1. Introduction

Reversible Data Hiding in Encrypted Images by Reversible Image Transformation is used for increasing security of images while transferring it from a source to destination. The major challenge lies in encrypting an image so as to improve its security. Besides, to fit different image resolutions, the image must be properly merged with another image on top of it. This document proposes the implementation of an algorithm for transformation and anti-transformation of an image. This document describes the functional and non-functional requirements of the software application of Reversible Data Hiding in Encrypted Images by Reversible Image Transformation. The working and objectives is briefly summarized. The main aim of this project, ‘Reversible Data

Hiding in Encrypted Images by Reversible Image Transformation’ is to improve the security of images. The system first accepts an image from client and then it will be encrypted with another image so as to obtain a merged image by keeping the new image over the original image. For this encryption we need to make resolution of both images same and then divide the original into four pieces of equal resolution and then rotate the four pieces in different directions using a transformation algorithm and merge new image over it. Also create a key and set access rights to users so that others can’t access it. A new client will access this image using his login and authentication so that others can’t access. If user has access right and if key is correct then provide the original image to client else produce a fake image.

2. System Design

2.1 High Level Design

2.1.1 Product Modules

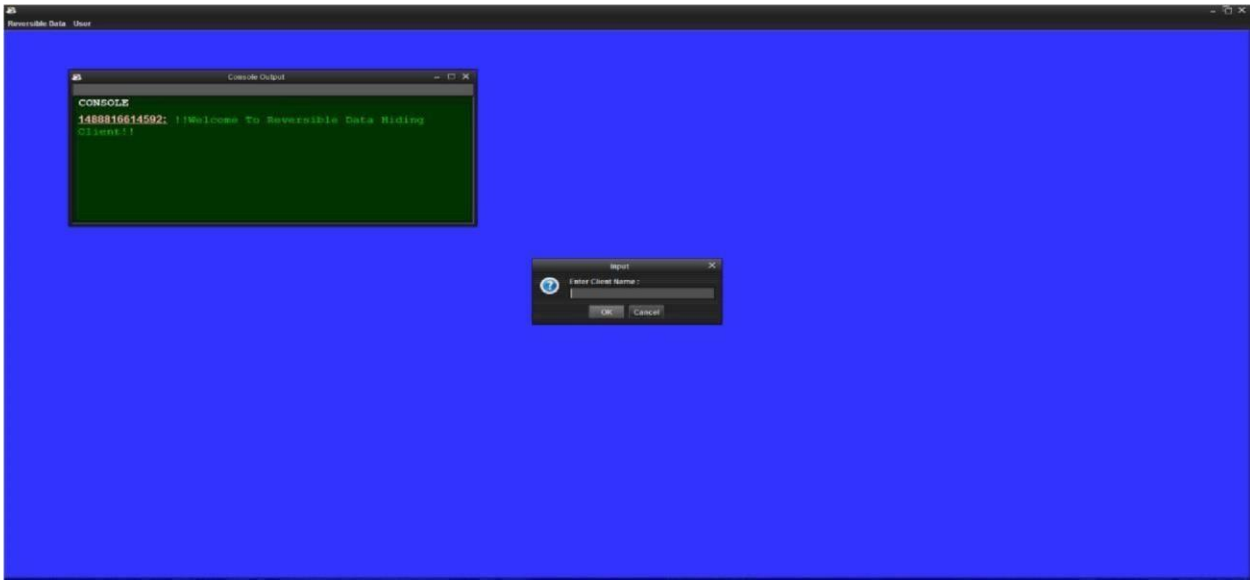
The Application has mainly four modules, when it comes to development. They are: -

- Image Acquisition
- Image Transformation
- Image Anti-transformation
- Image Retrieval

