International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 5– No.2, January 2013 – www.ijais.org

LRUD Shift based Encryption Method using Matrices

Pankesh Bamotra
SCSE, VIT University
Vellore
Tamil Nadu, India

ABSTRACT
This paper deals with a novel and secure method of encryption using matrices and public key cryptography algorithm. Matrices have a very well-known property of invertibility i.e. if AX=B then $X=A^{-1}B$[1]. This paper exploits this property to achieve encryption of plain text into cipher text. The idea is simple and intriguing. A non-singular integer valued square matrix Ai.e. $\det|A| \neq 0$ is taken so that A is invertible and it is possible to obtain A^{-1}. This matrix A serves as the encryption matrix. The plain text to be encrypted is taken and ASCII values of each character is obtained and matrix X is formed such that $A_{q\ x\ q} \times X_{q\ x\ r} = B_{q\ x\ r}$. Before performing the multiplication of A and X to obtain cipher text ASCII value matrix B, circular shift operation is performed on rows and/ or columns of A. A single operation is stored as a string of 4 values viz. R32L9 or C8D61. The first character tells whether the circular shift operation is to be done on a row or a column. Second value tells the row or column number. Third character tells whether row is to be shifted left or right or if column is to be shifted then either up or down. The last value tells the number of times shift operation has to be performed. Successive operations are concatenated with a hyphen viz. R2R45-C6U8-C32D9 and encryption matrix A' is obtained. This key along with the matrix B' $(=A' \times X)$ is encrypted using RSA algorithm and sent to the other user who has the original matrix A. Receiver decrypts the message using private key, performs shift operation, finds the inverse of matrix A' as $A'^{-1}$and finds $A'^{-1} \times B'$ to eventually obtain X and hence the original plain text message.

General Terms
Cryptography, matrix theory, security, combinatorics

Keywords
Encryption, matrices,circular shift, RSA

1. INTRODUCTION
Encryption of text is not at all a modern concept but has been in use since ages. Encryption methods are usually application based.Highly confidential information requires highly reliable and secure encryption mechanism. The method described here is not intended to ensure integrity of the message but is only for encryption purpose. Various properties of matrices are used for this purpose. First is the property of matrix multiplication. Two matrices can be multiplied if and only if number of columns in first matrix is equal to the number of rows in second matrix.Second property that follows is inverse of a matrix. A square matrix of size n×n is said to be invertible if its determinant is not zero or in other words it is non-singular. Putting in mathematical notations,

$$A^{-1} = \frac{1}{\det(A)}(adjoint\ of\ A)$$

where $A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$ is a square matrix of order n and $\det(A) \neq 0$. To formally describe the meaning of adjoint of a matrix two definitions are given as follows,

Definition.1 The $(i, j)^{th}$ cofactor of A, denoted by C_{ij}, is the number $(-1)^{i+j}A_{ij}$.

Definition.2 If A is a n×n matrix, then matrix $B = [b_{ij}]$ with $b_{ij} = C_{ji}$, for $1 \leq i, j \leq n$ is called the adjoint of A.

As given in the paper title LRUD shifting is used which shortened notation for left-right-up-down circular is shifting. In combinatorics, the term circular shifting [2]refers tooperation of rearranging the entries in a tuple, either by moving the final entry to the first position, while shifting all other entries to the next position, or by performing the inverse operation.For rows of a matrix circular shift operation can be done either in left or right direction while for columns it is possible only in up or down direction. Hence the name is LRUD shifting. These operations are performed on rows and columns and each operation is saved as a string of 4 values as mentioned in the abstract part. Successive operations are concatenation of these strings with hyphen in between them. For example a 2 operation key could be R2R45-C6U8 with A being a 64×64 matrix. This key is interpreted as, firstly circular shift row 2 of matrix A towards right 45 times and then circular shift column 6 of matrix obtained from the previous operation upwards 8 times. After performing all such operations matrix A' is obtained.

