International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 07 | July 2020 www.irjet.net p-ISSN: 2395-0072

An Improved Reversible Data Hiding In Encrypted JPEG Bitstreams

Sruthi Suresh[1], Dr. Sreela Sreedhar[2], Dr.N. Vishwanath[3], Saira Varghese[4]

[1]PG Student, Dept. Computer Science Engineering, Toc H Institute of Science and Technology, Kerala
[2]Assoc. Professor & HOD, Dept. Computer Science Engineering, Toc H Institute of Science and Technology, Kerala
[3]Professor, Dept. Computer Science Engineering, Toc H Institute of Science and Technology, Kerala
[4]Asst. Professor, Dept. Computer Science Engineering, Toc H Institute of Science and Technology, Kerala

---

Abstract - *Reversible data hiding can be defined as an approach where the data can be made hidden in a host media such as image, audio and video files. It embeds the additional and secret message into a cover media such as images and performs reversible procedure to reconstruct the original message. Since there are great research achievements on Reversible Data Hiding in Encrypted Images (RDHEI), many of the jpeg encryption algorithms cannot keep format complaint to JPEG decoders and have low embedding capacity and security. The proposed RDHEI framework is designed for secure storage and transmission of Joint Photographic Experts Group (JPEG) images. The proposed system provides a JPEG encryption to encipher a JPEG image and keep the format applicable to JPEG decoders. The image owner can encrypt a JPEG bitstream and upload it to the cloud. The cloud server embeds the additional information into the encrypted bitstream to generate a marked encrypted bitstream. A combination of Hamming code and histogram shifting is used as embedding algorithm during data hiding procedure. The server extracts additional information from marked encrypted JPEG bitstream and recovers the original encrypted bitstream losslessly, when an authorized user requires a download. The user obtains the original JPEG bitstream by a direct decryption, after downloading. The proposed system can provide high embedding capacity and image privacy. Moreover, it reduces client/user workload where client/user requires to do no extra tasks except encryption/decryption.*

***Key Words*: Image privacy, RDHEI, JPEG, Hamming code, Histogram shifting.**

1. INTRODUCTION

Data hiding is referred to as a process to hide data, such as image or text, into cover media. The data hiding process consists of two sets of data, one is a set of the embedded data and another is a set of the cover media data. Data hiding schemes are categorized into two: irreversible and reversible data hiding schemes. Irreversible data hiding means once the cover image has embedded on the original image, the original image is lost i.e, from stego image original image cannot be recovered in extraction process. Reversible data hiding allows the user to embed the additional and secret message into cover media and to perform a reversible procedure that extracts the hidden secret message and perfectly reconstructs the original cover content. Many literature works proposed varieties of reversible data hiding techniques in which most of them relies on unencrypted images.

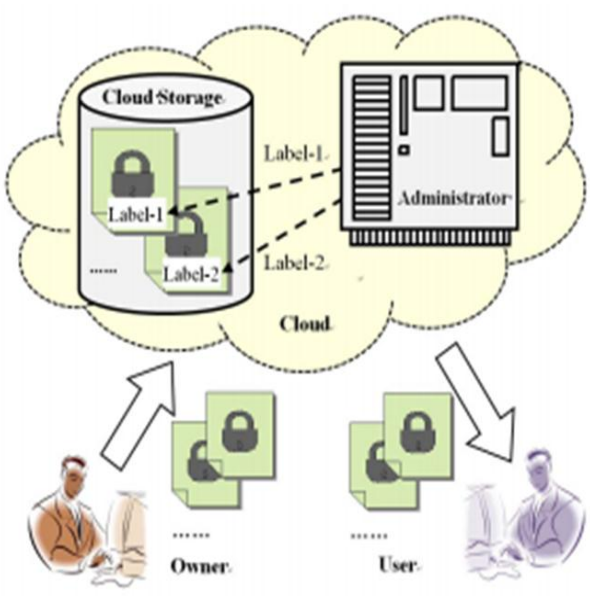

Fig -1: Labelling In Cloud

Reversible Data Hiding in Encrypted Images (RDHEI) was a new direction of RDH over Encrypted images. Areas where original image is as important as secret message, like military and medical fields, any changes on original image during the transmission can alter the intelligence of original image and affect the overall results. RDHEI scheme achieves owner privacy i.e, image privacy. A trusted party is authorized to insert some additional data such as the origin information, image notations or authentication data, within the encrypted image, where the original image content is unknown to this party. Indeed, the medical images are encrypted for preserving the patient privacy, and a database administrator only embeds a few data into the corresponding encrypted images. For the consistency of a medical image, it must guarantee that the original content can be perfectly reconstructed after decryption-then-extraction of the secret message by the receiver. That is, RDHEI method not only ensures the accuracy of the reconstructed cover-image and extracted secret message, but also preserves the privacy of the cover-image. Since JPEG

images are widely used over Internet, this paper focuses on RDHEI in JPEG bitstreams.

RDHEI is especially useful for labeling the ciphertext in cloud storage, as shown in Fig.1. When image owner wants to protect their privacy, RDHEI first provides a secure encryption algorithm for the owners to encrypt their images before uploading. On the cloud side, RDHEI allows the server to label an encrypted image through data hiding, e.g., hiding the identities, timestamps, and remarks into the ciphertext to generate a marked encrypted copy. Therefore, the labels are attached inside the ciphertext, providing a better management of files for administrators. On the other hand, when an authorized user downloads the encrypted image from the cloud, the original content can be losslessly recovered after image decryption. The server constructs a metadata file to record the information of the uploaded images in traditional systems of file management. The RDH-EI technique provides an alternative way, which accommodates additional information of the image inside the encrypted bitstream. Therefore, no metadata files are needed anymore for labeling the uploaded images.

