IMAGE SPLITTING

AND

RECONSTRUCTION

FOR

SECRET COMMUNICATION

ABSTRACT

In this paper Image splitting for secure storage and communication has been considered through two different techniques. Firstly it is shown how by using a Linear Feedback Shift Register (LFSR), an image can be split in such a way that none of the split parts make any sense on its own, but when all the split parts are superimposed modulo-2, the original image emerges. Secondly we generalize the splitting technique using two-dimensional programmable Hybrid Cellular Automata. Cellular Automata based splitting techniques provide a more difficult to break scheme and the number of split parts can be increased to a very large number adding flexibility and security to the distribution scheme to different agents. Implementation of both the schemes has been carried out and results are demonstrated.

CHAPTER 1

INTRODUCTION

1. Image Processing For Security

An image is a media by which the information can be presented in effective and better manner to the receivers for its better understanding. Visual information has vital role in many defence as well as civil applications and its protection becomes essential to avoid losses due to its leakage. Images are handled in digital form to achieve higher security. Techniques for securing images can be considered as cryptographic or steganographic. In cryptographic techniques, transforming it into an unintelligible form conceals the contents of an image, which looks as a random mess of pixels. In steganographic techniques, an image to be secured is embedded in an ordinary image known as a cover or carrier image. The embedded image is then communicated. Yet another technique used for secure storage or communication of an image is based on Image splitting. This technique is an extension of a scheme known earlier for message security under the name secret splitting [2].

Secret Splitting is a method to take a message and divide it up into pieces. Each piece by itself contains no information but when all the pieces are combined, the original message is recovered. Any individual piece is of no consequence, but if put together the message appears.

The simplest sharing scheme splits the message between A and B participant according to the following steps:

- Dealer generates a random bit string, R, of the same length as that of the message M.
- Dealer logically Exclusive ORs M with R to generate S
  M XOR R = S
- Dealer gives R to A and S to B

Message Reconstruction:

- A and B XOR their pieces together to reconstruct the message R XOR S = M.

In this technique no encryption is carried out. The original image is split (divided into a number of images of same size) in such a way, that only superposition of either all the parts or minimum of k parts only lead to the original image and no information about the image is available if less than k parts are superimposed.

Mathematically, an image is defined as a 2D function f(x, y) where (x, y) denotes the position of a pixel in xy plane and f(x, y) represents the gray level of pixel located at (x, y) [1]. An image is categorized as binary, panchromatic/monochromatic or multispectral image. A binary image is a black and white image recorded in two levels black and white and has dynamic range {0, 1}. Monochromatic image is a multi level image. An image recorded in 8 bits / pixel has 256 gray values ranging from 0 to 255. Multispectral image is a colour image recorded in three different bands, viz. red (R), green (G) and blue (B) bands.

The term image processing refers to manipulating gray value of the pixel within the image frame. Mathematically, a function T transforms f(x, y) to some new value g(x, y) i.e. $T(f(x, y)) = g(x, y)$

For securing images in cryptographic techniques, the transformation T holds the following properties:

(i) Easy implementation of T to encrypt image.

(ii) Hard reversing process, T^{-1}.

(iii) Encrypted images unintelligible and look random.

For securing images in steganographic techniques, the transformation T holds the following properties:

(i) Easy implementation of T to embedded image.

(ii) Hard reversing process, T^{-1}.

(iii) Stego images as similar as covers.

Cryptographic methods conceal the content and steganographic methods conceal the existence of secret image.

In the split image technique of secure storing or communication the transformation T holds the following properties:

(i) T splits every unit of image (say an 8-bit recorded image) in such a way that only superposition of a minimum number of splits can lead to the original image unit.

i.e. $T(U) = T_1(U) \oplus T_2(U) \oplus \ldots \oplus T_m(U)$

(ii) T is a lossless transformation. No information is lost during processing.

(iii) Size of each constituent split is equal to the size of the original unit.

In stream ciphers the message is converted into bits EXORed bitwise with key sequence. The techniques for **concealing the content** are:

(i) **Contrast Reversal**: Normally, in contrast reversal, the pixel values are inverted such that the brighter pixel becomes darker and vice versa. If an image F(x, y) has a dynmic range R, then resulting image G(x, y) is given by

$G(x, y) = R - F(x, y)$

For an image of size N x N, $2^{(N \times N)}$ trials have to be applied in an exhaustive approach to get back the original image.

(ii) **Image Scanning:** Image Scanning refers to observing each pixel for its value. The images are scanned as raster scanning, from left to right and top to bottom while recording, transmitting or retrieving.

A scan function with t number of scan terms for given image of size N x N, where $N = 2^x$ and x is an integer, is defined as

$S_t = L_1n_1 \# L_2n_2 \# \ldots \# L_in_i \# \ldots \# L_tn_t$

L_i is scan letter. The symbol # connects two scan terms which are performed hierarchically.

The techniques for **concealing the existence** are:

(i) **Shift Register based Hiding:** A shift register of suitable length is used to hide the image.

(ii) **Image Splitting:** There can be two different ways to split an image (without any alteration or encryption of the image):

(i) Only a small portion of an image is supplied to different agents who can, when need be, get together to get the full image. If some portions of the split image are kept away from agents, they would not be in a position to get the full image and (secret) image would therefore remain secure.

(ii) A very robust method of image splitting is to split each pixel in such a way that a full size image (with split pixels) is kept or sent to each agent. Thus each agent has a full size image with him with split pixels. Nothing can be made out of this full size image, unless all such images are superimposed (modulo 2).

This is the technique that has been studied in this dissertation using two different methodologies.

In order to further classify the technique of Image splitting and Image reconstruction a simple key – based image splitting technique is first discussed and the technique is illustrated through an example where the image is split in 4 parts and then reconstructed. The shift register based and Celllular Automata based image splitting are later detailed in Chapters 4 and 6.

Next it is shown in Chapter 4, how a simple methodology based on Linear Feedback Shift Registers (LFSR) can be used for this purpose. In order to increase the number of splits a maximal length (primitive) polynomial is used. A primitive polynomial of degree m over GF(2), containing elements (0, 1) as coefficients generates a length of $(2^m – 1)$ and therefore provides $(2^m – 2)$ splits --- one of the splits being the original pixel.

Secondly, a two dimensional programmable Hybrid Cellular automata has been used to split each pixel of the image to be communicated securely. For this we use an 8 – neighborhood two – dimensional Cellular Automata.

2. An elementary example of Image splitting and Reconstruction:

Suppose the BMP image consists of four columns and four rows of pixel vectors shown as, where every pixel vector is of length 8:

A1	A2	A3	A4	=	A
B1	B2	B3	B4	=	B
C1	C2	C3	B4	=	C
D1	D2	D3	D4	=	D

To generate four splits, we will use four keys: K1, K2, K3 and K4.

The BMP image where the pixels are represented by 8 bit binary data is shown as:

Image I represented as:

1001 1010	1010 0011	1101 0110	0011 1011	=	A
1000 1000	0110 0110	1001 1101	1100 1010	=	B
0110 1001	0011 0011	1101 1110	1111 1111	=	C
1111 0110	0000 1101	0101 0111	1001 1101	=	D

Split 1 generation:

Assuming K1 = 1010 0011 to obtain first split
Image I1 represented as:

1001 0011	1010 0011	1101 0011	0011 0011	=	A’
1000 0011	0110 0011	1001 0011	1100 0011	=	B’
0110 0011	0011 0011	1101 0011	1111 0011	=	C’
1111 0011	0000 0011	0101 0011	1001 0011	=	D’

Split 2 generation:

Assuming K2 = 1000 1010 to obtain second split
Image I2 represented as:

1001 1010	1010 1010	1101 1010	0011 1010	=	A’’

1000 1010	0110 1010	1001 1010	1100 1010	=	B''
0110 1010	0011 1010	1101 1010	1111 1010	=	C''
1111 1010	0000 1010	0101 1010	1001 1010	=	D''

Split 3 generation:

Assuming K3 = 1000 1101 to obtain third split
Image I3 represented as:

1001 1101	1010 1101	1101 1101	0011 1101	=	A'''
1000 1101	0110 1101	1001 1101	1100 1101	=	B'''
0110 1101	0011 1101	1101 1101	1111 1101	=	C'''
1111 1101	0000 1101	0101 1101	1001 1101	=	D'''

Split 4 generation:

The nth split is generated as:
Image XOR (n-1) splits

Thus image I4 obtained as:
I4 = I XOR I1 XOR I2 XOR I3

0000 1110 0000 0111 0000 0010 0000 1111

0000 1100 0000 0010 0000 1001 0000 1110

0000 1101 0000 0111 0000 1010 0000 1011

0000 0010 0000 1001 0000 0011 0000 1001

Method Of Image Reconstruction:

The secret image can be regenerated back if we take the XOR of n splits, which are with n participants.
Thus, I = I1 XOR I2 XOR I3 XOR I4

Reconstructing row A of image I

1001 0011 1010 0011 1101 0011 0011 0011
XOR
1001 1010 1010 1010 1101 1010 0011 1010
XOR
1001 1101 1010 1101 1101 1101 0011 1101
XOR
0000 1110 0000 0111 0000 0010 0000 1111
EQUALS
1001 1010 1010 0011 1101 0110 0011 1011 = A

Reconstructing row B of image I

1000 0011 0110 0011 1001 0011 1100 0011

XOR

1000 1010 0110 1010 1001 1010 1100 1010

XOR

1000 1101 0110 1101 1001 1101 1100 1101

XOR

0000 1100 0000 0010 0000 1001 0000 1110

EQUALS

1000 1000 0110 0110 1001 1101 1100 1010 = B

Reconstructing row C of image I

0110 0011 0011 0011 1101 0011 1111 0011

XOR

0110 1010 0011 1010 1101 1010 1111 1010

XOR

0110 1101 0011 1101 1101 1101 1111 1101

XOR

0000 1101 0000 0111 0000 1010 0000 1011

EQUALS

0110 1001 0011 0011 1101 1110 1111 1111 = C

Reconstructing row D of image I

1111 0011 0000 0011 0101 0011 1001 0011

XOR

1111 1010 0000 1010 0101 1010 1001 1010

XOR

1111 1101 0000 1101 0101 1101 1001 1101

XOR

0000 0010 0000 1001 0000 0011 0000 1001

EQUALS

1111 0110 0000 1101 0101 0111 1001 1101 = D

Thus the entire image I has been reconstructed as:

Image I :

1001 1010 1010 0011 1101 0110 0011 1011 = A

1000 1000 0110 0110 1001 1101 1100 1010 = B

0110 1001 0011 0011 1101 1110 1111 1111 = C

1111 0110 0000 1101 0101 0111 1001 1101 = D

Therefore, the original image has been reconstructed by taking the XOR of 4-splits in possession of the 4-participants.

i.e. I = I1 XOR I2 XOR I3 XOR I4

3. About this dissertation:

Two methods of splitting an image for secure communication of the image have been studied. Firstly, a technique of splitting based on Linear Feedback Shift Register with a primitive polynomial feedback is studied. The degree of the primitive polynomial is chosen to be equal to the number of bits in each pixel vector and the initial state of the shift register is taken to be any non-zero binary vector except the pixel vector itself. The subsequent vectors so generated by the shift register are distributed to different agents randomly or each agent being given a certain number of vectors (pixels) as per the scheme of the sender. As the vectors are distributed randomly, no one agent can get at the 'secret' pixel. However when all the agents get together and add modulo-2 all the vectors (pixels) with them, they reach at the correct pixel. This scheme of splitting the image is explained fully in Chapter 4.

Another scheme of splitting the image based on Cellular Automata is explained in Chapter 6. Here again the various stages of Cellular Automata produced pixels are distributed amongst a number of agents randomly and only when all the agents come together they can arrive at the correct pixel. Chapter 5 first explains the basics of Cellular Automata Theory and then applies it into image splitting. Cellular Automata based splitting is much more complex and provides many more choices in the number of choices for distribution of split images.

CHAPTER 2

BMP FORMATS AND THEIR STRUCTURE IN C

BMP files are commonly used file format for commonly used operating system called "Windows". BMP images can range from black and white (1 byte per pixel) up to 24 bit colour (16.7 million colours). While the images can be compressed this is rarely used in practice and won't be discussed in detail here.

Structure

A BMP file consists of either 3 or 4 parts. The first part is a header, this is followed by a information section, if the image is indexed colour then the palette follows, at last of all pixel data. The position of the image data with respect to thestart of the file is contained in the header. Information such as the image width and height, the type of compression, the number of colours is contained in the information header.

