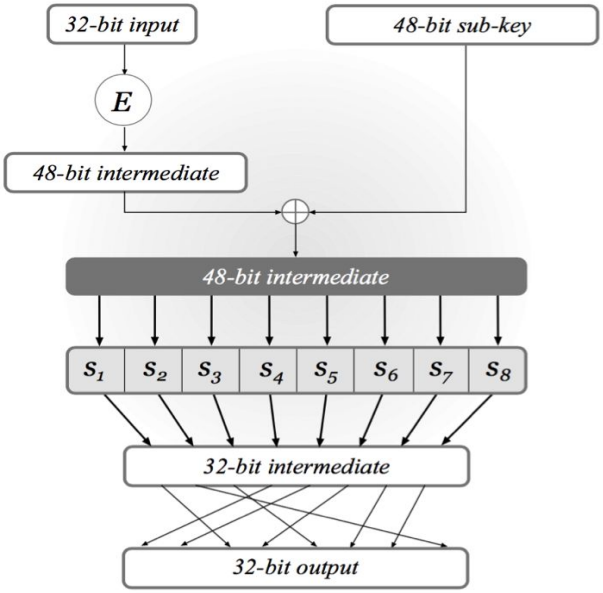

Figura 8. Funzione di round del DES.

La funzione di round è strutturata in due fasi.
Nella *prima fase* R viene esteso attraverso una *funzione di espansione* e arriva a 48 bit.
Nella *seconda fase* E(R) va in *XOR* con K_i ed il risultato viene mandato in input alle ***S-Box***.

Le *Substitution Box* (**S-Box**) sono degli elementi fondamentali per la sicurezza del DES ed hanno le seguenti caratteristiche:

- ciascuna S-Box è una funzione 4-a-1;
- non sono invertibili;
- ciascuna riga della tabella contiene tutte le possibili stringhe di 4 bit esattamente una volta;
- cambiando un bit di input ad una S-box, se ne cambiano almeno due di output.

	Column Number															
Row No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Figura 9. Esempio di una S-Box del DES.

La struttura a round del DES fa si che esibisca un forte **effetto valanga**, piccoli cambiamenti nel testo in chiaro o nella chiave producono, infatti, molti bit di differenza nel testo cifrato.

La debolezza del DES non risiede nella sua struttura ma nella lunghezza dei suoi parametri. La chiave di 56 bit, infatti, rende il DES vulnerabile agli **attacchi di ricerca esaustiva**.

4.2.2 AES

L'*Advanced Encryption Standard* (**AES**) è un algoritmo di cifratura adottato ufficialmente dal NIST nel 2001.
Tale standard ha un singolo blocco lungo 128 bit e può usare chiavi di 128, 192 e 256 bit.

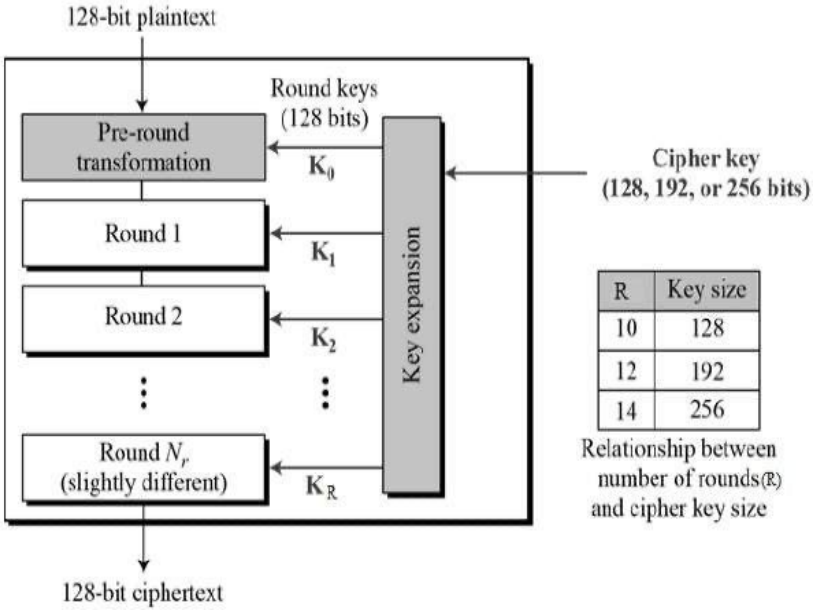

Figura 9. Algoritmo di cifratura di AES.

La lunghezza della chiave influisce sulla schedulazione delle chiavi e sul numero di round, ma non influisce sulla struttura di alto livello di ciascun round.

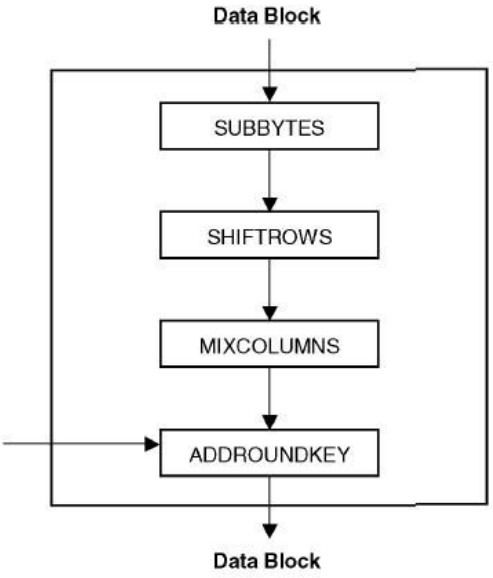

Figura 10. Funzione di round di AES.

La funzione di round è strutturata in quattro fasi: **aggiunta della chiave di round**, **sostituzione di bytes, shift delle righe** e **moltiplicazione matriciale.**

Nella *prima fase* (**AddRoundKey**) una sottochiave di round di 128 bit viene aggiunta tramite XOR allo *stato* (input i-esimo del round).

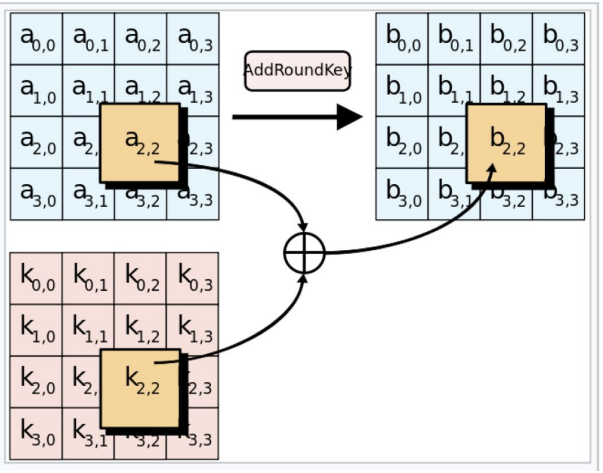

Figura 11. Aggiunta della chiave di round nella prima fase della funzione di round di AES.

