ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

High Capacity Data Embedding in Image Steganography using Hard Sudoku puzzle

Usha B A[1], Dr N K Srinath[2], Anvith E[3]

Assistant Professor, Department of CSE, R.V.C.E, Bangalore[1]

Professor and Dean, Department of CSE, R.V.C.E, Bangalore[2]

B.E, 8th Semester, Department of CSE, R.V.C.E, Bangalore[3]

Abstract: The practice of concealing a file, message, image, audio or any other media in another file, image, audio or video in such a way that only the sender and receiver know about the message being transmitted is called steganography. In this paper, first an enhancement is proposed to the work of Sanmitra Ijeri et al. wherein a hard sudoku is used instead of a soft sudoku. The embedding and extraction algorithms are generalized to work with any NxN sudoku where N is a perfect square. A comparison is then made between the hard and soft sudoku techniques. A study is also performed to compare parameters such as PSNR and payload capacity for different sudoku sizes.

Keywords: Image steganography, Cover Image, Stego Image, Hard Sudoku, Reference Matrix, Pixel, PSNR

1. INTRODUCTION

Security of information is a vital aspect for most communication mediums today. Encryption, which is the process of converting messages that need to be transmitted securely to a format that can only be understood by authorised parties, plays an important role in the secure transmission of data. However, the fact that a message is in fact being sent is not hidden in the case of encrypted communication. Encrypted messages, no matter how unbreakable, arouse interest if they are plainly visible. The practice of concealing a file, message, image, audio or any other media in another file, image, audio or video is called steganography. Encryption deals with only the protection of data whereas steganography is concerned with concealing the fact that a message is even being sent in the first place, along with protecting the contents of the message being sent. The two techniques can even be combined by sending encrypted data using steganographic means. The use of images as a medium to conceal other forms of data is called image steganography. A technique that uses a Sudoku for embedding secret data in images is discussed in this report. Roshan Shetty B R et al. [1], whose work was inspired by C. Chang et al. [2], made use of the red and green components of a 24-bit RGB image to store secret data. Data embedding and extraction was done with the help of a 27x27 reference matrix that was constructed by repeating a 9x9 sudoku 9 times. The embedding capacity was 3 bits per pixel. Sanmitra Ijeri et al. [3] proposed a revised version of Roshan Shetty B R et al. [1], where the red, green and blue components of the cover image is used. The embedding capacity was increased to 4.5 bits per pixel. The reference matrix used here was 9x9. Before sending data compression and encryption was used so that more, and various forms of digital data could be securely embedded in the cover image.

Embedding capacity per pixel, visual quality and security are three important factors that are used to judge an image steganography technique. The work covered in this report seeks to further build upon the embedding capacity and security of Sanmitra Ijeri et al. [3], discussed in section 2, while maintaining the visual quality above permissible levels. Embedding capacity is improved by generalising the previous algorithm to work for any NxN reference matrix where N is a perfect square, and this is covered in section 3. The embedding capacity increases to 6 bits per pixel for a 16x16 reference matrix, 7.5 bits per pixel for a 25x25 reference matrix, and so on. However, as embedding capacity increases, the visual quality of the image comes down. Depending on the type of application a balance needs to be found between embedding capacity and visual quality. Sanmitra Ijeri et al. [3] relied on the transmission of a completely solved sudoku from the sender to the receiver. If this reference matrix was intercepted by a third party, they could go on and extract the secret data from the cover image. The security of this process is improved here by sending an unsolved and almost empty sudoku from the sender to the receiver. This is also covered in section 3. Section 4 compares the security of this system with the previous one, and section 5 compares other factors such as PSNR and payload capacity for different sudoku sizes.

2. PREVIOUS WORK

Here we discuss the soft sudoku image steganography technique used by Sanmitra Ijeri et al. [3]. The sender uses a random completely filled sudoku, referred to as a soft sudoku, for embedding the data in a cover image. The hidden data to be sent is compressed and encrypted before starting the data embedding procedure. Once data embedding is complete, the cover image containing the hidden data is sent to the receiver. However, the receiver also requires the reference 9x9 sudoku used during the embedding procedure to extract the data from the cover image. For this purpose, the complete 9x9 soft sudoku is encrypted and sent separately by the sender to the receiver

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

by embedding it using LSB technique in another cover image. The receiver extracts the sudoku, decrypts it, and uses it to extract data from the cover image containing the hidden data. This data is then decrypted to obtain the original image.