Header

The header consists of the following fields.We are assuming short int of 2 bytes, int of 4 bytes, and long int of 8 bytes. Further we are assuming byte ordering as for typical (Intel) machines. The header is 14 bytes in length.

```
typedef struct {
        unsigned short int type ;                   /* Magic identifier            */
        unsigned int size ;                         /* File size in bytes          */
        unsigned short int reserved 1, reserved 2 ;
        unsigned int offset ;                       /* Offset to image data, bytes */
```

} HEADER;

The useful fields in this structure are the type field (should be 'BM') which is a simple check that is likely to be a legitimate BMP file, and the offset field which gives the number of bytes before the actual pixel data (this is relative to the start of the file). This structure is not a multiple of 4 bytes for those machines/compilers that might assume this, these machines will generally pad this structure by 2 bytes to 16 which will unalign the future fread() calls.

Information

The image info data that follows is 40 bytes in length, it is defined in the struct given below. The fields of most interest below are the image width and height, the number of bits per pixel (should be 1, 4, 8 or 24), the number of planes (assumed to be 1 here), and the compression type (assumed to be 0 here).

```
typedef struct {
        unsigned int size ;                 /* Header size in bytes          */
        int width, height ;                 /* Width and height of image */
        unsigned short int planes ;         /* Number of colour planes   */
        unsigned short int bits ;           /* Bits per pixel            */
        unsigned int compression ;          /* Compression type          */
        unsigned int imagesize ;            /* Image size in bytes       */
        int xresolution, yresolution ;      /* Pixels per meter          */
        unsigned int ncolours ;             /* Number of colours         */
        unsigned int importantcolours ;     /* Important colours         */
} INFOHEADER ;
```

The compression types supported by BMP are listed below:

0 – no compression

1 – 8 bit run length encoding

2 – 4 bit run length encoding

3 – RGB bitmap with mask

type 0 (no compression)

24 bit Image Data

The simplest data to read is 24 bit true colour images. In this case the image data follows immediately after the information header, that is, there is no colour palette. It consists of three bytes per pixel in b, g, r order. Each byte gives the saturation for that colour component, 0 for black and 1 for white (fully saturated).

Indexed colour data

If the image is indexed colour then immediately after the information header there will be a table of infoheader .ncolours colours, each of 4 bytes. The first three bytes correspond to b, g, r components, the last byte is reserved/unused but could obviously represent the alpha channel. For 8 bit greyscale images this colour index will generally just be a greyscale ramp. If you do the sums….then the length of the header plus the length of the information block plus 4 times the number of palette colours should equal the image data offset. In other words

14 + 40 + 4 * infoheader .ncolours = header .offset

CHAPTER 3

SEQUENCES GENERATED BY LINEAR FEEDBACK SHIFT REGISTERS

Primary sequences generated by Linear Feedback Shift Registers have been extensively studied in literature [2, 3, 4] in the context of their usefulness in cryptography especially in the design of stream ciphers.

The operational disadvantages of one-time-pad have led to the development of Synchronous Stream Ciphers, which encipher the plain text in much the same way as the one-time-pad with deterministically random generated sequence (generated by key-stream generators or Pseudo-random generators such as shift registers) with perfect synchronization between the encrypting and the decrypting devices. The security of a synchronous Stream Cipher now depends on the randomness of the key stream and the way the system is synchronized.

The working of the stream cipher is explained in Fig 3.1. The plaintext (or message) is encrypted on bit by bit basis by adding modulo-2 or XOR (exclusive or) with the key stream, which is a binary sequence generated by electronic machine with memory or without, to produce a ciphertext (encrypted message), which is sent through the channel. At the receiving end, the same key stream is added modulo-2 to the ciphertext to get the plaintext (or message). Example 3.1 illustrates the principle.

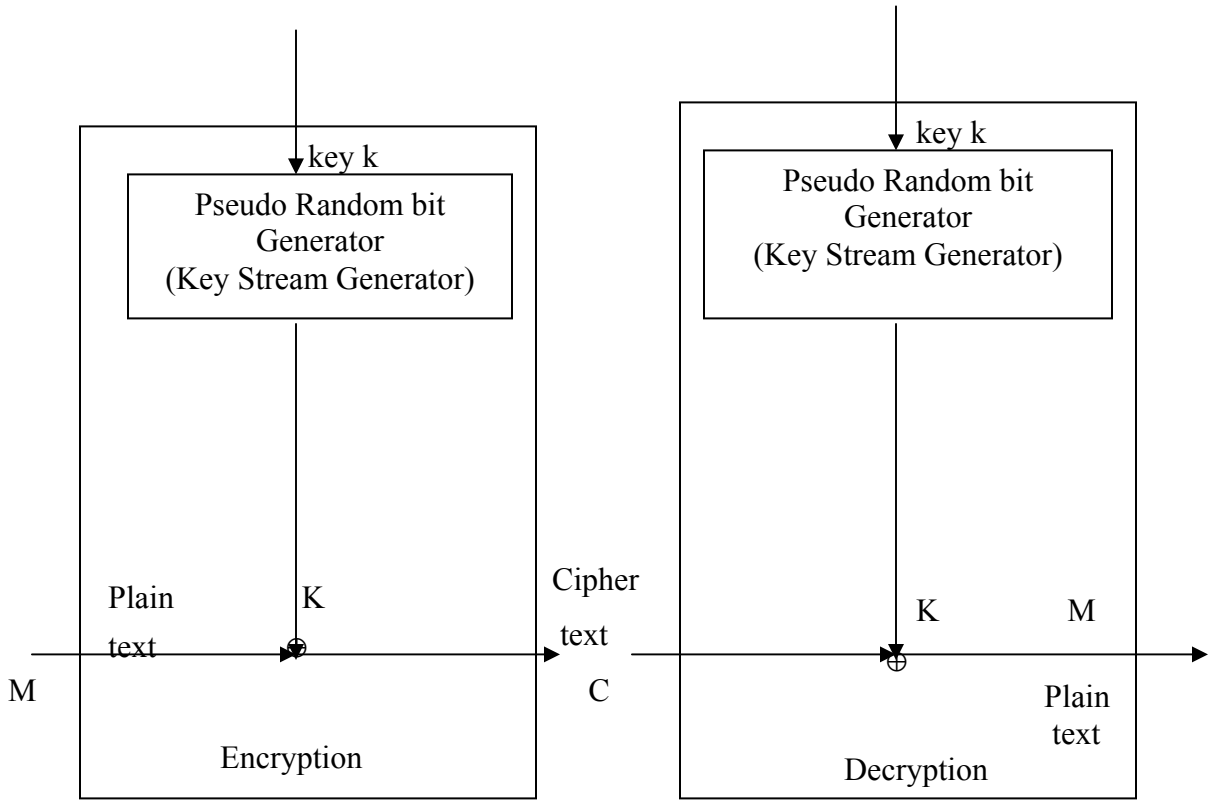


Fig 3.1 A Stream Cipher System

Example 3.1

Plaintext $\overrightarrow{M}$	:	10011000101011101001 10
Key Stream $\overrightarrow{K}$	:	10001010010011110111 10
Ciphertext $\overrightarrow{M} \oplus \overrightarrow{K} = \overrightarrow{C}$	:	00010010111000011110 00
Key Stream $\overrightarrow{K}$	:	10001010010011110111 10
Plaintext $\overrightarrow{C} \oplus \overrightarrow{K} = \overrightarrow{M}$	:	10011000101011101001 10

The machine that produces the key stream from the actual key k and the internal state is called the key stream generator. Whenever the key k and the internal states are identical at the sender and the receiver end, the key streams are also identical, and

deciphering is easily accomplished. One says that the key generators at the sending and receiving ends are synchronized with each other. Whenever the key generator looses synchronism, deciphering becomes impossible and means must be provided to reestablish synchronization.

In self synchronizing stream ciphers, the deciphering transformation has a finite memory with respect to the influence of the past bits, so that an erroneous or lost ciphered bits cause only a fixed number of errors in the deciphered plaintext, after which, again, the correct plaintext is produced. In any system with initial contents (IC) and logic F, if each key stream bit is derived from a fixed number of preceding ciphertext bits, then the system becomes self synchronizing. A serious disadvantage of self synchronizing stream ciphers is their limited analyzability because of the dependence of the key stream on the message stream (Fig 3.2).

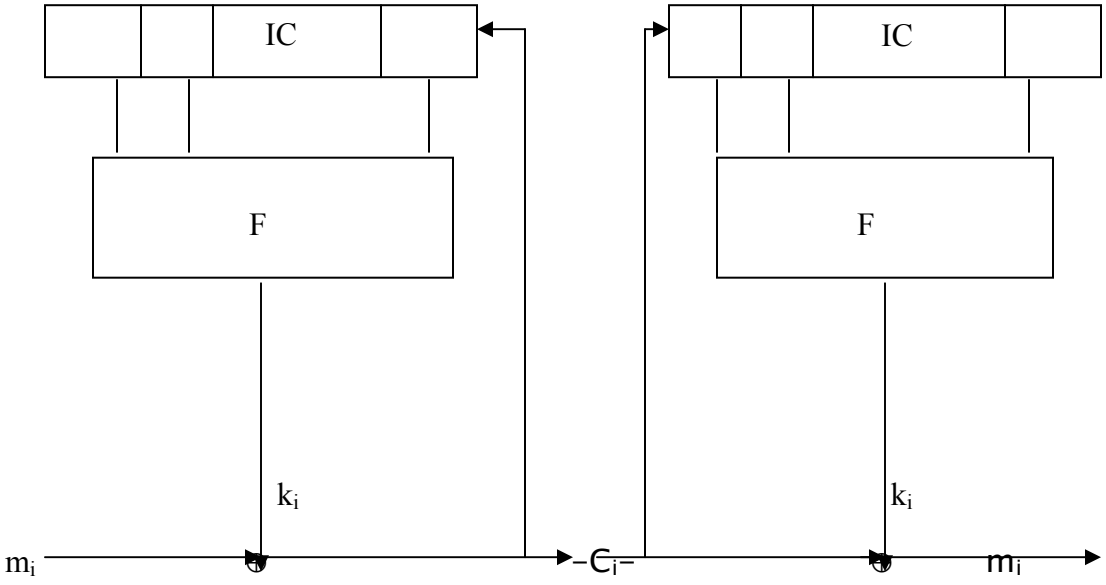


Fig 3.2 Principle of self synchronizing stream cipher

When a communication link employs some means of frame synchronization (as in the case of digital communication), the synchronous stream cipher may be supplied with some sort of self-synchronizing property without decreasing the security level.

CONSIDERATIONS IN DESIGN OF STREAM CIPHER SYSTEMS FOR MESSAGE / IMAGE ENCRYPTION

In order to ensure that a stream cipher designed by a cryptographer takes some minimum amount of effort on the part of the cryptanalyst to be able to reach at the communicated plaintext, it is necessary to give due consideration to the design of the key stream generator so that the produced encryption sequences satisfies some properties. As the encryption sequences are generated through shift registers, first of all we describe various types of shift registers and the properties of the sequences generated by them. Some of the important design considerations for stream cipher are:

(i) The encryption sequences should have a large period. As we are generating the encryption sequences deterministically and such a sequence have a finite period, it is to be ensured that the sequence does not repeat itself within a reasonable period taking into account the amount of plaintext to be encrypted into a particular key.

(ii) The most important property of encryption sequences is unpredictability. To ensure this property we have to have a large complexity in the sequence as also proper distribution of ones and zeroes in the sequences.

(iii) One way to ensure the above two requirements is to generate a sequence by a non linear combining function whose arguments are the shift register sequences generated by linear feedback shift registers. Such sequences, however, are vulnerable to a new type of attack called the correlation attack if the ciphertext sequence can be correlated to one of the constituent sequences generated by linear shift register.

(iv) From a cryptographer's point of view it is necessary to ensure that the adversary may not be able to launch a "Brute Force Attack" to find the key used for encryption. Having ensured a large period, a large complexity and randomness, there should be a large variability of the possible keys to be used for encryption. This number should be so large that taking into account the speed of the latest computers, including parallel processing possibilities, it should not be feasible for the cryptanalyst to arrive at the plaintext. From the

point of view of key management however, it is better to have as small a key set as possible. To meet the twin requirements of ease of key set management and security from brute force attack, usually a key structure is introduced in the stream cipher system, which is arrived at by taking into account enciphering requirements.

It may be appreciated that all the requirements can only be met if the designed system lends itself to analysis. We can therefore add analyzability as an important requirement.

