Image Encryption using the Sudoku Matrix

Yue Wu[a] ,Yicong Zhou[a], Joseph P. Noonan[a], Karen Panetta[a], Sos Agaian[b]
[a] Department of Electrical and Computer Engineering, Tufts University, Medford, MA 02155
[b] Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249

ABSTRACT

This paper introduces a new effective and lossless image encryption algorithm using a Sudoku Matrix to scramble and encrypt the image. The new algorithm encrypts an image through a three stage process. In the first stage, a reference Sudoku matrix is generated as the foundation for the encryption and scrambling processes. The image pixels' intensities are then changed by using the reference Sudoku matrix values, and then the pixels' positions are shuffled using the Sudoku matrix as a mapping process. The advantages of this method is useful for efficiently encrypting a variety of digital images, such as binary images, gray images, and RGB images without any quality loss. The security keys of the presented algorithm are the combination of the parameters in a 1D chaotic logistic map, a parameter to control the size of Sudoku Matrix and the number of iteration times desired for scrambling. The possible security key space is extremely large. The principles of the presented scheme could be applied to provide security for a variety of systems including image, audio and video systems.

Keywords: image encryption, Sudoku, scrambling, logistic map

1. INTRODUCTION

In the past half century, computers and network technology have become prevalent in all aspects of our daily lives. Email, remote video conference, online music and movies and other applications are now commonplace in this digital age. Online personal albums and data storage are widely used and are easily accessed and shared across the internet. In order to improve the quality of service and reduce record storage costs, many medical institutions and hospitals digitize and stored diagnostic images and/or medical records of their patients. This information can be shared, and transmitted over networks from the laboratories to medical centers or to a doctor's office. However, the privacy of these images and data is susceptible for unauthorized used and might be disclosed to some unauthorized individuals. Therefore, digital images should be encrypted before they are sent over network.

Many image scrambling and image encryption algorithms have been developed based on different principles. Generally these algorithms can be divided into two categories based on the domain of encryption works, namely the transform domain and the spatial domain. In the transform domain encryption, the image first is transformed to some frequency domain, and then encryption is applied and finally transformed back to a spatial domain image. Dang et al. proposed his approach based on the Discrete Wavelet Transform[1]. Hui et al. gave an encryption algorithm with fractional discrete cosine and sine transform[2]. However, in spatial domain encryption, the algorithm directly modifies image pixels. Fridrich et al. employed a 2D chaotic baker map[3]. Zou et al. applied a classical Fibonacci number to scramble image in spatial domain[4]. Zhang et al. used discrete Chebyshev chaotic sequences[5].

Sudoku puzzles and their variants have become extremely popular in the last decade, and can now be found daily in most major U.S. newspapers[6]. Sudoku was popularized in 1986 by the Japanese puzzle company Nikoli, under the name Sudoku, meaning "single number"[7]. A standard Sudoku is a logic-based, combinatorial number-placement puzzle consisting of a 9 by 9 grids. The total number of different Sudoku solutions was 6.67×10^{21}[8].

In the past, efforts have focused on how to quickly generate, solve and rate the Sudoku puzzle [9-11]. The Sudoku puzzle/matrix has been used for image security (data hiding and encryption) recently. Shirali-Shahreze et al. applied Sudoku solutions for encrypting short message service (SMS)[12]. They used a 9 by 9 Sudoku matrix and hid data in one row or one column of the puzzle. Unless the user solved the puzzle and knew the exact row number or column number, one cannot retrieve the correct digit sequence. Chang et al. used a selected Sudoku solution as a reference matrix to

Mobile Multimedia/Image Processing, Security, and Applications 2010, edited by Sos S. Agaian, Sabah A. Jassim,
Proc. of SPIE Vol. 7708, 77080P · © 2010 SPIE · CCC code: 0277-786X/10/$18 · doi: 10.1117/12.853197

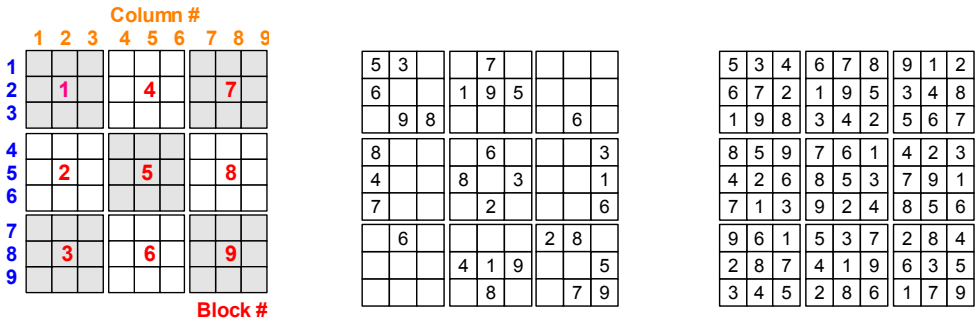
guide cover pixels' modification for embedding secret data as a modification on Mielikainen's Least Significant Bit(LSB) matching method[13]. Later, Hong et al [14, 15] improved Chang's approach by applying the searching idea of minimal Euclidean distance.

In this paper, we introduce a new image encryption scheme based on the Sudoku Matrix. Instead of using an unfinished Sudoku Puzzle, which is employed by previous Sudoku based encryption algorithm, we used the full solution to a Sudoku Puzzle, i.e. a Sudoku matrix to encrypt the image directly. In addition, we broadened the conception of the Sudoku matrix from 9 by 9 to any N by N matrix, where N is some square number. Our algorithm also employs a 1D Chaotic Logistic Map to generate a random-like Sudoku Matrix and it is used as our reference matrix. By changing the pixels' values according to the Sudoku reference matrix, the histogram after encryption is dramatically changed compared to the original one. Furthermore, with the property of the Sudoku matrix that no two digits in the same block can be aligned in the same row, column or box, the input image can be scrambled to a desired output. Therefore, no two pixels originally in the same block will be in the same row, or the same column or the same box in the output.

The presented algorithm can be used to encrypt other types of images such as color images, gray images, binary images and etc. The security key is selective and has a very large number space. The rest of the paper is organized as follows: in Section 2, the proposed encryption and decryption algorithms are discussed in details; in Section 3 some experimental results are displayed and security analysis is given; in Section 4 conclusion remarks are drawn in Section 5.

Background

Here we define a Sudoku Matrix as an X by X matrix with the numbers from 1 to N with the constraints that X is a square number and N = X, such that each number occurs exactly once in each row, exactly once in each column and exactly once in each block. The following Figure 1 shows an example of a Sudoku puzzle and its solution when X = 9. We call the solution of the Sudoku puzzle a Sudoku matrix.