Nella *seconda fase* (**SubBytes**) ciascun byte dello stato viene sostituito con un altro byte, in accordo ad una **tabella di lookup**.

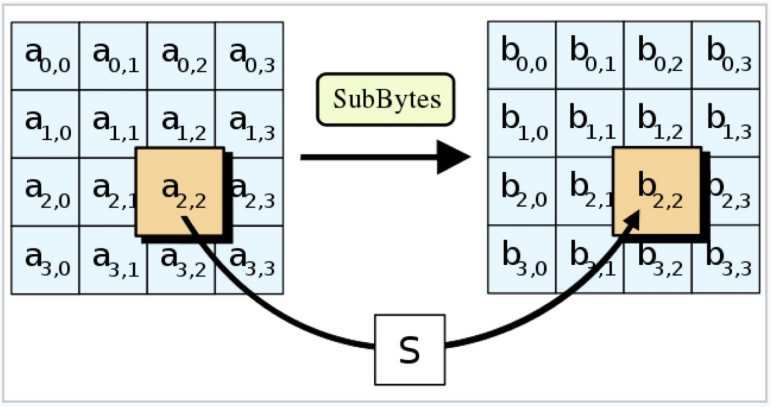

Figura 12. Sostituzione di byte nella seconda fase della funzione di round di AES.

Nella *terza fase* (**ShiftRows**) i byte di ogni i-esima riga vengono shiftati a sinistra di i posizioni.

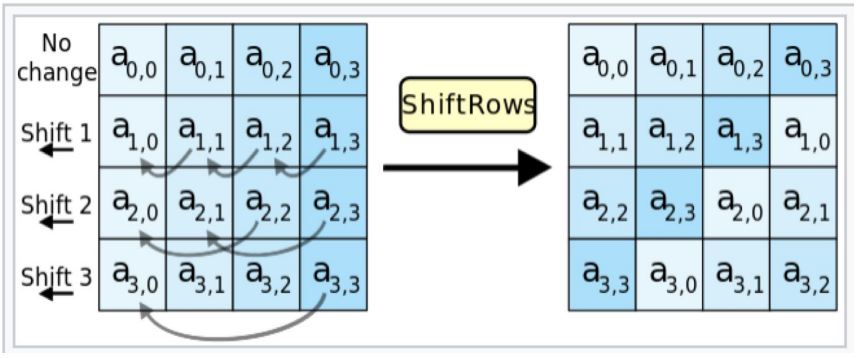

Figura 13. Shift delle righe nella terza fase della funzione di round di AES.

Nella *quarta fase* (**MixColumns**) viene applicata una trasformazione invertibile (moltiplicazione matriciale) ai quattro bytes di ogni colonna.

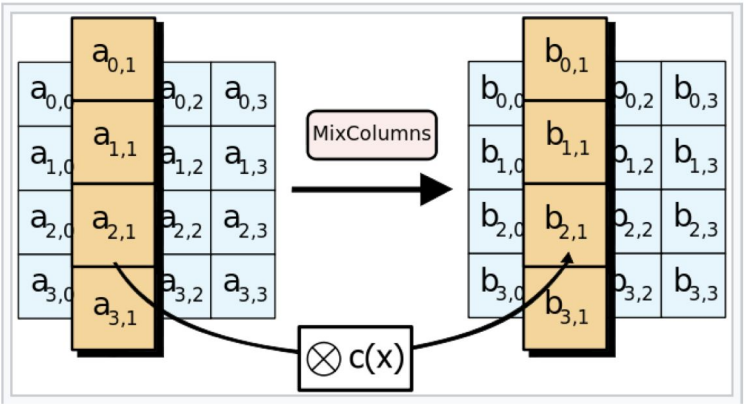

Figura 14. Moltiplicazione matriciale nella quarta fase della funzione di round di AES.

La struttura, la lunghezza della chiave e l'assenza di attacchi di crittoanalisi rendono AES, ad oggi, un **cifrario sicuro**.

5.0 Metodo proposto

Lo scopo del nostro studio è il **masking** e l'**unmasking** delle biometrie sensibili, ovvero **volto** e **occhi**, contenute all'interno di un'immagine.

Il principio di funzionamento alla base di *Secure JPEG Masking* è quello di utilizzare i metadata di JPEG, per preservare segretamente le informazioni riguardanti le biometrie specifiche dell'immagine originale.

Nella *fase di masking* viene scelta la biometria da mascherare, lo schema e la modalità di cifratura. Una volta individuata la biometria, la *Region of Interest* (**ROI**) viene sostituita con la maschera e le informazioni sensibili vengono salvate in maniera criptata nei *metadata* della nuova immagine ottenuta tramite compressione *JPEG*.

Nella *fase di unmasking* le informazioni sensibili vengono estrapolate dai metadata dell'immagine mascherata ed utilizzate per ricostruire l'immagine di partenza, sostituendo la maschera con le *ROI*.

Gli algoritmi di masking e unmasking verranno descritti in maniera dettagliata nei paragrafi seguenti.

5.1 Algoritmo di masking

L'algoritmo di masking è schematizzato nella figura sottostante.

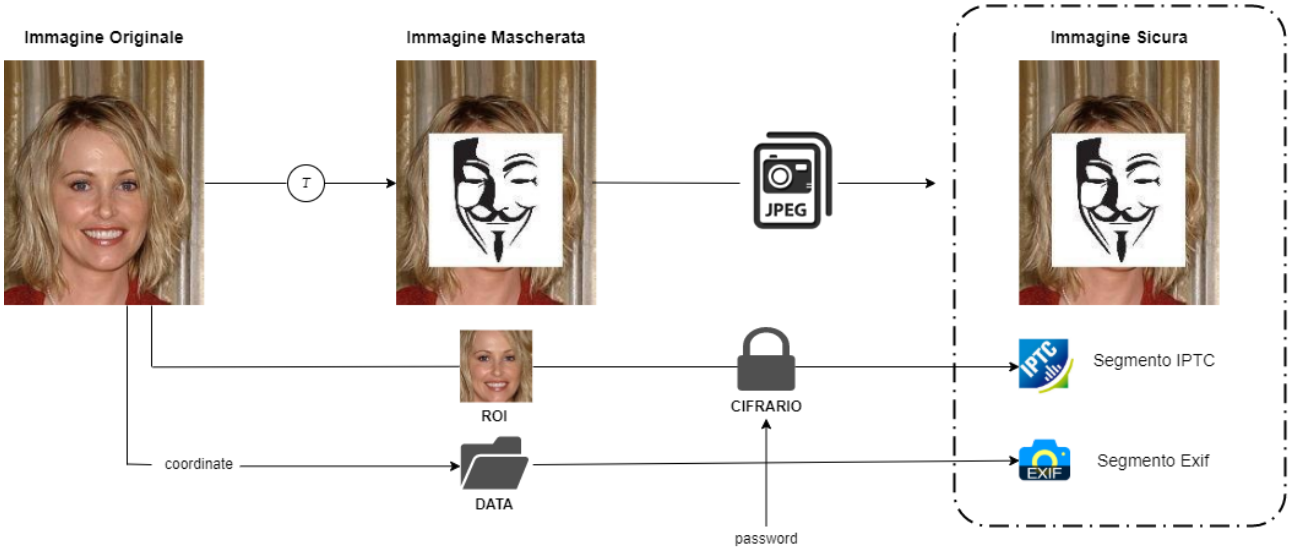

Figura 15. Algoritmo di masking di Secure JPEG Masking.