2.1 Data Embedding Procedure

First a completely filled 9x9 sudoku is selected as a reference matrix. Each element in the matrix is subtracted by 1, as the data to be sent is in Base-9, i.e., between 0 and 8. Next, a cover image to be used for hiding the data is picked. The data to be sent is compressed and encrypted. The encrypted data is converted to a stream of Base-9 numbers S1, S2, S3, etc. Two pixels from the cover image are chosen and the RGB components are paired as follows: C1(R1,G1), C2(B1, R2), C3(G2, B2). This can be generalised as Ci(x, y). Pi(x, y) is computed as shown below:

$$Pi.x = Ci.x \% 9; Pi.y = Ci.y \% 9$$

The 9x9 reference matrix is considered to be an XY grid with 0,0 pointing to the top left element and 8,8 referring to the bottom right element. Therefore (Pi.x, Pi.y) refer to an element in the matrix. The data to be embedded is Si. The minimum change in Pi.x and Pi.y such that they point to the element Si needs to be found. Suppose Pi.x = 6, Pi.y = 4 and Si = 8 as shown in figure 2.1. (Pi.x, Pi.y) refers to the element 4 in the figure. First the column Pi.x is taken with the element 4 in the centre by rotating the column in cyclic fashion to obtain '0 6 7 8 4 5 2 1 3'. The change required in Pi.y is -1 to make (Pi.x, Pi.y) point to 8. Next the row Pi.y is considered. Again, the element 4 is brought to the centre and we have '7 2 5 1 4 6 3 8 0'. Here, a change of 3 is required in Pi.x in order

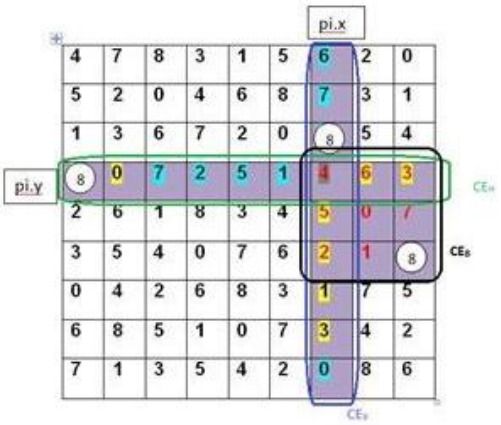

Figure 2.1: Data Embedding Procedure[3]

for the coordinates (Pi.x, Pi.y) to refer to 8. Lastly, the 3x3 sub-matrix is analysed. Here, Pi.x and Pi.y need to be changed by 3 each leading to a total resultant change of 6 in order for (Pi.x, Pi.y) to refer to 8. From these 3 scenarios, the one requiring the least total absolute change in Pi.x and Pi.y is selected and the values of Pi.x and Pi.y are changed accordingly so that they point to the element containing the serial data Si to be embedded. This entire process is repeated for all 3 pairs obtained by taking 2 pixels at a time from the cover image and pairing the RGB components, till no more data is left to embed. The first 10 pixels in the image are used to store the total length of the data that is embedded in the image.

2.2 Data Extraction Procedure

The receiver first needs to obtain the encrypted sudoku from the sender via another cover image. Once the sudoku has been received and decrypted, it is used as the 9x9 reference matrix. This reference matrix needs to be the same as the one used while embedding the data at the sender end. Two pixels from the cover image with hidden data are chosen and the RGB components are paired as follows: C1(R1,G1), C2(B1, R2), C3(G2, B2). This can be generalised as Ci(x, y). Pi(x, y) is computed as shown below:

$$Pi.x = Ci.x \% 9; Pi.y = Ci.y \% 9$$

The 9x9 reference matrix is considered to be an XY grid with 0,0 pointing to the top left element and 8,8 referring to the bottom right element. Therefore (Pi.x, Pi.y) refer to an element in the matrix. This element is the hidden data element Si. The process is repeated for all 3 pairs obtained by taking 2 pixels at a time from the cover image and pairing the RGB components. The length of data to be extracted is stored in the first 10 pixels of the cover image.

3. ENHANCEMENTS

The work that was detailed in section 2 is extended in this paper by generalising the algorithm to be used with a NxN reference matrix. Also, the soft sudoku sent from the sender to receiver in the previous case is replaced by a hard sudoku, i.e, one that is partially filled.