(a) (b) (c)

Figure 1. A sample Sudoku puzzle and its solution
(a) Row#, Column # and Block# notation; (b) A sample Sudoku puzzle; (c) The solution to Sudoku puzzle (b)

Sudoku puzzles are normally generated from the Sudoku matrix by removing some elements but keeping some hints for unique solution. Researchers made efforts on generating many different ways[9, 11]. In this paper we employed the most straight-forward method, namely, the Latin square method, to generate a square size Sudoku matrix. The trade-off of this fast and systematic method is that the set of Sudoku matrices generated by this method is a subset of the universal set of all possible Sudoku matrices. A more detailed discussion is made in later sections.

A Latin square is a *Y* by *Y* table filled with Y different symbols in such a way that each symbol occurs exactly once in each row and exactly once in each column. An example of Latin squares is shown in Figure 2. Since the Latin Square does not care about the repetition in blocks, it could be considered as a simplified version of the Sudoku matrix. In other words, a completed Sudoku grid is a special type of Latin square with the additional property that there be no repeated values in any of the 9 blocks of contiguous 3×3 cells[16].

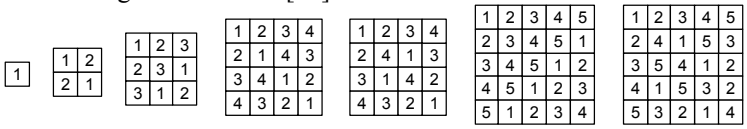


Figure 2. Samples of Latin squares

2. IMAGE ENCRYPTION USING THE SUDOKU MATRIX

In this section we present a novel encryption scheme using the Sudoku Matrix. The basic steps are provided in Figure 3.

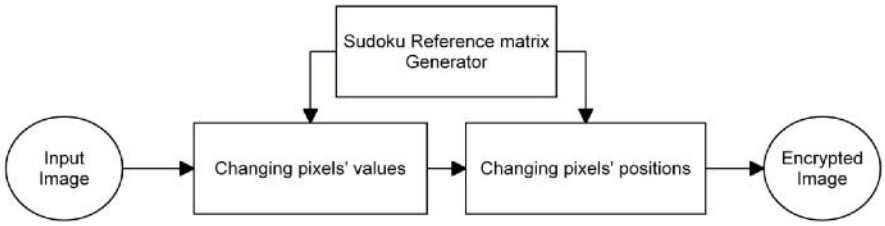


Figure 3: Block diagram of the proposed Image Encryption Schema

The entire algorithm could be divided into three main stages: generating the Sudoku Reference Matrix, changing Pixels' values and changing pixels' positions. The whole encryption process can be described as follows: first, a Sudoku Reference Matrix **Ref** is generated and then the input image is resized to match the size of **Ref**. Then, the input image's pixel values are changed according to the **Ref** matrix. Finally, the input image's pixel positions are also shuffled according to the **Ref** matrix.

The decryption process is simply the reverse of the encryption process. A security KEY is used to generate Sudoku Reference Matrix **Ref**. We decrypt an encrypted image by first changing the positions of image pixels and then changing pixels' value according to the **Ref** matrix. Finally we obtain the decrypted image as the output.

In the remainder of this section, we will describe the each processing stage as depicted in the block diagram of Figure 3.

Generating the Sudoku Reference matrix: the flow chart of the Sudoku Reference matrix Generator

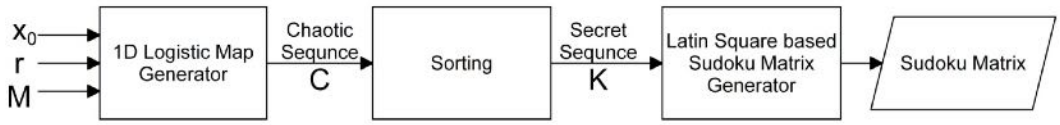


Figure 4: Flow chart of the Sudoku Reference matrix Generator

Generating a Chaotic Sequence C using a Logistic Map

In the inputs (x_0, r) determines the initial value for the logistic map while M determines the length of the Chaotic Sequence **C**, where $Length(C) = N = M^2$. The definition of a discrete Logistic map is as follows

$$x_{n+1} = r \cdot x_n \cdot (1 - x_n) \tag{1}$$

From (1), it is clear to see that once $(\boldsymbol{x_0}, \boldsymbol{r})$ are given, the whole chaotic sequence is determined.

Generating a Secret Sequence K by the sorting sequence C

The Secret Sequence **K** is generated from the order of the indices from a sorted Chaotic Sequence **C**. The Chaotic Sequence $\mathbf{C} = [C_1, C_2, \ldots, C_N,]$ and then after sorting **C** according to some certain order (without loss generality, the ascendant order is used here), $\mathbf{C} = [C_{k1}, C_{k2}, \ldots, C_{kN},]$. Finally, the Secret Sequence $\mathbf{K} = [k1, k2, \ldots, kN]$ is obtained.

Generating the Sudoku Matrix

First of all, we present the algorithm for generating a Latin square using a ring shift method. The input is a sequence of numbers **B** with length *M* and the output is a Latin square **L** with size *M* by *M*.

Algorithm I: Generating the Latin Square using Ring shift method

```
if (Row# = 1)
  { L[Row#1] = B}
for (Row# = 2: M)
  {L[Row#m]= ring_shift {L[Row#(m-1)]}}
```

Where the function of 'ring_shift' is to always and only move the 1st element to the end.

Suppose the Secret Sequence is **K** with length N (Note: $N = M^2$). Then for generating an N by N Sudoku matrix using Latin Squares can be described as following algorithm:

Algorithm II: Generating the Sudoku Matrix using Latin Squares

1. Generate a size M by M Latin square $\mathbf{L_{seed}}$ using ring shift method with sequence **B**, where B = [1, 2, .., M]
2. Group the secret sequence **K** into M subgroups (each subgroup has M elements) and generate the ith sub-Latin square for the ith subgroup.
3. For each element in $\mathbf{L_{seed}}$, substitute itself with its corresponding sub Latin square and generate **L**.
4. Resample the matrix of **L** for every M rows and get its corresponding Sudoku matrix **S**.

A concrete example of generating a Sudoku Reference Matrix

Suppose $(x_0, r) = (0.4, 3.8)$ and $M = 3$, $N = 9$. Then the chaotic sequence **C** yielded is [0.4000, 0.9120, 0.3050, 0.5954, 0.2943, 0.7892, 0.6322]. After sorting according to be in ascending order, $\mathbf{C} = [C_7, C_3, C_1, C_5, C_9, C_8, C_4, C_2 C_6]$. Therefore, the Secret Sequence **K** is [7, 3, 1, 5, 9, 8, 4, 2, 6]. In the next step, this Secret Sequence **K** should be used for generating a Sudoku reference matrix as stated in section 2.2.4. Here **K** = [7, 3, 1, 5, 9, 8, 4, 2, 6], and the size of expected Sudoku matrix is 9 by 9, i.e. $N = 9$. First, we generate a Latin square with $M = 3$ and **B** = [1, 2, 3] as Algorithm II step 1 says.

