International Journal of Engineering & Technology, 7 (3.3) (2018) 174-180

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

A dynamic web based color image encryption using Latin square image cipher

Katta sugamya [1] *, Suresh pabboju [1], A. Vinay Babu [2]

[1] Dept. of IT, CBIT, Gandipet, Hyderabad
[2] Retd. Professor, Jntuceh, Hyderabad
**Corresponding author E-mail: sugamya.cbit@gmail.com*

Abstract

A new Latin square image encryption provide a new way of integrating probabilistic encryption in image by embedding random noise in the least significant image bit-plane . LSIC may achieve many desired properties of a secure cipher including a large key space, high key sensitivities, uniformly distributed cipher text, semantically secure, and robustness against channel noise.

Keywords: *Advanced Encryption Standard (AES); Digital Encryption Standard on (DES); Latin Square Image Cipher (LSIC); Substitution Permutation Network (SPN).*

1. Introduction

With the rapid development of the computers and networks, the security of digital data becomes an important issue. Data encryption [1] is one of the most common ways to fulfill the needs of secure requirement for digital data. A method of encrypting images comprises the steps of creation of an encrypted image by alteration of the original image. Hence, the images are encrypted before transmitting.

This paper focuses on development of an image encryption technique where it generates a tough cipher image in a comparatively short span of time. In an encryption scheme, the message or information (referred to as plaintext) is encrypted using an encryption algorithm, and turns it into an unreadable form. This is usually done with the use of an encryption key, which stipulates about how the message is to be encoded. An encryption scheme usually needs a key-generation algorithm which is randomly produce keys. There are two types of encryptions: Symmetric-key and public-key encryption. According to symmetric-key schemes, the encryption and decryption keys are the same and the communicating parties must agree on a secret key before they wish to communicate [3]. In public-key schemes, the encryption key is published for anyone to use and encrypt messages. But, only the receiving party has access to the decryption key and is capable of reading the encrypted messages. Public-key encryption is a quite recent invention: generally, all encryption schemes have been Symmetric-key (also called private-key) schemes. For every encryption process, there is a "key", which is normally a binary sequence with a length from 40 to 256 bits. Generally speaking, the greater the number of bits in the key, the more ciphers text possible key combinations are possible and the longer it would take to break the key.

1.1. Disadvantages of existing system

The main disadvantage of the existing system is that the user cannot choose their required image as input because they needs to edit the source code i.e. change the path to the input image whenever he needs to choose a different input image[7] for encryption. Hence it is static MATLAB application. It takes only gray scale images as input which is disadvantageous because user usually provides colored images as input.

1.2. Proposed system

A dynamic web based application is developed in such a way that the user gets an option to select any image from system. This application is not just limited to gray scale images but also can encrypt colored images. A web application is developed which is similar to online file converters. E.g.: freefileconverter.com which converts one file format to any other file format. Here, in this web application the user can select any of his images, give this image as an input and can download the resultant encrypted image.

The system encrypts the image in short span of time. In this application MATLAB will be run as backend process. The downloaded encrypted image can be communicated over the internet in a secured manner. The proposed system is very user-friendly. The web application developed can be accessed easily even by layman since the MATLAB runs in the background and people accessing this web application don't need to have any technical knowledge regarding MATLAB tool.

To standardize the encryption/decryption processing, the cipher processing block is set to a 256x256 gray scale block, i.e. its pixel intensity is denoted as a 8-bit byte.

In the rest of the paper, P was used to denote a 256x256 plaintext image block, C to denote a corresponding cipher text image block of P, L denotes a keyed Latin square of order 256, and K denotes a 256-bit encryption key.

The new proposed Latin square image cipher [1] is of a SPN structure [1] with eight rounds as shown in fig.1. It is composed of the probabilistic encryption stage noise embedding in LSB and the SPN stage containing three encryption primitives such as Latin Square Whitening, Latin Square Substitution and Latin Square Permutation. It is worthwhile to note that this SPN [1] is of a loom-like structure designed for image data, which encrypts plaintext image

along rows and columns iteratively. Rest of the paper, will discuss the stages and the detail encryption/decryption algorithms for LSIC.