L'*algoritmo di masking* è strutturato in cinque fasi: **selezione dei parametri**, **riconoscimento delle ROI**, **applicazione della maschera**, **compressione** e **aggiunta dei metadata**.

Nella *prima fase* vengono selezionati i seguenti parametri: **immagine da mascherare**, **biometria** da rilevare, **cifrario e modalità di cifratura** per crittografare i dati sensibili e **password** per la crittografia.
In particolare, le biometrie che possono essere rilevate sono due: **volto** e **occhio**.
Si può scegliere, inoltre, tra due cifrari (**AES** e **DES**) e quattro modalità di cifratura: **CBC**, **CFB**, **ECB** e **OFB**.

Nella *seconda fase*, tramite l'utilizzo di specifici **classificatori**, viene eseguito l'algoritmo di **Viola-Jones** che permette di rilevare la biometria scelta in precedenza all'interno dell'immagine.

Nella *terza fase* viene applicata la maschera ad ogni ROI rilevata in precedenza.

Nella *quarta fase* l'immagine appena mascherata viene compressa e salvata nel formato JPEG con una **qualità di compressione** pari a *0.85*.

Nella *quinta fase* le informazioni per effettuare l'unmasking vengono salvate all'interno dei metadata dell'immagine JPEG.
In particolare, i bytes delle ROI vengono salvate in maniera criptata all'interno del segmento **Application2.ObjectName** di *IPTC*, mentre le coordinate di ogni ROI vengono salvate all'interno del segmento **Photo.UserComment** di *EXIF*.
La scelta di salvare le coordinate in chiaro e non in maniera criptata è ponderata dal fatto che non vengono considerate contenuto sensibile e, di conseguenza, per evitare di appesantire inutilmente il peso della foto.

5.2 Algoritmo di unmasking

L'algoritmo di unmasking è schematizzato nella figura sottostante.

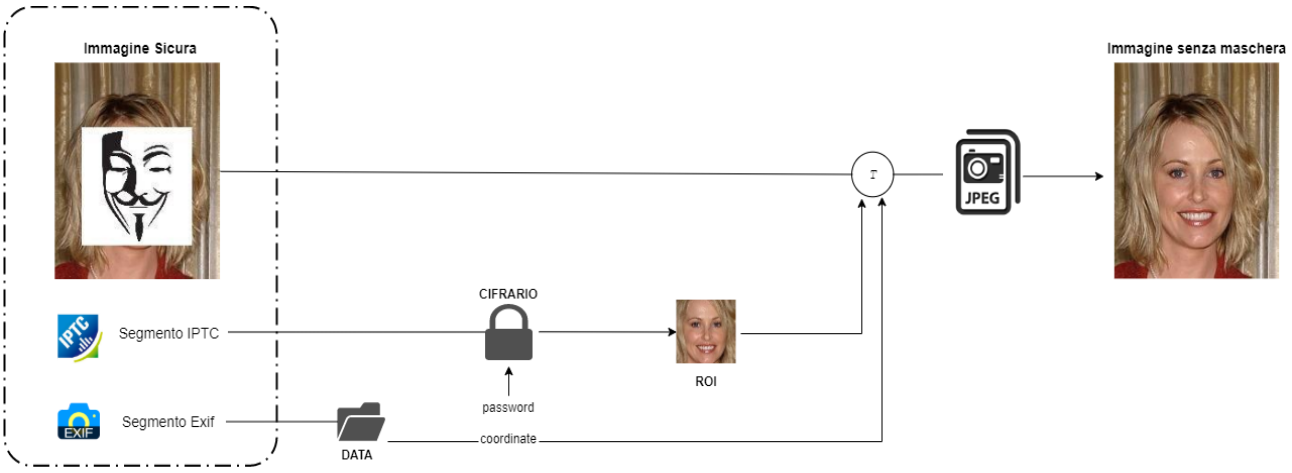

Figura 16. Algoritmo di unmasking di Secure JPEG Masking.

L'*algoritmo di unmasking* è strutturato in quattro fasi: **selezione dei parametri**, **estrazione dei metadata**, **rimozione della maschera** e **compressione**.

Nella *prima fase* vengono selezionati i seguenti parametri: **immagine da smascherare**, **cifrario e modalità di decifratura** per decrittografare i dati sensibili e **password** per la decrittografia.

Nella *seconda fase* vengono estratte le informazioni necessarie per l'unmasking dell'immagine dai metadata.
In particolare, i bytes delle ROI vengono estratte e decifrate dal segmento **Application2.ObjectName** di *IPTC*, mentre le coordinate di ogni ROI vengono estratte dal segmento **Photo.UserComment** di *EXIF*.

Nella *terza fase* viene ripristinato il contenuto dell'immagine originale andando a sostituire le maschere nelle coordinate estratte con le ROI corrispondenti.

Nella *quarta fase* l'immagine appena smascherata viene compressa e salvata nel formato JPEG con una **qualità di compressione** pari a *1*.
Ciò per minimizzare la perdita di informazione in seguito all'*unmasking*.

6.0 Implementazione

6.1 Hardware e software

Per la realizzazione di questo lavoro sono stati utilizzati il linguaggio **Java** (1.8.0) ed il linguaggio **Python** (3.7.4).
In particolare, in Python sono state sfruttate le seguenti librerie: **PIL** (6.2), **pycrypto** (2.6.1) e **pyexiv2** (1.3).

Per l'applicazione dell'algoritmo di Viola-Jones è stata, inoltre, utilizzata la libreria **OpenCV** (4.2.0).

La macchina su cui è stato svolto il lavoro è stata un **Dell XPS 9570** Intel Core i5 2.30 GHz con sistema operativo **Windows 10**.

6.2 Interfaccia grafica

La **GUI** del software sviluppato si presenta nel seguente modo.

Figura 17. GUI del software sviluppato.