2. METHODOLOGY
To begin with a square matrix A of order n was chosen. Matrix A has to be an invertible matrix, so the matrix A has non-zero determinant value. Using the traditional names of sender as Alice and receiver as Bob theprocedure is as follows. Alice after choosing an invertible matrix A chooses the number of shift operations to be performed. For a matrix of order n as total of $n^2(n-1)^2$ operations can be performed that makes this algorithm a variable key algorithm because each time Alice can choose the number of operations to be performed on the matrix. As explained earlier these operations are stored as a concatenated string of individual operations. After these operations are performed on the original matrix A, Alice takes the ASCII values of the plain text to be encrypted and stores into a matrix X of size $n \times \lceil l \div n \rceil$ where l is the length of text to be encrypted. Alice then multiplies the two matrices A' and X and obtain a matrix B of the same size as of X. To send the encrypted message Alice concatenates three things so that Bob can properly decrypt the message. She takes the concatenated string of operations, number of columns in matrix B and the list of elements of matrix B. All these things are separated by suitable flags and encrypted using the public key of Bob generated using the well-known

RSA[3] algorithm. On the receiver side Bob receives the encrypted message. He decrypts the message using his private key. He retrieves the three things mentioned earlier. One thing to be noted is that this protocol assumes that both Alice and Bob are having the matrices A. Bob then performs similar operations on matrix A from the concatenated string of operations to obtain matrix $A^{'}$. Then he finds the inverse of this matrix $A^{'-1}$ and multiplies with matrix obtained by suitably arranging the matrix B obtained from the message received. This will lead him to matrix X which can then be translated back to plain text. Both can use latest matrix $A^{'}$ for future communication.