2.2 Detailed Design

2.2.1 Image Acquisition

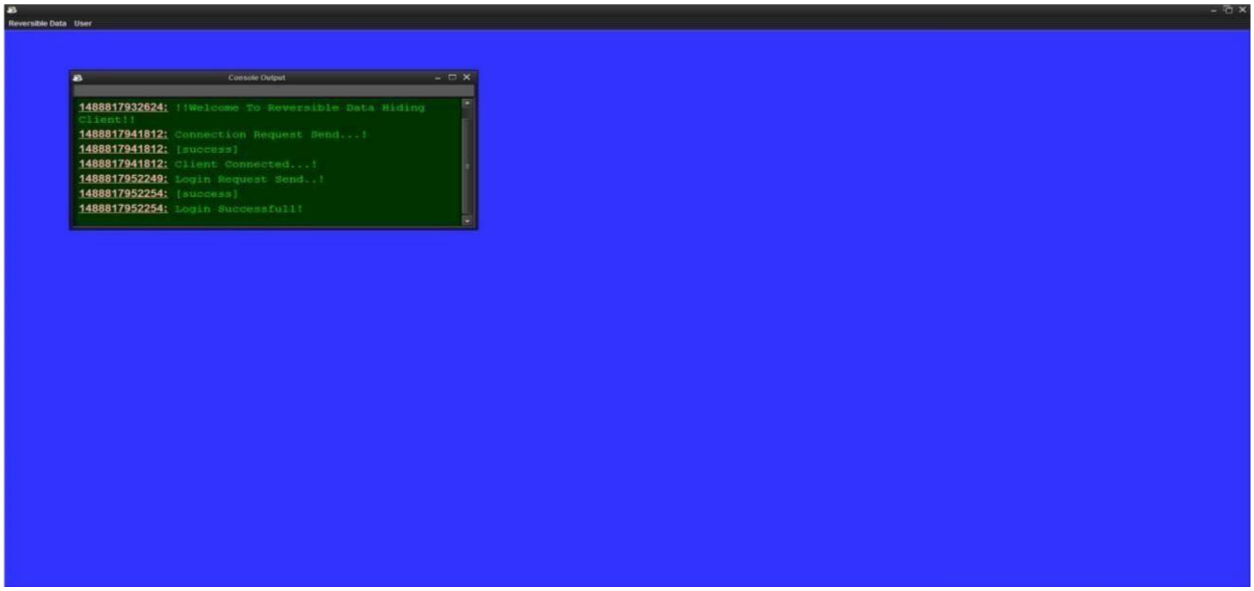
In this module, we will accept the image from the user and the image to be transferred is then send to the server for encryption.



```
CONSOLE
1488816614592: !!Welcome To Reversible Data Hiding
Client!!
```

Enter Client Name :

Fig 2.1 (a)



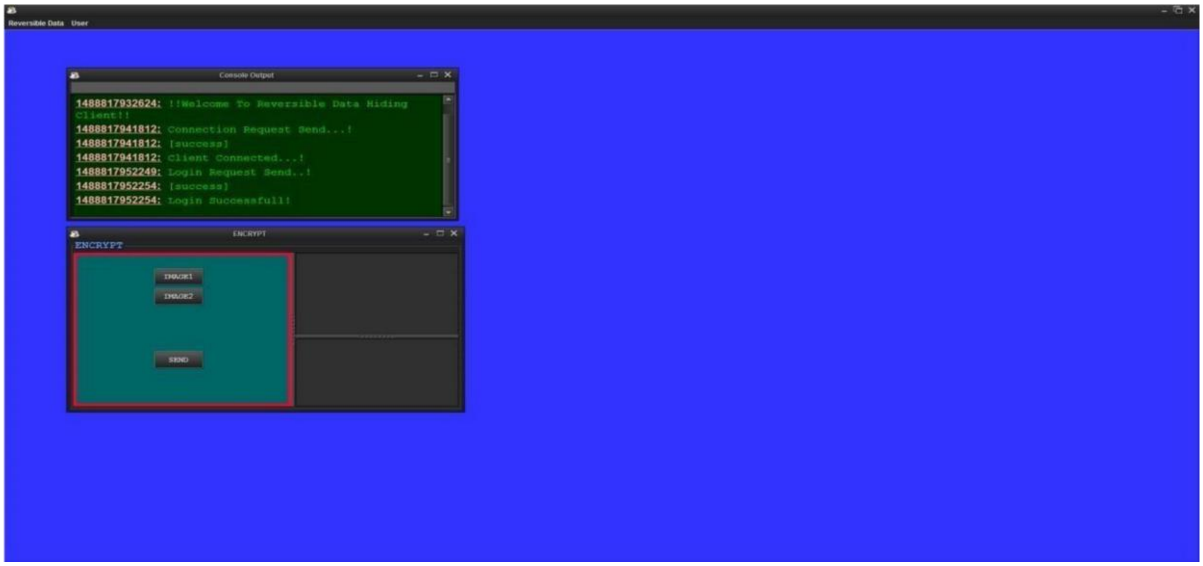
```
1488817932624: !!Welcome To Reversible Data Hiding
Client!!
1488817941812: Connection Request Send...!
1488817941812: [success]
1488817941812: Client Connected...!
1488817952249: Login Request Send..!
1488817952254: [success]
1488817952254: Login Successfull!
```