1.3. Image encryption using Latin squares

Because of high sensitivity for initial values and random-like behaviors, chaotic systems are widely used in image encryption [5].Conventionally, encryption keys for these image encryption systems are limited to manipulate the initial values or parameters of the chaotic system [2] [4]. However, effective attacks are found to crack encryption systems by using prior information about the system chaotic map.

Conventionally, digital data is encrypted by bit-stream ciphers and block ciphers. The two well-known block ciphers are the Digital Encryption Standard (DES) and its successor Advanced Encryption Standard (AES). A digital image is a specific type of digital data and can be encrypted by these conventional ciphers. However, they are not ideal ciphers for digital images. Hence development of a Latin Square Image Cipher system for image encryption is chosen.

1.4. Design flow of web application

The Figure 1 describes the design flow of the web application. Here the user selects image from system and uploads it. Then the uploaded image will be moved to a particular directory.

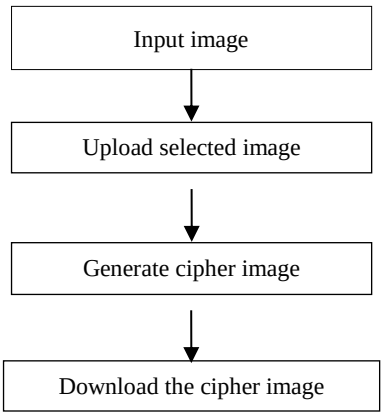

Fig. 1: Design Flow of Web Application.

The mat lab encryption function will be called on that image in backend. Then the cipher image will be generated.

This will be provided to the user for him to download the cipher image which he can communicate over internet securely.

1.5. Design of SPN network

In cryptography, an input message and its corresponding output message of a crypt system are referred to as plaintext and cipher text, respectively. A substitution-permutation network is a cipher structure composed of a number of substitution and permutation ciphers with multiple iterations. This structure is widely used in many well-known block ciphers, e.g. Rijndael i.e. AES [8].

A typical M-round SPN for block ciphers has a structure shown in Fig. 2 Conventionally in a SPN, plaintext, commonly in the form of a bit stream and denoted as P, which is the original message to be encrypted; Key Whitening denotes an operation to mix the plaintext P with a round key; S-Box denotes a substitution-box, which maps one input byte to another in a deterministic way;

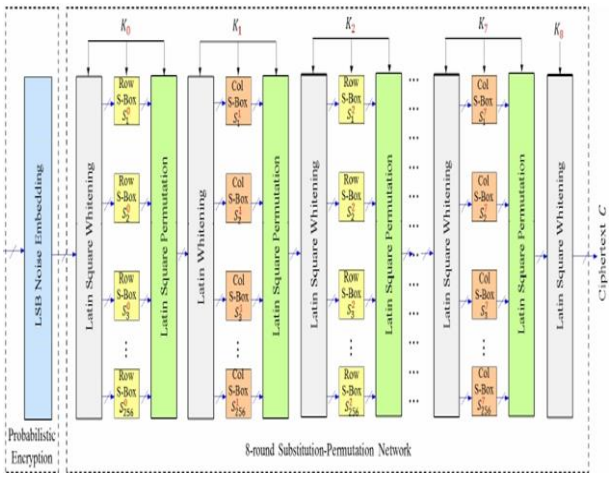

Fig. 2: SPN Network.

P-Box denotes a permutation-box, which shuffles bit positions within the input bit stream in a deterministic way; and cipher text denotes the output bit stream C, which is an encrypted message by the SPN. The decryption process of a SPN cipher is only to reverse the arrow directions of all processing and to use inverse S-Box and inverse P-Box instead. The classic SPN ciphers are able to obtain good Shannon's confusion and diffusion properties. For the diffusion property: if one changes one bit in plaintext P, the corresponding cipher text C changes in many bits. This one-bit change results in a different byte after passing through a S-Box, then leads more byte changes after passing through a P-Box, so on in each cipher round. Finally, one-bit change leads to substantial changes in cipher text C. The confusion property is the similar to the diffusion property. One bit change in encryption key K; will spread over all bits and result significant changes in cipher text C.

2. Implementation

Latin Square Generator

Although Latin squares can be generated a variety of means, for simplicity: Algorithm 1 described below for Latin square generation.