3. IMPLEMENTATION IN PYTHON

```
1  import random
2  import numpy
3  import re
4  import RSA
5  def randomDigits(digits):
6      lower = 10**(digits-1)
7      upper = 10**digits - 1
8  return random.randint(lower, upper)
9  def zero(m,n):
10     new_matrix = [[0 for row in range(n)] for col in
range(m)]
11  return new_matrix
12  def show(matrix):
13  for col in matrix:
14  print col
15  def mult(matrix1,matrix2):
16  if len(matrix1[0]) != len(matrix2):
17  print 'Matrices must be m*n and n*p to multiply!'
18  else:
19         new_matrix = zero(len(matrix1),len(matrix2[0]))
20  for i in range(len(matrix1)):
21  for j in range(len(matrix2[0])):
22  for k in range(len(matrix2)):
23                     new_matrix[i][j] +=
matrix1[i][k]*matrix2[k][j]
24  return new_matrix
25  dim=64
26  matrix=[]
27  index=0
28
mat_in_file=open('C:\Users\Pankesh\Desktop\Matrix.txt','r
')
29  for line in mat_in_file.readlines():
30     matrix.append([])
31  for x in line.split():
32         matrix[index].append(int(x))
33     index += 1
34  matrix_ENC=numpy.matrix(matrix)
35  plain_text=raw_input("Enter the plain text to be
encrypted :\t")
36  text_length=len(plain_text)
37  plain_matrix_no=[ord(char) for char in plain_text]
38  no_o_array=0
39  if len(plain_matrix_no)%dim == 0:
40     no_o_array=text_length/dim
41  else:
42     no_o_array=int(text_length/dim)+1
43  no_o_ele=0
44  plain_2d_matrix=[]
45  for i in xrange(no_o_array):
46     plain_2d_matrix.append([])
47  for j in xrange(dim):
48  if no_o_ele != (text_length):
49
plain_2d_matrix[i].append(plain_matrix_no[no_o_ele])
50             no_o_ele +=1
51  else:
52             plain_2d_matrix[i].append(0)
53
plain_2d_matrix_numpy=numpy.matrix(plain_2d_matrix)
54  fkey_length=4
55  def get_key_length():
56     key_length=raw_input("Enter no. of operations:\t")
57  return int(key_length)
58
59  def get_key():
60     key=raw_input("Enter the key in terms of quadruple
viz. [ R|C,1-64,L|R|U|D,1-63] > R3L9-C25U8-C2D39-
R8R8:\n")
61  return key
62  key_length=get_key_length()
63  while key_length%4!=0:
64     key_length=get_key_length()
65  key=get_key()
66  keys=key.split('-')
67  while len(keys)!=key_length:
68     key=get_key()
69     keys=key.split('-')
70  def
valid(row_col,row_col_no,left_right_up_down,shift_n_times)
:
71     flag=False
72  if row_col == "R" and 1 <= row_col_no <= dim and (
73         left_right_up_down == "L" or left_right_up_down
== "R" and 1 <= shift_n_times <= dim - 1):
74         flag=True
75  if row_col == "C" and 1 <= row_col_no <= dim and (
76         left_right_up_down == "U" or left_right_up_down
== "D" and 1 <= shift_n_times <= dim - 1):
77         flag=True
78  return flag
79  def shift_row(row_no, n):
80     actual_row=row_no-1
81     matrix[actual_row]=matrix[actual_row][n:] +
matrix[actual_row][:n]
82  def flatten(*args):
83  for x in args:
84  if hasattr(x, '__iter__'):
85  for y in flatten(*x):
86  yield y
87  else:
88  yield x
89  def shift_col(col_no, n):
90  global matrix
91     matrix_NUM=numpy.matrix(matrix)
92     matrix_NUM=matrix_NUM.T
93     matrix=numpy.array(matrix_NUM).tolist()
94     shift_row(col_no,n)
95     matrix_NUM=numpy.matrix(matrix)
96     matrix_NUM=matrix_NUM.T
97     matrix=numpy.array(matrix_NUM).tolist()
98  inc=0
99  for i in xrange(key_length):
100     digits=re.findall(r'[0-9]+', keys[i])
101     R_C_L_U_D=re.findall(r'[R|C|L|U|D]', keys[i])
```

```
if valid(R_C_L_U_D[0],int(digits[0]),R_C_L_U_D[1],int(digits[1])):
        times=int(digits[1])
if key[inc]=="R":
if key[inc+2]=="R":
                times=-times
            shift_row(int(key[inc+1]),times)
if key[inc]=="C":
if key[inc+2]=="D":
                times=-times
            shift_col(int(key[inc+1]),times)
else:
print "Wrong Key Formation - Correct [ R|C,1-64,L|R|U|D,1-63]"
encryption_matrix=numpy.matrix(matrix)

cipher_text=encryption_matrix*plain_2d_matrix_numpy.T

cipher_message=list(flatten(numpy.array(cipher_text).tolist()))
message_2b_sent=str(cipher_message).strip('[]')
pubkey, privkey = RSA.keygen(2 ** 64)
print('=' * 100)
msg = "KEY_START__"+key+"__KEY_END__"+str(no_o_array)+"__MSG_START__"+message_2b_sent+"__MSG_END"
h = RSA.encode(msg, pubkey, 1)
p = RSA.decode(h, privkey, 1)
print('=' * 100)
print "Plain Msg to be sent: "
print(msg)
print('-' * 100)
print "Encrypted Msg to be sent: "
print repr(h)
print('=' * 100)
decrypted_key=p.split("__")
SHIFTING_KEY=str(decrypted_key[1]).strip("u'")
no_o_array=int(str(decrypted_key[3]).strip("u'"))
ENCRYPTED_MSG=str(decrypted_key[5]).strip("u'")

ENCRYPTED_MSG=ENCRYPTED_MSG.replace(',','').split(" ")
ENCRYPTED_2D_MATRIX=[]
no_o_ele=0
index=0
for i in xrange(no_o_array):
    ENCRYPTED_2D_MATRIX.append([])
    index=i
for j in xrange(dim):

ENCRYPTED_2D_MATRIX[i].append(int(ENCRYPTED_MSG[index]))
        index +=no_o_array
        no_o_ele+=1

ENCRYPTED_MSG=numpy.matrix(ENCRYPTED_2D_MATRIX).T

plain_text_recovered_list=numpy.array((encryption_matrix.I*ENCRYPTED_MSG).T).tolist()

plain_text_ascii=numpy.array(numpy.rint(numpy.abs(plain_text_recovered_list))).tolist()
actual_plain_text=""
for ascii_value_element in plain_text_ascii:
for ascii_value in ascii_value_element:
if ascii_value>0:
            actual_plain_text+=str(chr(int(ascii_value)))
print "Decrypted text :-\n",actual_plain_text
```

Note: RSA.py has not been included

4. RESULT

In a sample run with matrix A or order n = 64 was taken and the output was as shown below :-

Enter the plain text to be encrypted : *The problem isn't the Internet. The problem is the horribly insecure computers attached to the Internet.*

Enter no. of operations : *8*

Enter the key in terms of quadruple viz. [R|C,1-64,L|R|U|D,1-63] > R3L9-C25U8-C2D39-R8R8:

R12L9-R56R9-C4U55-R32L7-C63D7-R2L45-R6R60-C6U45

Plain Msg to be sent:

KEY_START__R12L9-R56R9-C4U55-R32L7-C63D7-R2L45-R6R60-C6U45__KEY_END__2__MSG_START__1834544, 1337282, 1623133, 1224102, 1909734, 1125149, 1475997, 1033510, 1496190, 1018634, 2018857, 1302755, 2004275, 1237261, 1659189, 1097920, 1952101, 1316477, 1826417, 1318375, 2286842, 1402102, 1946031, 1091303, 1723829, 1222494, 1904809, 1187619, 1736257, 1269208, 1558951, 967769, 1684807, 1215314, 1671468, 1129749, 1428909, 793472, 1863473, 1090548, 2142459, 1197649, 1650908, 851131, 1779642, 1227430, 2307479, 1402936, 1806224, 1268525, 1757853, 908742, 1732359, 1160997, 1782153, 1067480, 1872091, 1081062, 1866687, 1503432, 1695913, 1196393, 1826153, 1224043, 1633271, 1000557, 1791986, 911289, 1904894, 1191340, 1698194, 1262844, 2081148, 1611992, 1914818, 1382358, 2212078, 1434227, 1938624, 1325327, 1701981, 930700, 2064776, 937547, 1441466, 923877, 1658078, 1054027, 2063384, 1130256, 1292635, 739008, 1559323, 991824, 1463358, 827579, 1734296, 1219273, 1950518, 1222260, 1499886, 1147414, 2077106, 1447225, 2242602, 1316584, 1817398, 1165107, 1920194, 1262992, 1467509, 871960, 1470831, 938968, 1864842, 883755, 1607932, 1111523, 1705166, 1081365, 1767416, 1346150, 1559422, 1199135, 1859281, 1306843, 1364113, 1061997__MSG_END

Encrypted Msg to be sent:

'\x03\xa2)s\x81\xad4\x9dpD^\xd87\x18[\xe9\x00c\xac\xc4\x80\xf3\xc5sA\x05mLZ\xdd\xae\xd8\x02\xb36\xfd\xabsMw\xa4\xf4+\x93K^}>\x03\x8d#8t\t\x85V\xb4\x85\x1aw\x0e\xac\xfc\xd5\x00\xec\xb7\xf2\xc0\x9f-\x10\xc0\xc8u\xaf\x1cv,\x8b\x01\xbc\xdc\xf6\xa32\x8c\xdcK\xaa\x96\xf2G\x05\x13\xbc\x03#\xa2\x8a\x89\xf3M<\xe1\x94\x18%\xe0\xaeU\x95\x019\xdb\xb7\x06\x84&5sm\x1e\xc1\x95\tf\x04\x01\x12=w\x16b{\x12^S\xc7G\x83\xccT"\x01\xce\xcb\xeek&Z\xeb\xbe\x837\x89,R\x05\xb7\x00\x97t|\xea<\xcf\x99\xcf\xbc\xc1\x10\x9b\xbe\xa8\x81\x03\xfe\xbf\x0bF\xef\xc8<\xbc\\\xb4\xa8\xca\xac\xcdS\x00\xe9\xe5\'\xe6\x81m%\x11\xbf(\xde\x80\xaa\x96\xe4\x01\x93\x05zK~e\x17\x84\x05@\xea\xed\xc6p\x00|\xff\x006\t\xbeG"\xc0b\x00\xd8\x10;\xad\x01,\xd1\xa8\xd