In the above we have given a detailed account of the requirements of a good stream cipher based on shift register sequences, which produce an enciphering sequence to be added mod (2) to the message sequence (or pixel vectors).

Although, we carry out only image splitting and no encryption, it is to be appreciated that the technique of producing an encryption sequence or pixel vector splits is just the same. In pixel vector splits we use the vectors generated by LFSR instead of the sequence generated in the process. Therefore in the methodology suggested here for splitting the image is closely related to encryption by LFSR generated sequences. The connection between LFSR and Cellular Automata is brought out in Chapter 5 on Cellular Automata.

GENERATION OF BINARY SEQUENCES BY LINEAR FEEDBACK SHIFT REGISTER (LFSR):

A linear feedback shift register (LFSR) of length r is shown in Fig 3. It consists of a cascade of r unit delay cells or registers with a provision to form a linear combination of cell contents, which then serves as the input to the first register or stage. After each time unit, the contents of the registers are shifted one place to the right and a new bit, which is modulo-2 addition of some of the contents of the shift register, is placed at the first stage. The output of LFSR is assumed to be taken from the last stage. The initial

contents $a_0, a_1, \ldots, a_{r-1}$ of the r stages coincide with the first r output bits and the remaining output bits are uniquely determined by the recursion

$$a_n = \sum_{i=1}^{r} c_i a_{n-i} \quad n > r-1$$

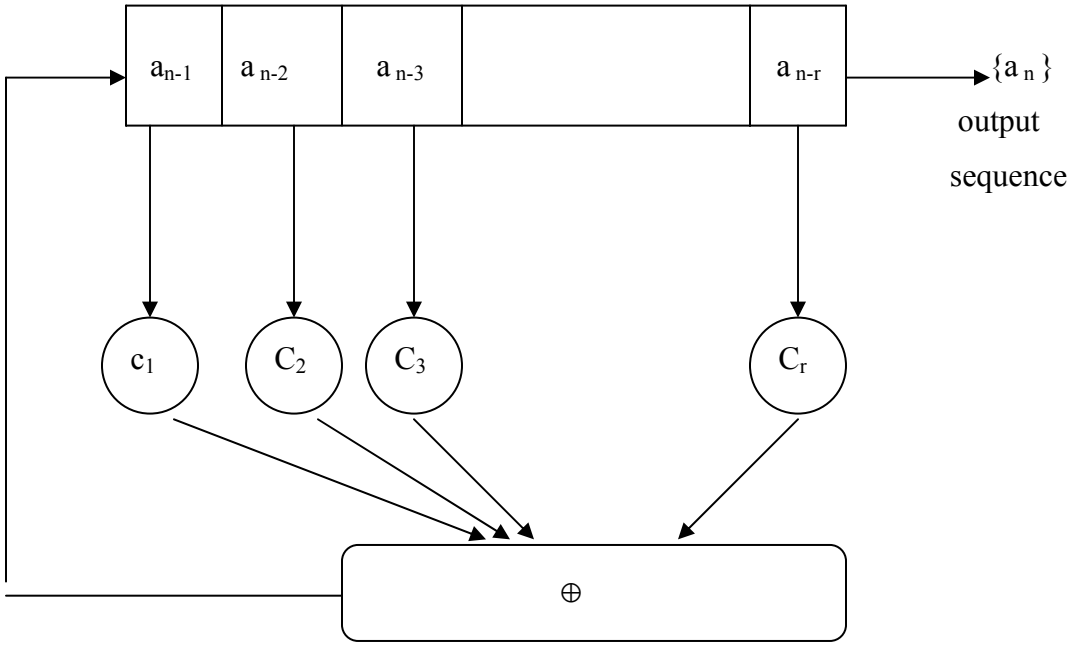


Fig 3 An r-stage Linear Feedback Shift Register

The feedback coefficients $c_1, c_2, \ldots, c_r$ are assumed to lie in the field GF(2), so as to take the value one or zero according as the ith register is, or is not involved in the feedback circuit. An output sequence generated by LFSR satisfying the difference is also called a linear recurring sequence. The polynomial $1 + \sum_{i=1}^{r} c_i x_i$ is called the characteristic polynomial or feedback polynomial. The output sequence $\{a_n\} = a_0\, a_1\, a_2 \ldots$ is called a linear feedback shift register sequence and generator is called Linear Feedback Shift Register Generator (LFSR).

THE MATRIX METHOD:

If we treat the contents of an LFSR as an r-dimensional state vector, the shift register can then be interpreted as a linear operator, which changes each state into next. It is a familiar fact that a linear operator operating on r-dimensional vector is most conveniently represented by an r x r matrix. In general, a shift register matrix takes the form with all ones along the diagonal above the main diagonal; the feedback coefficients down the first column and zero at all other positions. Thus the matrix

$$T = \begin{bmatrix} c_1 & 1 & 0 & \dots & 0 \\ c_2 & 0 & 1 & \dots & 0 \\ . & . & . & & . \\ . & . & . & & \\ . & . & . & & . \\ c_{r-1} & 0 & 0 & \dots & 1 \\ c_r & 0 & 0 & \dots & 0 \end{bmatrix}$$

and $[a_{n-1}, a_{n-2}, \dots, a_{n-r}]\, T = [a_n, a_{n-1}, \dots, a_{n-r+1}]$

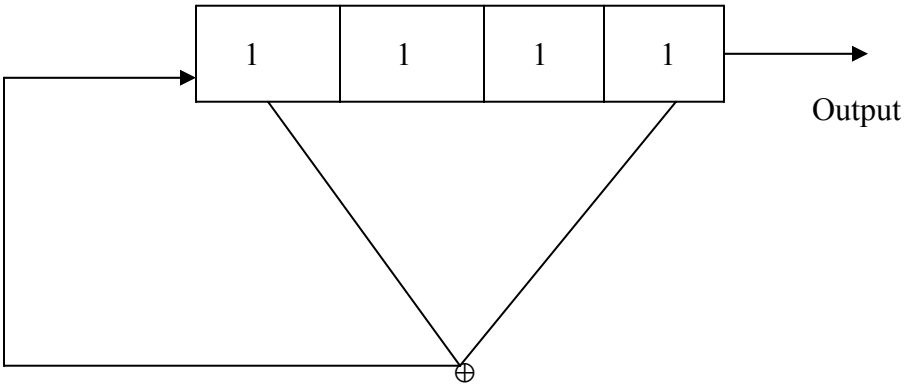
$[a_n, a_{n-1}, \dots, a_{n-r+1}]\, T = [a_{n+1}, a_n, \dots, a_{n-r+2}]$

The repeated application of T on vector $S = [a_{n-1}, a_{n-2}, \dots, a_{n-r}]$ will introduce power of T in the successive register state vectors i.e. $\vec{S}, \vec{ST}, \vec{ST^2}, \vec{ST^3}, \vec{ST^4}, \dots$. It follows that a periodic output of period p will result if we can find an exponent p such that $T^p = I$, the identity matrix. Also the characteristic equation of the matrix $T = \det [T - xI]$ is the reciprocal polynomial of the characteristic polynomial of the shift register.

CHAPTER 4

SPLITTING OF PIXEL VECTOR USING LFSR

Sequences generated by a four stage shift register with feedback polynomial : $1 + x + x^4$.



Generated polynomial is : $1 + x + x^4$.

Starting vector is 1111. The stages through which such a register will go through are as follows :

M =

1 1 1 1------- 1

0 1 1 1------- 2

1 0 1 1------- 3

0 1 0 1------- 4

1 0 1 0------- 5

1 1 0 1------- 6

0 1 1 0------- 7

0 0 1 1------- 8

1 0 0 1------- 9

0 1 0 0------- 10

0 0 1 0------- 11

0 0 0 1------- 12

1 0 0 0-------- 13

1 1 0 0-------- 14

1 1 1 0-------- 15

M is a 15 x 4 matrix ---- each row is 4-length vector.
Also every column of Matrix M is the Pseudo - random sequence generated through the above shift register. One can say we have generated 4 sequences of length 15 each. All the four sequences are cyclic shifts of each other. Each one of these sequences can be used as encryption sequence as shown earlier.

However for Image splitting we use the 14 vectors (No. 2 – 15) in the following way. Let the pixel we want to split be represented by 4 bit vector. Let it be the pixel 1111. In order to split this pixel we use a 4 stage Linear Feedback Shift Register with the feedback polynomial: $1 + x + x^4$ which is a primitive polynomial of 4^{th} degree and the initial contents of the register can be any non-zero vector except 1111.

To achieve splitting this pixel in $2^4 - 2 = 14$ parts (the number of parts is one less than the length of the sequence which in this case is 15), we start with any one pixel except 1111 and generate a sequence of 14 vectors. These 14 vectors can be distributed in any manner amongst various Agents. An agent may get more than one pixel or a single pixel, but no one agent would be in a position to get the correct pixel 1111. This pixel can be achieved by mod(2) addition of all the splits.

For an m-bit pixel, using an m degree primitive polynomial, we can split each of the $2^m - 1$ pixels into $2^m - 2$ splits which when added mod 2 altogether would yield the required pixel but any number of pixels added together mod 2 taken separately would be of no consequence if the number of distinct pixels is less than $2^m - 2$.
Thus with a simple technique stated above we can distribute the pixels amongst different people in

$$ {}^mC_1 + {}^mC_2 + \ldots + {}^mC_{m-1} $$

different ways. In case of above example where m = 4, we have

$^4C_1 + {}^4C_2 + {}^4C_3 = 14$ possible ways to distribute the splits.

In each possible distribution, a bit by bit mod 2 additions would yield the correct pixel. The above is achieved because of the following:

All the binary vectors of length m together form a vector space of dimension m. Therefore any combination of less than m vectors cannot give any desired vector and any vector can be produced by addition of m or more than m vectors. It has to be ensured for distribution of the pixel vectors that no single agent may get more than m vectors. Also the addition of $2^m - 2$ vectors would yield $(2^m - 1)^{th}$ vector which is the desired pixel vector with $2^{m\,th}$ vector being all zero vector. Thus we can split each pixel of the image to be communicated in the above manner. The size of the image is maintained and if each pixel is split as per the above scheme, one can easily send the split images to different agents who would be able to get the image only when all of them get together and superimpose $2^m - 2$ split images mod 2, the correct original image would appear.

A number of other techniques of securing images have been given by Ratan [4]. He has specifically discussed Shift register based hiding. But our approach is based on Pixel wise splitting while his hiding of image is Coordinate based.

CHAPTER 5

CELLULAR AUTOMATA

The Cellular automation (CA) first introduced by John Von Neumann [1] in the 1950s, has been accepted as a good computational model for the simulation of complex physical systems. It can be used to simulate readily the complex growth pattern of a snowflake and it has also been suggested that the analysis of the general features of the CA may yield better insight of the behaviour of such complex models. Wolfram et al [8 &9] have studied one-dimensional, periodic, boundary additive CAs with the help of polynomial algebra. Mathematical studies of null and periodic boundary additive CAs and some experimental observations have been reported by Pries et al [10]. The treatment is based on a similar kind of polynomial algebra and is confined mainly to uniform additive CAs.

A more generalised treatment of additive CAs was introduced by Das et al [10] as a new tool based on matrix algebra. Treatments of both null and periodic boundary CAs, uniform and hybrid, non complemented and complemented – are reported. In this paper, some of the properties reported were verified and established with the help of Matrix based formulation. It was shown by Das et al [10] that the use of LFSR as a pseudorandom pattern generator is based on sound mathematical tools around polynomial algebra. LFSR was also shown to be a special case of additive Cellular Automata.

A CA is a collection of simple cells usually arranged in a regular fashion. The next state of the cell depends on the present state of 'k' of its neighbours, for a k-neighbourhood CA, specified by its neighbourhood function. There can be various

boundary conditions namely '00' (null, where extreme cells are connected to ground level), 'periodic' (extreme cells are adjacent), etc.

Mathematically, the next state transition of the ith cell can be represented as a function of the present states of the ith, (i+1)th and (i – 1)th (for 3-neighbourhood) cells:

$q_1(t:) = f(q_i(t), q_{(i+1)}(t), q_{(i-1)}(t))$

where 'f' is known as the rule of the CA denoting the combinational logic.