Fig 2.1 (b)

Fig 2.1 (a) & (b) : Image Acquisition

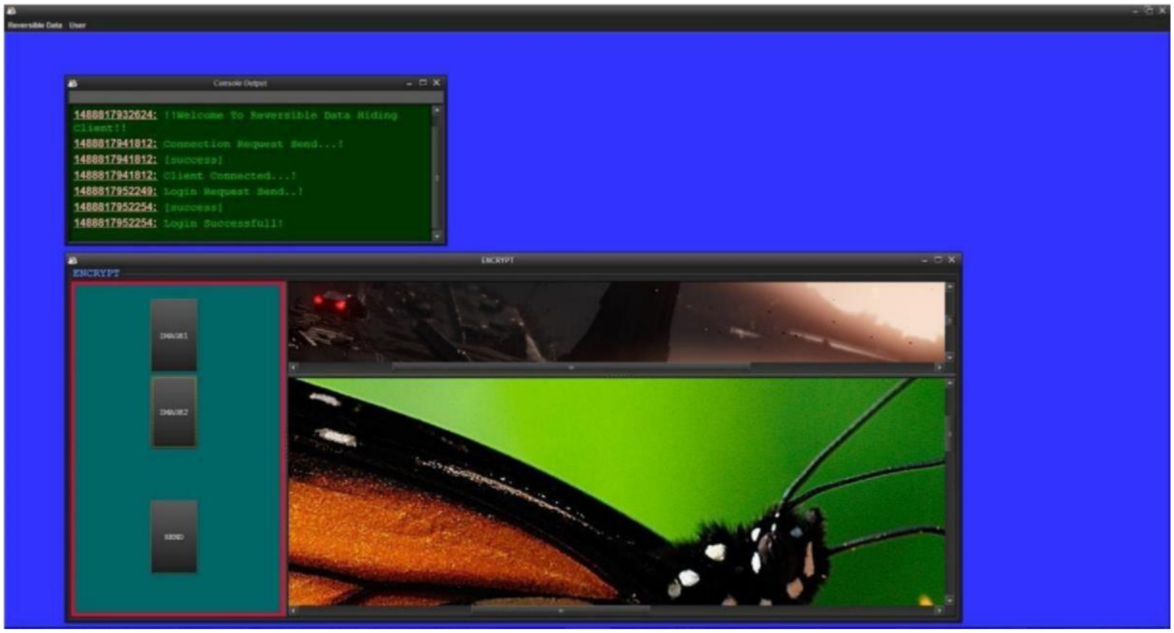
2.2.2 Image Transformation

Here the accepted image that is to be encrypted is transferred to server and then the target image is selected and using transformation algorithm we will encrypt the original image to transformed image. And this is done in the server part.



```
1488817932624: !!Welcome To Reversible Data Hiding
Client!!
1488817941812: Connection Request Send...!
1488817941812: [success]
1488817941812: Client Connected...!
1488817952249: Login Request Send..!
1488817952254: [success]
1488817952254: Login Successfull!
```

Fig 2.2 (a)



```
1488817932624: !!Welcome To Reversible Data Hiding
Client!!
1488817941812: Connection Request Send...!
1488817941812: [success]
1488817941812: Client Connected...!
1488817952249: Login Request Send..!
1488817952254: [success]
1488817952254: Login Successfull!
```

Fig 2.2 (b)

Fig 2.2 (a) & (b) Image Transformation

2.2.3 Image Anti-Transformation

In this section we will perform the anti-transformation algorithm. The image that is to be obtained is decrypted from the target image using anti transformation algorithm. Here we will have the key and then the transformed image is decrypted using the key.

2.2.4 Image Retrieval

In this module, the original image is retained from the target image after performing anti transformation. And the original image is given to the new client who enters the correct key.

3. Test Plan

Testing is the process of executing a program with the intent of finding any errors Testing are of the following types:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

3.1 Unit Testing

The software units in a system are modules and routines that are assembled and integrated to perform a specific function. Unit testing focuses first on modules, independently of one another, to locate errors. This enables, to detect errors in coding and logic that are contained within each module. This testing includes entering data and ascertaining if the value matches to the type and size supported by java. The various controls are tested to ensure that each performs its action as required.

The purpose is to validate that each unit of the software performs as designed.

