International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878
Volume 3 Issue 3, March 2015

# Separable Reversible Data Hiding With Data Optimization for Minimum Loss to Original Image

**Vrishali Gaikwad[1], Todmal S. R.[2]**

[1]Post Graduate Student
Department Of Computer Engineering
Imperial College of Engineering & Research
Wagholi Pune, Maharashtra, India

[2]Professor
Department Of Computer Engineering
Imperial College of Engineering & Research
Wagholi Pune, Maharashtra, India

**Abstract:** ***In this paper, the problem of transmitting large amount of data over an insecure, communication channel with minimum loss to the image in which the data is hidden is discussed. A data owner encrypts the data using a secret key. The secret key is taken and LFSR algorithm is applied to the secret key to obtain a set of unique keys. Each key from the set is embedded with the actual data and checked against the image's pixel bit values (LSB of RGB channel) as to obtain the optimal index, using this index the data will be hidden into the image, so that it causes minimum change in the original image's pixel bit values (Noise reduction).Once the data is hidden into the image using the optimal index the image is then encrypted in parallel using the encryption key (reducing encryption time) to obtain the encrypted stego image. This image at receiver's end can be decrypted with the decryption key and the original data can be obtained from the image only if the receiver has the secret key. If the receiver has both the data-hiding key(secret key) and the encryption key, the receiver can extract the additional data and the original image without any loss.***

**Keywords:** LFSR; AES; Image recovery; Datahiding; LSB replacement; UIQI.

## 1. Introduction

One of the secure ways of protecting data is encryption which converts the normal data into a meaningless piece of data, this is the reason why the traditional data processing takes place before encryption and lastly after decryption. In certain scenarios it is been observed that an data owner does not rely on the processing service provider, the ability to alter the encrypted data when keeping the plain content unrevealed is desired. Here, when the secret data is to be transmitted, a channel provider or an unauthorized user may tend to destroy the data or may try to get the secret message. So to avoid this threat the source is first embedded with the secret key using a standard algorithm. Then, the embedded source is hidden into an image using the Data hiding algorithm .Then the image along with the hidden data is encrypted with the standard encryption algorithm. At the receiver, decryption is performed first, followed by data extraction from the received image.

Sending the data to a receiver or keeping that data in an shared location and being sure that only the legitimate user only can get the original data from it is the essence of the paper/project. In this way we can easily put our data into shared drives which will be available for all the users who are able to access that shared drive, but the person who will have the password keys only will be able to make the use of that data. For other users it will just be a non-intellectual piece of data. However, in some application scenarios, a sender needs to transmit some data to a receiver and hopes to keep the information confidential to a network operator in provides the channel resource for the transmission. That means the sender should encrypt the original data and the network provider may tend to compress the encrypted data without any knowledge of the cryptographic key and the original data. At receiver side, a decoder integrating decompression and decryption functions will be used to reconstruct the original data.

The art of protecting data by encrypting it into an unreadable format is called cipher text. Only users who possess a secret key can decrypt the message into plain text. Encrypted messages can be broken by cryptanalysis, also called code breaking, although modern cryptography techniques are almost unbreakable. Cryptography systems are classified into symmetric-key systems that use the same key(single) that both the sender and recipients posses, and public-key systems that use two keys, a public key known to everyone and a private key that only the receiver of data uses.

## 2. Different Ways of Encryption

*A.* Hashing
In this method a unique, fixed-length signature is created for data set. Hash function is used to create the hashes and people commonly use them to compare data sets. Since a hash is unique to a specific message, even minor changes to that message result in a dramatically different hash, there by alerting a user to potential tampering.

*B.* Symmetric Encryption
Symmetric cryptography is commonly called as private-key cryptography. Term private key refers to that key which is used to encrypt and decrypt the data and must remain secret because anyone having the private key can decode messages. A sender encrypts a message into cipher text using a key, and the receiver uses the same key to get encrypted messages.

