A Block Cipher using Key based Random Permutations and Key based Random Substitutions

K. Anup Kumar† and S. Udaya Kumar†

SreeNidhi Institute of Science and Technology, Hyderabad, India†

Summary
In this paper, we have developed a large block cipher by introducing the basic concepts of permutations and substitutions. The permutations and substitutions are key based. We have taken the key and the plaintext in the form of numbers and characters respectively. Where, each one is converted into its 8 bit binary equivalent based on its ASCII values. In the process of encryption, we have represented the plaintext as a block of 256 bits and developed a block cipher of 256 bits by using the classical feistel network. In each round, we have performed different key based permutations with the help of the equation that we have derived. Similarly, in each round, we have also permuted the key based S-Boxes. The cryptanalysis carried out at end shows that the cipher cannot be broken by any cryptanalytic attack.

Key words:
Block cipher, plaintext, permutation, substitution, XOR, encryption and decryption, SBox.

1. Introduction

In the development of Cryptography, majority of the block ciphers found in literature, are based upon feistel network. The basic elements of this type undergo a series of diffusions and confusions. This is achieved through permutation and substitution of plaintext that is to be encrypted. In the classical feistel network, which involves a round function, wherein the number of rounds is sixteen, provides good strength to ciphers.

In the present paper, our interest is to develop a large block cipher, using 16 rounds classical feistel network, which makes use of key based random substitutions and key based random permutations. In this analysis, we use a key containing 16 numbers, represented as a block of 128 bits. A plain text of 32 characters is represented as a block of 256 bits. In each round, we have performed different key based permutations with the help of the equation that we have derived. Similarly, in each round, we have also permuted the key based S-Boxes. The cryptanalysis carried out at end shows that the cipher cannot be broken by any cryptanalytic attack.

2. Development of Cipher

Consider a key vector ‘K’ containing 16 numbers. Consider a vector
\[ D = \{d_0, d_1, \ldots, d_{15}\} \]  
(2.1)\n
Where \[ d_i = K_i \mod 4 \]  
(2.2)\n
The elements of vector D allows us to implement various different permutations in different rounds. Let the binary equivalent of these 16 numbers be represented as a matrix \( K_{16x8} \). So that,
\[ K^0 = \{k_0x8 + k_1x8 + \ldots + k_{14}x8 + k_{15}x8\} \]  
(2.3)\n
Consider a plaintext vector ‘T’ containing 32 characters. Let the binary equivalent of these 32 characters be represented as a matrix \( T_{32x8} \). So that,
\[ T = \{t_0x8 + t_1x8 + \ldots + t_{30}x8 + t_{31}x8\} \]  
(2.4)\n
Here, ‘+’ is the concatenation of bits, \( t_{ix8} \) and \( k_{ix8} \) are the 8 bits binary equivalent of the i th character of the plaintext vector ‘T’ and i th number of the key vector ‘K’ respectively. Let
\[ C_0 = \{t_0x8 + t_{1x8} + \ldots + t_{30x8} + t_{31x8}\} \]  
(2.5)\n
be the initial plaintext. Let \( C_1, C_2, \ldots, C_{15}, C_{16} \) be the 256 bits intermediate cipher text. Such that, \( C_i \) is obtained after the \( i^{th} \) round during encryption. The linear equation used for Permutation \( P^i \) in the \( i^{th} \) round is given by
\[ s = (r + n/2 + d_i) \mod n \]  
(2.6)\n
Such that, ‘n’ specifies the number of bits on which permutation is applied and \( d_i \) is the value which makes permutation distinct in different rounds. In each permutation \( P^i \),
\[ r = 0 \text{ to } (n/2) - 1 \]  
(2.7)\n
We interchange \( i^{th} \) and \( i^{th} \) bits to get required permutation \( P^i \) on ‘n’ bits.

The generation of S-Box from the key is explained in detail with an algorithm in (3.6). During encryption/decryption, in all the 16 rounds, we have used...
S-Boxes; Each S-Box contains 8 rows and 32 columns; takes 8 bits input and gives 8 bits output. In each round, we have permuted the S-Boxes so that, a set of bits of intermediate cipher will not enter into the same S-Box. So that, there is no scope for cryptanalysis with respect to the substitution boxes.

Let the 256 bits initial plaintext \( C^0 \) be divided into two equal parts \( L^0 \) and \( R^0 \). Such that, \( L^0 \) is the most significant 128 bits of \( C^0 \) and \( R^0 \) is the least significant 128 bits of \( C^0 \). Such that Copy the bits of \( K^1 \) to \( kr^1 \). By using the algorithm given for permutation \( P^i \) in (3.3), let us permute the bits of \( kr^1 \) by using the value \( d_0 \). Thus, round key \( kr^1 = P^i( kr^1, d_0 ) \).