La sezione **Compression** permette l'esecuzione dell'*algoritmo di masking,* la sezione di **Decompression** permette l'esecuzione dell'*algoritmo di unmasking* ed, infine, la sezione di **Test** permette il *testing* completo di tutte le immagini.

6.3 Masking

Il masking dell'immagine viene implementato sia in *Java* che in *Python*.

In Java la classe di riferimento è **Compression.java**.
In Python gli script sviluppati sono i seguenti: **compression-cbc.py**, **compression-cfb.py**, **compression-ebc.py** e **compression-ofb.py**.
Ciascuno script è stato implementato due volte, uno per il cifrario *AES* e l'altro per il *DES*.

La classe **Compression.java** implementa la funzionalità relativa al riconoscimento della biometria ed effettua la *compressione JPEG*.
In particolare:

Il caricamento della libreria **OpenCV,** utilizzata per il riconoscimento delle biometrie, avviene nel seguente snippet di codice.

```
static {
	System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
}
```

Il metodo **compress** effettua il *riconoscimento* delle biometrie e la compressione *JPEG*.
Tale metodo viene lanciato nel seguente snippet di codice.

```
compress(img, type, mask, cipher, mode, password);
```

I parametri che prende in input il metodo *compress* sono: *l'immagine* da mascherare*, il tipo di classificatore* da utilizzare*, la maschera* di sostituzione*, il cifrario, la modalità e la password* per la cifratura dei dati sensibili.

L'applicazione del classificatore che permette il riconoscimento delle ROI all'interno dell'immagine avviene nel seguente snippet di codice.

```
CascadeClassifier classifier = new CascadeClassifier(type);
MatOfRect detections = new MatOfRect();
classifier.detectMultiScale(matrixImgIn, detections);
```

L'applicazione della maschera avviene nel seguente snippet di codice.

```
Rect roi = new Rect(rect.x, rect.y, rect.width, rect.height);
Mat matrixImgROI = matrixImgInCopy.submat(roi);

Mat matrixMask = Imgcodecs.imread(mask);
Mat matrixMaskResized = new Mat();
Imgproc.resize(matrixMask, matrixMaskResized, new Size(rect.width,
rect.height));
Mat matrixImgSecure = matrixImgIn.submat(new Rect(rect.x, rect.y,
matrixMaskResized.cols(), matrixMaskResized.rows()));
matrixMaskResized.copyTo(matrixImgSecure);
```

La compressione e la scelta dei parametri di JPEG avviene nel seguente snippet di codice.

```
ArrayList < Integer > list = new ArrayList();
list.add(Imgcodecs.IMWRITE_JPEG_QUALITY);
list.add(85);
MatOfInt params = new MatOfInt();
params.fromList(list);

Imgcodecs.imwrite(PATH_DATA + "/imgs/out/secure/secure.jpg", matrixImgIn,
params);
```

In particolare all'interno dell'oggetto **list** viene definita la qualità di compressione di JPEG per scelta impostata a *0.85*.

L'esecuzione dello script in python è effettuato tramite il metodo **execPythonScript**.
Tale metodo viene lanciato nel seguente snippet di codice.

```
execPythonScript(script, PATH_DATA + "/imgs/out/secure/secure.jpg", coords,
password);
```

I parametri che prende in input il metodo *execPythonScript* sono: *lo script* da lanciare in base al cifrario e la modalità di cifratura scelta*, il path dell'immagine* in cui aggiungere i metadata*, le coordinate* delle ROI e *la password* per effettuare la cifratura.

Una volta, dunque, lanciato il metodo *execPythonScript* verrà eseguito lo script in python.
Per la stesura di questo documento verrà discussa l'implementazione dello script **compression-cbc.py** del cifrario *AES,* poiché gli altri script mantengono la struttura invariata avendo delle differenze solo nel metodo di **encrypt.**
In particolare:

La manipolazione dei metadata avviene tramite la libreria **pyexiv2**.
La lettura e l'assegnamento dei metadata dell'immagine ad una variabile avviene nel seguente snippet di codice.

```
metadata = pyexiv2.Image(file_name)
```

Il parametro *file_name* indica il *path dell'immagine* in cui occorre manipolare i metadata.

L'aggiunta dei metadata all'immagine sicura è effettuata nel seguente snippet di codice.

```
metadata.modify_iptc({"Iptc.Application2.ObjectName":
img_encrypted_stringed})

metadata.modify_exif({"Exif.Photo.UserComment": new_info})
```

Per la grandezza del segmento, per ospitare le informazioni criptate (*i byte del contenuto delle ROI*) è stato scelto il segmento **Application2.ObjectName** di *IPTC.* Per scelta arbitraria, per il salvataggio delle informazioni in chiaro (*le coordinate delle ROI)* è stato selezionato il segmento **Photo.UserComment** di *EXIF.*

All'interno dei metadata deve essere salvata una *stringa*. Nel segmento *IPTC* viene salvata una stringa composta dalla concatenazione di tutte le stringhe cifrate contenti i byte delle ROI, mentre nel segmento *EXIF* viene salvata una stringa composta dalla concatenazione di tutte le stringhe contenenti le coordinate delle ROI.

Le coordinate di ogni ROI vengono salvate in chiaro nel seguente formato.

```
new_info = str(len_img_decoded) + "," + str(id) + "," + str(x) + "," +
str(y)
```

Per il salvataggio dei byte di ogni ROI si effettuano, invece, le seguenti operazioni: **codifica** dei byte della ROI in stringa, **cifratura** e **ricodifica** della stringa ottenuta.
Tale processo è descritto nel seguente snippet di codice.

```
img_stringed = base64.b64encode(roi.read())
img_decoded = img_stringed.decode('utf-8')

len_img_decoded = len(img_decoded)
img_decoded += ((16 - len(img_decoded) % 16) * "0")

img_encrypted = encrypt(img_decoded, password)
img_encrypted_stringed = str(base64.b64encode(img_encrypted), 'utf-8')
```

Come specificato in precedenza, il metodo **encrypt** e l'allungamento dei byte della variabile *img_decoded* dipendono dal cifrario e dalla modalità di cifratura scelta.

L'esecuzione dello script in python, una volta aggiunti i metadata, termina con un numero che verrà poi catturato dalla classe java e decodificato tramite il metodo **decodeReturn**.
I numeri che possono essere catturati sono stati da noi scelti in modo tale da poter verificare al meglio se l'esecuzione dello script ha avuto *successo* o se, e dove, è andata in *fallimento*.

6.4 Unmasking

L'unmasking dell'immagine viene implementato sia in *Java* che in *Python*.

In Java la classe di riferimento è **Decompression.java**.
In Python gli script sviluppati sono seguenti: **decompression-cbc.py**, **decompression-cfb.py**, **decompression-ebc.py** e **decompression-ofb.py**.
Ciascuno script è stato implementato due volte, uno per il cifrario *AES* e l'altro per il *DES*.