For a 2-state 3-neighbourhood CA, there can be a total of 2^3 distinct neighbourhood configurations. For such a CA with cells having only 2 states there can be a total of 2^8 distinct mappings from all these neighbourhood configurations to the next state. Each mapping is called a rule of the CA. Two particular sets of transition from a neighbourhood configuration to the next state have been shown below:

111	110	101	100	011	010	001	000	
0	1	1	1	1	0	0	0	rule 120
0	1	0	1	1	0	1	0	rule 90

Two sets of transition

These are two rules of mappings, designating a transition from the neighbourhood configuration (consisting of the present states of the cells q_{i-1}, q_i and q_{i+1}) to the next state of q_i. This 8-bit binary number expressed in equivalent decimal form gives a convenient scheme for representing the CA rule. Rules 120 and 90 are illustrated above.

The combinational logic equivalent for rule 120 is given as

$$q_i(t+1) = q_{i+1}(t)\, \bar{q}_{i-1}(t) + q_{i+1}(t)\, \bar{q}_i(t) + \bar{q}_{i+1}(t)\, q_i\, q_{i-1}$$

The minimized expression for rule 90 is

$\bar{\quad}$ $\bar{\quad}$

$q_i(t+1) = q_{i+1}(t)\, q_{i-1}(t) + q_{i+1}(t)\, q_{i-1}(t)$

$q_i(t+1) = q_{i+1}(t) \oplus q_{i-1}(t)$

In rules 120 and 90, we have assumed a CA with two states per cell. We shall adhere to this kind of CA only.

Definition 1: If in a CA the same rule applies to all the cells, then the CA is called as uniform or regular CA.

Definition 2: If in a CA different rules are applied over different cells, then the CA is called as a hybrid CA.

Definition 3: If in a CA (which is a 2-state per cell CA) the neighbourhood dependence is on EXOR or EXNOR only, then the CA is called as an additive CA. The next state of a cell in such a CA can be expressed as a modulo-2 sum of the neighbours.

Definition 4: If in a CA the neighbourhood dependence is EXOR, then it is called a noncomplemented CA and the corresponding rule is referred to as a noncomplemented rule. For neighbourhood dependence on EXNOR only (where there is an inversion of the modulo-2 logic), the CA is called a complemented CA. The corresponding rule involving the EXNOR function is called the complemented rule.

A hybrid CA may have both complemented and noncomplemented rules.

LFSR as special case of hybrid additive group CA:

An n-bit LFSR can be conveniently modeled as a hybrid additive CA where the neighbourhood dependence of the leftmost cell (input cell) extends to n and for each of the rest of the cells, the dependence is confined to the cell towards its left only. Fig shows a 4-bit LFSR associated with the characteristic polynomial

$f(x) = 1 + a_1x + a_2x^2 + a_3x^3 + x^4$, $a_i \in \{0, 1\}$ $i = 1$ to 3

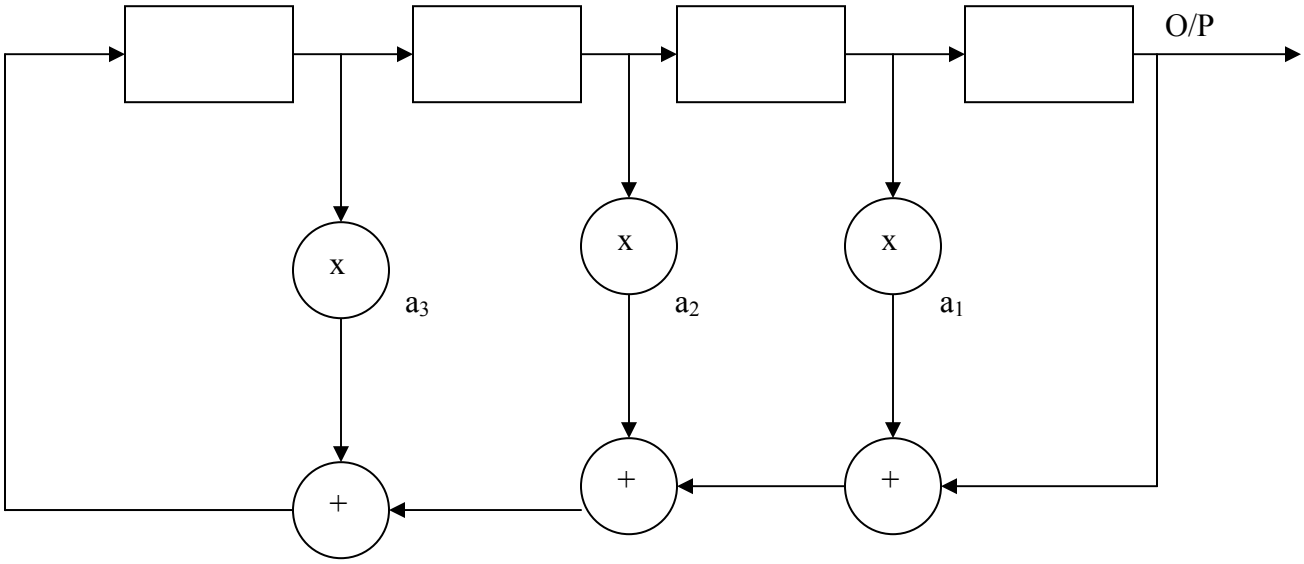


Fig : 4-bit LFSR

$a_1 = 1$ when feedback connection physically exists

The corresponding T matrix when it is modeled as a hybrid CA is as follows:

$$T = \begin{bmatrix} a_3 & a_3 & a_3 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

By virtue of the fact that the T matrix is nonsingular, it becomes immediately evident that LFSR generates cycles of states, i.e. periodic sequences.

Lemma: The characteristic equation of the T matrix for an LFSR is identical to the characteristic polynomial f(x) of the LFSR with x replaced by T.

TWO-DIMENSIONAL CELLULAR AUTOMATA:

Two-dimensional finite cellular automata (2D-CA for short) are discrete dynamical systems formed by a finite two-dimensional array of r x s identical objects called cells, such that each of them can assume a state. The state of each cell is an element of the finite state set, S. We will consider $S = Z_c$ where $c = 2^b$ is the number of colours of the image; i.e., if the image is a black and white image, then b = 1; for gray level images the value is b = 8, and if it is a colour image, then b = 24 Maronon et al [11].

The (i, j)-th cell is denoted by <i, j>, and the state of this cell at time t is $a_{ij}^{(t)} \in Z_c$. The 2D-CA evolves deterministically in discrete time steps, changing the states of all cells according to a local transition function,

$$f: (Z_c)^n \rightarrow Z_c$$

The updated state of each cell depends on the n variables of the local transition function, which are the previous states of a set of cells, including the cell itself, and constitute its neighbourhood. For 2D-CA, there are some classic types of neighbourhood, but in this work only the extended Moore neighbourhood will be considered;that is, the neighbourhood of the cell <i, j> is formed by its nine nearest cells:

$$V_{i,j} = \{<i-1, j-1>, <i-1, j>, <i-1, j+1>, <i, j-1>, <i, j>, <i, j+1>, <i+1, j-1>, <i+1, j>, <i+1, j+1>\}$$

Graphically it can be seen as follows:

$< i-1, j-1 >$	$< i-1, j >$	$< i-1, j+1 >$
$< i, j-1 >$	$< i, j >$	$< i, j+1 >$
$< i+1, j-1 >$	$< i+1, j >$	$< i+1, j+1 >$

Consequently the local transition function

$$f : (Z_c)^9 \rightarrow Z_c \quad \text{is}$$

$$a_{ij}^{(t+1)} = f\left(a_{i-1\, j-1}^{(t)}, a_{i-1\, j}^{(t)}, a_{i-1\, j+1}^{(t)}, a_{i\, j-1}^{(t)}, a_{i\, j}^{(t)}, a_{i\, j+1}^{(t)}, a_{i+1\, j-1}^{(t)}, a_{i+1\, j}^{(t)}, a_{i+1\, j+1}^{(t)} \right),$$

or equivalently,

$$a_{i\, j}^{(t+1)} = f\left(V_{ij}^{(t)} \right), \quad 0 \le i \le r-1, \quad 0 \le j \le s-1,$$

where $V_{ij}^{(t)} \quad (Z_c)^9$ stands for the the states of the neighbour cells of $< i, j >$ at time t.

The matrix

$$C^{(t)} = \begin{bmatrix} a_{i00}^{(t)} & \cdot & \cdot & \cdot & a_{0,\, s-1}^{(t)} \\ \cdot & \cdot & & & \cdot \\ & & \cdot & \cdot & \cdot \\ \cdot & & & \cdot & \\ a_{r-1,\, 0}^{(t)} & \cdot & \cdot & \cdot & a_{r-1,\, s-1}^{(t)} \end{bmatrix}$$

is called the configuration at time t of the 2D-CA, and $C^{(0)}$ is the initial configuration of the CA. Moreover, the sequence $\{C^{(t)}\}_{0 \le t \le k}$ is called the evolution of order k of the 2D-CA, and C is the set of all possible configurations of the 2D-CA; consequently $|C| = c^{r.s}$.

CELLULAR AUTOMATA AS BIT STREAM GENERATORS

In stream ciphers pseudorandom pattern generators are widely used to generate the key streams for encryption. Nandi et al [12] have demonstrated use of Hybrid Cellular Automata for sequence generation based on the two programmes and then implemented through PCA_1 and PCA_2. We describe this technique in the following and then use the same scheme for creating split images.

KEY STREAM GENERATORS:

Many key stream generators are based on combining two or more generators (i.e. LFSR's) by using nonlinear functions. It is already established that maximum length CA's generate patterns having high quality of pseudo randomness. Using CA properties two types of key stream generators are proposed:

i) PCA with ROM

ii) Two stage PCA

Fig 6.1 shows a 90 / 150 PCA cell used in the key stream generators.

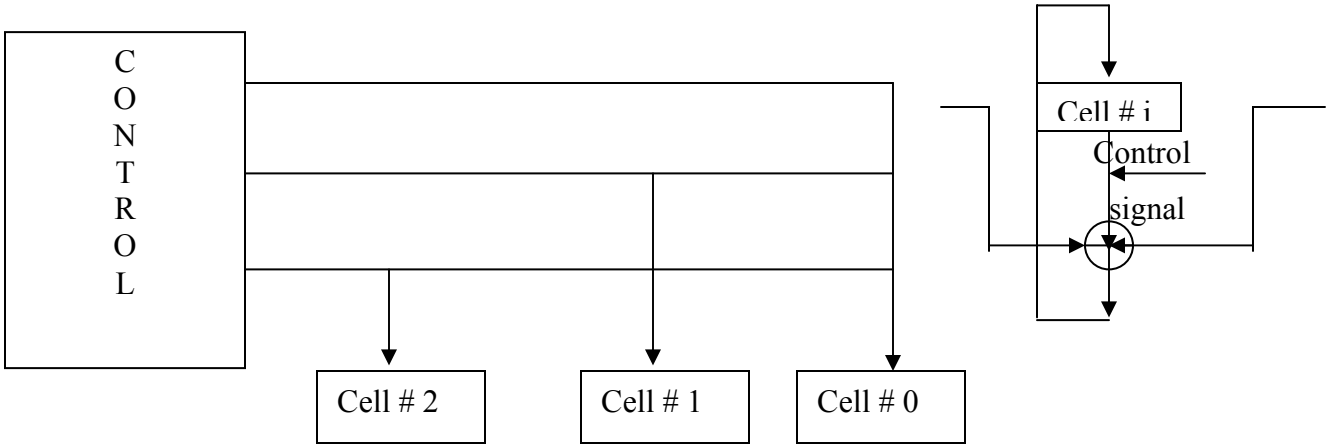


Fig A 3-cell Programmable CA structure and a PCA cell

PCA With ROM as a Key Stream Generator:

Let L be the number of cells in the PCA and w be the number of maximal length CA's with rule 90 and rule 150. Assume that l maximal length CA's are chosen out of w maximal length CA's. These rules are noted as $\{R_0, R_1, R_2, \ldots, R_{l-1}\}$. The rule configuration control word corresponding to a rule R_i is stored in a ROM word. Initially the PCA is configured with rule R_0 and loaded with a non-zero seed. With this configuration the PCA runs one clock cycle. Then it is reconfigured with the next rule (i.e., R_1) and runs another cycle. This process repeats until CLOCK SIGNAL to PCA is made inactive. The rule configuration of PCA changes after every run, i.e., if in the ith run rule configuration is R_i, then in the next run, rule is $R_{(i+1)\text{mod}l}$. After each clock cycle, the output of PCA is taken as a pseudorandom pattern.

Now our objective is to show that this output sequence is a pseudorandom pattern sequence. The following Theorem provides the background.

Theorem 1 : If the characteristic polynomial of a CA is primitive then it generates pseudorandom pattern.