Algorithm 1. A Latin Square Generator L=LSG(q_1, q_2)

Require: q_1 and q2 are two sequences of length-N

Ensure: L is a Latin square of order N

Q_{seed} =SortMap (q_1)

Q_{shift} =SortMap (q_2)

For i = 0: 1: N−1do

L (i, :) = RowShift(Q_{seed},Q_{shift}(i))

End for

In Algorithm 1, both q1 and q2 are length-N sequences from a pseudo-random number generator (PRNG), e.g. Linear Congruential Generators (LCG) [3][2]; Sort Map(Q) is a function which finds the index mapping between a sequence Q and its sorted version Q in the ascending order; and Row Shift t(Q; v) ring shifts the sequence Q with v elements towards left.

For example, if a 4x4 Latin square L is to be generated with $q_1 = [.1, .6, .9, .7]$ and $q_2 = [.3, .9, .4, .2]$

Then function Sort Map (.) first calculates the sorted version of the input sequence and obtain

$$q_1^* = SortMap(Q_1) = [0, 1, 3, 2]$$

it then compares q_1 with $\boldsymbol{q_1^*}$ and q_2 with $\boldsymbol{q_2^*}$ and obtains the element mapping sequences as

$$Q_{seed} = SortMap(q_1) = [0, 1, 3, 2]$$

And

$Q_{shift} = SortMap(q2) = [3, 0, 2, 1]$

where the permutation sequencesQ_{seed} andQ_{shift} indicate for i∈{0, 1, 2,3}

$$q_1^*\left(Q_{seed}(i)\right) = Q_i(i) \; and \; q\left(Q_{shift}(i)\right) = q_2(i)$$

Finally, function Row Shift (Q, v) left shiftsQ_{seed} with the amount v =Q_{shift}(r) indicated by the r[th] element in Q_{shift}, and assigns this row to be the r[th] row in L. Therefore the 4x4 Latin square L is:

$$L = \begin{bmatrix} 2 & 0 & 1 & 3 \\ 0 & 1 & 3 & 2 \\ 3 & 2 & 0 & 1 \\ 1 & 3 & 2 & 0 \end{bmatrix}$$

2.1. Latin square

A Latin square of order N is an NxN array filled with a symbol set of N distinctive elements, with each symbol appears exactly once in each row and each column. The name Latin Square is motivated by the mathematician Leonhard Euler, who used Latin characters as symbols. Mathematically, [6] it defines a Latin square L of order N via a tri-tuple function f_L of (r, c, i) as follows

$$f_L(r, c, i) = \begin{cases} 1, L(r, c) = S_i \\ 0, otherwise \end{cases}$$

where r denotes the row index of an element in L with r∈ N ={0, 1,… , N– 1}; c denote the column index of an element in L with c∈ N; i denotes the symbol index of an element in L with I∈ N; and S_i is the i[th] symbol in the symbol set S ={S_0, S_1,… , S_{N-1}}. Therefore, if L is a Latin square of order N, then

For arbitrary c, I∈ N, then $\sum_{r=0}^{N-1} f_L(r, c, i) = 1$

For arbitrary r, i ∈ N, then $\sum_{c=0}^{N-1} f_L(r, c, i) = 1$

It implies that each symbol appears exactly once in each row and each column in L.

2.2. Steps involved in LSIC system

1) LSB Noise Embedding
2) Key Translation
3) Latin square Whitening
4) Latin Square Row and Column Bijections
5) Latin Square Substitution
6) Latin Square Permutation

2.3. LSB noise embedding

Probabilistic [6] encryption means to use randomness in a cipher, so that this cipher is able to encrypt one plaintext with the exact same encryption key to distinctive cipher texts. It is well known that such randomness is crucial to achieve semantic security. Here, such randomness is introduced by embedding noise in the least significant bit-plane of an image.

More specifically, an EX-OR operation is applied on a randomly generated 256 x 256 bit-plane with the least significant bit-plane of the plaintext image, where the generation of this random bit-plane is completely independent of the encryption key. Once again, this introduced noise in LSB does not affect any visual quality of image from the human visual perceptibility point of view. However, any slight change in plaintext will lead to significant changes in cipher text after it is encrypted by the SPN.