The result we get is (2):

$$\mathbf{L_{seed}} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \tag{2}$$

In the following step, we shall divide **K** into 3 subgroups, and they are subgroup#1=[7, 3, 1] , subgroup#2=[5, 9, 8], subgroup#3=[4, 2, 6]. Each of subgroup is used to generate its corresponding sub Latin square as (3) shows:

$$\text{Sub Latin square\#1} = \begin{bmatrix} 7 & 3 & 1 \\ 3 & 1 & 7 \\ 1 & 7 & 3 \end{bmatrix}, \text{Sub Latin square\#2} = \begin{bmatrix} 5 & 9 & 8 \\ 9 & 8 & 5 \\ 8 & 5 & 9 \end{bmatrix}, \text{Sub Latin square\#3} = \begin{bmatrix} 4 & 2 & 6 \\ 2 & 6 & 4 \\ 6 & 4 & 2 \end{bmatrix} \tag{3}$$

In the third step, we substitute the corresponding elements in $\mathbf{L_{seed}}$ with the sub Latin squares and we get:

$$\mathbf{L} = \begin{bmatrix} \text{Sub Latin square\#1} & \text{Sub Latin square\#2} & \text{Sub Latin square\#3} \\ \text{Sub Latin square\#2} & \text{Sub Latin square\#3} & \text{Sub Latin square\#1} \\ \text{Sub Latin square\#3} & \text{Sub Latin square\#1} & \text{Sub Latin square\#2} \end{bmatrix} \tag{4}$$

Equivalently

$$\mathbf{L} = \begin{bmatrix} \begin{bmatrix} 7 & 3 & 1 \\ 3 & 1 & 7 \\ 1 & 7 & 3 \end{bmatrix} & \begin{bmatrix} 5 & 9 & 8 \\ 9 & 8 & 5 \\ 8 & 5 & 9 \end{bmatrix} & \begin{bmatrix} 4 & 2 & 6 \\ 2 & 6 & 4 \\ 6 & 4 & 2 \end{bmatrix} \\ \begin{bmatrix} 5 & 9 & 8 \\ 9 & 8 & 5 \\ 8 & 5 & 9 \end{bmatrix} & \begin{bmatrix} 4 & 2 & 6 \\ 2 & 6 & 4 \\ 6 & 4 & 2 \end{bmatrix} & \begin{bmatrix} 7 & 3 & 1 \\ 3 & 1 & 7 \\ 1 & 7 & 3 \end{bmatrix} \\ \begin{bmatrix} 4 & 2 & 6 \\ 2 & 6 & 4 \\ 6 & 4 & 2 \end{bmatrix} & \begin{bmatrix} 7 & 3 & 1 \\ 3 & 1 & 7 \\ 1 & 7 & 3 \end{bmatrix} & \begin{bmatrix} 5 & 9 & 8 \\ 9 & 8 & 5 \\ 8 & 5 & 9 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 7 & 3 & 1 & 5 & 9 & 8 & 4 & 2 & 6 \\ 3 & 1 & 7 & 9 & 8 & 5 & 2 & 6 & 4 \\ 1 & 7 & 3 & 8 & 5 & 9 & 6 & 4 & 2 \\ 5 & 9 & 8 & 4 & 2 & 6 & 7 & 3 & 1 \\ 9 & 8 & 5 & 2 & 6 & 4 & 3 & 1 & 7 \\ 8 & 5 & 9 & 6 & 4 & 2 & 1 & 7 & 3 \\ 4 & 2 & 6 & 7 & 3 & 1 & 5 & 9 & 8 \\ 2 & 6 & 4 & 3 & 1 & 7 & 9 & 8 & 5 \\ 6 & 4 & 2 & 1 & 7 & 3 & 8 & 5 & 9 \end{bmatrix} \tag{5}$$

Finally, we shall resample L for every $M = 3$ rows.

$$\mathbf{L} = [\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{R4}, \mathrm{R5}, \mathrm{R6}, \mathrm{R7}, \mathrm{R8}, \mathrm{R9}]^{\mathrm{T}} \xrightarrow{\text{resample for every 3rows}} [\mathrm{R1}, \mathrm{R4}, \mathrm{R7}, \mathrm{R2}, \mathrm{R5}, \mathrm{R8}, \mathrm{R3}, \mathrm{R6}, \mathrm{R9}]^{\mathrm{T}} = \mathbf{S} \tag{6}$$

Therefore,

$$\mathbf{S} = \left[\begin{bmatrix} 7 & 3 & 1 \\ 5 & 9 & 8 \\ 4 & 2 & 6 \end{bmatrix} \begin{bmatrix} 5 & 9 & 8 \\ 4 & 2 & 6 \\ 7 & 3 & 1 \end{bmatrix} \begin{bmatrix} 4 & 2 & 6 \\ 7 & 3 & 1 \\ 5 & 9 & 8 \end{bmatrix}\right. \\ \left.\begin{bmatrix} 3 & 1 & 7 \\ 9 & 8 & 5 \\ 2 & 6 & 4 \end{bmatrix} \begin{bmatrix} 9 & 8 & 5 \\ 2 & 6 & 4 \\ 3 & 1 & 7 \end{bmatrix} \begin{bmatrix} 2 & 6 & 4 \\ 3 & 1 & 7 \\ 9 & 8 & 5 \end{bmatrix}\right. \\ \left.\begin{bmatrix} 1 & 7 & 3 \\ 8 & 5 & 9 \\ 6 & 4 & 2 \end{bmatrix} \begin{bmatrix} 8 & 5 & 9 \\ 6 & 4 & 2 \\ 1 & 7 & 3 \end{bmatrix} \begin{bmatrix} 6 & 4 & 2 \\ 1 & 7 & 3 \\ 8 & 5 & 9 \end{bmatrix}\right] \tag{7}$$

It is easy to verify that in **S**, in each row and each column and each block, each number occurs exactly once and hence **S** is a Sudoku matrix, which we named **Ref** and is used as a reference matrix in the further processing.

Changing image data values using a Sudoku Matrix

Pixels' intensities or values in a digital image carry abundant information. For example, the brightness and contrast are closely related to the pixels' intensity in an image, and they are crucial for identifying the objects. Therefore, we can encrypt a digital image by changing its data values. In reality, it is expected to see a very different histogram of the encrypted image from the original one. And the ideal output histogram follows a uniform distribution.

The process of changing image data values using a Sudoku Matrix is shown in Figure 5. Firstly, the Sudoku **Ref** matrix subtracts 1 such that its possible values become [0, 1, 2, …, N-1] and then it is rescaled, such that the numbers [1, 2, …, N] now are replaced by [0, 255/(N+1),…, 255*(N-1)/(N+1)] respectively.