Corollary 1: A PCA built with maximal length CA configurations generates pseudorandom patterns.

Proof: All the maximal length CA's generate pseudorandom sequences, individually. The sequence generated by the PCA can be taken as a set of subsequences generated by a particular maximal length CA. As the subsequences are pseudorandom in nature so the overall sequence is also pseudorandom in nature.

The above key stream generator scheme may be implemented through a circuit whose block diagram is shown in Fig

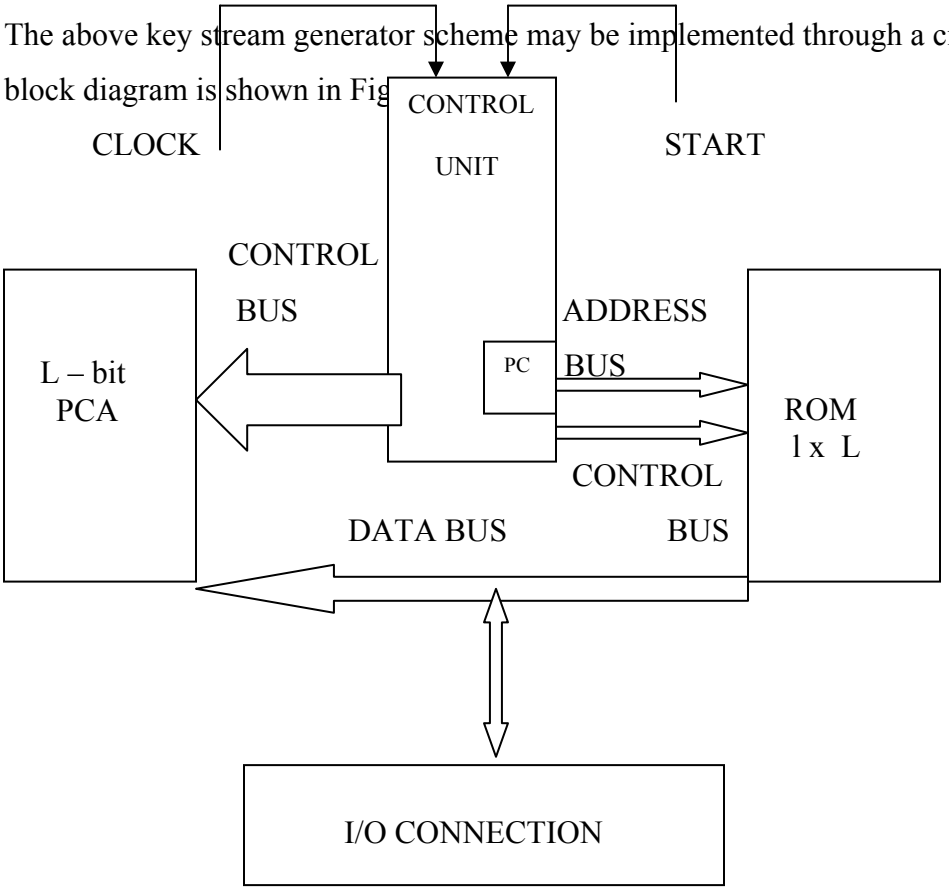


Fig 6.2 PCA based psedorandom pattern generator

Description of the Circuit:

PCA (Programmable CA): It is an L-bit null boundary, uniform or hybrid CA configured with the rule 90 and 150. The control signals corresponding to a CA configuration are stored in the ROM and loaded into the PCA via the DATA BUS.

ROM(Read Only Memory): It is of size l x L (l words, each of L bits), and it stores the control signals for the PCA.

Control Unit: It consists of several counters to generate different types of control signals for PCA and ROM. The control sequences of the circuit are described in the Algorithm 1

below. Program Counter (PC) is ($\log_2 l$)-bit (i.e., l is power of 2) up counter and is used to store address of the ROM where next PCA rule is present.

I/O Connection: It is an input/output unit for data transfer between PCA and outside world.

Only two external signals are required to operate the whole circuit, i.e., CLOCK for running the circuit and START for reset and start of the circuit. The working is explained through Algorithms 1 & 2 given below.

Algorithm 1:

Step 1: Reset all counters in the Control Unit.
Step 2: Load PCA with L-bit initial seed from I/O connection.
Step 3: Read the ROM control word and configure the PCA.
Step 4: Run PCA for one cycle.
Step 5: Read pseudorandom pattern from I/O connections and increment PC by 1.
Step 6: If (CLOCK active) then go to Step 3.
Step 7: Stop.

In the above scheme, the PCA configured by a rule (stored in the ROM) is assumed to run one cycle only. By using some extra ROM bits and additional control, we can specify the number of cycles the PCA should run for each configuration. Such modification can substantially enhance the quality of encipher.

Two Stage PCA as a Key Stream Generator:

In the ROM based Key Stream Generator, the main disadvantage is the increased area overhead with lower speed of operation due to use of ROM for storage of control signals. This can be avoided by replacing the ROM with another maximal length 90/150 rule PCA (i.e., PCA_2) as shown in Fig 6.3. A single chip can be fabricated with PCA_1,

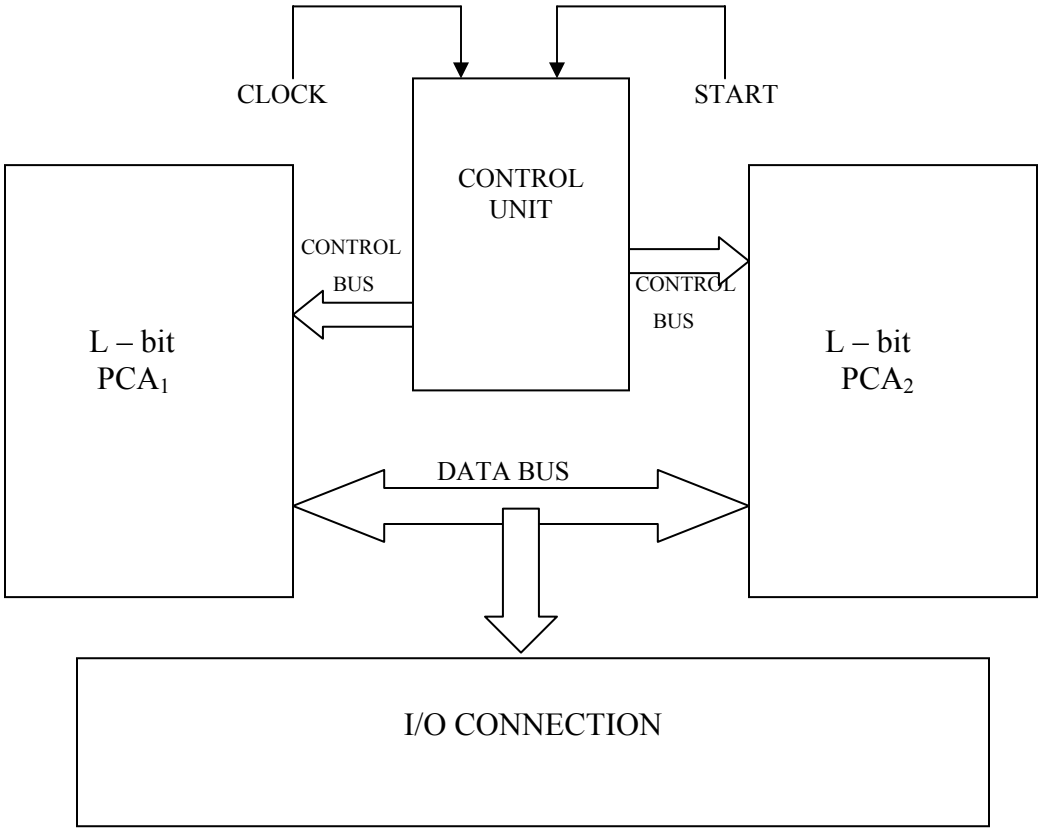


Fig Two stage PCA baed pseudorandom pattern generator

PCA_2 and the CONTROL UNIT. The control signals (R) to configure PCA_2 and the input seed (I_2) for PCA_2 can be concatenated to the input seed (I_1) of PCA_1 to form the key (i.e., <R, I_2, I_1 >) for the key stream generator. The PCA_2 generates the control signals to configure PCA_1 (Program 1).

Description of the Circuit:

PCA_1 (Programmable CA_1): It is an L-bit null boundary, uniform or hybrid CA loaded with the rule 90 and 150. The control signals to configure PCA_1 are loaded from the output of PCA_2 via the DATA BUS.

PCA_2 (Programmable CA_2): It is an L-bit null boundary, uniform or hybrid CA configured with the rule 90 and 150. Rule (R) is part of the key and it is loaded into the PCA_2 via the DATA BUS.

Control Unit: It consists of several counters to generate different control signals for PCA_1 and PCA_2. The control sequences of the circuit are described in the Algorithm 2 below.

I/O Connection: It is an input/output unit for data transfer between PCA_1, PCA_2 and the outside world.

Algorithm 2:

Step 1: Reset all counters in the Control Unit.
Step 2: Configure PCA_2 with the control signals (R) from the I/O connections.
Step 3: Load PCA_1 and PCA_2 with initial seed I_1 and I_2.
Step 4: Run PCA_2 for one cycle.
Step 5: Configure PCA_1 with the control signals from the output of PCA_2.
Step 6: Run PCA_1 for one cycle.
Step 7: Output pseudorandom pattern from I/O connections (i.e. output of PCA_1).
Step 8: If (CLOCK active) then go to Step 4.
Step 9: Stop.

The enciphering process using this type of generator fails if PCA_1 goes to all zero graveyard state. Analogous to modified LFSR design, with some extra logic it is possible to design the PCA_1 to have a transition out of all- zero state. On the other hand, it is necessary to avoid a situation where PCA_2 enters in a graveyard state resulting in PCA_1 being configured with the same rule all through out. So, the user of the scheme must avoid such a key combination from the simulation study.

CHAPTER 6

CELLULAR AUTOMATA IN IMAGE SPLITTING

Let each pixel vector in the image proposed to be securely stored or communicated be of 8 - bit length. We propose to use an 8 - neighborhood cellular automata for splitting this 8 - bit
pixel. For this purpose, we use a two - dimensional, Hybrid, Programmable Cellular Automata.

c_4	s_1	c_1
s_4	S_c	s_2
c_3	s_3	c_2

In this scheme, we number those cells which have a side common with the central cell to evolve according to Program - 1 and cells which have a corner common with the central cell evolve according to Program - 2. In this formation we have tried to generalize the LFSR approach followed in Chapter 3. Please refer to figure for the circuit for simple implementation of two-dimensional hybrid programmable automata used for generating splits of an 8-bit pixel vector. The two parts of the Hybrid use different programs Program 1 and Program 2 for evolution. The close connection with LFSR and Cellular Automata has been brought out [ ]. While the cells having a side common with the central cell are marked as s_1, s_2, s_3 and s_4, the ones which have only a corner common with the central cell to be marked as c_1, c_2, c_3 and c_4. While s_1, s_2, s_3 and s_4 follow a program governed by $s_1 + s_4$ replacing s_1 followed by a clockwise shift in s_2, s_3, s_4 ---- c_1, c_2, c_3 and c_4 follow a program governed $c_3 + c_4$ replacing c_4 followed by the following rule:

Program 1:

s_1, s_2, s_3 and s_4 follow the following program

1. The initial seed for the cells s_1, s_2, s_3 and s_4 is 1101.
2. S_c is 1.
3. At time t+1 : S_1 is replaced by $S_1 \oplus S_4$ at time t

   S_2 is replaced by S_1 at time t

   S_3 is replaced by s_2 at time t

   S_4 is replaced by s_3 at time t

4. S_c at t+1 = $S_1^{t+1} \oplus S_2^{t+1} \oplus S_3^{t+1} \oplus S_4^{t+1}$

In short $S_1^{t+1} = S_1^{t} \oplus S_4^{t}$

$$S_c^{t+1} = S_1^{t+1} \oplus S_2^{t+1} \oplus S_3^{t+1} \oplus S_4^{t+1}$$

and $S_2^{t+1} = S_1^{t}$, $S_3^{t+1} = S_2^{t}$ and $S_4^{t+1} = S_3^{t}$

Program 2:

c_1, c_2, c_3 and c_4 follow the following program

1. At time t+1 : c_1 is replaced by $c_1 \oplus c_4$ at time t

   c_2 is replaced by c_1 at time t

   c_3 is replaced by c_2 at time t

   c_4 is replaced by c_3 at time t

2. At time t+1: S_c is replaced by $c_1 \oplus c_2 \oplus c_3 \oplus c_4$ at time t+1

   i.e. $S_c^{t+1} = c_1^{t+1} \oplus c_2^{t+1} \oplus c_3^{t+1} \oplus c_4^{t+1}$

   $c_1^{t+1} = c_3^{t} + c_4^{t}$

   $c_2^{t+1} = c_1^{t}$

   $c_3^{t+1} = c_2^{t}$

   $c_4^{t+1} = c_3^{t}$

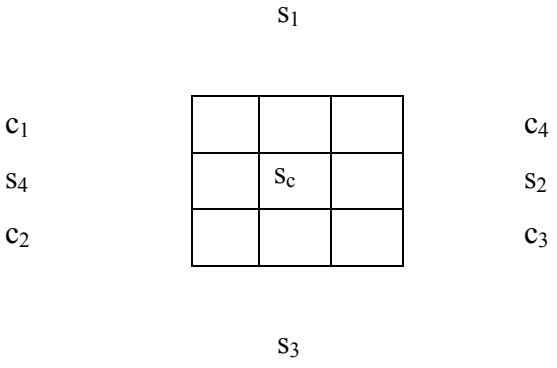
The first few assigned values are shown. Let the starting position be:

s_1	s_2	s_3	s_4	c_1	c_2	c_3	c_4
1	1	0	1	1	1	0	1

The automata evolve through two different rules, one each for 4 – neighborhood case.