2.4. Key translation

In conventional block ciphers, keys are used directly without translation. For example in key whitening process, the proposed LSIC uses a 256-bit encryption key K with key translation to eight key-dependent Latin squares of order 256 before using in LSIC. Specifically, for a given 256-bit encryption key K,

1) Divide the encryption key K, into eight 32-bit sub keys, using function Sub Key Division i.e.K = [k_0, k_1, k_7]
2) Generate pairs of pseudo-random sequences $(q_1^0, q_2^0), (q_1^1, q_2^1), \ldots, (q_1^8, q_2^8)$ each pair with 2x 256 elements by using PRNGs by feeding these sub keys as seeds.
3) Generate key-dependent Latin squares i.e., L_0, L_1, L_8 with the order of 256. Namely, ∀n ∈ {0, 1, 8}, then L_n = LSG (q_1^n, q_2^n)

2.5. Latin square whitening

In conventional SPN for block ciphers, the whitening stage normally combines a plaintext message P with a round key, (e.g. XOR operation) such that

- The statistics of the plain text message P is redistributed after combining.
- The relationship between cipher text and encryption key is very complicated and involved.

In image encryption, a plaintext message is an image block, P, composed of a number of pixels. Each pixel is represented by several binary bits (a byte). Therefore, XOR whitening scheme become inefficient for image data, in the sense that it requires to extend an encryption key to be an equal size to a plaintext image, and to impose bitwise XOR to byte pixels. And this type of image encryption is called a naive algorithm. Since the objective of key whitening is to mix plaintext data with encryption keys, therefore whitening is defined as a transposition cipher over the finite field GF(2^8) for image data, as shown in equation

$$y = [x+l] \; 28 \tag{1}$$

where x is a byte in plaintext, l is a corresponding byte in the keyed Latin square, y is the whitening result and $[.]_2{}^8$denotes the computations over GF (2^8). The above whitening process can be easily reversed by applying

$$x = [y+l]_{\;2^8} \tag{2}$$

In image encryption, plaintext byte x is a pixel, say it is located at the intersection of r[th] row and c[th] column i.e. x = P(r, c) Now let l = L(r, c) be an element located at the corresponding position in the keyed Latin square L, and y be the cipher text byte with y = C(r, c), then the pixel-level equation

$$\begin{cases} C(r, c) = [SR\,(P(r, c), [D]_3) + L(r, c)]_{2^8} \\ P(r, c) = SR([C(r, c) + L(r, c)]_{2^8}, [D]_3) \end{cases} \tag{3}$$

where symbol n denotes current round number (n ∈ [0, 7]), D = L(0, 0) is the rotating parameter, and SR denotes the spatial rotating function (X; d) rotates an image X according to different values of the direction d as defined in Equation below

$$Y = SR(X, d) \tag{4}$$

Notice that if Y =SR(X, d), then the following identity always holds

$$X = SR(Y, d) \tag{5}$$

Apply key whitening for all pixels using the pixel-level Eq. (3), the Latin Square Whitening (LSW) in the image level then can be denoted as

$$LSW: \begin{cases} C = Ecr_w(L, P, D) \\ P = Dcr_w(L, C, D) \end{cases} \tag{6}$$

Therefore, plaintext image block P can be restored from the cipher text image block C. Figure5 shows an example of Latin Square

Whitening, where the first row shows images and the second row shows corresponding histograms of these images. From this example, it is easy to verify that the cipher text image after the Latin Square Whitening is unrecognizable and its pixels are redistributed uniformly.

2.6. Latin square row and column bijections

Since Eqs. (1) and (3) hold, each row and each column in a Latin square L of order N is a permutation of the integer number sequence [0, 1, …, N - 1], define bijections (one-to-one and onto mapping) by mapping this integer number sequence to either a row or a column in a Latin square, which is a permutated sequence of the integer number sequence. In other words, to construct forward and inverse row mapping functions (FRM and IRM) with respect to the r^{th} row in L as shown in Eq. (7), and also forward and inverse column mapping functions (FCM and ICM) with respect to the c^{th} column in L as shown in Eq. (8), where x and y denote the input and output of the mapping functions, respectively.

$$\begin{cases} y = FRM(L,r,x) = L(r,x) \\ x = IRM(L,r,y) = arg\,max(f_L(r,z,y)) \end{cases} \tag{7}$$

$$\begin{cases} y = FCM(L,x,c) = L(x,c) \\ x = ICM(L,y,c) = arg\,max(f_L(z,c,y)) \end{cases} \tag{8}$$