- Testing the path
- Input document
- Check the validity of path

- Check for matching path

3.2 Integration Testing

Data can be lost across any interface, one module can have an adverse effect on another, sub functions when combined, may not produce the desired major functions. The objective is to take unit tested modules and build a program structure. All the modules are combined and tested as a whole.

3.3 System Testing

The process of testing an integrated system to verify that it meets specified requirements. Formal testing with respect to user needs and requirements conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user or other authorized entity to determine whether or not to accept the system.

3.4 Acceptance Testing

User acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the system users at time of developing and making changes whenever required.

4. Implementation Details

4.1 Hardware Requirements:

- Processor: Pentium IV or above.
- RAM: Minimum 64 MB.
- Standard Keyboard and Mouse.

4.2 Software Requirements:

- Operating System: Windows XP and above.
- NetBeans IDE 8.2

- MySQL

4.3 Details of Algorithms

Firstly, we divide the original image I and the target image J into N non-overlapping blocks respectively, and then pair the blocks of I and J as a sequence such that (B1,T1),...,(BN,TN), where Bi is an original block of I and Ti is the corresponding target block of J, 1 ≤ i ≤ N. We will transform

Bi toward Ti and generate a T ′ i similar to Ti. After that, we replace each Ti with T ′ i in the target image J to get the transformed image J′. Finally, we embed some accessorial information into J′ with an RDH method and generate the ultimate “encrypted image” E(I). These accessorial information is necessary for recovering I from J′. Before being embedded, this accessorial information will be compressed and encrypted with a key K shared with the receiver, so only a receiver having K can decrypt E(I). The proposed transformation process consists of three steps: block pairing, block transformation and accessorial information embedding. We will mainly elaborate the first two steps in the subsections and the third step can be implemented by any traditional RDH method.

4.3.1 Block Pairing

To make the transformed image J′ look like target image J, we hope, after transformation, each transformed block will have close mean and standard deviation (SD) with the target block. So we first compute the mean and SD of each block of I and J respectively. Let a block B be a set of pixels such that B = {p1, p2, ⋯, pn}, and then the mean and SD of this block is calculated as follows.

$$u = {}^{1}/_{n} \sum_{i=1}^{n} p_i$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(p_i - u)^2}{n}} \qquad \text{Eqn. (4.1)}$$

When matching blocks between original image and target image, we hope two blocks with closest SDs to be a pair. In Lee et al.’s method, the blocks of original image and target image are sorted in ascending order according to their SDs respectively, and then each original block is paired up with a corresponding target tile in turn according to the order. To recover the original image from the transformed image, the positions of the original blocks should be recorded and embedded into the transformed image with an RDH method. If the image is divided into N blocks, N logN bits are needed to record block indexes. Obviously, the smaller the block size is, the better the quality of transformed image will be, but which will result in a large N. Therefore, the amount of information used to record the index for each block may be so large that it will cause much distortion when embedding this information into the transformed image. In fact, there may not exist enough redundant space to store this additional information. For instance, if we divide a 1024 × 1024 image into 4 × 4 blocks, 216 × 16 bits are needed to record the positions of blocks. To compress the block indexes, we first classify the blocks according to their SD values before pairing them up. In fact, we found that the SD values of most blocks concentrate in a small range close to zero and the frequency quickly drops down with the increase of the SD value, which is depicted from various sizes of 10000 images from the BossBase image database. Therefore, we divide the blocks into two classes with unequal proportions: class 0 for blocks with smaller SDs, and class 1 for blocks with larger SDs, and pair up the blocks belonging to the same class. By assigning the majority of blocks to the class 0, we can avoid the large deviation of SDs between a pair of blocks and efficiently compress the indexes at the same time.

In the present paper, we propose to divide both the original and target images into nonoverlapping 4 × 4 blocks and calculate the SDs of each block. We first divide the blocks of original image I into 2 classes according to the quantile of SDs. Denote that the %α quantile of SDs by Nα. We assign the blocks with SDs ∈ [0, Nα] to "Class 0", and blocks with SDs ∈ (Nα,N100] to "Class 1". And then we will scan the blocks in the raster order, i.e., from left to right and from top to bottom, and assign a class label, 0 or 1, to each block. Next, we label the blocks of target image based on the classes' volumes of original image. Assuming that the i'th class in the original image includes ni blocks for i = 0 or 1, we scan the target image in the raster order, and label the first n0 blocks with the smallest SDs as Class 0, and the rest n1 blocks as Class 1. As a result, each class in the target image includes the same number of blocks as the corresponding class in the original image. We pair the original block up with target block in the following manner. Scan the original image and target image in raster order respectively and pair the jth block of the class i in the original image up with the jth block of the class i in the target image for i = 0,1 and j = 1...,ni.