The number of splits that can be achieved through cellular automata evolution are much more than with LFSRs. Also it is possible to deal with pixel vectors of greater length, thus suggesting the use of cellular automata in quality colour images.

Let the image pixel vector be eight bit binary vector 1 1 0 1 1 0 1 1 – we start with the configuration



Thus the neighbour of s_c (the central cell) consists of two types:

(i) s_1, s_2, s_3, s_4 are cells which have a side common with s_c

(ii) c_1, c_2, c_3, c_4 are cells which have a corner common with s_c

We start with the following configurations:

t = 0

We start with some pixel vector 0110 0111

At t = 0 s_1 has a 0, s_2 has a 1, s_3 has a 1 and s_4 has a 0.

$s_1 \oplus s_2 \oplus s_3 \oplus s_4 = s_c = 0$ (boundary)

1	0	0
0	0	1
1	1	1

t = 1

Apply the law. Let s_1 get $s_1 \oplus s_4$ and s_2, s_3 and s_4 get the values s_1^0, s_2^0, s_3^0 respectively, i.e. $s_2^{t1} \rightarrow s_1^{t0} \rightarrow s_3^{t1} \rightarrow s_2^{t0}$ and $s_4^{t1} \rightarrow s_3^{t0}$ and c_1, c_2, c_3, c_4 remain unchanged

1	0	0
1	0	0
1	1	1

t = 2

Apply the law c_1 gets $c_3 \oplus c_4$ and c_2, c_3 and c_4 at (t+1) get a shift of c_2, c_3 and c_4 at t as follows: $c_2^{t+1} = c_1^{t}$, $c_3^{t+1} = c_2^{t}$, $c_4^{t+1} = c_3^{t}$ and s_1, s_2, s_3, s_4 remain unchanged

and $s_c = c_1 \oplus c_2 \oplus c_3 \oplus c_4$

1	0	0
1	1	0
1	1	1

t = 3

Apply the law s_1 gets $s_1 \oplus s_4$ and s_2, s_3 and s_4 get the values of s_1, s_2, s_3 at t=2 and c_1, c_2, c_3, c_4 remain unchanged and

$s_c = s_1 \oplus s_2 \oplus s_3 \oplus s_4$

1	1	0
1	0	0
1	0	1

Repeatedly applying the above two laws of evolution; we get at

t = 4

1	1	1
1	0	0
1	0	1

t = 5

1	1	1
1	1	0
1	0	1

t = 6

0	1	1
1	1	0
1	0	1

t = 7

0	0	1
0	1	1
1	0	1

t = 8

0	0	1
0	0	1
0	0	1

t = 9

0	0	1
0	1	0
0	1	1

t = 10

0	0	1
0	1	0
0	1	0

t = 11

0	0	1
1	1	0
0	0	0

t = 12

1	0	1
1	1	0
0	0	0

t = 13

1	1	1
0	1	0
0	0	0

t = 14

0	1	0
0	1	0
1	0	0

t = 15

0	1	0
0	0	1
0	0	1

t = 16

0	1	0
0	1	1
1	0	0

t = 17

0	1	0
0	1	1
1	1	0

t = 18

1	1	1
0	0	1
0	1	0

t = 19

1	1	1
1	0	1
0	1	0

t = 20

1	1	0
1	0	1
1	1	0

t = 21

1	0	0
1	1	1
1	1	0

t = 22

0	0	0
1	0	1
1	1	1

t = 22

0	0	0
1	0	1
1	1	1

t = 23

0	0	0
1	1	1
1	1	1

t = 24

1	1	1
1	1	0
0	1	1

t = 25

1	0	1
1	1	1
0	0	1

t = 26

1	1	0
1	0	0
0	1	1

t = 27

1	1	0
0	0	0
0	1	1

t = 28

0	1	1
0	1	0
1	1	0

If we superimpose 29 stages from t = 0 to t = 28 modulo-2 we get the original pixel vector, i.e.

1	1	1
1	0	1
1	0	0

Proceeding like this one complete cycle of the Programmable Hybrid Cellular Automata can be written down

	s		**c**
	1101		1011
t = 1	0110	t = 0	0111
t = 3	0011	t = 2	1111
t = 5	1001	t = 4	1110
t = 7	0100	t = 6	1100
t = 9	0010	t = 8	1000
t = 11	0001	t = 10	0001
t = 13	1000	t = 12	0010

t = 15	1100	t = 14	0100
t = 17	1110	t = 16	1001
t = 19	1111	t = 18	0011
t = 21	0111	t = 20	0110
t = 23	1011	t = 22	1101
t = 25	0101	t = 24	1010
t = 27	1010	t = 26	0101
	1101		1011

It can be seen that the above scheme leads to two groups of splits to be distributed. One group consisting of splits shown at t = 0, 2, 4, …, 26 and the other group splits at t = 1, 3, 5, …, 27. Each group separately can reach the original pixel vector by adding mod-2 fourteen pixel vectors of one group. The shares of Group 2 also lead to the original pixel vector 1101 1011 by addung mod 2.

Now having more programs p_3,p_4,---, p_5, p_6 we can enlarge the cycle length to desirable levels and when all the n agents get together, only then they are able to arrive at each of the correct pixels.

It is relevant to remark here that the entry in the cell S_c can be easily used for making the split sequence to be more complicated and for increasing the number of splits using the following rule:

(i) When S_c is zero, use the s cycle

(ii) When S_c is 1, use the c cycle

instead of the alternate use of s & c cycles regularly.

We have intentionally resisted from using the value of S_c as

(i) it would make mathematical analysis of the splits so generated extremely complicated and consequently

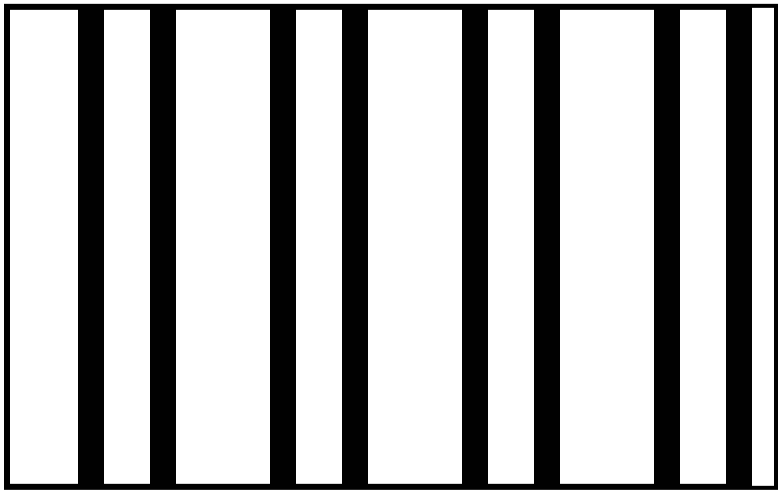
(ii) it would be difficult to compare the results achieved by the two techniques of LFSR and 2-dimensional CA worked out here.

The comparison of results achieved through the above technique is presented in Chapter 7.