La classe **Decompression.java** si occupa principalmente di raccogliere i parametri per la decompressione e lanciare lo script in python.
In particolare:

Il metodo **decompress** viene lanciato nel seguente snippet di codice.

```
decompress(cipher, mode, password);
```

I parametri che prende in input il metodo *decompress* sono: *il cifrario, la modalità e la password* per la decifratura dei dati sensibili.

L'esecuzione dello script in python è effettuato tramite il metodo **execPythonScript**.
Tale metodo viene lanciato nel seguente snippet di codice.

```
execPythonScript(script, PATH_DATA + "/imgs/out/secure/secure.jpg", cipher,
mode, password);
```

I parametri che prende in input il metodo *execPythonScript* sono: *lo script* da lanciare in base al cifrario e la modalità di cifratura scelta*, il path dell'immagine* su cui effettuare l'unmasking*, il cifrario, la modalità* e *la password* per effettuare la decifratura.

Una volta, dunque, lanciato il metodo *execPythonScript* verrà eseguito lo script in python.
Per la stesura di questo documento verrà discussa l'implementazione dello script **decompression-cbc.py** del cifrario *AES,* poiché gli altri script mantengono la struttura invariata avendo delle differenze solo nel metodo di **decrypt.**
Tale script si occuperà di effettuare *l'unmasking* dell'immagine.

In particolare:

La manipolazione dei metadata avviene tramite la libreria **pyexiv2**.
La lettura e l'assegnamento dei metadata dell'immagine ad una variabile avviene nel seguente snippet di codice.

```
metadata = pyexiv2.Image(file_name)
```

Il parametro *file_name* indica il *path dell'immagine* in cui occorre manipolare i metadata.

Il recupero dei metadata dell'immagine sicura è effettuata nel seguente snippet di codice.

```
iptc = metadata.read_iptc()
exif = metadata.read_exif()

rois = iptc["Iptc.Application2.ObjectName"].split("separator")
info = exif["Exif.Photo.UserComment"].split("-")
```

Lo split viene effettuato poiché, come spiegato nel paragrafo precedente, i metadata contengono stringhe contenenti la concatenazione delle informazioni di ogni ROI individuata nell'immagine.

Il recupero delle coordinate di ogni ROI, essendo salvate in chiaro, viene effettuato senza problemi.

Per il recupero dei byte di ogni ROI, invece, devono essere effettuate le seguenti operazioni: **decodifica** e **decifrarura** della stringa e **decodifica** della stringa in bytes.
Tale processo è descritto nel seguente snippet di codice.

```
img_decoded = base64.b64decode(rois[count])

img_decrypted_decoded = decrypt(img_decoded, password)[: int(lenght_roi)]

img_decrypted_decoded = base64.b64decode(img_decrypted_decoded)
```

Come specificato in precedenza, il metodo **decrypt** e l'eliminazione dell'allungamento dei byte della variabile *img_decoded* dipendono dal cifrario e dalla modalità di cifratura scelta.

L'eliminazione della maschera e il ripristino delle ROI originali alle esatte coordinate è effettuato nel seguente snippet di codice.

```
img_decompressed.paste(roi, (int(x), int(y)))
```

La compressione ed il salvataggio dell'immagine smascherata viene effettuata nel seguente snippet di codice.

```
img_decompressed.save(decrompressed_path + "/decompressed.jpg",
format='JPEG', quality=100)
```

Questa volta la qualità di compressione di JPEG è stata impostata ad 1, in modo tale da minimizzare la perdita dell'informazione e ricostruire nel modo giusto l'immagine smascherata.
L'esecuzione dello script in python, una volta effettuato l'unmasking, termina con un numero che verrà poi catturato dalla classe java e decodificato tramite il metodo **decodeReturn**.
I numeri che possono essere catturati sono stati da noi scelti in modo tale da poter verificare al meglio se l'esecuzione dello script ha avuto *successo* o se, e dove, è andata in *fallimento*.

7.0 Risultati sperimentali

Per testare in maniera completa e diretta tutte le immagini scelte per questo lavoro è stata prevista una funzione di **Test** avviabile nella schermata principale del software sviluppato.

Tale funzione crea il file **results.txt** al cui interno è possibile trovare tutte le informazioni relative al testing di ogni immagine per ogni biometria, cifrario e modalità di cifratura. All'interno di questo file è possibile trovare, anche, una serie di medie in modo tale da poter effettuare l'analisi da una prospettiva di più alto livello.

I risultati ottenuti sono schematizzati nei paragrafi successivi.

7.1 Dataset

Per la realizzazione di questo lavoro sono stati utilizzati due dataset differenti, il primo (**CelebA**) contenente solo immagini di persone singole, il secondo (**Group2A**), invece, contenente immagini con più persone.

Dal primo dataset sono state estrapolate 75 immagini, mentre, dal secondo ne sono state estrapolate 25.

La *fase di testing* è stata programmata, dunque, su un totale di *100 immagini,* in maniera tale da poter effettuare un'analisi più completa.

7.2 Analisi dei risultati

Per l'analisi dei risultati si è adottato un approccio di tipo *bottom-up.*

La *prima fase* dell'analisi aveva l'obiettivo di selezionare il cifrario e la modalità di cifratura migliore in modo tale da poterlo utilizzare, poi, per l'analisi più approfondita.
Per fare ciò, si è scelto di testare le immagini sicure ottenute con il masking della biometria del **viso**.

Il confronto delle modalità di AES è schematizzato nel grafico sottostante.

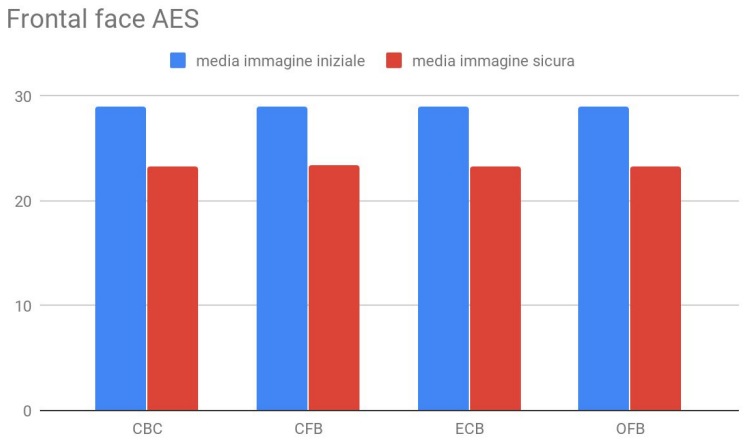

Figura 17. Grafico delle modalità di AES per la biometria viso.