First, the sender uses the sudoku generation algorithm to generate a NxN sudoku that has only N elements filled in. It then feeds this hard sudoku to a sudoku solver that completes the NxN sudoku based on a seed value. The seed value can be any arbitrarily selected number, the creation time of the selected cover image for data hiding in milliseconds, the time in milliseconds for 12pm on the day the cover image is to be sent to receiver, or any other value. The NxN sudoku fed to the sudoku solver has only N elements filled in, due to which it has multiple solutions. The solution that is picked depends on the seed value chosen. Once the complete NxN sudoku is obtained, the sender uses this as a reference matrix to embed the required data onto the cover image selected. The data is compressed, encrypted and converted to a serial stream of Base-N numbers prior to embedding. Once embedding is complete, the cover image is sent to the receiver. The sender also needs to send the partially filled NxN hard sudoku to the receiver. This can be sent as an inconspicuous puzzle through a mail, or by encrypting and embedding it in a separate cover image like was the case in section 2. The receiver also needs to be made aware of the seed value used by the sender. If the seed value is a predetermined dynamic value such as the creation time of the cover image, it does not need to be explicitly communicated. The receiver extracts and decrypts the NxN sudoku that it receives from the sender and feeds it

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

into the same sudoku solver algorithm as was used by the sender, along with the seed value. The output NxN complete sudoku obtained is used as a reference matrix by the receiver to extract data from the cover image. The extracted data is decrypted to obtain the original message. In the following sub sections, the algorithms for sudoku generation, sudoku solving, data embedding and data extraction are detailed.

3.1 SUDOKU Generation and Solver Algorithm

The generation phase uses back tracking to obtain a completely filled sudoku. It uses the system time at which it is executed as the seed value. It starts with the first cell. For the current cell being filled by the algorithm, the procedure is as follows:

- A random number between 0 and N-1 which hasn't been tried yet for this cell is selected and placed in the cell and the number is marked as tried. If no numbers are left to try, the algorithm clears all data associated with the present cell and moves back 1 cell. Step 1 is repeated.
- The sudoku is then validated by checking the row, column and box containing the current cell to see if the number that was filled into the cell is already present.
- If the sudoku is in a valid state, the algorithm moves to the next cell and starts again from step 1.
- If it is in an invalid state, the algorithm moves back to step 1 for the same cell.
- The process stops once the entire sudoku is filled, and is in a valid state.

After a complete NxN sudoku is generated, all but N elements are randomly removed from the sudoku.

The solver phase uses the same approach as the generation phase above to fill the sudoku. The only difference is that the cells that are already filled in the input are skipped by the algorithm, and the seed value to be used is given by the user. It is assumed that the pseudo random number generator function used gives the same sequence of random numbers for a given seed value. This is true for the 'rand' function available in the 'stdlib' header file for C and C++ programs. The pseudo code for this algorithm is given below, where 'rand' refers to the pseudo random number generator:

sudoku -> NxN array initialised to 0 if generating new sudoku, else contains the sudoku to be solved.
tried -> NxNxN array initialised to 0. tried(x,y,0) set to -1 for any sudoku(x, y) not equal to 0.

```
i = 0
j = 0
flag = 1
backTracking = 0
set seedValue to current time if generating new sudoku,
else input from user
while (i < N)
        if (flag)
        r = (rand() mod N) + 1
        if (tried[i][j][0] EQUALS -1)
                if (backTracking EQUALS 1)
                if (j EQUALS 0)
                        j = N - 1
                        i = i - 1
                else
                        j = j - 1
                end if
                flag = 1
                continue to next iteration
        end if
    else if (tried[i][j][r] EQUALS 0)
        tried[i][j][r] = 1
    if (r is can be placed at position i,j in sudoku)
                sudoku[i][j] = r
    else
        flag = 1;
                continue to next iteration
        end if
    else if (tried[i][j][r] EQUALS 1)
        if (tried array is checked and a number is
yet to be tried for position i,j)
                if(r EQUALS N)
                        r = 1
                else
                        r = r+1
                end if
                flag = 0
                continue
    else
        backTracking = 1
                clear tried array for i,j
                sudoku[i][j] = 0
                if(j EQUALS 0)
                        j = N-1
                        i = i-1
                else
                        j = j-1
                end if
                flag = 1
                continue
        end if
    end if
    backTracking = 0
    if(j EQUALS (N-1))
        j = 0
    i = i+1
    else
    j = j+1
    end if
    flag = 1
end while
```