*C.* Asymmetric Encryption

Asymmetric also referred as public key cryptography is more secure than symmetric method. This method uses two keys, a private and a public key to perform encryption and decryption. The use of two keys overcomes a major weakness in symmetric key cryptography, since a single key does not need to be securely managed among multiple users. In this method of cryptography, a public key is available to all and is used to encrypt messages before sending them. A different, private key remains with the receiver of cipher text messages, who uses it to decrypt them the messages.

## 3. AES Algorithm

AES is most widely used “symmetric block cipher” for encrypting data which can be decrypted with the encryption key. AES works fast in case of both software and hardware. AES operates on a 128 bit 2D matrix of bytes, termed as state, Few versions of Rijndael uses states of larger sizes with additional columns in it. AES calculations are done in a special finite field. The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the cipher text.
The number of cycles of repetition is as follows:

- 10 cycle for 16 byte key.
- 12 cycle for 24 byte key.
- 14 cycle for 32 byte key.

Description of AES Algorithm:

1. KeyExpansion—Rijndael’s key schedule is used to derive round keys from the cipher key.
2. Initial Round
    1. AddRoundKey—here every byte of the state is bitwise xor’ed with the round key.
2. Rounds
    1. SubBytes—This is an non-linear substitution step where every byte is substituted with another as per the lookup table.
    2. ShiftRows—This is an transposition step where every row of the state is cyclically shifted by a number of steps.
3. MixColumns—Here a mixing operation operates on the state’s columns, combining the four bytes in every column.
4. AddRoundKey- Here a sub-key is combined with the state.
5. Final Round (This round excludes MixColumns)
    1. SubBytes
    2. ShiftRows
    3. AddReoundKey

## 4. Non-Seperable Reversible Data Hiding and Encrypting the Image

A content owner after deciding on the carrier image and the secret data to be sent gives a password to encrypt the data. Here initially we take the password (key1) for creating a unique set of key by applying the LFSR algorithm. These uniquely generated keys are then checked one by one after embedding secret data with it, as to find the optimal key which will save all the data with minimum modification to the original values of the image pixels.

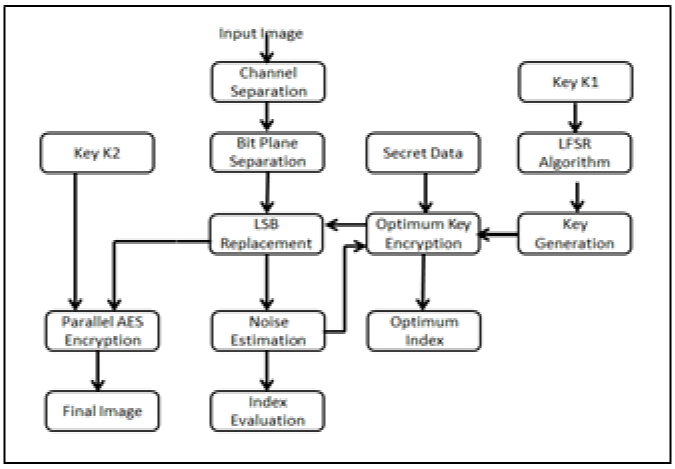

**Figure 1:** Non-separable reversible data hiding in Carrier Image

This way the optimal index key is found with which we hide the embedded data into the image along with that we also hide the optimal index in the image itself instead of sending it separately. Now once the data is hidden into the image we encrypt the carrier image in parallel with the AES algorithm. A new password (key 2) is used for AES encryption of the Image. Once the encryption of image is done we get an encrypted stego image .This stego final file then can be sent to the receiver or can be shared on shared resources with the insurance that the legitimate user only can get the original image and data back who has both the correct keys to decrypt the image and data.