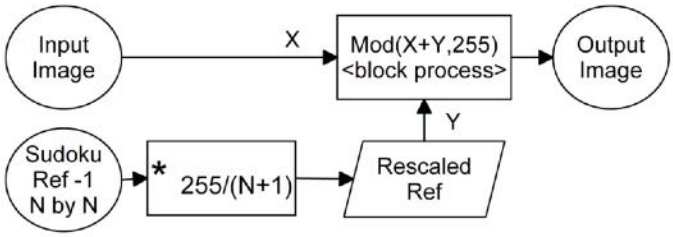
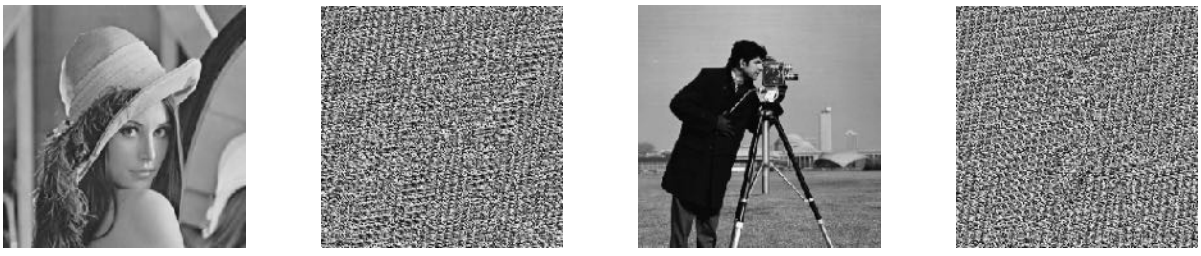


Figure 5: Flow chart of changing pixel values using the Sudoku Reference Matrix

The original image is padded/resized to fit the size of the Sudoku **Ref** such that the new image could be divided exactly *R* integer blocks. This new image will be used for the future processing. Later on, the input image is processed block by block. For each block, the operation of Mod(X+Y,255) is applied, where X is some selected block in the input image and Y is the rescaled **Ref** matrix. Figure 6 shows the differences between before and after applying the proposed algorithm.



(a) (b) (c) (d)

Figure 6: Changing image pixel values using a Sudoku Reference matrix
(a) Lenna Image; (b) Image of (a) after changing pixels' value;
(c) Cameraman Image; (d) Image of (c) after changing pixels' value;

Changing image data positions using a Sudoku Matrix

A shuffling algorithm can be expressed as a mapping function $\boldsymbol{f}$, such that $\forall(x, y) \in \boldsymbol{I}$, where (x, y) is the (row, column) pair for some pixel in image $\boldsymbol{I}$, $\exists\ (r, c) = \boldsymbol{f}(x, y)$, where (r, c) is the new (row, column) pair for the input pixel after shuffling. The mapping function $\boldsymbol{f}$ has to be a one-to-one and onto function. In addition, if we do not want to change the image size after shuffling then $\boldsymbol{f}$ has to be a self-map function as well. Here the 'self-map' property guarantees the domain and the range of $\boldsymbol{f}$ are the same, 'one-to-one' and 'onto' property impose the one to one corresponding between the domain and the range.

In any Sudoku reference matrix **Ref**, the mapping relationship between (row, column) pair and (block, digit) pair is a self-map and bijective (both one-to-one and onto) function. Therefore, this relationship can be also used for shuffling image and we call this mapping a Sudoku mapping.

$$Sudoku\ Mapping\colon (x, y) \rightarrow (r, c), where\ (r, c) is\ \text{the}\ (block\#, digit\#) for\ (x, y)\ \ in\ \textbf{Ref}\ respectively \tag{8}$$

Figure 7 shows an example of using the Sudoku mapping for a 9 by 9 image. Figure 7 (a) is a sample linear image, whose pixels' values change linearly with 1 for the upper left corner and 81 for the lower right corner. Figure 7 (b) is our Sudoku reference matrix, which exactly represents Equation (7). Figure 7 (c) is the resulting image after the Sudoku mapping (Note the ranges of the colorbars in Figure 7). As we expected the linearity of original image has been disordered after shuffling.

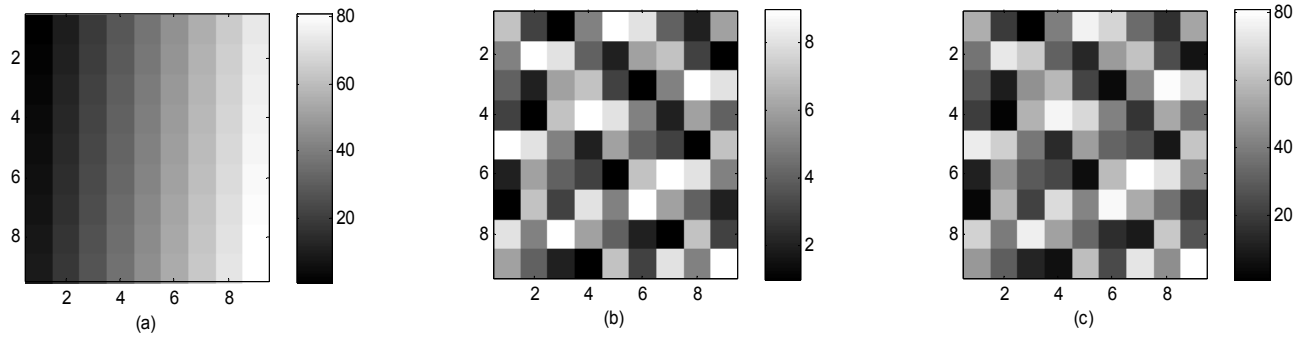


Figure 7: A Sudoku Mapping
(a) Original 9 by 9 image; (b) A Sudoku Reference Matrix; (c) Output image after scrambling according to Ref

Since the image is processed block by block, shuffling pixels only occurs within each block but not inter-blocks. In order to compensate for this drawback, one can either change to a bigger size **Ref** matrix such that the image will has fewer blocks or we can repeat the shuffling process until the degree of disorder is sufficiently large. Here, we simply define the degree of disorder $\boldsymbol{d}$ as follows. Note, in the formula, ***std*** denotes the standard deviation function:

$$\boldsymbol{d} = Degree\ of\ disorder = \boldsymbol{std}(original\ image - shuffled\ image)/\boldsymbol{std}(original\ image) \tag{10}$$

Figure 8 (a) shows the shuffling result for 'cameraman.tif', whose size 256 by 256. As we see, the bigger **Ref** we choose the larger disorder degree we achieved. Note, in (b)-(d) the shuffling algorithm is only applied once. An alternative way to improve the shuffling result is to apply the shuffling process repeatedly to the image and shift the image for several pixels. This idea is very common, since we want the pixels to be shuffled into a larger region. Although one pixel is still shuffled within its own blocks, its own blocks are different from time to time when we shift the image for each iteration. Once the action of how to shift an image for each iteration is predefined, it is possible for a pixel to be shuffled into a larger region and thus the scrambling result has been improved. The following result Figure 8 (e) (f) displays the outputs using this scheme. Here the predefined shifting actions are: when the iteration# is odd, shift the image rightwards and when it is even, shift it downwards. As we seen, the degree of disorder increases as the iteration times increases.