RDHEI schemes are applicable in medical and military purposes. In healthcare social networks (HSN), the patients can outsource their encrypted health records such as scan reports including CT scan, MRI scan etc., to cloud storage and share them with doctors in a secure and efficient manner. RDH-EI allows the cloud server to label an encrypted image by data hiding, e.g., hiding the identities, timestamps, and remarks of patients into the ciphertext to generate a marked encrypted copy.

This paper presents a cloud application comprises of an image owner, cloud server and a user. The image owner can encrypt a jpeg image and upload it. The cloud server acts as data hider who hides the details of owner in the encrypted image. When an authorized user requires the image, he can request the cloud and the cloud will extract the hidden message and user can download it.

The rest of this paper is organized as follows: Section 2 presents various reversible data hiding methods and related works. Section 3 describes the proposed framework and Section 4 analyzes the proposed work based on security and performance.

2. RELATED WORKS

Many reversible data hiding methods on plaintext images have been reported in past years. In [1], the host image is divided into nonoverlapping blocks so that each block contains n pixels. By counting the frequency of the pixel-value-array sized n of each divided block, an n-dimensional histogram is generated. Finally, data embedding is done by modifying the resulting n- dimensional histogram.

Great researches were done on RDHEI schemes. It was first realized in uncompressed nature images [2]. First encrypt the original image using a stream cipher algorithm and then upload the encrypted image onto the cloud by the content owner. By flipping three least significant bits (LSB) of half pixels in each block the cloud server embeds additional bits into ciphertext. On the recipient end, the authorized user can decrypt the marked encrypted image and generate two candidates for each block by flipping LSBs again. Since the original block of a nature image is smoother than the interfered block, one hidden bit can be extracted and the original block can be recovered.

Although the methods in [3]–[7] have good embedding and recovery capabilities, data extraction must be done after image decryption. This limitation makes the technique less useful in cloud storage.

Another type of RDHEI is realized by preprocessing the original image. Some works require the image owner to vacate spare rooms in the plaintext image before image encryption. This additional operation of vacating spare rooms in plaintext is referred as preprocessing. After that, the image owner encrypts the processed image and uploads it onto the server. In methods [8] – [12], the image owner first reserves enough space on original image and with the encryption key owner encrypt the image. Now, the data hider only needs to accommodate data into the spare space previous emptied out. Then a receiver can extract the embedded data with the data hiding key and further recover the original image from the encrypted image according to the encryption key. In [13], a stream cipher algorithm is used to encrypt the original image and user upload ciphertext to a remote server. On server side, combination of cyclic-shifting and data-swapping is used to embed messages into the ciphered image. A receiver can extract the hidden messages and losslessly recover the original image. But data extraction must be done after image decryption thus less useful in cloud storage.

The above-mentioned methods are for uncompressed images thus not applicable in several cases where images transmitted over Internet are compressed e.g., the popular JPEG format. As a result, some RDHEI works were proposed for JPEG bitstreams.

In [14], a histogram shifting-based RDH scheme is used. In this method non-zero AC coefficients in DCT are represented by $N = (N_1, \ldots, N_m)$ Most of the peak points of the AC coefficient histograms are located at points 1 and -1. Zero coefficients remain unchanged and only coefficients with values 1 and –1 are expanded to carry message bits and thus have a low hiding capacity.

In [15], here for the JPEG bitstream encryption, a new JPEG bitstream is constructed by selecting a portion of blocks from the whole image. Bitstreams of rest of blocks are

encrypted and hidden in the JPEG header. With a compression algorithm, some bits of the encrypted JPEG bitstream are compressed to accommodate additional bits in server side. On the receiver side, it uses an iterative algorithm to recover the original JPEG bitstream. Even though the encryption is format compliant, recipient must do a recovery task after decryption.

In [16], the proposed system uses "reserving-room-before-encryption (RRBE)" framework. Before encrypting the JPEG bitstream, content owner first reserves space on original JPEG bitstream and encrypts modified JPEG bitstream. Server embeds secret data using data hiding key and receiver can extract the secret data by the data-hiding key and can recover the original JPEG bitstream by the encryption key. Here, the owner required to do pre-processing, i.e., generating spare rooms before encryption and also JPEG encryption is not format complaint.

In [17], it proposes an RDHEI method for jpeg images. Even if the jpeg encryption algorithm maintains the structure of the jpeg after encryption, the embedding procedure is little complex. Embedding procedure includes two stages: code mapping-based embedding and ordered embedding.

This paper proposes a new JPEG RDHEI framework to develop a client/user burden free application where client/user requires to do no extra works except encryption/decryption. Also, to incorporate a JPEG encryption algorithm [17] that is format complaint to JPEG decoders with data hiding technique [18]. Compared to previous techniques it can achieve a larger embedding payload during data hiding in encrypted jpeg image and to recover the original content losslessly after image decryption.