## 5. Finding Optimal Index Key and Data Embedding

In this project we first take the input as the carrier image in which the encrypted data will be hidden. Once carrier image is specified we calculate the amount of data that can be sent based on the size of the image. After receiving the carrier image we take the key (k1) with which the data is to be embedded and hidden into the carrier image. The key that we receive is in the plain text format which is converted into a 32 bit sequence. This 32 bit sequence is then passed to the LFSR algorithm which then creates a unique set of 32 bit keys. This unique set of 32 bits is obtained by XOR’ing the fixed numbered bits(2,4,8 numbered bits) and we shift all the bits one place to the left and put the result bit of XOR’ing in the first position of 32 bit sequence. This process is carried out until we get the original 32 bit sequence back .After having all the 32 bit key we then get the data from the file which has the users secret data in the chunk of 32 bits and XOR it with the 1st key of the key set until we are done with the complete data of the Data file. This XOR’ed data is then checked against the Least Significant Bit values of the RGB channel of each pixel. While doing this we keep track of how many bits are needed to be changed to put this data into an image. This count helps us to do Noise estimation for each key. This process of getting the data from data file and XOR’ing it with different key is carried out for all the keys in the key set and estimation of noise is done for each key in the key set. The key for which the Noise is minimum is used for hiding the data into the Image. This way we can hide all the data with minimum loss to the Original Image. Once this is done we are ready for committing LSB Steganography for finally storing the data with the Optimal Key.

## 6. LSB Steganography and Parallel AES

Least significant bit steganography is a, simple approach to embedded information into a cover image. In case of 8 bit representation least significant bit of few or of all the bytes inside an image are altered to bits of the secret message. While considering a 24-bit image, a bit of every R,G,B color components is used, since they are represented by a byte each. Thus we can store about 3 bits in every pixel. An 1024×768 pixel image, can store a total amount of 2,359,296 bits or 294,912 bytes of secret data.

Least Significant Bit (LSB) is a technique to implement steganography. Like all steganographic algorithms, it embeds the data into the carrier image so that it cannot be detected by mere observation. The technique works by changing some of the information in a pixel with information from the data in the image. It is possible to embed data on any bit-plane of image, so LSB embedding is performed on the least significant bit(s). This reduces the variation in color that embedding creates. For instance, embedding into the least significant bit of each colour channel of each pixel. Every colour pixel has 3 colour channels i.e. R,G,B given by a 8 bit data values, we can embed 3 bits per pixel one in each channel of a pixel. Steganography prevents introduction od variation, to minimize the possibility of getting detected. In a LSB embedding, we lose some information from the original image. This is due to embedding directly into a pixel. To do this we must discard some of the cover's information and replace it with information from the data to hide. LSB algorithms have 2 ways about how they embed and hide the data. Lossless embedding preserves all information about the data, or the data should be generalized so that it takes up less space.

In case of an 8-bitgrayscale bitmap image where every pixel is stored as a byte presenting a gray scale value. Suppose the first eight pixels of the original image have the following grayscale values:

```
{
 11001010 01001010
 10001111 10001100
 00001101 01010111
 00110110 01010011
 }
```

To hide the text character 'C' with binary value as 01000011, we need to replace the LSBs of these pixels to have the following new grayscale values:

```
{
 11001010 01001011
 10001110 10101100
 00001100 01010110
 00110111 01010011
 }
```

However, its major limitation is that small size of data can be embedded in such type of images using LSB. LSB techniques implemented on 24-bit formats are difficult to detect as compared to 8-bit format images. For instance, assume the original three pixels are represented by the 3 24-bit words as below:

```
 {
 (00100111, 11001001, 11011000)
 (00100111, 11001000, 10101001)
 (11001000, 10100111, 10101001)
}
```

The binary value for the text character 'A' is (001000001). Inserting the binary value of 'A' into the 3 pixels, starting from the top left byte, would result in:

```
{
 (00100110 11001000 11011001)
 (00100110 11001000 11101000)
 (11001000 10100110 10101001)
}
```

*D.* Improvements to LSB Steganography

As a standard approach in LSB steganography the data bits are replaced sequentially for each and every pixel, but in this project we have added 2 new features:

1. Embedding based on optimal index: This will decrease the Decrypted Image's distortion.

2. Offset Addition: using this we are hiding the data at specific offsets i.e. we aren't hiding the data into the image sequentially but are hiding it based on the offset value that we calculate from the given key k1.Thus this will make impossible even to guess the sequence of the data bits stored into the image.