Where f_L is the tri-tuple function defined in Eq(1). Its maximum is equal to [1], i.e. $f_L(r, x, y) = [1]$, only for the column number x satisfying the constraint, L(r, x) = y. Further, row mapping identities hold for arbitrary x and y within a Latin

$$\text{Square L:} \begin{cases} IRM\big(L,r,FRM(L,r,x)\big) = x \\ FRM\big(L,r,IRM(L,r,y)\big) = y \end{cases} \tag{9}$$

Similarly, it also has column mapping as follows:

$$\begin{cases} ICM(L,FCM(L,x,c),c) = x \\ FCM(L,ICM(L,y,c),c) = y \end{cases} \tag{10}$$

2.7. Latin square substitution

An S-Box in cryptography is a basic component performing byte substitution. Each S-Box can be defined as a bijection, also known as a one-to-one and onto mapping. In image encryption, an image pixel is commonly represented as a byte, i.e. a sequence of bits. For example, 8-bit gray scale image has 256 gray intensity scales with each intensity scale represented in an 8-bit sequence.
Because of the existence of FRM/IRM and FCM/ICM bijections in a Latin square, which is able to perform byte substitution in an image cipher using bijections from rows and columns in a Latin square? The substitution with respect to a row in a Latin square is called Latin Square Row S-box

$$\text{(LSRS): } LSRS: \begin{cases} C = Ecr_s^{row}(L,P) \\ P = Dcr_s^{row}(L,C) \end{cases} \tag{11}$$

In regard to pixel-level function of LSRS, each cipher text byte is determined by the FRM function see Eq (7) using the keyed Latin square L with function parameters given by plaintext bytes and cipher text bytes as follows:

$$Ecr_s^{row}: C(r,c) = \begin{cases} FRM\big(L,C(r-1,c),P(r,c)\big), ifr \neq 0 \\ FRM\big(L,0,P(r,c)\big), ifr = 0 \end{cases} \tag{12}$$

Clearly, plaintext bytes then can be perfectly restored from cipher text bytes, if IRM is used instead of FRM as follows

$$Dcr_s^{row}: P(r,c) = \begin{cases} IRM\big(L,C(r-1,c),C(r,c)\big), ifr \neq 0 \\ IRM\big(L,0,C(r,c)\big), ifr = 0 \end{cases} \tag{13}$$

Similarly, bijections from columns in a Latin square are used to perform byte substitutions. And this is called Latin Square ColumnS-box (LSCS) i.e.

$$LSCS: \begin{cases} C = Ecr_s^{col}(L,P) \\ P = Dcr_s^{col}(L,C) \end{cases} \tag{14}$$

In addition, the corresponding LSCS encryption and decryption process then can be defined as:

$$Ecr_s^{col}: C(r,c) = \begin{cases} FCM\big(L,P(r,c),C(r,c-1)\big), ifc \neq 0 \\ FCM(L,P(r,c),0), if\ c = 0 \end{cases} \tag{15}$$

$$Dcr_s^{col}: P(r,c) = \begin{cases} ICM\big(L,C(r,c),C(r,c-1)\big), ifc \neq 0 \\ ICM(L,C(r,c),0), ifc = 0 \end{cases} \tag{16}$$

Figure6 Shows encryption results of Latin Square Row S-box and Latin Square Column S-box. As can be seen, the plaintext image block P becomes unrecognizable after applying either LSRS or LSCS. Histogram analysis also shows that the statistics of the pixel intensity changes dramatically after substitution.Latin square row/column substitution defined above has excellent diffusion properties.
One pixel change in the plaintext P will diffuse to a column/row of pixels after a round of LSRS or LSCS. This diffusion quickly spreads to the entire cipher text image in several cipher rounds.