3. PROPOSED SYSTEM

The proposed RDH-EI framework consists of three parties, the image owner, the cloud server and the authorized user. The owner can encrypt a JPEG bitstream and uploads it to the cloud. The cloud server can embed additional information into the encrypted bitstream to generate a marked encrypted bitstream. When an authorized user requires a downloading operation, the server extracts hidden information from the marked encrypted bitstream and recovers the original encrypted bitstream. After decryption, the user obtains the original JPEG image. The proposed framework is shown in Fig. 2.

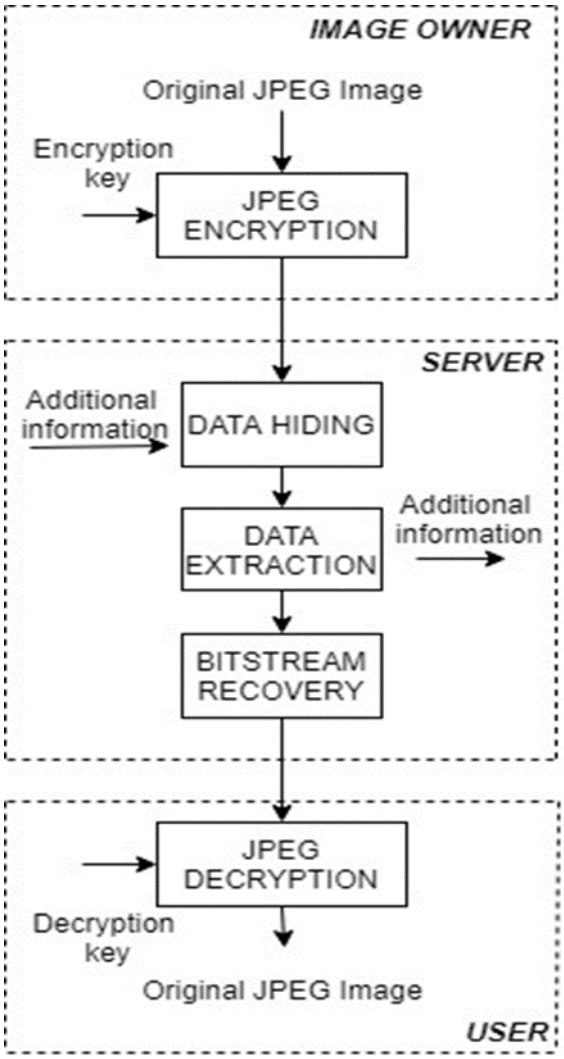

Fig -2: Proposed System

A JPEG bitstream consists of a marker of start-of-image, a JPEG header, the entropy encoded data, and an end-of-image marker as shown in Fig. 3

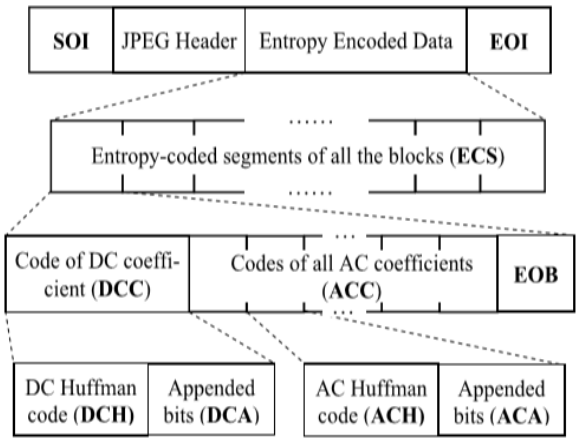

Fig -3: Syntax of JPEG Baseline

The section of entropy encoded data is constructed by entropy-coded segments of all blocks. Each ECS includes one code of DC coefficient (DCC), several codes of the AC coefficients (ACC), and an end-of-block marker. The DCC part consists of a DC Huffman code (DCH) and DC appended bits (DCA), and each ACC part consists of AC Huffman codes (ACH) and AC appended bits (ACA). Therefore, a JPEG bitstream J can be represented by

$$J = \{SOI, JH, ECS_1, ..., ECS_N, EOI\}$$

Table -1: Acronyms Used

Acronyms	Terms
SOI	*start-of- image*
EOI	*end-of-image*
EOB	*end-of-block*
JH	*JPEG header*
ECS	*entropy coded segments*
DCC	*code of a DC coefficient*
DCH	*Huffman code in a DCC*
DCA	*appended bits in a DCC*
ACC	*code of an AC coefficient*
ACH	*Huffman code in an ACC*
ACA	*appended bits in a ACC*

There are N entropy-coded segments when an H × W image is compressed into a JPEG bitstream, where N =[H/8] ·[W/8]. and denoted as ECS_i, i =1,2,...,N. According to the JPEG syntax, each segment can be represented by

$$ECS_i = \{DCC^{<i>}, ACC^{<i,1>}, ACC^{<i,2>}, ..., EOB\}$$
$$= \{[DCH^{<i>}, DCA^{<i>}], [ACH^{<i,1>}, ACA^{<i,1>}], [ACH^{<i,2>}, ACA^{<i,2>}], ..., EOB\}$$

The proposed framework includes a four-phase procedure for this application. First phase is the JPEG Encryption algorithm that encrypts the jpeg image from user. Second phase is Data Embedding that embeds the user details in jpeg image using Hamming code and Histogram shifting. Third phase is Data Extraction and Bitstream Recovery that extracts the original information from stego image. Fourth phase is JPEG Decryption that the user can decrypt the jpeg image and download it.