AES algorithm is used to encrypt the final stego image containing hidden data. Here we are applying the AES algorithm in parallel i.e. dividing the complete image into 3 parts, so that algorithm can be applied on the 3 parts at one time simultaneously by creating 3 different threads each working on defined set of pixels. The algorithm takes the key k2 for encrypting the image.

## 7. Data Extraction and Image Recovery

The proposed scheme is made up of image encryption, data embedding and data-extraction /image-recovery phases. The end user/receiver on receiving the encrypted image needs to have the key k1 and k2 to get the data as well as image respectively. Once the encrypted image is received by the receiver the below steps are followed to get the data and the image back.

Steps:

1. Get the Encrypted Image.
2. Get the stego image after decrypting, using the Key K2
3. Load the image and extract the RGB values for each pixel into a variable col.
4. Perform Logical AND operation with 0xff on the R,G,B values to get the values of LSB bits for each pixel channel and store it in an array pixdata.

```
5. Get the optimal index stored in the last pixel of the image.
6. Get the stego key from the user, to find the optimal key.
7. Get the size of the data into the variable dataLength.
8. If dataLength <1 then
    Display "Image is not Stegnographed"
9. If dataLength >1 then
 Open file //for writing data
 for each pixdata[index] till end of dataLength
 if (pixdata[index] &1)==1 then
 file.writebit(1)
 else
 file.writebit(0)
 Close file
 end if
10.display message "File Desteganographed Successfully".
11.End.
```

After performing the Desteganography we get the original data into a file .Thus the receiver having both the keys k1 and k2 will be able to get original image as well as the secret data. The word "data" is plural, not singular.

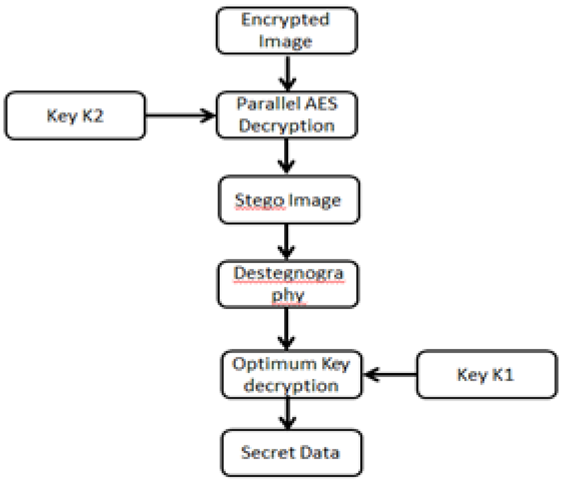

**Figure 2:** Image Decryption

## 8. Implementation and Evaluation Models

We propose a mathematical model for implementing this project and also for evaluating its results by calculating the quality of the decrypted image in terms of MSE, SNR, PSNR and the Universal Image Quality Index.

### *E.* Mathematical Implementation Model

Implementation can be broken down into functions which operate on each other's output.

Let S be the complete system as given
$S=\{U , I , k1 , k2 , I_E , I_S , F\}$
Where
U=Set of User.
I=Set of Input Images.
K1=Set of LFSR keys.
K2=Set of encryption keys
$I_E$=Set of Encrypted Images.
$I_S$=Set of Stego Images.
F=Set of functions used.
F={getLFSRKey(),Image_AES_Encryption(),Image_AES_Decryption(),calBit(),dataembed()}

*1)* Function getLFSRKey():

```
Steps:
1. Get the key k1 from the user to embed the data.
2. Convert the key into a 32 bit sequence save it as originalValue.
3. Pass this 32 bit sequence to calBit()
 calBit()
{
 1.xor the bit values at position 1 and 4 .
 2.output of xor is again xor'ed with 8th bit to
 obtain the calbit,save it as bitToMask.
 }
4. Replace this bitTomask at the 1st position of the 32 bit sequence by shifting other bits to their right, thus creating a new bit sequence.
5. Different bit sequences are generated until we get the originalValue back.
6. Save these bit sequences into an byte array along with its cycleCount.
7. End
```