2.7. Latin square permutation

Unlike a S-Box performing byte substitution, a P-Box performs byte shuffling or scrambling. Each P-Box can also be defined as a bijection.
If both input x and output y are considered in FRM and IRM as indices (see Eq. (7), then FRM defines a mapping { 0, 1, … , 255 } { 0, 1, … , 255 } and IRM defines the corresponding inverse mapping. Therefore, the Latin square row p-box (LSRP) is defined with respect to rows in a Latin square L as follows,

$$LSRP: \begin{cases} C(r,c_y = P(r,FRM(L,r,c_x)) \\ P(r,c_x) = C(r,IRM\big(L,r,c_y\big)) \end{cases} \tag{17}$$

Where c_x and CY denotes the column indices before and after mapping. Consequently, for any pixel in P and its corresponding pixel in C are in the same row r after LSRP; and only column indices change before and after mapping with relationship c_y = FRM(L, r, c_x) holds.
Similarly, a Latin Square Column P-box (LSCP) can be constructed with respect to columns in a Latin square as

$$LSCP: \begin{cases} C\big(r_y,c\big) = P(FCM(L,r_x,c),c) \\ P(r_x,c) = C(ICM\big(L,r_y,c\big),c) \end{cases} \tag{18}$$

In general, Latin Square Permutation function can be written as

$$LSP: \begin{cases} C = Ecr_p(L,P) \\ P = Dcr_p(L,C) \end{cases} \tag{19}$$

The implementation part will give permutation results of using LSRP, LSCP and LSP. It is clear that cascading LSRP and LSCP in LSP helps LSP to achieve a better pixel permutation performance in the sense that, pixels in its cipher text image become more random-like and makes the cipher text image content unintelligible.

3. System design

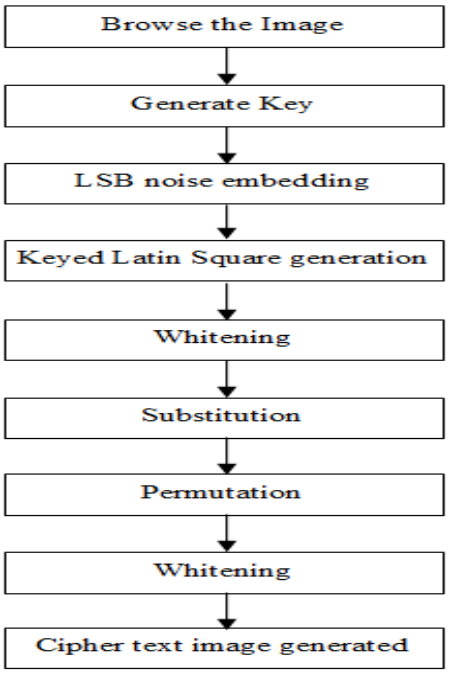

Fig. 3: Design Flow of Mat lab Application.

The Figure 3 describes the system design of matlab application. User selects any image and gives an input to matlab encryption function. Random Key is generated and a latin square is generated which is dependent on the key generated. Whitening, substitution and permutation steps are to be performed on the image and finally the cipher text image is generated. Decryption can also be performed on the cipher text image and plain text image can be obtained back again

Table 1: Algorithm for Encrypting Plaintext Image

Algorithm Latin square image cipher- encryption C=E (P, K)
Require: K is a 256-bit key
Require: P is a 256x256 8-bit gray scale image block
(q_1, q_2) = KDSG (K, 8)
For i = 0: 1: 7 do
If i ==0 then
C_{LSP}=LSBNoisEmbedding (P) [4]
End if
L_i = LSG (Q_1^i ,Q_2)
D_i= L_i (0, 0)
C_{LSW} = Ecr_w (L_i, C_{LSP}, D_n)
If mod (i, 2) ≠0 then
C_{LSS} = Ecr_s^{col}(L_i, C_{LSW})
Else
C_{LSS} = Ecr_s^{row}(L_i, C_{LSW})
End if
C_{LSP} = Ecr_p (L_i, C_{LSS})
End for
L_8 = LSG (q_1^8,q_2^8)
D_8= L_8 (0, 0)
C= Ecr_w (L_8, C_{LSP}, D_n)
Require: C is a 256x256 8-bit gray scale image block

Although Latin squares can be generated via a variety of means, for the sake of simplicity Algorithm 1 described below is used for Latin square generation in the paper.