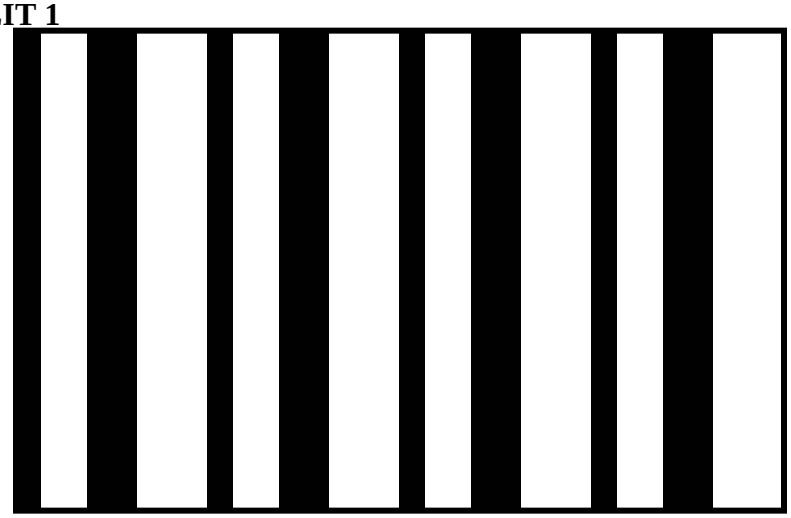
CHAPTER 7

RESULTS

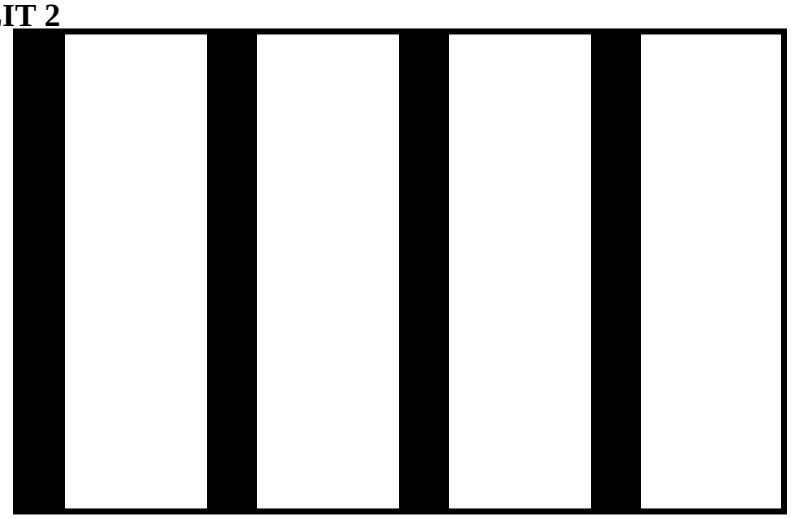
ORIGINAL IMAGE



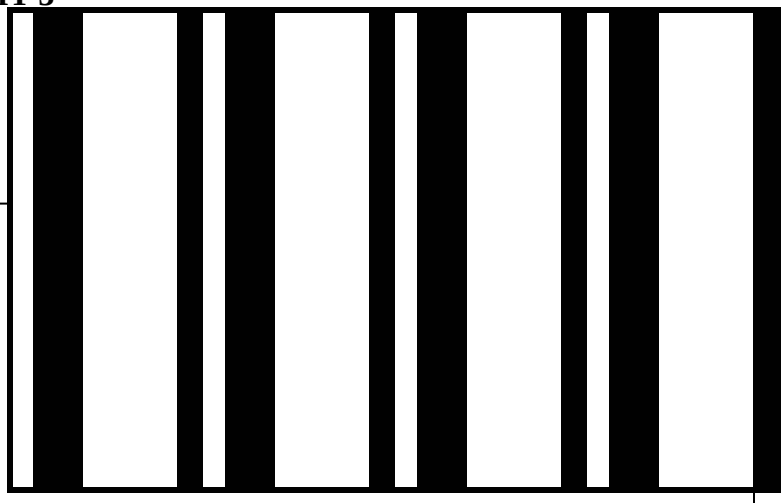
SPLIT 1



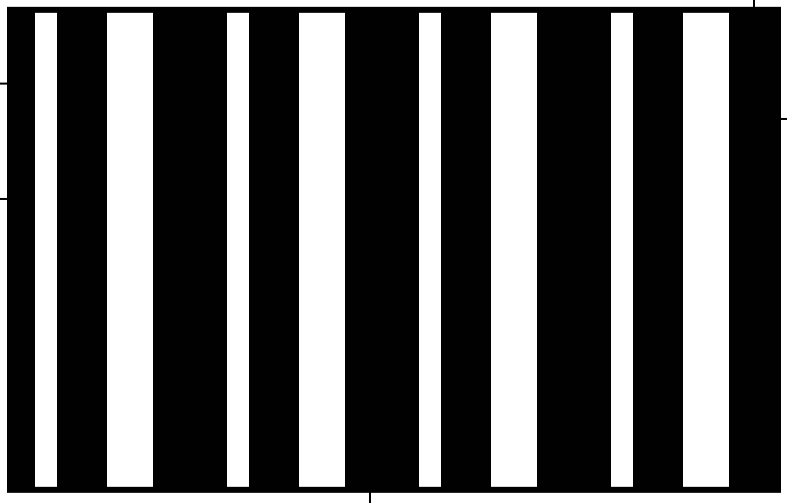
SPLIT 2



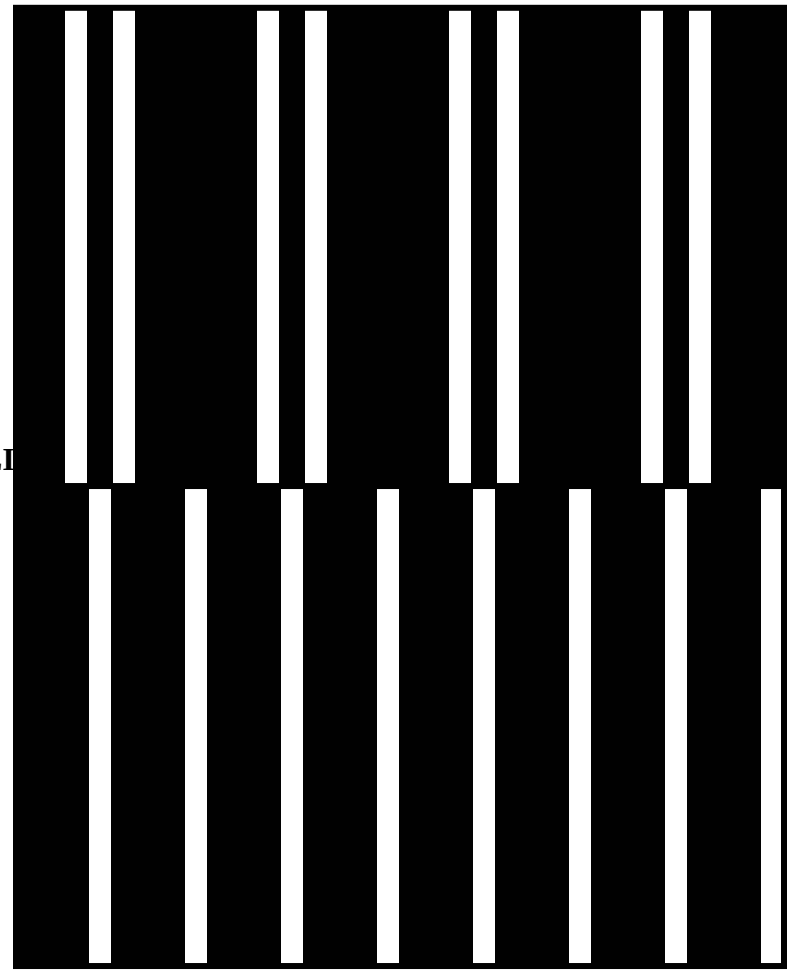
SPLIT 3



SPI

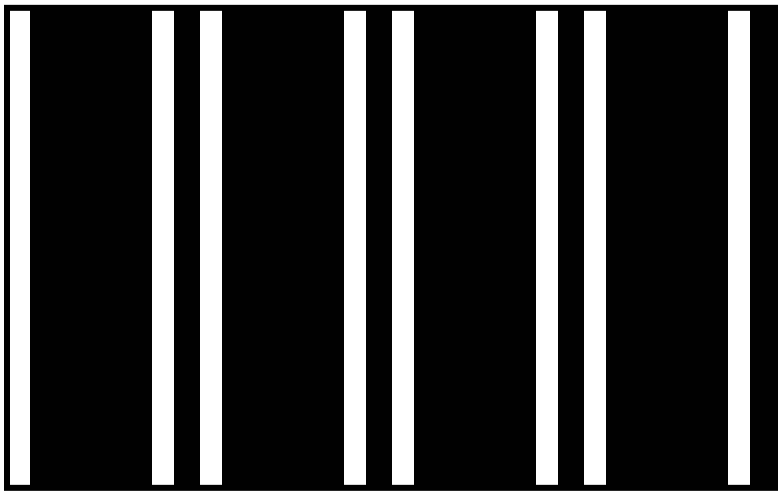


SPLIT 5

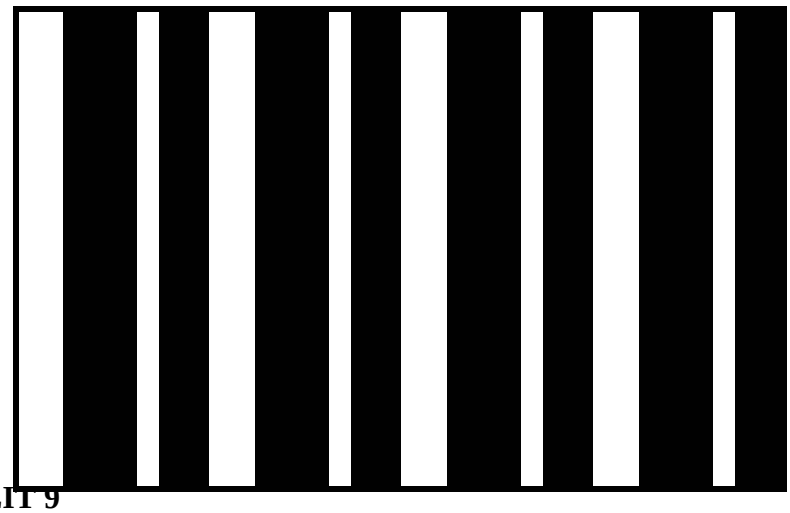


SPLI

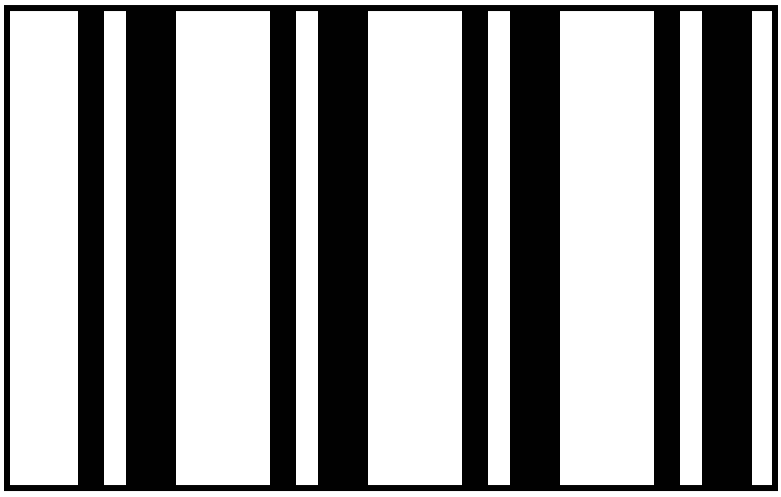
SPLIT 7



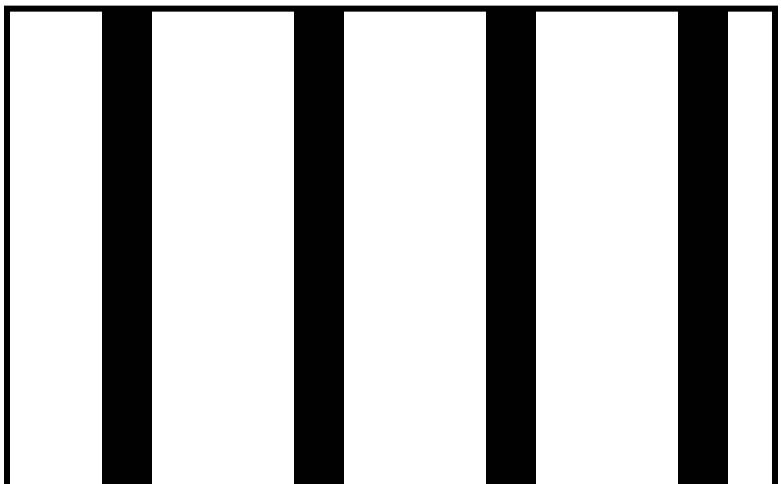
SPLIT 8



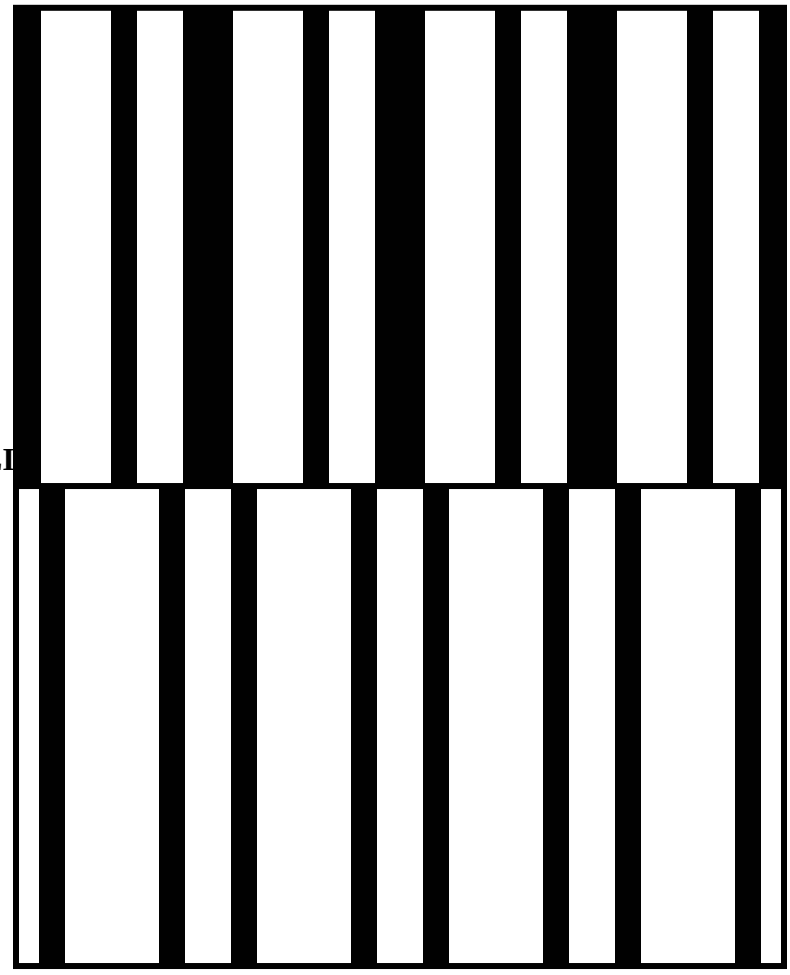
SPLIT 9



SPLIT 10

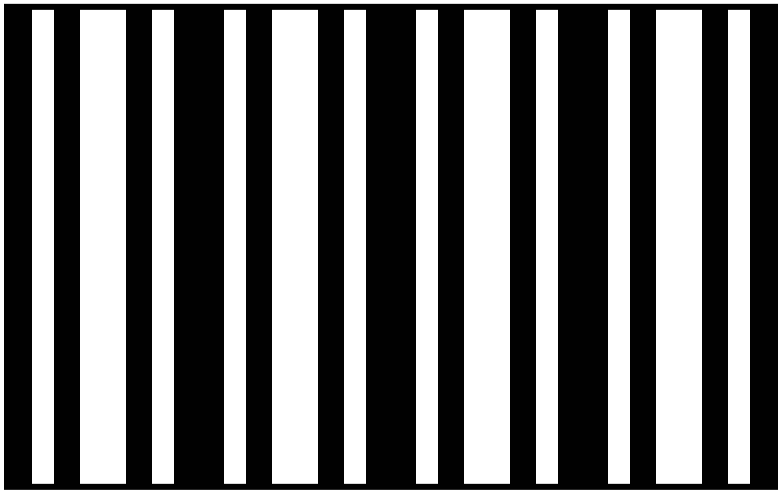


SPLIT 11

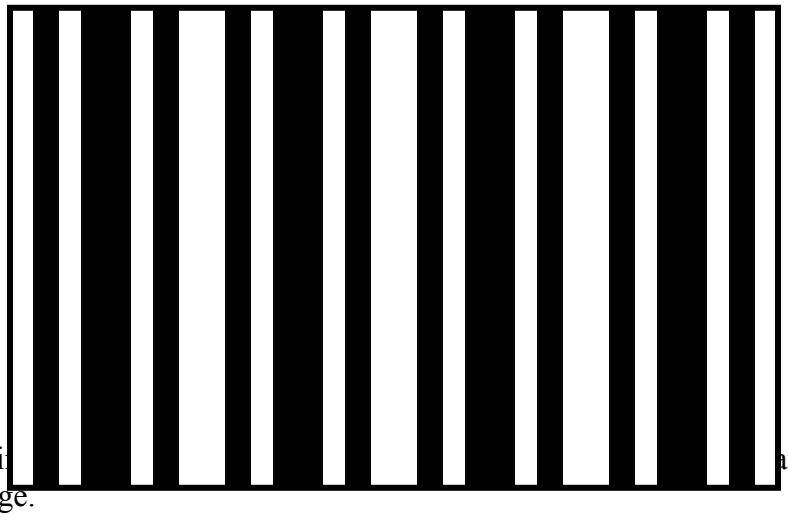


SPLI

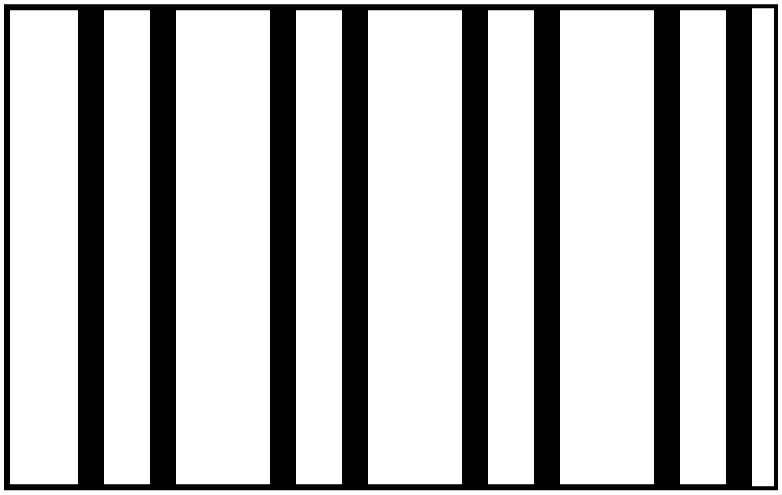
SPLIT 13



SPLIT 14



The i ame as the original image.



COMPARISON OF RESULTS: LFSR AND TWO-DIMENSIONAL HYBRID PCA

1. It is well known that LFSR's are only particular cases of Cellular Automata. See for example Nandi etal [3] and Das et al [10]. By using an r-stage LFSR one can generate $(2^r - 2)$ splits of each pixel vector, which together would reproduce the original $(2^m - 1)^{th}$ pixel vector. However we can generate split images using Cellular Automata with many more splits, which can be divided to various agents – who can get the original image only when all of them get together.

2. One can easily increase the number of splits in two ways by use of Cellular Automata by:
   (i) **Vertical Splitting**: As demonstrated above. The number can be increased by choosing Program I and II in such a way that the cycle length of the CA is larger.

   (ii) **Horizontal Splitting**: The dimension of the pixel vector can be increased by either:

(i) Increasing the size of the CA – for example a 5 x 5 or 7 x 7 square can easily be used as two dimensional CA to increase the pixel size to 10, 12 or any other convenient size – If the dimensions of the pixel vectors are p_1 and p_2, we can increase the cycle length to $(2^{p1} – 2) \times (2^{p2} – 2)$.

(ii) By taking a larger number of pixel vectors in each. If the size of the pixel vector is s we may increase the dimension of the pixel to s x n where n is the number of pixels in the pixel vector.

(iii) By interspersing the pixel bits and progressing the evaluation by different programs one can achieve much more security.

It is clear from the above discussion that cellular automata offer much superior techniques of secure communication of images, in terms of :

(i) Number of splits: This is much larger in case of cellular automata based scheme than in case of LFSR.

(ii) Cellular automata technique can handle much longer pixels and can therefore be useful in high quality colour images where the constituent pixels are of much longer length to cater for different colours.

(iii) As cellular automata are capable of creating highly complex patterns, these can be used for providing much higher level of security than in the case of LFSR's where 2^m-1 splits security from a cryptanalyst's point of view is only m, as m bits are sufficient to predict the rest of the bits in a period of 2^m-1.

Thus it can be concluded that Cellular Automata provides endless possibilities of generating splits by adopting various variants of the CA two dimensional, programmable and Hybrid CA's.

CONCLUSION & FURTHER SCOPE OF WORK

Many methods of securing images for secure preservation and communication have been suggested in literature. For reasons explained in Chapter 2, we choose to study the methods of image splitting for this purpose.

We studied two methods for secure storage and secure communication for images in this dissertation. Two methods of splitting images studied here are

(i) LFSR based splitting

(ii) Image splitting through a two-dimensional 8-neighbourhood hybrid cellular automata in which each half of the 8-bit pixel (8-neighbourhood) cellular automata evolve according to different rules.

The following problems are suggested for investigation by the work presented in this dissertation:

1. The number of the splits generated and the scheme of split distribution if the entries at S_c are taken to drive the PCAs, for example PCA_1 is activated when $S_c = 0$ and PCA_2 is activated when $S_c = 1$.
2. To discuss the security aspects of the above schemes.

3. To study the strength of encryption schemes suggested by the above two-dimensional Hybrid CA driven schemes.

REFERENCES

1. Ram Ratan : "Techniques of Securing Images", Proceedings of the National Conference on Information Security. BVCOE, New Delhi Jan 8 – 9, 2003
2. Bruce Steiner Applied Cryptography
3. Paul Bourke [1998], BMP image format
4. I.J.Kumar Cryptology, System Identification and key clustering Agean Park Press, 1997
5. Golomb S W [1980] On the classification of Balanced Binary Sequences IEEE Transactions on Information Theory Vol 26 pp 730 – 732
6. H. Becker and F. Piper [1982] Cipher Systems, Worthwood Books