3.1 JPEG Encryption

On the image owner side, an owner can register the application using his username and password. And the owner can login using his valid credentials. After successfully logged in, the owner can browse for the jpeg image he wants to encrypt. After selecting the jpeg image, the owner can perform the jpeg encryption on the selected image. Finally, owner can upload the encrypted jpeg image to the server.

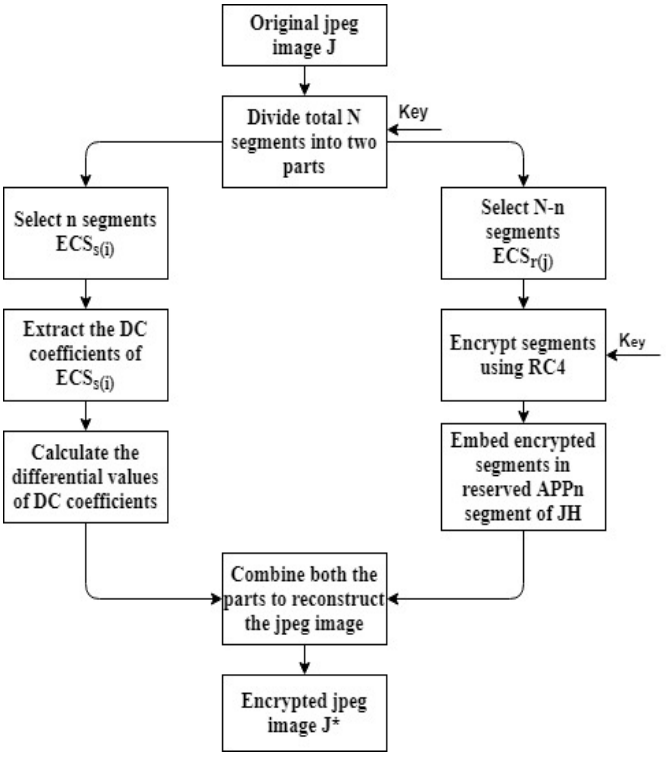

Fig -4: JPEG Encryption

The encryption procedure is divided into six steps:

(1) After selecting the jpeg image to be encrypted, using encryption key K the image owner pseudo-randomly selects n entropy encoded segments from the original jpeg bitstream, $\mathbf{ECS_{s(i)}}$, where i=1, 2..,n & 1<n<N. Let the remaining segments be $\mathbf{ECS_{r(j)}}$, where j= 1,2,..N-n.

(2) Owner encrypts the remaining N-n segments by stream cipher RC4 using the encryption key K.

(3) Using two positive integers h and w, the owner reconstructs the new jpeg image of size h x w.

(4) The encrypted bits, along with the values of H and W, are then embedded into the reserved application segments marked with APPn in JPEG header.

(5) Also, owner specifies the new image size as h × w in modified header.

(6) By decoding the DC Huffman code and the appended bits, the owner extracts the DC coefficient from n selected segments ECSs(i) i.e.,$[DCH^{<s(i)>}, DCA^{<s(i)>}]$, to $\{d_{s(1)}, d_{s(2)},...,d_{s(n)}\}$.

(7) The owner generates the differential values $\{d'_{s(1)}, d'_{s(2)},...,d'_{s(n)}\}$. of these coefficients where

$$d's(i) = \begin{cases} ds(i), & i = 1 \\ ds(i) - ds(i-1), & i = 2,..,n \end{cases} \quad (1)$$

(8) These are further encoded by Huffman codes to generate [DCH*, DCA*].

(9) Then owner replaces all $ECS_{s(i)}$ in the new entropy encoded data with $ECS^*_{s(i)}$, where $ECS^*_{s(i)}$ stands for the encrypted segments.

(10) Finally, the owner constructs the encrypted JPEG bitstream J* shown in Fig.5. as ,

$$J^* = \{SOI, JH^*, ECS^*_{s(1)},..., ECS^*_{s(n)}, EOI\}$$

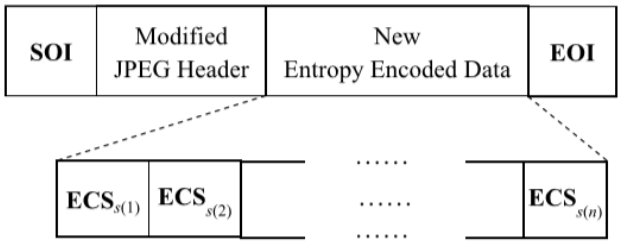

Fig -5: Structure of Encrypted JPEG Bitstream

3.2 Data Embedding

The embedding algorithm comprises of (7,4) Hamming code and Histogram shifting methods. (7, 4) Hamming code-based RDH scheme rearranges the seven cover bits from their normal form of R= (C_1, C_2 , D_1, C_3,D_2, D_3, D_4) to R= (D_1, D_2, D_3, D_4, C_1, C_2, C_3), where the first four bits consist of four data bits (D_1, D_2, D_3, D_4) and the last three bits consist of the parity check bits. For a given stream with seven bits, R= (R_1,R_2,R_3,R_4,R_5,R_6,R_7); if the calculated three-bit vector defined in eq.(2), Y(R) = (y_1,y_2,y_3) is equal to (0, 0, 0), then the bit stream R is classified as a perfect stream.

$$Y(R) = \begin{cases} y_1 = r_1 \oplus r_2 \oplus r_4 \\ y_2 = r_1 \oplus r_3 \oplus r_4 \\ y_3 = r_2 \oplus r_3 \oplus r_4 \end{cases} \quad (2)$$

Table -2: Relationship between Error Location and Y(R')

Error Location	Y(R')
error free	000
1	110
2	101
3	011
4	111
5	100
6	010
7	001

A new bit stream R' is generated by flipping only one bit of the stream R, will generate another three-bit vector Y(R'). The location of the flipped bit is referred to as the error location. The relationship between the error location and the corresponding vector of Y(R') is shown in Table 2.