Table 2: Algorithm Latin Square Generator

A Latin Square Generator L=LSG (q_1, q_2)
Require: q_1 and q_2 are two length-N sequences
Ensure: L is a Latin square of order N
Q_{seed} =SortMap (q_1)
Q_{shift} =SortMap (q_2)
For i = 0: 1: N−1 do
L (i, :) = RowShift (Q_{seed}, Q_{shift} (i))
End for

In Algorithm 1, both q1 and q2 are length-N sequences from a pseudo-random number generator (PRNG), e.g. Linear Congruential Generators (LCG) [3][2]; Sort Map (Q) is a function which finds the index mapping between a sequence Q and its sorted version Q in the ascending order; and Row Shift t(Q; v) ring shifts the sequence Q with v elements towards left.
For example, if a 4x4 Latin square L is to be generated with

$$q_1 = [.1, .6, .9, .7] \; and \; q_2 = [.3, .9, .4, .2]$$

Then function Sort Map (.) first calculates the sorted version of the input sequence and obtain

$$q_1^* = SortMap(q_1) = [0, 1, 3, 2]$$

It then compares q_1 with $\boldsymbol{q_1^*}$ and q_2 with $\boldsymbol{q_2^*}$ and obtains the element mapping sequences as

$$Q_{seed} = SortMap(q_1) = [0, 1, 3, 2]$$

And

$$Q_{shift} = SortMap(q_2) = [3, 0, 2, 1]$$

where the permutation sequences Q_{seed} and Q_{shift} indicate for $i \in \{0, 1, 2, 3\}$

$$q_1^*\left(Q_{seed}(i)\right) = Q_i(i) \; and \; q_2^*\left(Q_{shift}(i)\right) = q_2(i)$$

Finally, function Row Shift (Q, v) left shifts Q_{seed} with the amount v =Q_{shift}(r) indicated by the r^{th} element in Q_{shift}, and assign this row to be the r^{th} row in L. Therefore the 4x4 Latin square L is:

$$L = \begin{bmatrix} 2 & 0 & 1 & 3 \\ 0 & 1 & 3 & 2 \\ 3 & 2 & 0 & 1 \\ 1 & 3 & 2 & 0 \end{bmatrix}$$

Table 3: Algorithm for Decrypting Plaintext Image

Algorithm Latin square image cipher- decryption P=D(C, K)
Require: K is a 256-bit key
Require: C is a 256x256 8-bit grayscale image block
Require: P is a 256x256 8-bit grayscale image block
(q_1, q_2) = KDSG (K, 8)
For n = 7: -1: 0 do
If n, ==seven then
L_8 = LSG (q_1^8,q_2^8)
D_8 = L_8 (0, 0)
P_{LSW}= Dcr_w(L_8, C, D_8)
End if
L_n = LSG (q_1^n,q_2^n)
D_n= L_n (0, 0)
P_{LSP} = Dec_p (L_n, P_{LSW})
If mod (n, 2) ≠ 0 then
P_{LSS}= Dcr_s^{col}(L_n,P_{LSP})

4. Results

Image Encryption results using the proposed Latin Square Image cipher are shown below.

Fig. 4: Input Image.

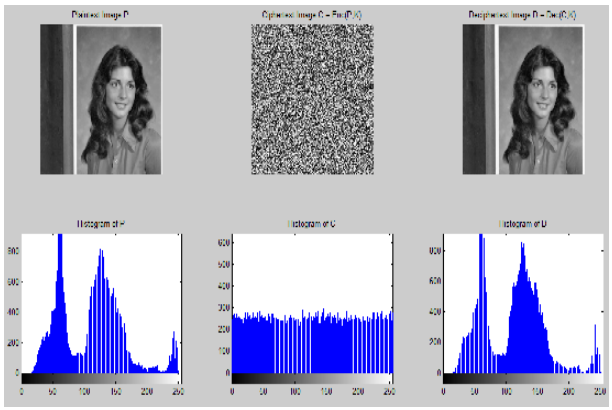

Fig. 5: Encryption and Decryption.

The figure 5 shows Encryption and decryption of input image with histograms.

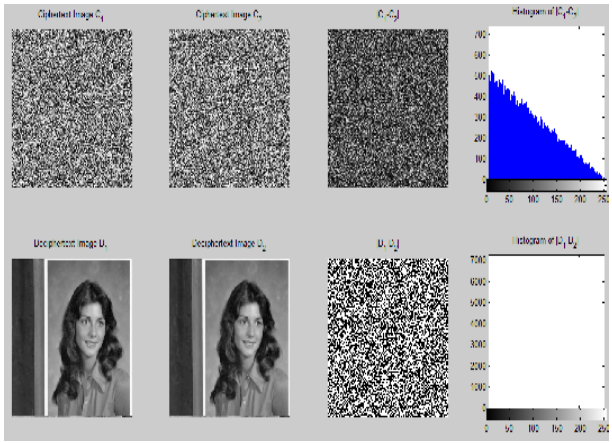

Fig. 6: Probabilistic Encryption and Decryption.