*2)* Function Image_AES_Encryption():

```
Steps:
1.Define sbox , inv_sbox , Rcon.
2.Get the key k2 from the user to encrypt the embedded data.
3. Divide the image into 3 blocks to apply AES simultaneously to all the blocks.
4.state :block from image to be encrypted.
 w: Cipher key
 state=AddRoundKey(state,w)
 for (round 1 to 9)
 state = SubBytes(state)
 state = ShiftRows(state)
 state = MixColumns(state)
 state = AddRoundKey(state,w)
 state = SubBytes(state)
 state = ShiftRows(state)
 state = AddRoundKey(state, w)
5.end
```

*3)* Function Image_AES_Decryption():

```
Steps:
1.Get the key k2 from the user/receiver to decrypt.
2.Divide the encrypted image into 3 blocks to apply AES decryption on all the blocks.
3.state= block from image to be encrypted.
w=Cipher key.
state=AddRoundKey(state,w)
for (round 1 to 9)
state = InvSubBytes(state)
state = InvShiftRows(state)
state = AddRoundKey(state, w)
state = InvMixColumns(state)
state = InvSubBytes(state)
state = InvShiftRows(state)
state = AddRoundKey(state, w)
4.end
```

*4)* Function dataembed():

Steps:
1. Here the data from data file is embedded with LFSR keys generated.
2. So a file is created with data embedded into it for each key sequence.
4. Offset is generated from the key k1 given.
5. Then each file is read on by one and checked against the nits of the image pixels.
6. if bits match
 move to next bit
 else
totallosscount variable is incremented and move to next bit.
7. Steps 5 and 6 are carried out for all the keys available.
8. The key sequence for which there is minimum value of totallosscount is taken and that cyclecount is also saved.
9. Using that cyclecount the data is embedded into an carrier image finally with the help of offset calculated from the key k1.
10. end.

*F.* Mathematical Evaluation Model

*1)* Mathematical Model for UIQI

Let x={$x_i$ |i=1,2,3,…,N} and y={$y_i$ | i=1,2,3,…,N} be the original and final image signals respectively. The Universal Quality index can be given as :

$$Q = \frac{4\sigma_{xy}\,\bar{x}\,\bar{y}}{(\sigma_x^2+\sigma_y^2)\,[(\bar{x})^2+(\bar{y})^2]}\ ,$$

where

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i\,, \qquad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i\,,$$

$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\bar{x})^2\,, \quad \sigma_y^2 = \frac{1}{N-1}\sum_{i=1}^{N}(y_i-\bar{y})^2\,,$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\bar{x})(y_i-\bar{y})\,.$$

The dynamic value of Q lies in between [-1,1] the best value is 1 where image quality is not compromised at all. This quality index models any distortion as a combination of factors: loss of correlation, luminous distortion and contrast distortion.

*2)* Mathematical Model for PSNR

PSNR is common way used to measure the quality of reconstructed or decrypted images. PSNR can be easily calculated using MSE (Mean Square Error).

$$PSNR = 20\log_{10}\left(\frac{MAX_f}{\sqrt{MSE}}\right)$$

$$MSE = \frac{1}{mn}\sum_{0}^{m-1}\sum_{0}^{n-1}\|f(i,j)-g(i,j)\|^2$$

where
f-gives the matrix data of the original image.
g-gives the matrix data of degraded image.
m-gives the no. of rows of pixels of the images.
i-gives the index of that row.
n -gives the no. of columns of pixels of the image .
j -represents the index of that column
MAXf -is the max signal value that exists in an original image.

## 9. Results and Discussions

The test image [1024*768] is used as a carrier image for our project.

**Figure 3:** Original Carrier Image

Here the key k1 is used as Apple which is converted to 32 bit key and is passed on to the LFSR algorithm to create key sets out of which an optimal key would be found out and will be used to hide the data into the image.