In Histogram Shifting method, all non-zero AC coefficients in DCT are represented by N = (N_1, N_2, …, N_{m-1}, N_m), where m represents the number of all nonzero AC coefficients in the JPEG image. Most of the peak points of the AC coefficient histograms are considered to be located at points 1 and -1. The embedding algorithm is described as:

$$\begin{aligned} N_i' &= N_i + sign(N_i)*s \quad \text{if } |N_i|=1 \\ N_i' &= N_i + sign(N_i) \quad \text{if } |N_i|>1 \end{aligned} \quad (3)$$

$$\text{sign } N_i = \begin{cases} 1 & \text{if } Ni > 0 \\ 0 & \text{if } Ni = 0 \\ -1 & \text{if } Ni < 0 \end{cases} \quad (4)$$

In eq (3), s ∈ {0,1}, s indicates the additional information bit to be embedded and N_i' denotes the corresponding hidden AC coefficients in the marked JPEG image. The information extraction and image restoration algorithm can be described as follows:

$$S' = \begin{cases} 0 & \text{if } |Ni|=1 \\ 1 & \text{if } |Ni|=2 \end{cases} \quad (5)$$

$$N_i = \begin{cases} sign(N'_i) & \text{if } 1 \le |N'_i| \le 2 \\ N'_i - sign(N'_i) & \text{if } |N'_i| \ge 3 \end{cases} \quad (6)$$

where S' and N_i denotes the extracted secret bit and the restored AC coefficient, respectively.

The data embedding procedure is divided into six steps:

(1) Input encrypted jpeg image J*.

(2) Cover JPEG image J* is decoded to get the quantized DCT coefficients D.

(3) Check the AC coefficients of D for every seven bits as a group X_i in zig-zag order.

(4) If X_i is a perfect string, it is embedded with additional information using the (7, 4) Hamming code. Otherwise, the Histogram Shifting method is used to embed the information. For each seven-tuple, we divide X_i in two parts, which consists of the LSB of each element, and which is composed of the non-LSB parts of each element. We take R into eq. (2) to calculate the vector value of Y(R). If Y(R) is equal to (0, 0, 0), then R is a perfect stream and the (7, 4) Hamming code is used to hide the data for this seven-tuple X_i and continue to step (5). Otherwise, R is a non-perfect stream, HS is used to hide the data for this seven-tuple X_i and then we continue to step 6.

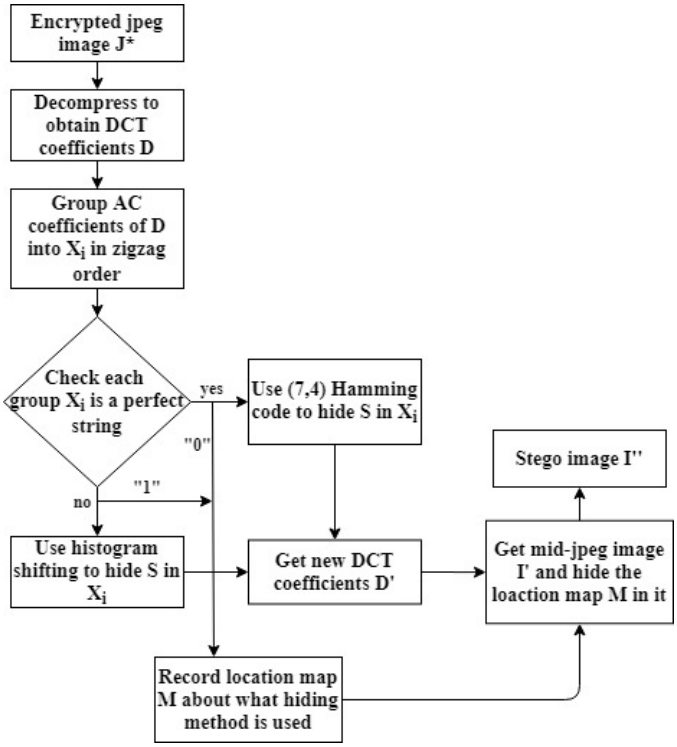

Fig -6: Data Embedding

(5) For seven-tuple $X_i = (x_1,x_2,x_3,x_4,x_5,x_6,x_7)$ whose R and P are defined as step 4, and information to be embedded be S, we first convert S to its corresponding decimal value Y (Y<8). If Y=0, then the stego perfect stream $R' = (r'_1,r'_2,r'_3,r'_4,r'_5,r'_6,r'_7)$ is equal to the cover seven-tuple R; otherwise, the stego perfect stream $R' = (r'_1,r'_2,r'_3,r'_4,r'_5,r'_6,r'_7)$ is generated by flipping Y^{th} bit of R. Finally, the vector of the stego seven-tuple $X'_i = (x'_1,x'_2,x'_3,x'_4,x'_5,x'_6,x'_7)$ is generated by re-combination of P and R' .