Figure 6 shows the Probabilistic encryption and decryptionof cipher text image

Fig. 7: Robustness to Noise in Cipher.

A good cipher should tolerate certain amount of noise; figure 7 shows the robustness to noise of a cipher.

Fig. 8: Sensitivity to Key Changes.

A secured cipher will give high key sensitiveness in both encryption and decryption. Figure [7] and [8] shows simulation results for encryption and decryption.

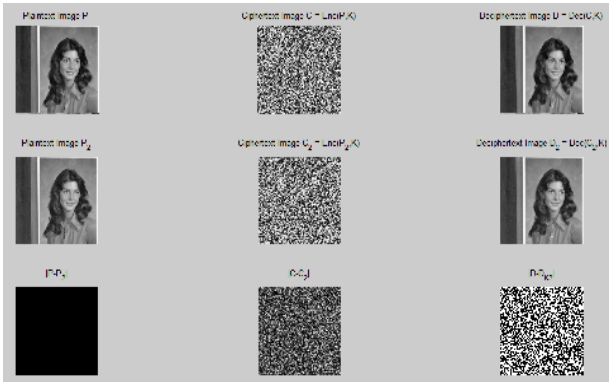

Fig. 9: Sensitivity to Plaintext Changes.

Figure 9 shows Plain text image of diffusion property of Latin square image cipher, the plain text image differs p and p_2 and c-c_2 differs in cipher text image.

Fig. 10: Result Images.

The above Figure 10 shows the results of decryption robustness of Latin Square Image cipher for various noise ratio in cipher text images.
The above fig.11 shows the performance of the system i.e. here Tic and Toc functions are placed before and after the encryption function call respectively. These indicate the time required for the execution of encryption function.

Fig. 11: Performance Verification.

5. Conclusion and future scope

LSIC integrates probabilistic encryption in a pre-processing stage and thus it allows encryption of a plain text image into different cipher text images when the same encryption key is used. LSIC's decryption stage is robust against a certain level of noise and thus is suitable to transmit cipher data over a corrupted channel.
In the proposed system the uploaded images as well as the resultant images are stored in a specific directory but not in a database. So, as a future work the proposed system can be integrated with database in order to store them securely and permanently without any loss of data as databases are more robust when compared to file systems.

References

[1] Yue Wu, Yicong Zhou, Joseph P. Noonan, SosAgaian, and C. L. Philip Chen, " A Novel Latin Square Image Cipher", a draft submitted to *IEEE transactions on information forensics and security*.

[2] V. Patidar, N. K. Pareek, G. Purohit, and K. K. Sud, "A robust and secure chaotic standard map based pseudorandom permutation-substitution scheme for image encryption," *Optics Communications, vol. 284, no. 19, pp. 4331–4339, 2011. [Online].Available: www.scopus.com https://doi.org/10.1016/j.optcom.2011.05.028.*

[3] G.K.Wallace,"Thejpeg still picture compression standard," *Communications of the ACM, vol. 34, no. 4, pp.30–44,Apr.1991.[Online].Available:http://doi.acm.org/10.1145/103085.103089*

[4] Yue Wu, Joseph P. Noonan, SosAgaian, "A Wheel-Switch Chaotic System for Image Encryption", *Proceedings of 2011 International Conference on System Science and Engineering, Macau, China - June 2011 https://doi.org/10.1109/ICSSE.2011.5961867.*

[5] Yue Wu, Joseph P. Noonan, SosAgaian, "Image Encryption using the Rectangular Sudoku Cipher", *Proceedings of 2011 International Conference on System Science and Engineering, Macau, China - June 2011 https://doi.org/10.1109/ICSSE.2011.5961994.*

[6] W. Press, Numerical recipes: the art of scientific computing. *Cambridge University Press, 2007. [Online]*

[7] R.Gonz´alez and R. Woods, *Digital image processing (Pearson/PrenticeHall2008). [Online].*