Next, we permute the S-Boxes according to the algorithm given in (3.4). Now according to the classical feistel network, the first step of the round function is Expansion Permutation table. This is mainly useful if the number of bits in \( R^0 \) is less than the number of bits of round key \( kr^1 \). Now since \( R^0 \) contains 128 bits equal to the number of bits in round key \( kr^1 \), we need not use the expansion permutation table and we move to the next step which is XOR of \( R^0 \) and round key \( kr^1 \). Let \( R^1 = R^0 \ XOR \ kr^1 \). Next based on the bits of \( R^1 \) we get the 8 bit outputs from the respective S-Boxes. Then permute \( R^1 \) and \( R^0 \) with \( L^0 \). Thus, \( R^1 = L^0 \ XOR (P^1(R^1) \ and \ L^1 = R^0) \). Now concatenation of all \( \{L^1, R^1\} \) of respective S-Boxes gives us the intermediate cipher \( C^i \) for the second round. Thus, by using this process for the remaining rounds finally we obtain the cipher \( C^16 \).

The decryption of cipher is done by using the same round function. We just need to follow the same procedure as explained for encryption. But the key is used in reverse order i.e. \( kr^{16} \) to \( kr^1 \) and the S-Boxes are reverse permuted in each round as explained in algorithms (3.2) and (3.5).

\[ C^0 = \{ L^0, R^0 \} \]. Now let, \( K^1 = LeftShift (K^0) \).

### 3. Algorithms

#### 3.1. Algorithm for Encryption

BEGIN
1. Read the Key vector \( K \) and plaintext vector \( T \)
2. Compute the vector \( D \) such that \( d_i = K_i \mod 4 \)
3. Convert the key vector \( K \) and plaintext vectors \( T \) to binary representation as a matrix \( k_{16 \times 8} \) and \( t_{32 \times 8} \) respectively.
4. \( K^0 = \{ k_{0 \times 8} + k_{1 \times 8} + \ldots + k_{15 \times 8} \} \)
5. \( C^0 = \{ t_{0 \times 8} + t_{1 \times 8} + \ldots + t_{31 \times 8} \} \)
6. for \( i = 0 \) to \( 15 \) \{ \[ K^{i+1} = LeftShift (K^i) \]
   Copy \( K^{i+1} \) to \( kr^1 \)
   \( p^{i+1}( kr^{i+1}, d_i ) \) \// permute 128 bits; see algo (3.3)
   Permute (\( bx, d_i \)) \// permute S-Boxes; see algo (3.4)
   \( C^{i+1} = F(kr^{i+1}, C^i, d_i) \)
\} \// end for
END.

#### 3.2. Algorithm for Decryption

BEGIN
1. Read the Cipher \( C^{16} \)
2. Read the last round key \( K^{16} \)
3. Read the vector \( D = \{d_0, d_1, d_2 \ldots d_{15}\} \).
   \{ \[ C^i = Switchbox (C^{i+1}) \]
   Copy \( K^{i+1} \) to \( kr^i \)
   Permute (\( kr^i, d_i \))
   \( C^{i-1} = F(kr^i, C^i, d_i) \)
   Rightshift (\( K^{i+1} \))
   \( C^{i-1} = SwitchBox (C^{i-1}) \)
   ReversePermute (\( bx \), \( d_i \)) \// permute S-Boxes; see algo (3.5)
\} \// end for
END

#### 3.3. Algorithm for Permutation

(To permute the key \( kr^{i+1} \))

BEGIN
1. for \( r = 0 \) to \( (r < n/2) \) \{
   \[ s = (r + n/2 + d_i) \mod n \]
   \[ temp = kr^{r+1} \]
   \[ kr^{r+1} = kr^{r+1} \]
   \[ kr^{r+1} = temp \]
\} \// end for
END

#### 3.4. Algorithm for Permuting S-Boxes

(To permute the S-Boxes during encryption in \( i^{th} \) round)
Note: Initial order of 16 S-Boxes before encryption is 
\( bx[16] = \{ 0,1,2,3, \ldots, 14,15 \} \).

4. for \( i = 15 \) to 0

\textbf{Permute ( bx, di )}

BEGIN

1. for \( r = 0 \) to ( \( r < 8 \) )

   \{ 
   \begin{align*}
   s &= ( r + 8 + d_i ) \mod 16 \\
   \text{temp} &= bx_s \\
   bx_s &= bx_r \\
   bx_r &= \text{temp}
   \end{align*}
   \}

2. for \( r = 7 \) to ( \( r \geq 0 \) )

   \{ 
   \begin{align*}
   s &= ( r + 8 + d_i ) \mod 16 \\
   \text{temp} &= bx_s \\
   bx_s &= bx_r \\
   bx_r &= \text{temp}
   \end{align*}
   \}

END.