A simple example on the proposed block pairing method in which the image only consists of 10 blocks. By setting α = 70, we assign 7 blocks with smallest SDs into class 0, and the rest 3 blocks into class 1 in the original image. In the target image, although the 8th and 9th block have the same SD value 5, the 8th block is assigned to class 0 but the 9th block is assigned to the class 1, because class 0 can only include 7 blocks as determined by the class 0 of the original image. After labelling the class indexes, we get a class index table (CIT) for original image and target image respectively, which will be helpful for understanding the procedure of block pairing.

According to the pairing rule, the first block of the original image is paired up with the forth block of the target image, because both of them is the first block of class 1 as shown in the CIT; the second block of original image is paired up with the ninth block of target image, because both of them is the second block of class 1, and so on. The pairing result is listed in Table I, which can be generated according to the CIT of original image and the CIT of the target image.

For each pair of blocks (B, T), as we will see in the next section, the original block B will be transformed to target block T by mean shifting and block rotation, yielding T ′. By replacing each T with T ′ in the target image, the sender will generate the transformed image. Note that both operations of mean shifting and block rotation will not change the SD value, so T ′ has the same SD as B. Therefore, the SDs in transformed image is only a permutation of those in original image.

When classifying the blocks of transformed image according to %α quantile of SDs, the receiver can get a CIT that is same with the CIT of target image. Therefore, to restore the original image from the transformed image, the receiver only needs to know the CIT of the original image. In fact, by CIT of original image and the CIT of transformed image (which is also the CIT of target image), the receiver can reconstruct Table I perfectly. Then according to the table he will know how to rearrange the transformed blocks to restore the original blocks. In the example, the first block of the transformed image should be put back to position 3, and the second block should be put back to position 4.

Note that CIT can be efficiently compressed because the ratio of 0 and 1 is bias. If the image is divided into N blocks, and these blocks are divided into two classes with %α quantile of SDs, we need N · H(α/100) bits to record S, where H is the binary entropy function. For instance, if we set α

= 75 and divide a 1024 × 1024 image into 4 × 4 blocks, we only need 216 ×H(0.75) ≈ 216 × 0.81 bits to record the positions of blocks, which is much less than 216 × 16 bits used by the method. The compressed CIT will be encrypted and embedded into the transformed image as a part of accessorial information (AI).

4.3.2 Block Transformation

By the block pairing method described above, in each pair (B, T), the two blocks have close SD values. Therefore, when transforming B towards T, we only need a mean shifting transformation that is reversible. However, the transformation used in Lee et al.'s method is not reversible because it changes the mean and SD at the same time. Let the original block B = {p1, p2,··· ,pn}, and the corresponding target block T = {p′ 1,p′ 2,··· ,p′n}. With Eq. (1), we calculate the means of B and T and denote them by uB and uT respectively. The transformed block T ′ =

{p′′ 1, p′′ 2,··· ,p′′ n} is generated by the mean shifting as follows. p′′ i = pi + uT − uB, where (uT − uB) is the difference between the means of target block and original block. We want to shift each pixel value of original block by amplitude (uT − uB) and thus the transformed block has the same mean with the corresponding target block. However, because the pixel value p′′ i should be an integer, to keep the transformation reversible, we round the difference to be the closest integer. Δu

= round (uT − uB), and shift the pixel value by Δu, namely, each p′′ i is gotten by p′′ i = pi + Δu, Note that the pixel value p′′ i should be an integer between 0 and 255, so the transformation may result in some overflow/underflow pixel values. To avoid such transformed blocks abstained by

Eq. (1), we assume that the maximum overflow pixel value is OV_{max} for Δu ≥ 0 or the minimum underflow pixel value is UN_{min} for Δu < 0. If overflow/underflow occurs in some blocks, we eliminate them by modifying Δu

$$\Delta u = \begin{cases} \Delta u + 255 - OV_{max} & , if\ \Delta u \geq 0 \\ \Delta u - UN_{min} & , if\ \Delta u < 0 \end{cases}$$

Eqn. (4.2)