The above algorithm is only to give the reader an idea as to how the same sudoku solution can be generated for a given input sudoku with multiple solutions and a given seed value. Since a sudoku solver is not the primary aim of this report, the efficiency of the algorithm was not of concern. By using standard sudoku solving techniques, the algorithm efficiency can be greatly improved for large values of N such as 25 and above. The above algorithm performs satisfactorily for a 4x4, 9x9 and 16x16 sudoku.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

3.2 Data Embedding Algorithm

Once a completely filled NxN sudoku is obtained from the sudoku solver algorithm, it is used as a reference matrix. Each element in the matrix is subtracted by 1, as the data to be sent is in Base-N, i.e., between 0 and N-1. Next, a cover image to be used for hiding the data is picked. The data to be sent is compressed and encrypted. The encrypted data is converted to a stream of Base-N numbers S1, S2, S3, etc. Two pixels from the cover image are chosen and the RGB components are paired as follows: C1(R1,G1), C2(B1, R2), C3(G2, B2). This can be generalised as Ci(x, y). Pi(x, y) is computed as shown below:

$$Pi.x = Ci.x \% N; Pi.y = Ci.y \% N$$

The NxN reference matrix is considered to be an XY grid with 0,0 pointing to the top left element and N-1,N-1 referring to the bottom right element. Therefore (Pi.x, Pi.y) refer to an element in the matrix at position (x,y). The data to be embedded is Si. The minimum change in Pi.x and Pi.y such that they point to the element Si needs to be found. First the column Pi.x is taken into consideration, and the position of Si is found. Then, the minimum change required in Ci.y to make (Pi.x,Pi.y) point to Si is computed and stored as D1. Next the row Pi.y is considered. Again, the minimum change required in Ci.x to make (Pi.x,Pi.y) point to Si is computed and stored as D2. Lastly, the 3x3 sub-matrix is analysed. Here, change in Ci.x and Ci.y in order for (Pi.x, Pi.y) to refer to Si is computed and the total change required stored as D3x + D3y = D3. From these 3 scenarios, the one requiring the least total absolute change in Ci.x and Ci.y is selected and the values of Ci.x and Ci.y are changed accordingly so that they point to the element containing the serial data Si to be embedded. This entire process is repeated for all 3 pairs C1, C2 and C3 obtained by taking 2 pixels at a time from the cover image and pairing the RGB components, till no more data is left to embed. The first 10 pixels in the image are used to store the total length of the data that is embedded in the image.

Pseudo code for finding change required in column Pi.x:

```
FOR i from 0 to N in steps of 1 DO
        if (sudoku(Pi.x, i) EQUALS Si)
                D1 = i
                break out of loop
        end if
end for
D1 = D1 - Pi.x

if ( |D1| > (N - |D1|) )
        D1 = (N - |D1|) * (-D1/|D1|)
end if
```

Note - Suppose Ci.x is 28 and N = 9. Pi.x (= 28 mod 9) = 1. If Pi.x needs to be changed to 8, we can increase Ci.x by 7 to 35. Now Pi.x (= 35 mod 9) = 8. Or we can decrease Ci.x by (9 - 7 =) 2 to 26. Even now, Pi.x (= 26 mod 9) = 8. However, the distortion in pixel value will be less if the change is less in this case. The above if condition makes use of this property of the mod operator to ensure that the value of the pixel is changed by the minimum amount required.

Pseudo code for finding change required in row Pi.y:

```
FOR i from 0 to N in steps of 1 DO
        if (sudoku(i, Pi.y) EQUALS Si)
                D2 = i
                break out of loop
        end if
end for
D2 = D2 - Pi.y

if ( |D2| > (N - |D2|) )
        D2 = (N - |D2|) * (-D2/|D2|)
end if
```

Note - Same reason as that stated above.