The input image is of total 1024*786 pixels. As each pixel has 3 channels R,G,B which can hold 1 bit each in their least significant bit position. So each pixel holds 3 bits so in all total bits that can be stores in the image will be:

totalBits Available=[1024*768]*3.
So total space available in bytes will be :
totalSpace Available=totalBits/8.
i.e [( [1024*768]*3)/8] bytes.

Once the optimal key is found, the data is hidden into the image using that key into the image and we get a stego image as in Fig 4.

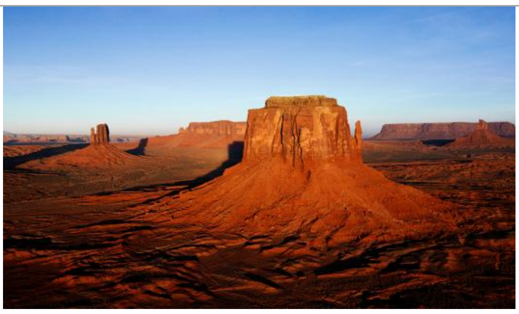
**Figure 4:** Stego Image

This stego image under goes AES encryption in parallel to give the final encrypted image as in Fig 5.

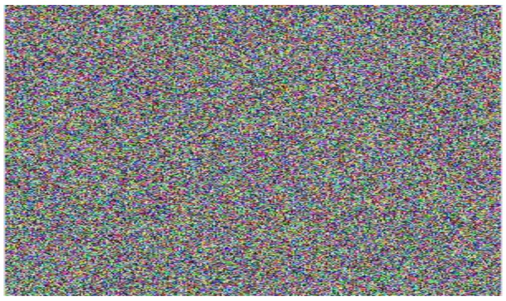
**Figure 5:**.Encrypted Image

This encrypted image is then sent at the receiver's end and who in turn uses the key k2 to get back the stego image from encrypted image as in Fig 6.

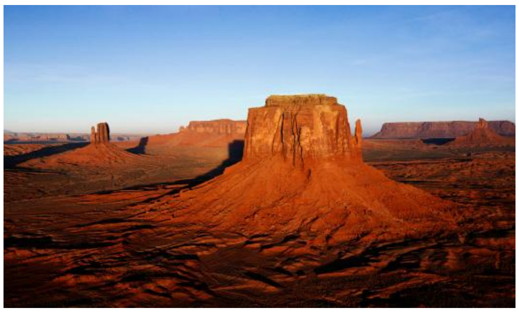
**Figure 6:** Decrypted Stego Image

From the stego image the user has to extract data to get back the original data using the key k1,thus user having both the correct keys will be able to get back the original image and data. Fig 7.

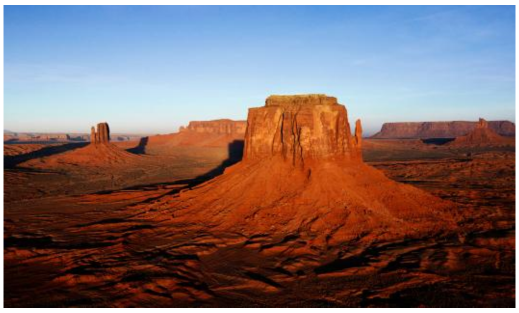
**Figure 7:** Image after data extraction

If in case the size of the data is small and the image has more capacity, then the remaining pixels of the image are embedded with random data so as to avoid steganalysis. The statistical analysis show that the PSNR ratio decreases as we go on hiding more data into the carrier image as more number of bits are needed to be altered for storing it. This can be easily studied from the comparison shown in the Table 1.