(6) For a vector of cover seven-tuple $X_i = (x_1,x_2,x_3,x_4,x_5,x_6,x_7)$ and binary secret bit streams S = (s1,s2,...,si), we first read x sequentially. If xi is a nonzero number, then we calculate x'_i using eq. (3) and (4). The binary secret bit is value corresponding to s is 1; otherwise, x remains unchanged, and x'i is equal to x, which is zero. Then, we get the stego seven-tuple to $X'_i = (x'_1,x'_2,x'_3,x'_4,x'_5,x'_6,x'_7)$ carry the secret message.

(7) Record the adopted embedding method of each group using a binary location map. "0" to represent (7, 4) Hamming code for hiding. "1" to represent HS method for hiding the additional information.

(8) Finally, obtain a new DCT coefficient carrying the additional information to get mid-JPEG image. Finally, the location map is embedded in the mid-JPEG image using HS method to generate the stego-image I".

3.3 Data Extraction and Bitstream Recovery

The data extraction procedure is divided into six steps:

(1) Get stego-JPEG image I''

(2) Extract location map M and recover mid-JPEG image I' .

(3) The quantized DCT coefficients D' is generated from mid-JPEG image I' that carries the embedded information.

(4) The AC coefficients of D' are grouped into X_i' in zig-zag order, and each group is seven tuple, just like the embedding phase.

(5) Using location map , judge which hiding method is used.

(6) If the corresponding location of X_i ' in location map M is "0" which means the (7, 4) Hamming code is used for X_i '. Otherwise, histogram shifting is used for X_i'.

(7) Corresponding to the hiding method used, extract embedded information S and obtain original DCT coefficients D.

(8) Through D, obtain the original JPEG image J.

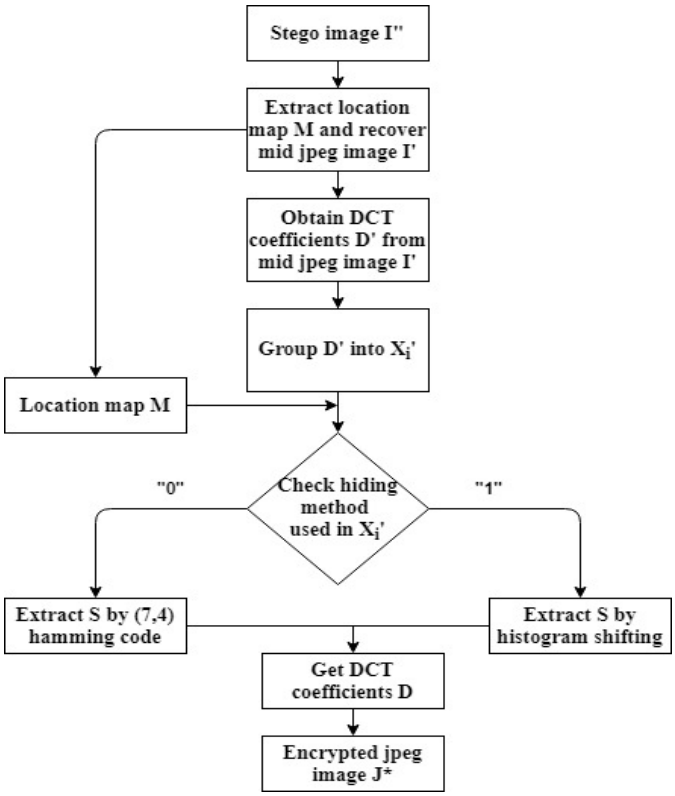

Fig -7: Data Extraction and Bitstream Recovery

3.4 JPEG Decryption

Decryption is the reverse process of encryption method of JPEG bitstream.

The decryption procedure is divided into six steps:

(1) User extracts N-n encrypted segments from the reserved application APPn in the JPEG header JH*.
(2) Using RC4 with decryption key K, user decrypt these segments back to $ECS_{r(j)}$, where j = 1,.., N – n.
(3) The n segments $ECS_{s(i)}$ are extracted from J*. The user reads [DCH* , DCA*] from all segments ECS∗ s(i) , and decodes them into the values of {$d'_{s(1)}$, $d'_{s(2)}$,..., $d'_{s(n)}$}. The original DC coefficients {$d_{s(1)}$, $d_{s(2)}$,..., $d_{s(n)}$} are reconstructed by

$$d_{s(i)} = \sum_{k=1}^{i} d'_{s(k)} \quad i = 1,2,..,n \qquad (7)$$

(4) The user reconstructs the original $ECS_{s(i)}$ after reencoding these DC coefficients into [DCH, DCA] by Huffman codes.
(5) After rearranging all segments of $ECS_{s(i)}$ and $ECS_{r(j)}$ the user reconstructs the original entropy encoded data.
(6) Finally, the original JPEG stream J is decrypted after modifying the JPEG header to specify the original image size H x W.