In this paper, the shuffling algorithm employed both improvements above. Both the **Ref** matrix and the times of iteration are contained in the encryption keyword.

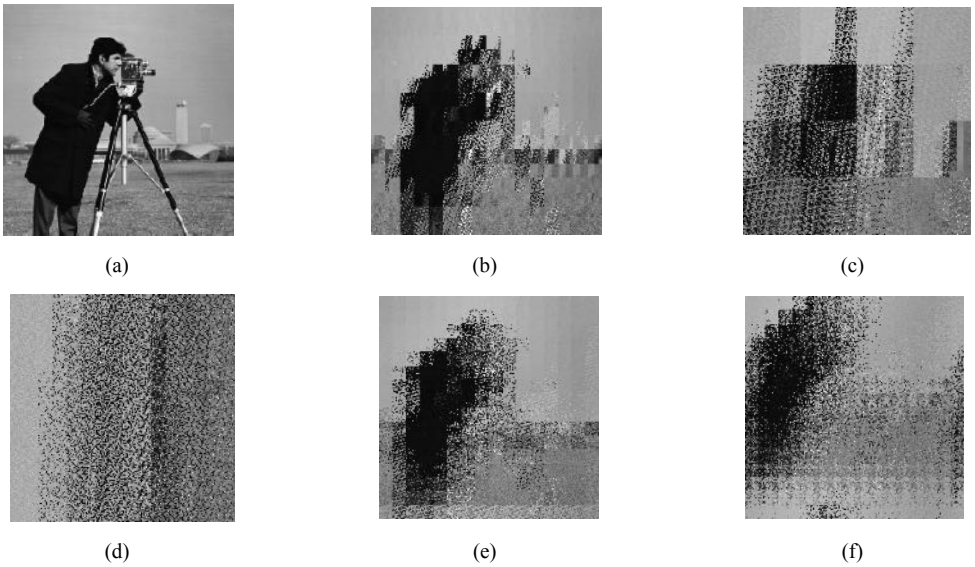


Figure 8:Shuffling image using the Sudoku Mapping with iterative process
(a) Original Image; (b) Ref size 16 by 16, #iteration = 1, d = 0.4830;
(c) Ref size 64 by 64, #iteration = 1, d = 0.6669;(d) Ref size 256 by 256, #iteration = 1, d = 0.8312
(e) Ref size 16 by 16, #iteration = 3, d = 0.6167; (f) Ref size 16 by 16, #iteration = 10, d = 0.8156

3. EXPERIMENTAL RESULTS AND SECURITY ANALYSIS

In this paper, the image processing software package (MATLAB) is used as the engine for the image processing experiments. An RGB image is stored in MATLAB as an *M*-by-*N*-by-*3* data array that defines red, green, and blue color components for each individual pixel. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location. A Gray image is stored in MATLAB as an *M*-by-*N* data array that defines gray intensities in the color plane at the pixel's location.

In our simulations, several images are used for test. They are a 512 by 512 gray Tank image a, a 364 by 256 gray Fingerprint image, 367 by 380 gray Brain MRI image and a 512 by 384 RGB Peppers image. Their results are shown in Figure 9 – 12. These results show that our algorithm is robust for different image formats and it is effective for encrypting various sensitive image data. Furthermore, we examined the histogram of the encrypted image, which appeared almost flat. In order to test the randomness of the encrypted image, we also apply the moment function analysis and correlation analysis. Table I and Table II show these results respectively.

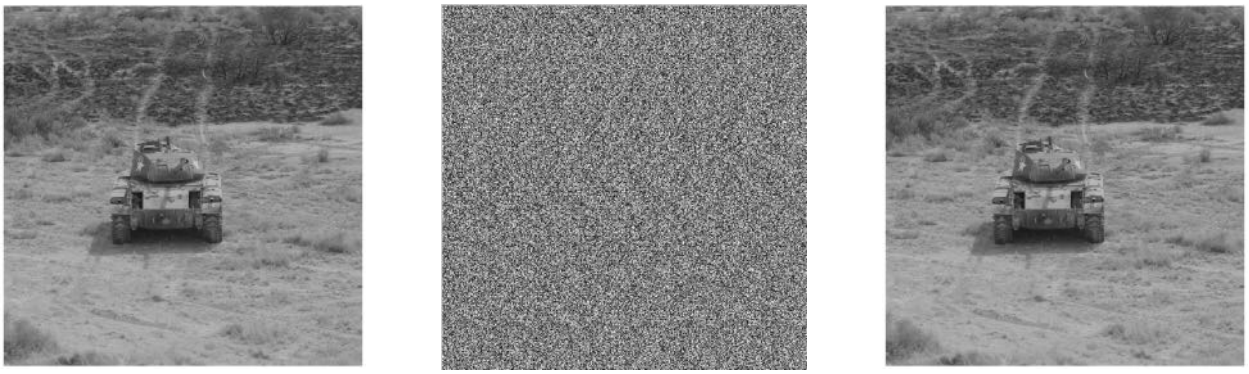
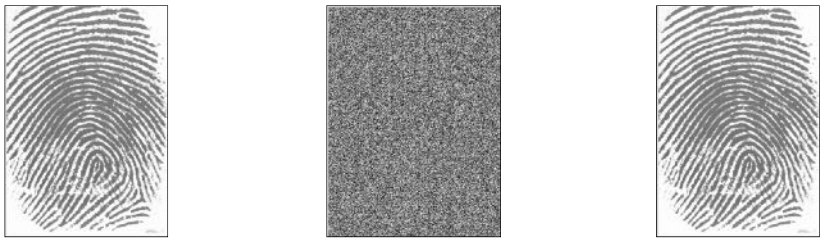
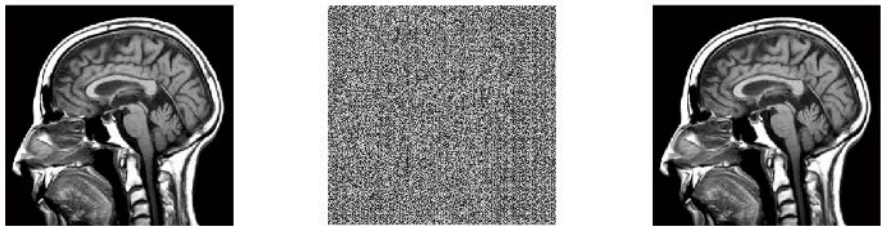


Figure 9: Experimental result of the proposed encryption and decryption algorithms – gray Tank image (**Ref** size = 64)
(a) original Tank image; (b) encrypted image; (c) decrypted image