Come si evince dal confronto le modalità, in termine di grandezza delle immagini sicure ottenute, si equivalgono. Verrà, dunque, scelta arbitrariamente la modalità **CBC.**

Il confronto delle modalità del DES è schematizzato nel grafico sottostante.

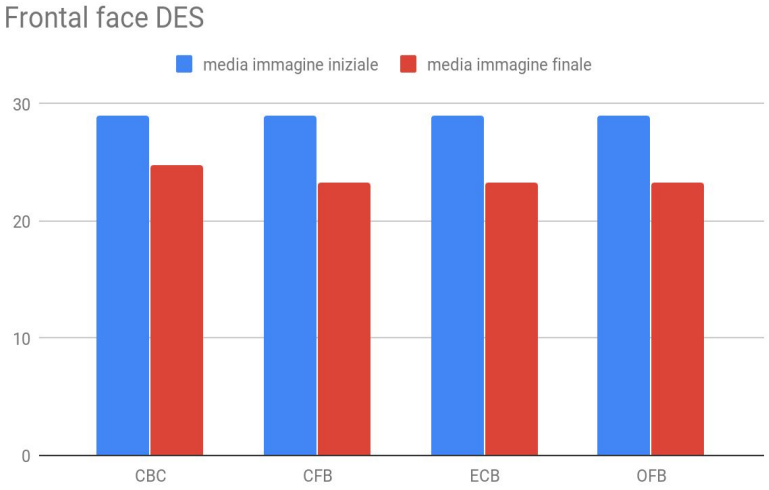

Figura 18. Grafico delle modalità del DES per la biometria viso.

Come si evince dal confronto le modalità migliori, in termine di grandezza delle immagini sicure ottenute, risultano essere, anche se di pochi kb, *CFB* ed *ECB*. Verrà, dunque, scelta arbitrariamente la modalità **ECB.**

Il confronto tra AES e DES con le loro modalità di cifratura migliori è schematizzato nel grafico sottostante.

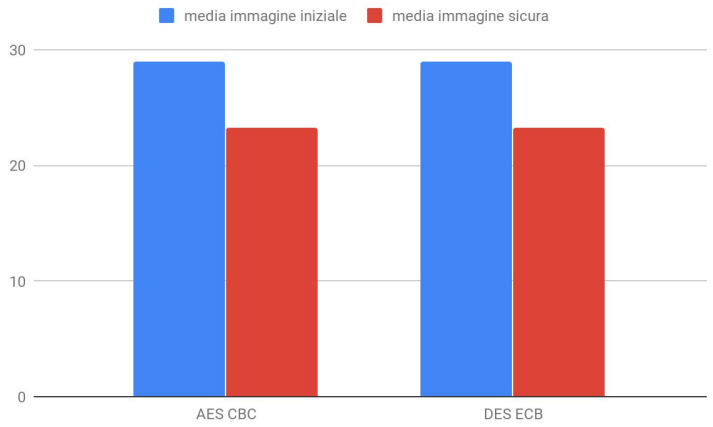

Figura 19. Grafico delle modalità migliori di AES e DES per la biometria viso.

Come si evince dal confronto tra i cifrari e le loro modalità migliori, in termine di grandezza delle immagini sicure ottenute, sono equivalenti. La scelta ricade su *AES* in quanto cifrario decisamente più sicuro del *DES*.

La *seconda fase* dell'analisi aveva l'obiettivo, una volta selezionato il cifrario migliore e la modalità di cifratura migliore, di confrontare ed analizzare i risultati delle immagini singole con le immagini di gruppo.

In particolare sono state testate le *75* immagini singole e le *25* immagini di gruppo per entrambe le biometrie presenti.

I risultati delle immagini singole per la biometria volto sono schematizzate nel grafico sottostante.

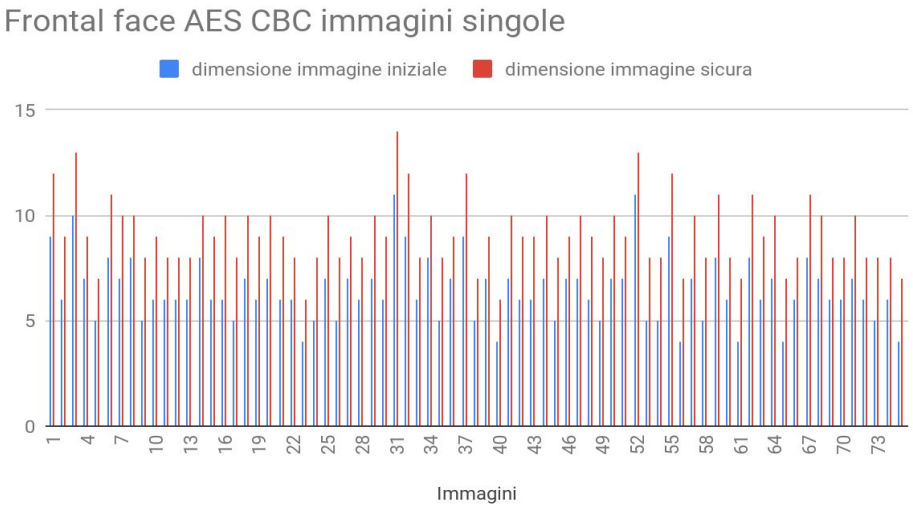

Figura 20. Grafico delle immagini singole con modalità AES CBC per la biometria volto.

I risultati delle immagini singole per la biometria occhi sono schematizzate nel grafico sottostante.

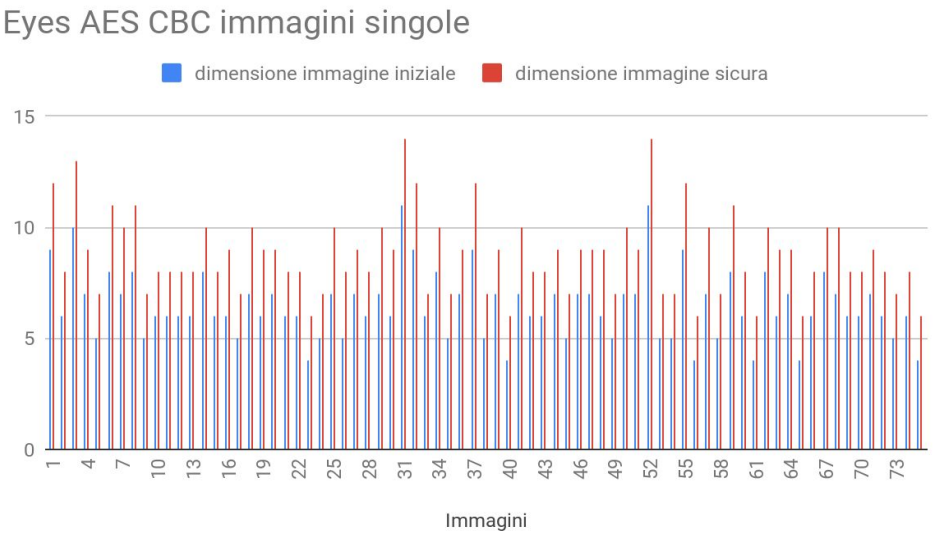

Figura 21. Grafico delle immagini singole con modalità AES CBC per la biometria occhi.