Pseudo code for finding change required in box containing (Pi.x, Pi.y):

```
boxX1 = x-lower bound of box
boxY1 = y-lower bound of box
boxX2 = x-upper bound of box
boxY2 = y-upper bound of box
FOR i from boxX1 to boxX2 in steps of 1 DO
        FOR j from boxY1 to boxY2 in steps of 1 DO
                if (sudoku(i, j) EQUALS Si)
                        D3x = i
                        D3y = j
                        break out of both loops
                end if
        end for
end for
D3x = D3x - Pi.x
D3y = D3y - Pi.y
D3 = |D3x| + |D3y|
```

3.3 Data Extraction Procedure

The receiver passes the hard NxN sudoku it gets from the sender, along with the seed value obtained from the sender, to the sudoku solver algorithm. The sudoku solver algorithm selects one of the multiple solutions for the hard sudoku based on the seed value. This solution is used as the NxN reference matrix for extraction of data. This reference matrix is the same as the one used while embedding the data at the sender end. Two pixels from the cover image with hidden data are chosen and the RGB components are paired as follows: C1(R1,G1), C2(B1, R2), C3(G2, B2). This can be generalised as Ci(x, y). Pi(x, y) is computed as shown below:

$$Pi.x = Ci.x \% N; Pi.y = Ci.y \% N$$

DOI 10.17148/IJARCCE.2015.46112

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

The NxN reference matrix is considered to be an XY grid with 0,0 pointing to the top left element and N-1,N-1 referring to the bottom right element. Therefore (Pi.x, Pi.y) refers to an element in the matrix. This element is the hidden data element Si. The process is repeated for all 3 pairs C1, C2 and C3 obtained by taking 2 pixels at a time from the cover image and pairing the RGB components. The first 10 pixels of the cover image represent the length of data stored in the image.

4. SECURITY COMPARISON

Improvement in the security of image steganography using hard sudoku as opposed to the same using soft sudoku is discussed here. A sudoku which is not completely filled is called a hard sudoku. Soft sudoku is a sudoku in which all the cells are filled with a number.

In the work by Sanmitra Ijeri et. al. [3], an image containing a completely solved sudoku or soft sudoku, which is to be used as reference matrix by the receiver, is sent along with the cover image containing the secret data. If a third party looking for secret data in images comes across this completely solved sudoku, it may raise suspicion. The third party may then proceed to analyse the image containing the secret data, and successfully extract the embedded data from it using the complete sudoku in their possession as the reference matrix.

Now the hard sudoku image steganography technique proposed in this report is considered. Here, on the senders end, a sudoku generation algorithm is used to generate a random fully solved NxN sudoku, S. All but N numbers in the sudoku are then removed and we get an unsolved sudoku or hard sudoku S2. Since only N numbers are filled in S2, it will always have multiple solutions. A sudoku solver algorithm is now used to solve this sudoku, based on a seed value. The seed value is a predetermined value such as the creation time of the cover image. Depending on the seed value, one of the many solutions of S2 is picked. Let the solution picked be S3. Unlike in the case of Sanmitra Ijeri et. al. [3], the complete sudoku S3 is not sent to the receiver. Instead the hard sudoku S2 is sent. This can be sent inconspicuously, as a daily email claiming to contain just another sudoku puzzle for the day, or by embedding in another image like was done by Sanmitra Ijeri et. al. [3]. Since the seed value is predetermined to be something like the creation time of the cover image, or the time the email is received, it need not be communicated every time. The receiver takes the sudoku S2 and runs in through the same sudoku solver algorithm that was used on the sender side, using the seed value that was either predetermined or communicated to the receiver. The solver generates the same sudoku S3 that the sender had used as reference matrix for embedding secret data. This sudoku is then used to extract data from the cover image.

Therefore, one must possess the unsolved sudoku S2, the seed value, and the sudoku solver algorithm used by the sender in order to be able to generate the required sudoku S3 and extract the embedded data from the cover image. Sending this sudoku from sender to receiver can also be done in as inconspicuous way, as it looks only like a sudoku puzzle that the sender wants to share with the receiver. Thus the overall security of the image steganography technique using sudoku is improved.

5. COMPARISON BETWEEN DIFFERENT SUDOKU SIZES

Here, the generalised image steganography using NxN sudoku is analysed for different values of N. The payload capacity, execution time, and distortion in cover image is compared for 4x4, 9x9, 16x16 and 25x25 reference sudoku matrices. As the secret data to be sent is compressed and encrypted just like is done in the case of Sanmitra Ijeri et. al. [3], the data is made more secured and also the overall size of the data is reduced. This compressed data is then converted to a serial sequence of Base-N numbers, i.e., numbers between 0 and N-1, by taking √N bits at a time. This sequence is of the form S1, S2, S3, etc.