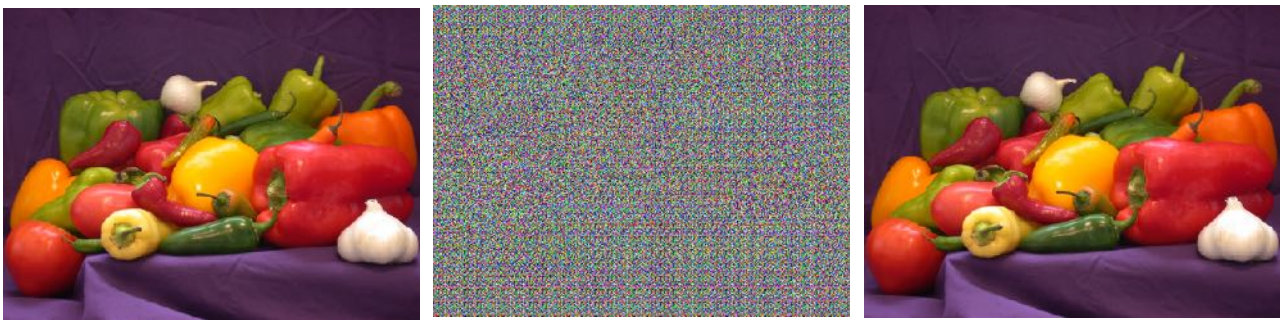
(a) (b) (c)

Figure 10: Experimental result of the proposed encryption and decryption algorithms – gray Fingerprint image (**Ref** size = 25)
(a) original Fingerprint image; (b) encrypted image; (c) decrypted image



(a) (b) (c)

Figure 11: Experimental result of the proposed encryption and decryption algorithms –gray MRI image (**Ref** size = 16)
(a) original MRI image; (b) encrypted image; (c) decrypted image



(a) (b) (c)

Figure 12: Experimental result of the proposed encryption and decryption algorithms – RGB Peppers image (**Ref** size = 16)
(a) original RGB Peppers image ; (b) encrypted image; (c) decrypted image

A good encryption algorithm is expected to resist all kinds of known attacks. In present attacks, the attacks based on statistical analysis and the key space consists of two main types. Based on these two types of attacks, we perform our security analysis.

Statistical Analysis

Using the image histogram is a very straight forward way to illustrate the confusion and diffusion properties of the encrypted image. We used three new images for this analysis. They are a 256 by 256 gray image named 'Cameraman', a 512 by 512 gray image named 'Lenna' and a 512 by 512 RGB image named 'Pepper'. In Figure 13, (a), (c) and (i) are the original images, the corresponding encrypted images are (e), (g) and (m). As we see, the histograms of encrypted images almost follow a uniform distribution and they are significantly different from the histograms of original images for both Gray and RGB images.

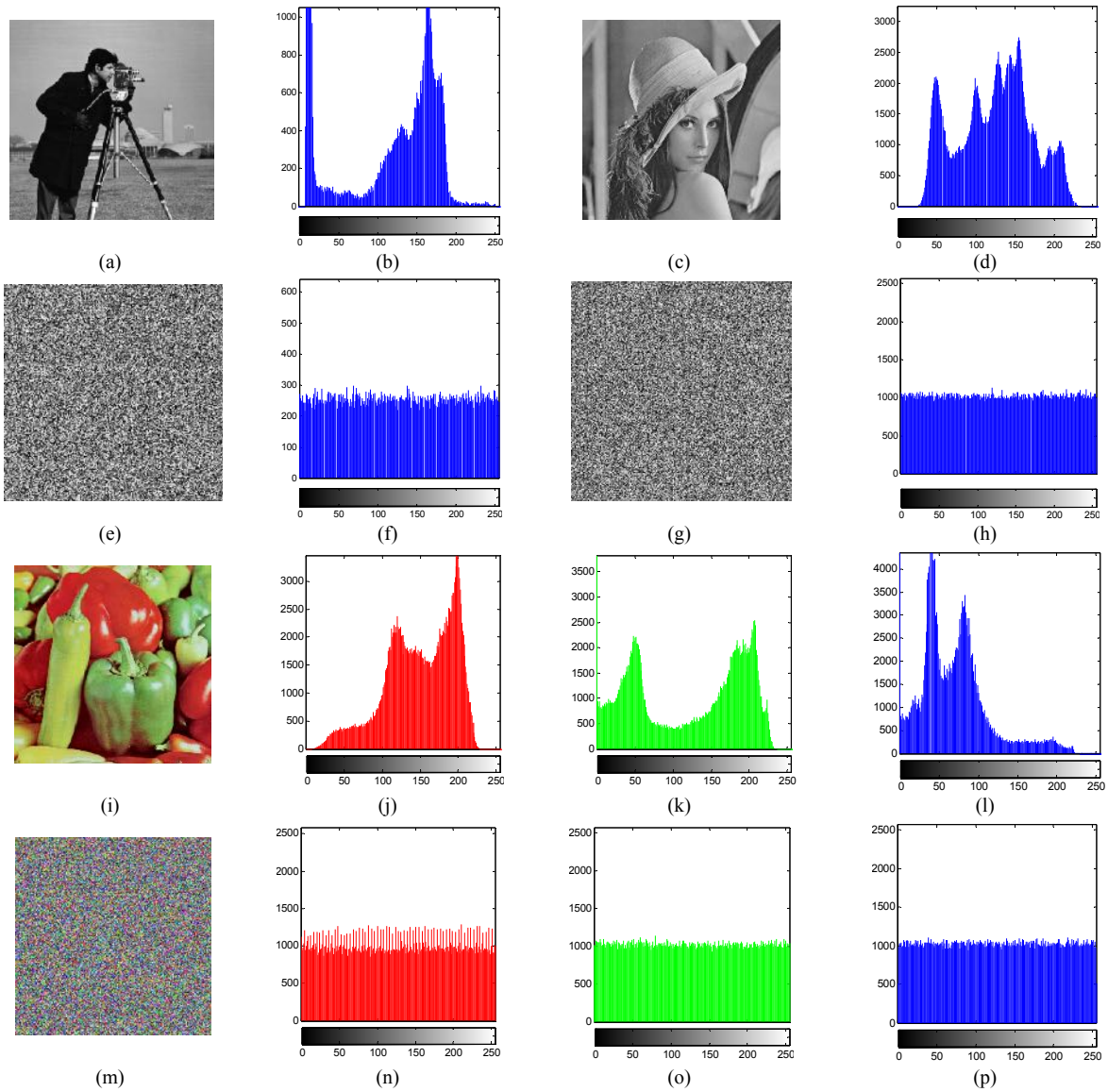


Figure 13: Histogram analysis on the images using proposed encryption and decryption algorithms
(a) The original Gray Cameraman image; (b) The histrogram of (a); (c) The original Gray Lenna image (d) The histogram image of (c)
(e) The encrypted image of (a) ; (f) The histogram of (e); (g) The encrypted image of (c); (h) The histogram of (h);
(i) The original RGB Pepper image; (j) The histogram of Red component in (i); (k) The histogram of Green component in (i);
(l) The histogram of Blue component in (i); (m) The encrypted image of (i); (n) The histogram of Red component in (m);
(o) The histogram of Blue component in (m); (p) The encrypted image of (m)

In order to simplify the analysis process, from this point we only use the Cameraman image and its encrypted image for further analysis. Now we apply the moment function analysis to Figure 13(a) and (e). In statistics the method of moments is to estimate population parameters such as mean, variance, median, skewness and kurtosis etc. In Table I, we compared four moments for an ideal uniform distributed image, the original Cameraman image and its encrypted image. From the table I, it is easy to see that the encrypted image's mean, standard deviation, skewness and kurtosis values are very close to those of the ideal uniform distributed image.