I risultati delle immagini di gruppo per la biometria volto sono schematizzate nel grafico sottostante.

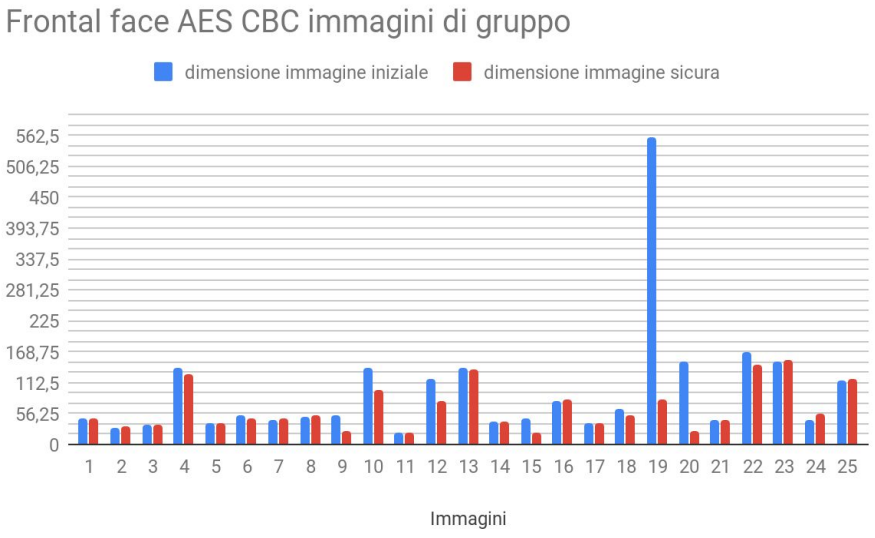

Figura 22. Grafico delle immagini di gruppo con modalità AES CBC per la biometria volto.

I risultati delle immagini di gruppo per la biometria occhi sono schematizzate nel grafico sottostante.

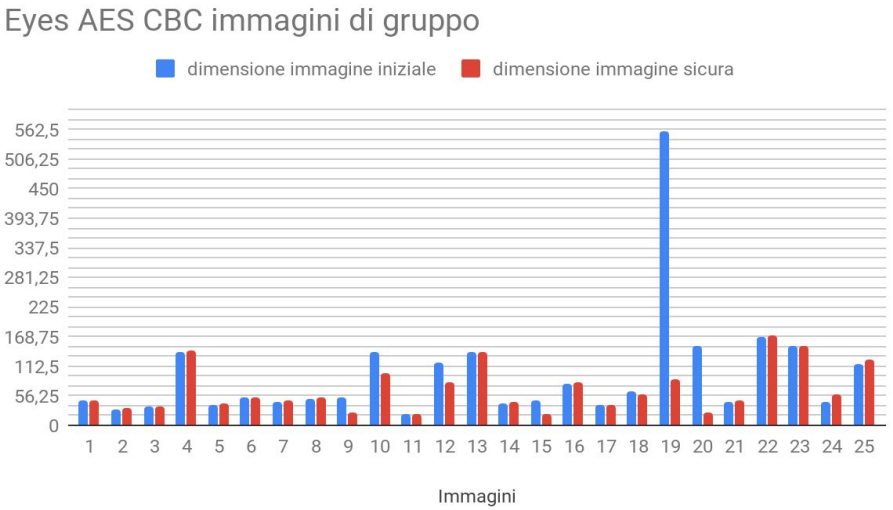

Figura 23. Grafico delle immagini di gruppo con modalità AES CBC per la biometria occhi.

Analizzando i risultati dati dai due tipi di immagini diverse prese in input si nota subito che l'algoritmo con immagini singole, avendo immagini più piccole e con meno particolari, tende ad aumentare, seppur di poco, la dimensione della sua immagine sicura, mentre con le immagini di gruppo ovvero immagini più grandi e con più particolari all'interno di essa la dimensione dell'immagine sicura talvolta tende a scendere.
Questo perché la compressione lossy determina un abbassamento della quantità di informazione salvata, che nel caso di immagini complesse risulta essere superiore.
In ogni caso, a livello visivo le immagini mascherate e smascherate non presentano differenze troppo marcate con l'immagine iniziale.

La *terza fase* dell'analisi aveva l'obiettivo di confrontare ed analizzare i valori medi dei risultati delle immagini singole con quelle di gruppo per entrambe le biometrie.

I risultati del confronto tra le immagini singole e le immagini di gruppo per la biometria volto sono schematizzate nel grafico sottostante.

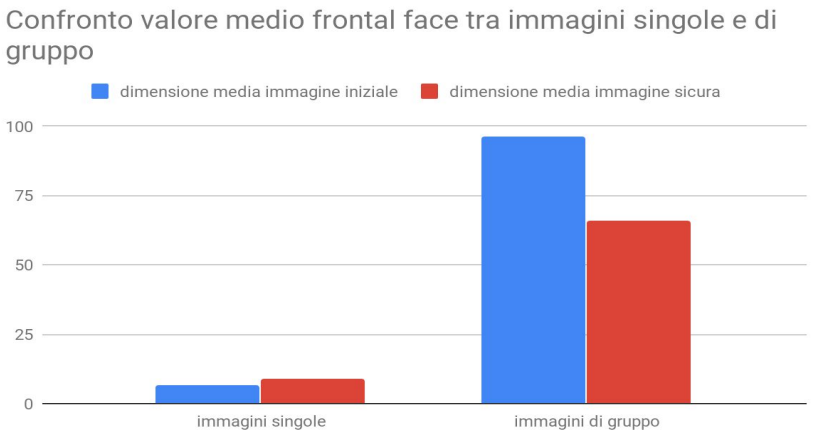

Figura 24. Grafico delle immagini singole e di gruppo per la biometria volto.

I risultati del confronto tra le immagini singole e le immagini di gruppo per la biometria occhi sono schematizzate nel grafico sottostante.