5.1 Payload Capacity

For every two pixels, we get 3 pairs of RGB components that can be used for embedding data. Therefore 3 numbers from the serial sequence of data to be hidden can be embedded for every two pixels. In other words, the payload capacity is 1.5 Base-N numbers per pixel. This translates to 3, 4.5, 6 and 7.5 bits per pixel for a 4x4, 9x9, 16x16 and 25x25 reference sudoku respectively. This is shown graphically in figure 5.1. The graph was constructed using the values from Table 5.1.

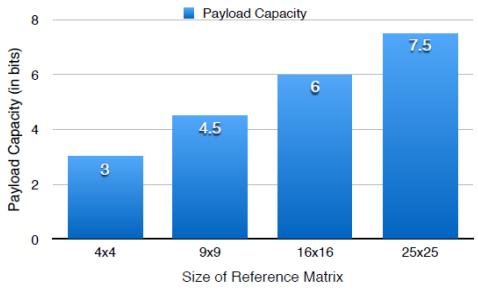

Figure 5.1: Payload Capacity for Varying Reference Matrix Sizes

Reference Matrix Size	Payload Capacity (in bits)
4x4	3
9x9	4.5
16x16	6
25x25	7.5

Table 5.1: Payload Capacity for Varying Reference Matrix Sizes

DOI 10.17148/IJARCCE.2015.46112

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

5.2 Execution Time

In this sub section, the execution time for embedding and extracting data for different sizes of reference matrix is analysed. This entire sub section uses a 1080x675 image for performing the required experiments. First we consider the case where the same number of pixels are altered in the image for different values of N, i.e., the total count of Base-N numbers to embed is the same. Figure 5.2 shows the execution time for embedding 57600 Base-N numbers for the corresponding NxN reference matrix used, or in other words the execution time for altering 38400 pixels as 2 pixels are altered for every 3 Base-N numbers. The reading are tabulated in Table 5.2.

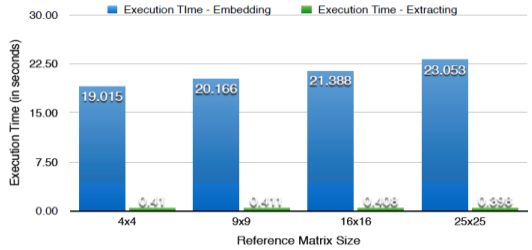

Figure 5.2: Execution Time for Embedding and Extracting 57600 Base-N Numbers

Reference Matrix Size	Embedding Time (in s)	Extraction Time (in s)
4x4	19.014651	0.410192
9x9	20.166478	0.410773
16x16	21.388300	0.407566
25x25	23.053394	0.398282

Table 5.2: Execution Time for Embedding and Extracting 57600 Base-N Numbers

The execution time for extracting data is almost the same in all cases because the algorithm executes the same number of iterations to extract the same number of Base-N numbers in each case. It is also clear from the above graph that as the size of the reference matrix increases, the execution time for embedding data increases. This is because the row, column and box in the case of a 25x25 matrixes larger than in the case of a 4x4 box, meaning a slight increase in complexity for checking the row, column and matrix as value of N increases. However, this measure is a bit misleading. Although execution time is increasing by a small fraction, the data being embed is also increasing as N increases as every Base-N number represents √N bits. This is because 57600 Base-4 numbers represent 115200 bits, whereas 57600 Base-25 numbers represent 288000 bits. A more accurate measure as to whether the efficiency of the algorithm increases or decreases with increasing N is obtained by comparing execution time for different values of N keeping the number of bits to be embedded a constant. Figure 5.3 and Table 5.3 show the execution time for increasing N, for embedding 230400 bits. This translates to 115200 Base-4 numbers, 76800 Base-9 numbers, 57600 Base-16 numbers and 46080 Base-25 numbers.

The execution time decreases as the size of reference matrix increases for a fixed number of bits to be embedded. This is because as the value of N increases, more bits can be represented by fewer Base-N numbers which means fewer pixels need to be altered, and the algorithm runs for fewer iterations overall.