Table I: Moment Function Analysis

Moments	Ideal uniform distributed image	Original Cameraman image	The encrypted image
Mean	127.5	118.7245	127.1893
Standard Deviation	74.0450	62.3417	74.1434
Skewness	0	-0.7381	0.0056
Kurtosis	1.8000	2.0915	1.7935

In order to test the correlation between two adjacent pixels in the original image and the encrypted image, we calculated the correlation coefficients along horizontal, vertical and diagonal directions.

Table II: Correlation coefficients of two adjacent pixels in original and encrypted images

Direction	Original Cameraman Image	The Encrypted Image
Horizontal	0.9333	-0.0058
Vertical	0.9565	-0.0157
Diagonal	0.9059	0.0168

In addition, we randomly select 1024 pairs of adjacent pixels in horizontal, vertical and diagonal direction from the original image and the encrypted image. Their correlation Figures are shown in Figure 15. From the Figure, it is clear to see that the correlation between a pair of pixels in the encrypted image is much smaller than that of pixels in the original image.

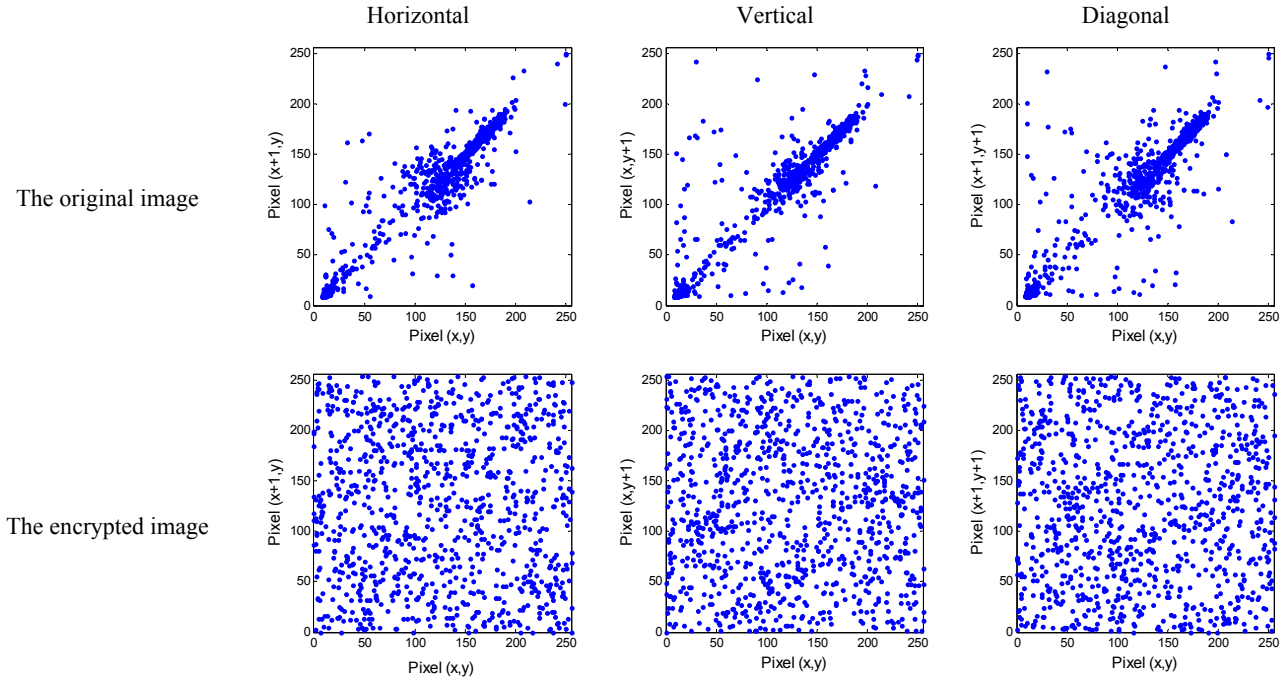
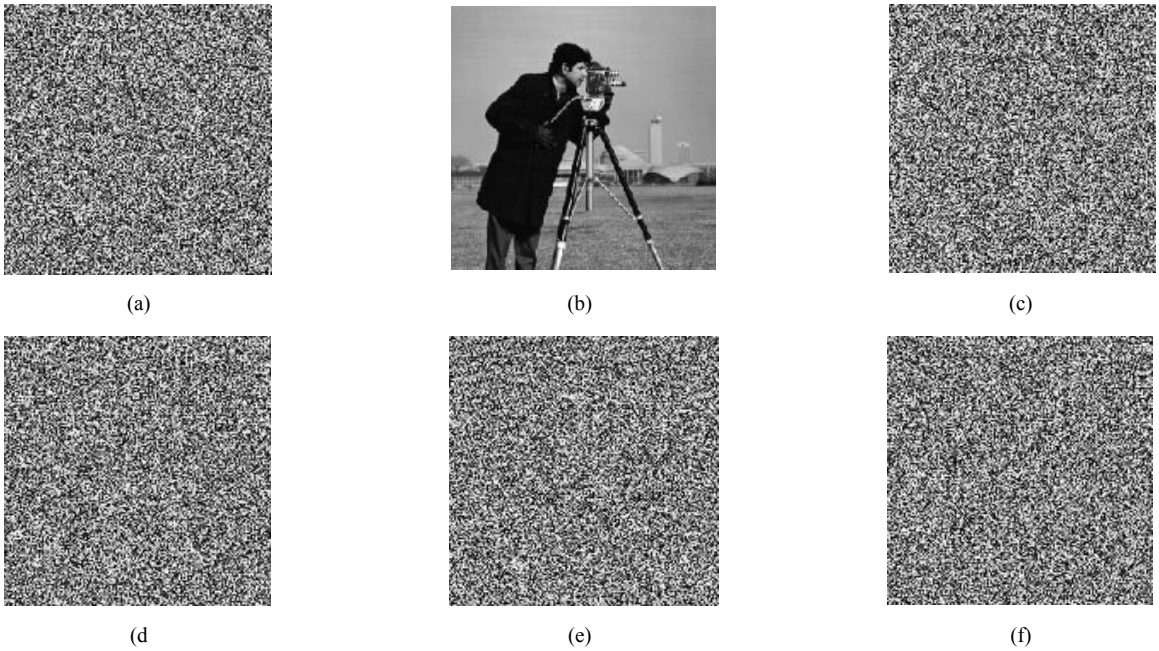


Figure 14: Correlations between pixel pairs along horizontal, vertical and diagonal direction

Key space Analysis:

A good encryption scheme should be sensitive to the secret keys, and the key space should be large enough to make bruteforce attacks infeasible[17]. The proposed Key has the format $[x_0, r, M, T]$: first two parameters (x_0, r) are used to generate the logistic chaotic sequence; the third parameter, an integer *M*, is used to control the size of Ref matrix; the fourth parameter, an integer *T*, is used to control the iteration times of using the Sudoku mapping.