cr\xeb_\x01\xa7r-v
R\x87\x00\x81\xd2\x1d\xb8\xa5&7\x1c\xa1\xc6\n
\xd3B\x1f\x03\xb4\x8b&\xc3\x9fS\xc6\x18\xb6o4M\xb5\n`\x03
\xfe\x9cK\xa2l\xf8\x04|C\xba\x11J\xa7\x00$\xf7\xb4\xf0_\x87
xd0\x1ePE\xf2\xa0u\xa5\xc5\x00\xcb}\xac\x98\x05s\xfc2\x1f\x
ae\t\x8aA\x15\xf2\x01\x02\xdf\xb5TdHehV\xc86m&{\x19\x02
VP\x1f\x1a\xe7\x90<\x15@\xfb=Uies\x02\x0b\xe6N\xbf\xa0\x
92\xb4\x81\xbc\x1b\x86\xabb\xb8C\x02\xa2U\x83\x0cV\xcd\xf
f\x8d\xc7r+\x97\x9a\x99@\x02X\xbds\xcbh\x0b\xe9\xfa\xc5\x
02\xb4\x83\xfd\x96\x82\x00t\xbb\xde\x84G\xba\xa9@\x92\x9f
2`\xd1q\x02\x04~\x0cg\xe1\x9d{D\x9f\xee\x9do
\xcct\x8d\x01\xcd\x87\x9dP\x17\xd1\x11\x1c\x0b\xdc\xc0\x11
xa7\x14\x81\x03\xa4\xff\x89\xc5d\x13\xa3\x9b\x94\xa2\x9c\xa
3)\xc7\xb4\x03\xc9d\xe5\x18\xeb\x8b\xbe\xab\xfc\xf0`k\xaelj\x
02p\xff8\xba1\t\xd8\x9e\xf2\xfa\x00\xc1\x1a\xdd6\x01\xff`0\xb
3\x04\xf7\xbbb\x00\x9b\xe9\xf3\x98\xff\x18\x01;\xf0\xb2}Tm\x
1c\xb7\xa2[8\x0f\xe2^"\x04K\xfb\x80\x9axd\xcc@2u\xf2\xe4\x
c0y,\x01(\xf3\xe8\x05\x833}\xecu\x88.\x012\xaf8\x02\x7f\x80
x1f\x1eg)\xf4i\x9aC\xae\xb2F]\xfd\x03h\xe7rC\x91C\xad\xa9
xcb\x8d\xa8\xfdC\x9c\x037\xcf(J\xaf&M\x8f\xe4[\x9f4R%\x01
\x02\xef\xa5F\x19\xe7\xc9\x08u<"\xddX\x94\x9d\x8d\x02\x94
\x1a\xb8\xa4\xc9%\x96\xa5\x12\x13\x17j|\xdb;\x00|pIb\x92-
F\xf6\x07\xb8OX\xbd\x11\x02\x034\xe7\xbd\xc5\xc9@\xb5\xb
9\xd2\xd5\x9d\xdc\x85\xc3)\x03\xa1\xee\xf9h\xdc\xe3\xe8\xb5
<]\xb9(\xcb\x8fN\x01sI\x94\x85s\x9d\xbf\x8a\xb7\xd5\x03E"
x19\x1d\x02_\xcf\xb1\xc5pL\xbd6d7\xb1L\x93\x02\x8a\x01\x1
8\x05\x9f<\xa9\xba\xaey#)\xd1{\xc2\xa1\x04Z\xfd\xf9\xc9<5Z
\x8e\xd0$}\t\x13\x12J\x03\xf9c\xd9\x92\x10\xc4K\xe5\x14\x1d
=\x9e\xc5\x8c\xe4\x02\xfaeM\xe7\x06r\x80\xf2\x98]\x13jS\xe
7/\x01x\xaf\xf1\x1eR\xe2>\xeb\xc5\xdd\xc5\x8f\x82N\x01iAD
>\xf1\xbf\xd3>\x9f}\x99\xb0\xe6\x07\xfa\x00B\x0ev\xc7j\xe0
xfb\xa7F\x8d\x00\x0c\xaf\x06\xe8\x045\x96\xf6\xff\t\xad\xff\xa
1"\x1b\x16D\xc5\xfd~\x03\xd8wtX\xa3\'\x8a\xb1\xcd\x15\xbd4
t\x8e\x9c\x00\x0f+d.\xbbh\xee\xd5E=I\xa6\xcd\x89\x04]q\xc5
\xe5_!\\Y\xb7\xd6\xad\xd8H\x0eZ\x02v\xfe\xfc\xf5Q\x87\x07X
xf7\xacaTea\xc1\x02F\xb1\xb0\xddL\xeb\xce13\x88\x18\xab\x
e1\x93\xe5\x02<\xfd\xa4\x900L\x86Q\x94\x98\n|\xd2)\x1b\x0
2o\xed\x8b\xf6wh\xb8\xab\xdb\x11\t\xe6\xdc\x14\xf1\x00\xc5
xb0\x93\x84)\xfaT\xfb\xeb\xa30r<\xe9}\x01\x8eM\xa1!a\xfer*
\x8f\xa9\xb0Q\x03a\xb8\x02\xbe\xfaH\xe3\xcc\x03\xab\x83!\x
ac\xa2Sf\x10\xbd\x02_\xbd)]w"\xbcWr\x83\xb4\xbd^\xd1=\x0
1\xba\xac\x19\x10F8\x0fp\x13]\xb0\x87\xd0\xf5\xde\x03;\x95`
w\xef\xd2D\xcf\xd0b\xa6\x91\xb3u[\x01g\x90%\x9d7K\xff{\x0
b\xd8\x16\xdb\xf5\xd6u\x03\xb7f\xe7\x10\xbc\x02\x80J\xd6\x1
6\xd0\xf0/\xc0\xbe\x02\xf4b=&\x86\x9b4\xc4\xc1$>\x01\xd7
xb2d\x03\x99A\x03=\x06L\x1c\x90GJ\xc4U<y\xf5\x02\x99\xd
e\x9b\x7f\x94FE\xf6\x9a\xd5\xaf\xe4\t\xadR\x00V\xf22\xa2\xc
3\xf1o\xa2\x9e\xe5\x9d\xeeKP\x02\x04\x0f(\t\x00\xbf\x96\xb*
b\xec\xc3\xb5w\xbb\x11\x16\x00o\x97dU\x1e\x91\xc3_1\xca\x
f8\xef@i\xb9\x03\x06\xf0\x08\xb9\x0e\xa8="\x13\xd91\n12U
x00\xecw\x12\x8b\xfe\xf0\x08:\x9f:Y\xba}\x81P\x00{\xf1\x18R
\xe7m\xbaw\xb9\n\t\xd3\xc2K\x02\xc2\xf1\xe1~G\xbc\xed=\xb
7g\xa4\xa8\xdf\xca\x00\'!%\xf5\xe3x\xf5+\xc6\xe6\xc2\x87\x*
cb\xfa\xe1\x01)\xc4\xde\xe6\xa5\xff\xf6\xe0Bn\xa1\x03\xac\xa
2R\x00\x1a\xafE,\xa4\xea\xc8\xd1\xc0\xd6\x9f\xf4\xa6\x97q'