We use the modified Δu to shift the pixels of block B, and thus all the pixels' values are controlled into the range of [0,255]. However, the range of Δu's value is still very large, which cannot be efficiently compressed. Thus we further modify Δu as

$$\Delta u = \begin{cases} \lambda \times round\left(\frac{\Delta u}{\lambda}\right) & , if\ \Delta u\ \geq\ 0 \\ \lambda \times floor\left(\frac{\Delta u}{\lambda}\right) + \frac{\lambda}{2} & , if\ \Delta u\ \geq\ 0 \end{cases}$$

Eqn. (4.3)

In which the quantization step, λ, is an even parameter. Then it just needs to record Δu′ =

$2|\Delta u|/\lambda$, by which it has the advantage of not to record the sign of Δu. Because when $\Delta u'$ is an even number it means $\Delta u \geq 0$ and when $\Delta u'$ is an odd number it means $\Delta u < 0$. Since when λ is large the amount of information recording $\Delta u'$ will be small but the offset between the modified

Δu and the original Δu will be large, a trade-off must have made by choosing λ. We set $\lambda = 8$ in the following experiments. Finally, to maintain the similarity between the transformed image and target image as much as possible, we further rotate the shifted block into one of the four directions 0º, 90º, 180º or 270º. The optimal direction is chosen for minimizing the root mean square error between the rotated block and the target block. After shifting transformation and rotation, we get a new block T′. With these new blocks, we replace the corresponding blocks in the target image and generate the transformed image J′. The parameters, $\Delta u'$ and rotation directions, will be compressed, encrypted and then embedded into the transformed image J′ as accessorial information (AI) to output the "encrypted image" E(I) called in this paper image. The transform and antitransformation procedures of the proposed method are described in Algorithm 1 and Algorithm 2 respectively.

Algorithm 1: Procedure of Transformation Input:

An original image I and a secret key K.

Output: The encrypted image E(I).

1) Select a target image J having the same size as I from an image database.

2) Divide both I and J into several non-overlapping 4×4 blocks. Assuming that each image consists of N blocks, calculate the mean and SD of each block.

3) Classify the blocks with %α quantile of SDs and generate CITs for I and J respectively. Pair up blocks of I with blocks of J according the CITs as described in subsection III-A.

4) For each block pair (B_i,T_i) ($1 \leq i \leq N$), compute the mean difference Δu_i. Add Δu_i to each pixel of B_i and then rotate the block into the optimal direction θ_i ($\theta^i \in \{0^o, 90^o, 180^o, 270^o\}$, which yields a transformed block T'_i.

5) In the target image J, replace each block T_i with the corresponding transformed block T'_i for $1 \leq i \leq N$ and generate the transformed image J′. 6) Collect Δui's and θi's for all block pairs, and compress them together with the CIT of I. Encrypt the compressed sequence and the parameter α by a standard encryption scheme such as AES with the key K.

7) Take the encrypted sequence as accessorial information (AI), and embed AI into the transformed image J′ with an RDH method, and output the encrypted image E(I). **Algorithm 2:**

Procedure of Anti-transformation

Input: The encrypted image E(I) and the key K.

Output: The original image I.

1) Extract AI and restore the transformed image J′ from E(I) with the RDH scheme.

2) Decrypt AI by AES scheme with the key K, and then decompress the sequence to obtain CIT of I, Δu_i, θ_i ($1 \leq i \leq N$) and α.

3) Divide J′ into non-overlapping N blocks with size of 4 × 4. Calculate the SDs of blocks, and then generate the CIT of J′ according to the %α quantile of SDs.

4) According to the CITs of J′ and I, rearrange the blocks of J′ as described in Subsection III-A.

5) For each block T'_i of J′ for $1 \leq i \leq N$, rotate T'_i in the anti-direction of θi, and then subtract Δu_i from each pixel of T'_i and finally output the original image I.

5. Database Design

Here we have two types of databases one is for storing the target image and during encryption we will store the image to the image database and then during decryption the image is retrieved from the database. And when the key is produced to the target image the original image is produced,

Hence the image database stores the target images. We have another type of database in which we will store the credentials of the authorised users and in this we will enter the details of clients who are given access rights to access the image and then when any client tries to access the target image and obtain the original image then the access rights are checked and if credentials are clear and access rights are given then the original image is produced. Thus we have 2 type of databases used in this.

6. Conclusion

The document proposes the implementation of a model to transfer images from one user to another through a server. It generates a target image along with a key so as to improve security. Using encryption algorithm, we create a target image from the original image. Finally, we transmit the target image and the new user will have to authenticate himself. After authentication when target image and key is produced original image is retrieved using anti transformation algorithm.

Hence image is safely transferred.