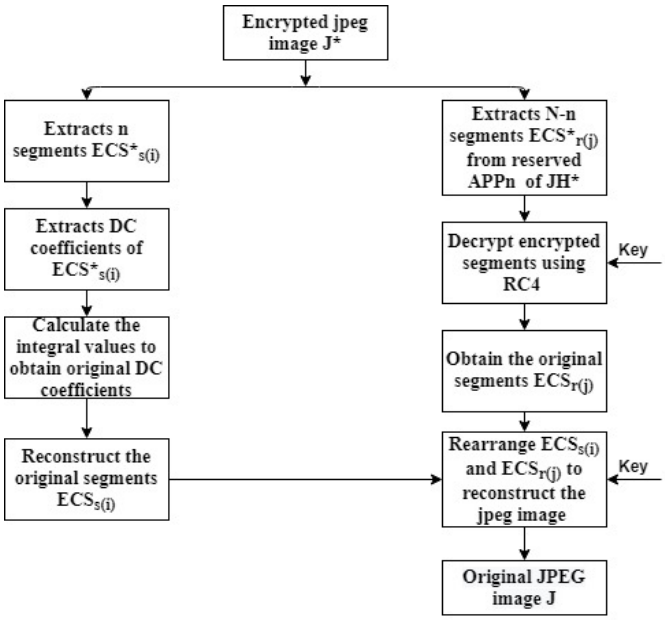

Fig -8: JPEG Decryption

4. RESULTS AND ANALYSIS

The histogram analysis of ciphertext image illustrates about the quality of the image encryption algorithm. If a uniformly-distributed histogram for the ciphertext image is generated, then the encryption method is considered to be a good image encryption. While performing the histogram analysis in the original JPEG image shown in Fig.9, we get a histogram as shown in Fig.10.

Fig -9: Original JPEG Image

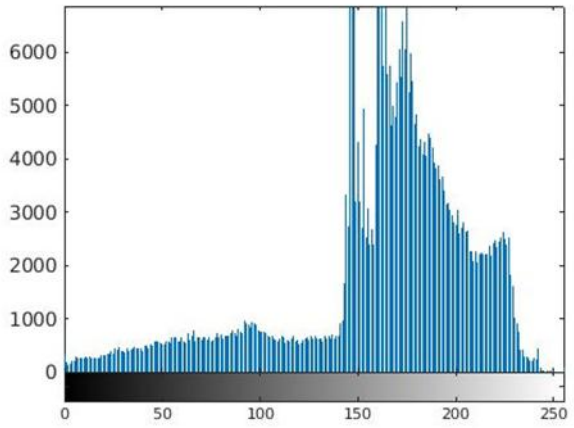

Fig -10: Histogram of Original JPEG Image

Similarly performing histogram analysis on ciphertext image of original JPEG image shown in Fig.11, we get a histogram as shown in Fig.12.

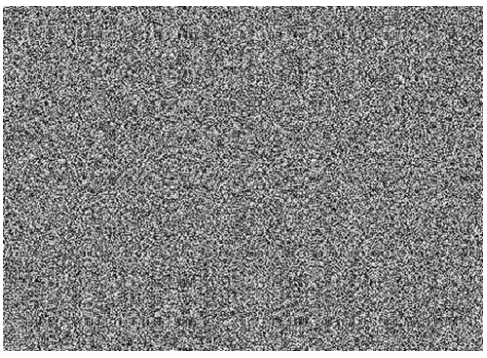

Fig -11: Encrypted JPEG Image

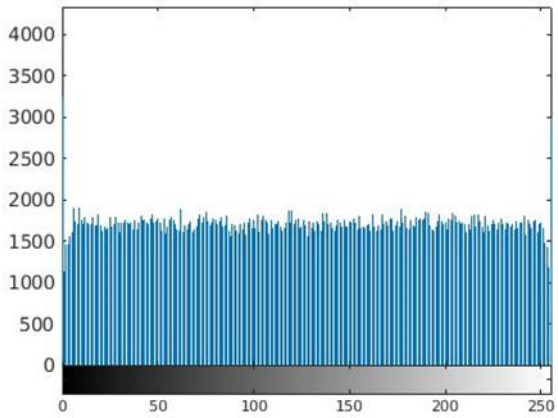

Fig -12: Histogram of Encrypted JPEG Image

While examining the histograms generated for both the original and ciphertext images, ciphertext image histogram is uniformly distributed and thus concluded that the encryption method had a good quality.

The proposed framework uses a two combined phase of data embedding method. For the experiment, the original image in Fig.9 with different quality factors (i.e, QF = 70,80,90,100) were taken. Then the quantity of perfect string (PS) and non-perfect string (NPS) for each DCT quantized block is shown in Table.3. The experimental results show that an increase in quality factor decreases the number of PS. But the results show that an average number of PS can be reached when the quality factor is 80. Even if the quality factor is 100, there is a considerable amount of PS. Three bits of data can be hidden in one PS by using (7,4) Hamming code. Thus, the embedding capacity increases with increase in PS. Compared to the previous RDHEI techniques which have low embedding capacity, this proposed embedding framework provide high embedding capacity as shown in Table.4.

Table -3: Number of PS and NPS

QF	PS	NPS
70	12443	5997
80	10990	7442
90	7487	10945
100	2316	16116

Table -4: Comparison of Embedding Capacity

Images	[16]	[17]	Proposed System
Lena	798	3667	9432
Man	1809	4856	13297
Peppers	960	4253	9884

The proposed JPEG encryption algorithm is also secure against the ciphertext-only attack. During JPEG bitstream encryption, it pseudo-randomly select n entropy-encoded segment from the bitstream to construct a new JPEG bitstream, which can be decoded to smaller sized ciphertext image. An attacker has no information of the original size of the image because only a part of Huffman codes are available. It is difficult for the attacker to find the original orders of all blocks, as long as the entropy encoded block number N is large enough. Thus, larger key space is enough to ensure security.