In the experiment, x_0 is some number in [0, 1]; r is the parameter in the Logistic map and it is in [3.6, 4] to keep chaotic behavior. In the second parameter, the integer *M* represents the size of Latin blocks and it also indirectly controls the length of chaotic sequence and the size of **Ref** matrix *N*, where $N = M^2$. Theoretically, the Key space could be at least as large as the space of possible **Ref** matrices, since (x_0, r) is supposed to be able to generate any random-like sequence but **Ref** matrix has its limitation on its size M^2. The total number of possible **Ref** matrices is $\sum_1^i M^2!$, where i is the max allowed size of **Ref** matrix. This number increase very fast as *M* increases. For example $\sum_1^3 M^2! \approx 2^{18.5}$, $\sum_1^4 M^2! \approx 2^{44.3}$, $\sum_1^5 M^2! \approx 2^{83.7}$ $\sum_1^6 M^2! \approx 2^{138}$, $\sum_1^7 M^2! \approx 2^{208}$, $\sum_1^8 M^2! \approx 2^{296}$. Therefore, the key space of our algorithm is sufficient large to resist all kinds of brute force attacks. The experimental results, as shown in Figure 16, also show that our scheme is very sensitive to the secret key.



(a) (b) (c)

(d (e) (f)

Figure 15: Key Sensitivity Results
(a) Encrypted Cameraman image with Key = [0.8239, 3.6511, 4, 12]; (b) Decrypted image with Key = [0.8239, 3.6511, 4, 12];
(c) Decrypted image with Key = [0.8240, 3.6511, 4, 12]; (d) Decrypted image with Key = [0.8239, 3.6512, 4, 12];
(e) Decrypted image with Key = [0.8239, 3.6511, 5, 12]; (f) Decrypted image with Key = [0.8239, 3.6511, 4, 13];

Actually the total number of possible Sudoku matrices is quite large. For M = 3, a 9 by 9 Sudoku matrix has a total possible number of 6,670,903,752,021,072,936,960 = $2^{72.5}$ arrangements. In the paper, we only defined one way to resample the Latin square **L**, if using all possible resample schemes in the Sudoku matrix generator, we can make many more possible Sudoku matrices for the same Secret Sequence K. Theoretically, this total number is $\sum_1^i M^2! \, (M!)^{M+1}$.

4. CONCLUSION

In this paper, we proposed a new image encryption scheme: the Sudoku matrix has been introduced for the encryption process, where both the pixels' values and positions are changed according to this Sudoku matrix. A general way of generating a Sudoku matrix is also given. Compared with traditional block ciphers such as DES, IDEA and AES, the proposed chaos-based Sudoku encryption system has some distinct properties. First, the proposed cryptosystem is suitable for large volume data encryption such as image, audio and perhaps video and works good for many areas, such

as military images (Figure 9), identification images (Figure 10), medical images (Figure 11) and private images (Figure 12). Second, the output of the proposed cryptosystem almost follows a uniform distribution. This nice statistical property helps to prevent attacks based on statistical analysis. Thirdly, the proposed cryptosystem is high sensitivity to the KEY. This implies that even a slight change in KEY will lead a great change in the ciphertext. This property makes various sensitivity-based attacks difficult to break the cryptosystem. Fourth, the proposed cryptosystem permutes the plaintext so greatly and the correlation between two adjacent pixels is very small. Finally, the encryption and decryption algorithms are easy to be implemented in hardware. For example, the generation of the Sudoku Matrix is able to be done with circular registers.

5. ACKNOWLEDGEMENTS

We acknowledge support of the Department of Electrical and Computer Engineering, Tufts University. Y. W thanks Shuai Nie and Zijing Li, Ph.D students in Tufts University, for their valuable helps.

REFERENCES

[1] P. P. Dang, and P. M. Chau, “Image encryption for secure Internet multimedia applications,” Consumer Electronics, IEEE Transactions on*,* 46(3), 395-403 (2000).
[2] L. Tang, [Methods for encrypting and decrypting MPEG video data efficiently] ACM, Boston, Massachusetts, United States(1996).
[3] J. Fridrich, "Image encryption based on chaotic maps." 2, 1105-1110 vol.2.
[4] Z. Jiancheng, R. K. Ward, and Q. Dongxu, "A new digital image scrambling method based on Fibonacci numbers." 3, III-965-8 Vol.3.
[5] D. Zhang, Q. Gu, Y. Pan *et al.*, "Discrete Chaotic Encryption and Decryption of Digital Images." 3, 849-852.
[6] L. Aaronson, “Sudoku Science,” Spectrum, IEEE*,* 43(2), 16-17 (2006).
[7] B. Hayes, “Unwed Numbers The mathematics of Sudoku, a puzzle that boasts" No math required!,” Amer. Scientist*,* 94, 12 (2006).
[8] B. Felgenhauer, and F. Jarvis, “Enumerating possible Sudoku grids,” Mathematical Spectrum. Available at http://www. afjarvis. staff. shef. ac. uk/sudoku/sudoku. pdf, (2005).
[9] T. Mantere, and J. Koljonen, "Solving, rating and generating Sudoku puzzles with GA." 1382-1389.
[10] T. K. Moon, J. H. Gunther, and J. J. Kupin, “Sinkhorn Solves Sudoku,” Information Theory, IEEE Transactions on*,* 55(4), 1741-1746 (2009).
[11] S. Baochen, S. Xiwei, W. Yue *et al.*, "A New Algorithm for Generating Unique-Solution Sudoku." 7, 215-217.
[12] M. H. Shirali-Shahreza, and M. Shirali-Shahreza, "Steganography in SMS by Sudoku puzzle." 844-847.
[13] C. Chin-Chen, C. Yung-Chen, and K. The Duc, "An Information Hiding Scheme Using Sudoku." 17-17.
[14] H. Wien, C. Tung-Shou, and S. Chih-Wei, "Steganography Using Sudoku Revisited." 2, 935-939.
[15] H. Wien, C. Tung-Shou, and S. Chih-Wei, "A Minimal Euclidean Distance Searching Technique for Sudoku Steganography." 1, 515-518.
[16] D. Berthier, “The Hidden Logic of Sudoku,” Lulu, Morrisville, NC, (2007).
[17] R. C. W. Phan, “Impossible differential cryptanalysis of 7-round Advanced Encryption Standard (AES),” Information Processing Letters*,* 91(1), 33-38 (2004).