**Table 1:** PSNR and QIUI values for Decrypted Image

| *Image* | *Available space (bytes)* | *Data to be hidden(bytes)* | *Actual bit loss %* | *PSNR without data padding* | *PSNR with data padding* | *UIQI* |
|---|---|---|---|---|---|---|
| Image | 294912 | 0 | 0.0000 | 97.2298 | 49.0432 | 0.9742 |
| | 294912 | 147456 | 12.5002 | 54.1401 | 51.1413 | 0.8919 |
| | 294912 | 294912 | 25.0005 | 51.1372 | 51.1447 | 0.8835 |

Table 2 shows the bits loss for each key in the unique key set. Depending on the total bit loss percentage the key is selected for embedding and hiding the data. The key which gives minimum bit loss is used. In this case while embedding full data into image the key used is key no.2 which gives minimum bit loss of only 25.0005%.

**Table 2:** Bit loss calculation for each LFSR key

| *Key No* | *Total Bit loss %* | *Key No* | *Total Bit loss %* | *Key No* | *Total Bit loss %* |
|---|---|---|---|---|---|
| 0 | 49.981 | 11 | 37.5006 | 22 | 50.0002 |
| 1 | 25.0006 | 12 | 50.0006 | 23 | 50.0005 |
| 2 | 25.0005 | 13 | 50.0019 | 24 | 50.0006 |
| 3 | 25.0009 | 14 | 37.5008 | 25 | 37.5008 |
| 4 | 37.5009 | 15 | 37.5008 | 26 | 25.0009 |
| 5 | 37.5008 | 16 | 50.0006 | 27 | 37.5007 |
| 6 | 37.5008 | 17 | 62.5004 | 28 | 37.5008 |
| 7 | 50.0006 | 18 | 75.0003 | 29 | 37.5009 |
| 8 | 50.0005 | 19 | 75.0001 | 30 | 50.0008 |
| 9 | 50.0003 | 20 | 62.5003 | 31 | 37.5008 |
| 10 | 37.5006 | 21 | 50.0004 | | |

Table 3 shows the percent time saved by applying the AES encryption algorithm in parallel as compared to normal AES algorithm.

**Table 3:** Percent time saved by applying AES in parallel

| *Image* | *Image Size (kb)* | *Encryption time( ms)* | *Parallel Encryption time(ms)* | *% time saved* |
|---|---|---|---|---|
| Image 1 | 4122 | 2590 | 890 | 65.64 |
| Image 2 | 2224 | 1780 | 630 | 64.61 |
| Image 3 | 827 | 360 | 130 | 63.89 |

## 10. Conclusion

Steganography is an effective way of hiding the sensitive information. In this paper we have used the LSB technique on images to get the secure stego-image which is encrypted and then sent to the receivers. As we are altering the last bit of each channel of every pixel so there is non-significant change in the image and also we check and use the key that makes minimum changes in the bit pattern for every pixel. This paper focused on hiding more data while increasing the PSNR and reducing the distortion rate. It also focuses in getting the QIUI to close to 1 so that we get minimum distortion in the decrypted Image.

As a future scope we can work on using audio, video instead of cover image for hiding the data. Also an addition of integrity check after the image is ready to be sent is considerable. This will make sure that the receiver is decrypting an original i.e. an unaltered Image. This will make sure that the data thus obtained from the image is 100 % correct.

## 11. Acknowledgement

I would like to thank project guide Prof. Todmal S., Prof. Phursule R., Prof. Wadane V. and our Principal Dr. Admane S. for their valuable comments and suggestions.

## References

[1] Announcing the ADVANCED ENCRYPTION STANDARD (AES), csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
[2] A Secure Image Steganography Using LSB Technique and Pseudo Random Encoding Technique.
[3] Linear feedback shift register ,from Wikipedia, the free encyclopedia.
[4] Separable Reversible Encrypted Data Hiding in Encrypted Image Using AES algorithm and Lossy Technique.
[5] Separable Reversible Data Hiding Using Rc4 IEEE 2013.pdf.
[6] Separable Reversible Data Hiding in Encrypted Image, Xinpeng Zhang, IEEE Transactions on Information Forensics and Security, VOL. 7, NO. 2, APRIL 2012.
[7] A Universal Quality Index,Zhou Wang,Student Member IEEE and Alan C.Bovik,Fellow,IEEE