5. CONCLUSIONS

Many unauthorized users try to get the protected information and therefore it is necessary to secure our data. There are also scenarios in which data hiding needs to be done in the encrypted domain or combined with the encryption, especially in the domain of big data and cloud computing. But the main challenge is the secure transmission of user's data. The proposed system is designed to develop a cloud application that can hide the details of the image owner who uploads the encrypted JPEG image to the cloud. The authorized user when require can request the cloud for the image, then cloud will extract the hidden data and send the recovered encrypted image to the user who can download it and decrypt it. The proposed system provides a high capacity embedding and image privacy. The tasks of data embedding, extraction and bitstream recovery are done by cloud, thus it is a client free application i.e, the proposed framework requires the owner or user to do no extra tasks except encryption/decryption.

REFERENCES

[1] Li, X., Li, B., Yang, B., & Zeng, T. (2013). General Framework to Histogram-Shifting-Based Reversible Data Hiding. IEEE Transactions on Image Processing, 22(6), 2181–2191. doi:10.1109/tip.2013.2246179

[2] X. Zhang, "Reversible data hiding in encrypted image," IEEE Signal Process. Lett., vol. 18, no. 4, pp. 255–258, Apr. 2011.

[3] W.-L. Tai, C.-M. Yeh, and C.-C. Chang, "Reversible data hiding based on histogram modification of pixel differences," IEEE Trans. Circuits Syst. Video Technol., vol. 19, no. 6, pp. 906–910, Jun. 2009.

[4] W. Hong, T.-S. Chen, and H.-Y. Wu, "An improved reversible data hiding in encrypted images using side match," IEEE Signal Process. Lett., vol. 19, no. 4, pp. 199–202, Apr. 2012.

[5] X. Liao and C. Shu, "Reversible data hiding in encrypted images based on absolute mean difference of multiple neighboring pixels," J. Vis. Commun. Image Represent., vol. 28, pp. 21–27, Apr. 2015.

[6] Z. Qian, S. Dai, F. Jiang, and X. Zhang, "Improved joint reversible data hiding in encrypted images," J. Vis. Commun. Image Represent., vol. 40, pp. 732–738, Oct. 2016.

[7] J. Zhou et al., "Secure reversible image data hiding over encrypted domain via key modulation," IEEE Trans. Circuits Syst. Video Technol., vol. 26, no. 3, pp. 441–452, Mar. 2016.
[8] K. Ma, W. Zhang, X. Zhao, N. Yu, and F. Li, "Reversible data hiding in encrypted images by reserving room before encryption," IEEE Trans. Inf. Forensics Security, vol. 8, no. 3, pp. 553–562, Mar. 2013.
[9] W. Zhang, K. Ma, and N. Yu, "Reversibility improved data hiding in encrypted images," Signal Process., vol. 94, pp. 118–127, Jan. 2014.
[10] X. Cao, L. Du, X. Wei, D. Meng, and X. Guo, "High capacity reversible data hiding in encrypted images by patch-level sparse representation," IEEE Trans. Cybern., vol. 46, no. 5, pp. 1132–1143, May 2016.
[11] M. Fujiyoshi, "Separable reversible data hiding in encrypted images with histogram permutation," in Proc. IEEE Int. Conf. Multimedia Expo, Jul. 2013, pp. 1–4.
[12] W. Zhang, H. Wang, D. Hou, and N. Yu, "Reversible data hiding in encrypted images by reversible image transformation," IEEE Trans. Multimedia, vol. 18, no. 8, pp. 1469–1479, Aug. 2016.
[13] Qian, Z., Dai, S., Jiang, F., & Zhang, X. (2016). Improved joint reversible data hiding in encrypted images. Journal of Visual Communication and Image Representation, 40, 732–738. doi:10.1016/j.jvcir.2016 .08.020.
[14] Huang, F.,Qu, X., Kim, H. J., & Huang, J.(2016). Reversible Data Hiding in JPEG Images. IEEE Transactions on Circuits and Systems for Video Technology, 26(9), 1610–1621. doi :10.1109/tcsvt.2015. 2473235.
[15] Qian, Z., Zhou, H., Zhang, X., & Zhang, W. (2016). Separable Reversible Data Hiding in Encrypted JPEG Bitstreams. IEEE Transactions on Dependable and Secure Computing, 1–1.doi:10.1109/tdsc.2016. 2634161.
[16] J. C. Chang, Y. Z. Lu, and H. L. Wu, "A separable reversible data hiding scheme for encrypted JPEG bitstreams," Signal Process., vol. 133,pp. 135–143, Apr. 2017.
[17] Qian, Z., Xu, H., Luo, X., & Zhang, X. (2018). New Framework of Reversible Data Hiding in Encrypted JPEG Bitstreams. IEEE Transactions on Circuits and Systems for Video Technology, 1–1. doi:10.1109/tcsvt.2018.2797897
[18] Chin-Cheng Chang, Ran Tang, Chia-Chen Lin, Wan-Li Lyu, "High-Capacity Reversible Data Hiding Method for JPEG Images," Journal of Software vol. 13, no. 1, pp. 1-17, 2018.