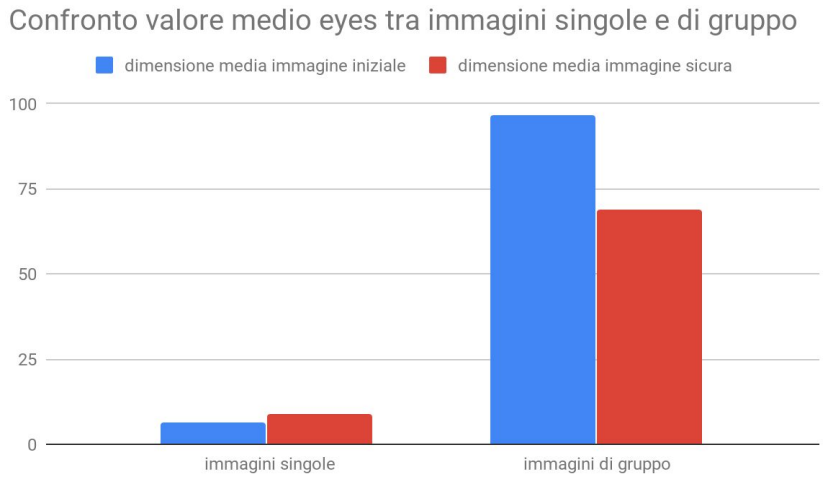

Figura 25. Grafico delle immagini singole e di gruppo per la biometria occhi.

Come da aspettative, le immagini di gruppo, essendo più complicate, hanno una dimensione maggiore rispetto alle immagini singole.
Occorre però considerare che il valore medio delle immagini iniziali delle immagini di gruppo è alterato dalla presenza dell'immagine *img_group_19* che è decisamente più grande rispetto alle altre.
Tale immagine, tuttavia, non incide sulla media delle immagini sicure poiché la compressione ne abbassa la dimensione e la porta alla media delle altre immagini.

La *quarta fase* dell'analisi aveva l'obiettivo, una volta selezionato il cifrario migliore e la modalità di cifratura migliore, di confrontare le medie dei tempi di *masking* e *unmasking* per entrambe le biometrie.

I risultati del confronto tra i tempi di masking e unmasking per entrambe le biometrie sono schematizzate nel grafico sottostante.

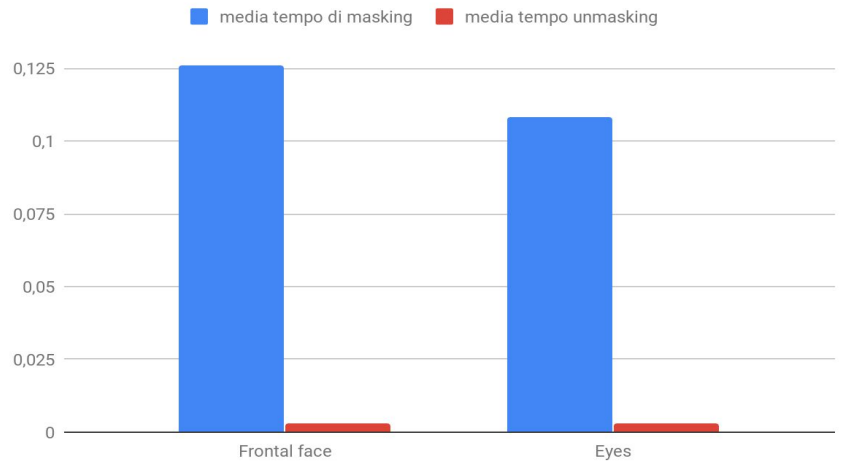

Figura 26. Grafico dei tempi di masking e unmasking per le biometrie volto e occhi.

Da questo grafico, si può notare che, effettivamente, i tempi per effettuare il *masking* sono superiori rispetto all'*unmasking* proprio a causa dell'operazione di riconoscimento delle biometrie.
Tale tempistica, tuttavia, risulta essere irrilevante poichè la differenza tra le due operazioni è nell'ordine dei centesimi di secondo.
Il *masking,* infatti, impiega in media *0,125 secondi* per completare l'operazione.

In generale, le operazioni di *masking* e *unmasking* risultano essere soddisfacenti dal punto di vista visivo e immediate dal punto di vista della velocità di completamento del task.

8.0 Conclusioni

In questo documento viene presentato *Secure JPEG Masking*, un metodo per la protezione delle immagini in modo sicuro, reversibile e personale.

Secure JPEG Masking permette, da un lato, di applicare una manipolazione visuale arbitraria su determinate ROI, mentre dall'altro, preserva segretamente le informazioni riguardanti le ROI originali nei metadata (*IPTC* e *EXIF*) dell'immagine JPEG offuscata.
L'immagine può essere smascherata andando a rimpiazzare le regioni offuscate dell'immagine protetta con le corrispondenti regioni originali estratte dai metadata.

In *Secure JPEG Masking*, le informazioni sensibili (i byte delle ROI) sono cifrate prima di essere inserite. Perciò, la sicurezza della soluzione proposta sta principalmente nello schema di cifratura applicato.
In ogni caso la scelta del cifrario e della modalità di cifratura è lasciata come scelta all'utente. Data, però, la differenza quasi nulla nelle dimensioni fra le immagini sicure generate con i due cifrari si consiglia l'utilizzo di *AES* per la sua maggiore sicurezza.

L'immagine sicura viene compressa tramite JPEG con una qualità di compressione pari a 0.85 (*compressione lossy)*, mentre, l'unmasking viene effettuato con una qualità di compressione pari ad 1. Ciò rende le qualità delle immagini mascherate e smascherate del tutto impercettibile ad occhio nudo.

L'aggiunta di ulteriori dati di masking nell'immagine JPEG causa un ovvio aumento della taglia del file. Si è cercato di mantenere al minimo tale aumento andando a cifrare solo le informazioni realmente sensibili.

Offuscare determinate porzioni dell'immagine potrebbe non offrire il miglior grado di protezione della privacy in certi scenari dato che le regioni non protette potrebbero ancora rivelare informazioni private; è abbastanza affidabile, invece, nei casi in cui le regioni non protette dell'immagine non rivelano informazioni esplicite sul contesto riguardanti l'obiettivo che potrebbero coincidere con altre informazioni pubbliche di tale obiettivo.

Usando *Secure JPEG Masking*, in conclusione, è possibile ottenere una protezione più intuitiva, personale e sempre reversibile.

Riferimenti

[1] Garante della Protezione dei Dati Personali - regolamento generale sulla protezione dei dati.

[2] Secure JPEG Scrambling - Rosario Di Florio, Raffaele Sabato, Steven Rosario Sirchia, Vincenzo Venosi.

[3] Dispense corso Compressione Dati - prof. Bruno Carpentieri.

[4] https://iptc.org

[5] https://it.wikipedia.org/wiki/Exchangeable_image_file_format

[6] Introduction to Modern Cryptography - Jonathan Katz, Yehuda Lindell - CRC Press, 2015.

[7] http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[8] http://chenlab.ece.cornell.edu/people/Andy/ImagesOfGroups.html