7. Von Neumann J: The Theory of self reproducing automata Burks, University of Illinois Press, London, 1966.
8. Wolform S. et al, Algebraic properties of Cellular Automata, Communication Mathematical Physics vol 93, pp219-258

9. N A Packard, Stephen Wolform [1985], Two dimensional Automata Journal of Mathematical Physics, Vol 38, 1985 pp 901 – 942.
10. A K Das el al : Efficient Characterisation of Cellular automata E E Proceedings Vol 137, 1990, pp 81 – 87.
11. Pries W et al : Group properties of cellular automata and VLSI application, IEEE Transaction on Computers,1986, c – 35 (12) pp 1013 – 1024.
12. G. Alvarez Maranon et al [2003], Sharing secret color images using cellular automata with memory
13. S. Nandi, B.K. Kar and P. Pal Chaudhary IEEE Transactions on Computers, Vol 43, pp 1346 – 1357

APPENDIX

IMPLEMENTATION OF IMMAGE SPLITTING USING TWO DIMENSIONAL HYBRIDS CELLULAR AUTOMATA

```
//implementation of CA by LFSR
#include<stdio.h>
#include<graphics.h>
#include<iostream.h>
#include<conio.h>

void side_corner_arrayfill();
void side_corner_square(int,int, int);
void display();
void display_square(int);
void one(int);
void final();
void final_check();
int s[16][4],c[16][4],temp[28][9],temp1[14][9],temp2[14][9],f1[9],f2[9],x=0,y=0,r=0;
```

```
void main()
{
int i,j,k;
clrscr();

//initialization of side and corner array
s[0][0]=1;s[0][1]=1;s[0][2]=0;s[0][3]=1;
c[0][0]=1;c[0][1]=0;c[0][2]=1;c[0][3]=1;

        cout<<"side       and     corner arrays contents\n";

        side_corner_arrayfill();
        display();
        getch();
        int count=0;

        for(i=1;i<15;i++)
        {
          for(j=i;j<=i+1;j++)
          {
                side_corner_square(count,j,i);
                display_square(count);
                cout<<endl;
                getch();
                count++;

                if(count==27)
                {
                side_corner_square(count,1,i);
                display_square(count);
                break;
```

```
                }
            }
                if(count==27) break;
        }

    //  cout<<count;
        getch();
        final();
    //  final_check();
 // one(2);
//  display_square(2);
getch();
}
//this function will fill the square with side and corner
void side_corner_arrayfill()
{
int x,y,cnt=0;

 while(cnt<16)
 {
 x=0,y=0;
 for(int i=0;i<3;i++)
    s[cnt+1][i+1]=s[cnt][i];

 for(i=0;i<3;i++)
```

```
    c[cnt+1][i]=c[cnt][i+1];


 x=x^s[cnt][0];
 x=x^s[cnt][3];

 y=y^c[cnt][0];
 y=y^c[cnt][1];

 s[cnt+1][0]=x;
 c[cnt+1][3]=y;

 cnt++;
 }

 c[0][0]=1;c[0][1]=0;c[0][2]=1;c[0][3]=1;
}

void display()
{
int i,j;
  for(i=0;i<16;i++)
  {
          for(j=0;j<4;j++)
            cout<<s[i][j]<<" ";

   cout<<"  "<<i<<"          ";

          for(j=0;j<4;j++)
            cout<<c[i][j]<<" ";
   cout<<endl;
```

```
    }
}

void side_corner_square(int k, int j, int i)
{
temp[k][1]=s[j][0];temp[k][5]=s[j][1];temp[k][7]=s[j][2];temp[k][3]=s[j][3];
temp[k][0]=c[i][0];temp[k][6]=c[i][1];temp[k][8]=c[i][2];temp[k][2]=c[i][3];
  if ( k%2 == 0)
{
          x=0;
          for(int r=0;r<4;r++)
          {
                    x=x^c[i][r];
          }
}
  else
{
           x=0;
          for( r=0;r<4;r++)
          {
                    x=x^s[j][r];
          }
}

temp[k][4]=x;

}

void display_square(int k)
{
cout<<"At t="<<k<<"....."<<endl;
```

```
  for(int i=0;i<9;i++)
  {
  cout<<"       "<<temp[k][i]<<" ";
  if( ( i==2 ) || ( i==5 ) || ( i==8 ) )
  cout<<endl;
  }
}

void final()
{
  int x,i,j,k=0;
  for(i=0;i<28;i=i+2)
  {
        for(j=0;j<9;j++)
        {
          temp1[k][j]=temp[i][j];
          temp2[k][j]=temp[i+1][j];
        }
         k++;
  }
//   cout<<"the value of k is :::"<<k<<endl;

  for(i=0;i<9;i++)
  {
        x=0; y=0;
    for(j=0;j<14;j++)
    {
         x=x^temp1[j][i];
         y=y^temp2[j][i];
    }
  f1[i]=x;
```

```
  f2[i]=y;

 }
        cout<<"Group 1 (t=0,2,4,8,.....,26)"<<endl;
        for(i=0;i<9;i++)
        {
                cout<<"      "<<f1[i]<<" ";
                if( ( i==2 ) || ( i==5 ) || ( i==8 ) )
                cout<<endl;
        }
        cout<<"Group 2 (t=1,3,5,7,.....,27)"<<endl;
        for(i=0;i<9;i++)
        {
                cout<<"      "<<f2[i]<<" ";
                if( ( i==2 ) || ( i==5 ) || ( i==8 ) )
                cout<<endl;
        }
}
//int t[9];
/*void final_check()
{
cout<<"displaying final\n";
  int x;
  for(int i=0;i<4;i++)
  {
        x=0;
    for(int j=1;j<=14;j++)
    {

        x=x^s[j][i];
    }
```

```
    t[i]=x;
  }

 for(int k=0;k<4;k++)
  {
            x=0;
    for(int j=1;j<15;j++)
    {

         x=x^c[j][k];
    }
  t[i]=x;
  i++;
 }

        for(i=0;i<9;i++)
        {
                cout<<t[i]<<" ";
                if( ( i==2 ) || ( i==5 ) || ( i==8 ) )
                cout<<endl;
        }
} */
```