The execution time for extraction of data also decreases. This is because fewer Base-N numbers need to be extracted from the cover image as value of N increases, for a fixed number of data bits.

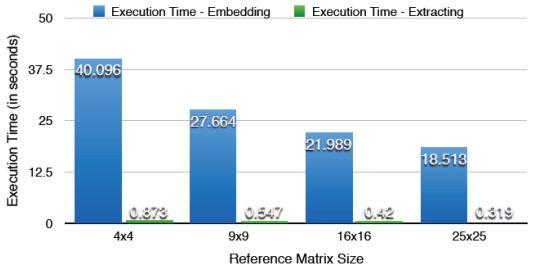

Figure 5.3: Execution Time for Embedding and Extracting 230400 Bits

Reference Matrix Size	Embedding Time (in s)	Extraction Time (in s)
4x4	40.095632	0.872564
9x9	27.664286	0.546550
16x16	21.989280	0.419970
25x25	18.513389	0.318936

Table 5.3: Execution Time for Embedding and Extracting 230400 Bits

5.3 PSNR

The distortion in the cover image depends upon the change in the value of pixels used for embedding data, as well as the total number of pixels of the cover image that are used for embedding data. This in turn depends upon the number of components of the pixel used and amount of input data. As the size of the reference matrix increases, the change in value of each pixel used for embedding increases. However, the number of pixels that are altered reduces because fewer number of pixels can now store more data. The Peak Signal-to-Noise Ratio (PSNR) value helps determine distortion in an image and is calculated as follows,

$$MSE = \frac{1}{3 \times M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (R_{ij} - R_{ij}')^2 + (G_{ij} - G_{ij}')^2 + (B_{ij} - B_{ij}')^2$$

$$PSNR = 10 \times \log_{10}(255^2/MSE) \text{ dB}$$

where MSE is the mean square error between the stego image and original image. It is defined as follows,

where R_{ij}, G_{ij}, B_{ij} nd $R_{ij}', G_{ij}', B_{ij}'$ are RGB value of original and stego images pixels respectively. M X N gives total number of pixels present in the image. A larger PSNR indicates that the the stego image is more identical to the original image. Figure 5.4 shows the PSNR value for increasing value of N. Size of the image used for experimentation purposes is 256x256. If the PSNR value is over 30 dB, the level of noise is considered to be of acceptable amount.

From figure 5.4, which was constructed using table 5.4, we infer that the PSNR value falls as the value of N

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

increases, if the percentage of image containing hidden data is kept fixed. This is because the change in value of the pixel increases as the size of the reference matrix increases. However, as the value of N increases, the total number of bits of hidden data embedded in the image also increases when the percentage of image filled with hidden data is kept the same. Figure 5.5 shows the variation in PSNR for increasing values of N when the number of bits of data hidden in the image is kept constant at 115200 bits.

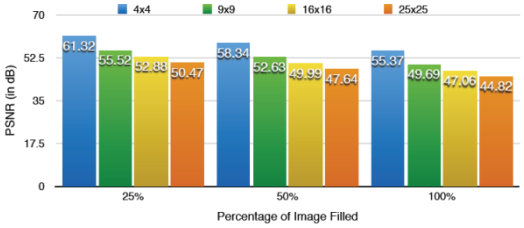

Figure 5.4: Variation in PSNR with Percentage of Image Filled for Different Values of N

Reference Matrix Size	Percentage of Image Filled	PSNR (in dB)
4x4	25	61.32
4x4	50	58.34
4x4	100	55.37
9x9	25	55.52
9x9	50	52.63
9x9	100	49.69
16x16	25	52.88
16x16	50	49.99
16x16	100	47.06
25x25	25	50.47
25x25	50	47.64
25x25	100	44.82

Table 5.4: Variation in PSNR with Percentage of Image Filled for Different Values of N

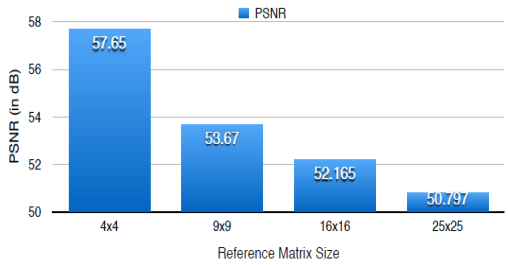

Figure 5.5: Variation in PSNR for Different Values of N - Number of Bits of Hidden Data Kept Constant

Reference Matrix Size	PSNR (in dB)
4x4	57.65
9x9	53.67
16x16	52.17
25x25	50.80

Table 5.5: Variation in PSNR for Different Values of N - Number of Bits of Hidden Data Kept Constant

It can be seen from figure 5.5 that even when total number of bits embedded is kept constant, the PSNR value decreases. Therefore, the impact of increasing change in the value of individual pixels is greater than the effect of reducing the overall number of pixels altered. Table 5.5 tabulates the values used to construct the graph in figure 5.5.