Decrypted text:-

The problem isn't the Internet. The problem is the horribly insecure computers attached to the Internet.

The code described above took 0.0390000343323 seconds to execute the procedure and has been optimized to keep the performance high which can be seen in the table below:-

Table 1 Comparison of performance vs. matrix order

Matrix Order (N)	Running Time (in sec.)
64	0.0390000343323
128	0.13499997139
256	0.339999866486
512	0.628999948502

The algorithm described till now can be summarized in the following block diagram Fig 1:-

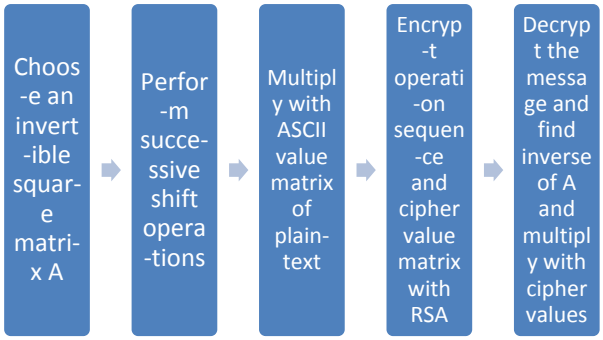

Fig 1: Algorithm block diagram

5. CONCLUSION

In nutshell it can concluded that the algorithm successfully encrypts the human readable message to be sent. The performance of the code is up to the mark. The encrypted message was retrieved and decrypted without any loss of information. Talking about the future prospects of the paper the algorithm can be used to encrypt gray scale images[4] as the image can be represented in terms of gray scale values.

6. REFERENCES

[1] Jin Ho Kwak and Sungpyo Hong, Linear Algebra, Second Edition

[2] Sesha Pallavi Indrakanti and P.S.Avadhani, Permutation based Image Encryption Technique, International Journal of Computer Applications (0975 – 8887), Volume 28 No.8, August 2011

[3] William Stallings, Cryptography And Network Security, 4/E

[4] Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing, 2nd edition, Prentice Hall