However, in all cases, the PSNR never falls below 30 dB. Therefore all the images are of acceptable quality.

5.4 Snapshots

Figure 5.6 is the original 256x256 image used. Figure 5.7, 5.8, 5.9 and 5.10 are the same image with 100% data embedded in them using a reference matrix of size 4x4, 9x9, 16x16 and 25x25 respectively.

Figure 5.6: Original Image

Figure 5.7: Original Image 100% Embedded with Data using 4x4 Reference Matrix

Figure 5.8: Original Image 100% embedded with Data using 9x9 Reference Matrix

DOI 10.17148/IJARCCE.2015.46112

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

Figure 5.9: Original Image 100% Embedded with Data using 16x16 Reference Matrix

Figure 5.10: Original Image 100% Embedded with Data using 25x25 Reference Matrix

6. CONCLUSION

An image steganography algorithm based on sudoku, that is generalised for any NxN reference matrix where N is a perfect square is developed. The security of the algorithm is improved by making use of hard sudoku instead of soft sudoku. For this, sudoku generation and solver algorithms are detailed.

Instead of just one key in the case of soft sudoku approach, which was the sudoku itself, the hard sudoku approach makes use of three keys - the hard sudoku, the seed value, and the solver algorithm. One must possess all three keys in order to obtain the hidden data from the stego image quickly.

The payload capacity for different sizes of reference matrices is compared. It is found that the payload capacity increases as the order of reference matrix increases. The time required to embed and extract data decreases with increase in size of reference matrix. These two observations are favourable as payload capacity is increasing an execution time is decreasing. The PSNR value on the other hand decreases with increase in size of reference matrix. However, it still does not fall below 30 dB. This level of PSNR is acceptable for most applications. For applications that require a higher PSNR value, a balance needs to be found between payload capacity and PSNR value, and the reference matrix size adjusted accordingly.

REFERENCES

[1] Roshan Shetty, B. R., J. Rohith, V. Mukund, Rohan Honwade, and Shanta Rangaswamy. "Steganography using Sudoku puzzle." In IEEE International Conference on Advances in Recent Technologies in Communication and Computing, 2009, pp. 623-626

[2] Chang, Chin-Chen, Yung-Chen Chou, and The Duc Kieu. "An information hiding scheme using sudoku.", In 3rd IEEE International Conference on Innovative Computing Information and Control, 2008, pp. 17-17

[3] Ijeri, Sanmitra, Shivananda Pujeri, B. Shrikant, and B. A. Usha. "Image steganography using Sudoku puzzle for secured data transmission.", International Journal of Computer Applications, Vol. 48, Issue 17, 2012, pp. 31-35

[4] C.-C. Chang, T. D. Kieu, and Y.-C. Chou. High capacity data hiding for gray scale images. In Proceedings of the First International Conference on Ubiquitous Information Management and Communication, pages 139–148. Seoul, Korea, February 2007.

[5] C.-C. Chang and C.-Y. Lin. Reversible steganography for vq-compressed images using side matching and relocation. IEEE Transactions on Information Forensics and Security, 1(4):493–501, 2006.

[6] Y.-T. Wu and F. Y. Shih. Digital watermarking based on chaotic map and reference register. Pattern Recognition, 40(12):3754–3763, December 2007.

[7] Yung-Chen Chou, Chih-Hung Lin, Pao-Ching Li, Yu-Chiang Li A (2, 3) Threshold Secret Sharing Scheme Using Sudoku 2010 Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing 978-0-7695-4222-5/10, 2010

[8] Wien Hong, Tung-Shou Chen, Chih-Wei Shiu, Steganography Using Sudoku Revisited Second International Symposium on Intelligent Information Technology Application 978-0-7695-3497-8/08, 2008