3.5. Algorithm for Reverse Permuting S-Boxes

\textbf{(To Reverse permute the S-Boxes during decryption in \( i^{\text{th}} \) round)}

Note: Initial order of 16 S-Boxes before decryption will be same as the order used in the last round during encryption.

\textbf{ReversePermute ( bx, di )}

BEGIN

1. for \( r = 7 \) to ( \( r > = 0 \) )

   \{ 
   \begin{align*}
   s &= ( r + 8 + d_i ) \mod 16 \\
   \text{temp} &= bx_s \\
   bx_s &= bx_r \\
   bx_r &= \text{temp}
   \end{align*}
   \}

2. for \( i = 1 \) to 15

   \{ 
   \begin{align*}
   \text{box}_{p,j} &= k_{i,j} \\
   \text{box}_{p+1,j} &= k_{15-i,j}
   \end{align*}
   \}

3. for \( i = 0 \) to 31

   \{ 
   \begin{align*}
   \text{for} \ j = 5 \ \text{to} \ 7 \\
   \text{for} \ j = 0 \ \text{to} \ 7
   \end{align*}
   \}

END.

3.6. Algorithm to generate Key based Substitution Boxes.

BEGIN

1. \( p = 0 \)

2. for \( i = 0 \) to 15

   \{ 
   \begin{align*}
   \text{for} \ j = 0 \ \text{to} \ 7 \\
   \text{for} \ j = 5 \ \text{to} \ 7
   \end{align*}
   \}

3. for \( i = 0 \) to 31

   \{ 
   \begin{align*}
   \text{for} \ j = 0 \ \text{to} \ 7 \\
   \text{for} \ j = 5 \ \text{to} \ 7
   \end{align*}
   \}

4. for \( i = 0 \) to 31

   \{ 
   \begin{align*}
   p &= p + ( \text{power} \ (2,j) \ * \ \text{box}_{p,7-j} ) \\
   \text{Tempbox1}_{i} &= p
   \end{align*}
   \}

5. \( \text{tempbox2}_{0,30} = \text{tempbox1}_{0} \)

6. for \( j = 0 \) to 29

   \{ 
   \begin{align*}
   \text{tempbox2}_{0,j} &= \text{tempbox1}_{j}
   \end{align*}
   \}

7. for \( i = 1 \) to 15

   \{ 
   \begin{align*}
   \text{tempbox2}_{i,30} &= \text{tempbox2}_{i-1,0} \\
   \text{tempbox2}_{i,31} &= \text{tempbox2}_{i-1,1}
   \end{align*}
   \}

8. for \( i = 0 \) to 15

   \{ 
   \begin{align*}
   \text{for} \ h = 0 \ \text{to} \ 7 \\
   \text{for} \ j = 5 \ \text{to} \ 7
   \end{align*}
   \}
\[ p = (j + 16 + (i \mod 2)) \mod 32 \]
\[
\text{tempbox2}_{i,j} = \text{tempbox2}_{i,j} + \text{tempbox2}_{i,p}
\]
\[
\text{tempbox2}_{i,p} = \text{tempbox2}_{i,j} - \text{tempbox2}_{i,p}
\]
\[
\text{tempbox2}_{i,j} = \text{tempbox2}_{i,j} - \text{tempbox2}_{i,p}
\]
\[
\text{for } j = 0 \text{ to } 31
\]
\[
\text{SBOX}_{i,h} = \text{temp}_{i,j}
\]
\[
\}
\]
\[
4. Illustration of Cipher
\]
Let the key vector be \( K = \{ 1, 254, 7, 200, 77, 16, 222, 53, 71, 40, 13, 67, 154, 0, 106, 153 \} \).
\((4.1)\)
Let the vector \( D = K_4 \mod 4 = \{ 1, 2, 3, 0, 1, 0, 2, 1, 3, 0, 1, 3, 2, 0, 2, 1 \} \).
\((4.2)\)
Consider the plaintext vector \( T = \{ \text{lets pray together for all of us} \} \).
\((4.3)\)
Let the initial order of S-Boxes be denoted by a vector \( bx[16] = \{ 0, 1, 2, 3, 4, 5, 6, 7 \} \).
\((4.4)\)
Let the 8 bit binary equivalent of plaintext and key elements be represented by the matrices \( t_{32 \times 8} \) and \( k_{16 \times 8} \).
\[
\text{According to (2.3) the key } K^0 \text{ is as follows.}
\]
\[
0000000111111110000001111100100010011010
0001000011011110001101010100011100101000
000011010000110001000000011010101
10011001.
\]
\((4.5)\)
According to (2.5) the plaintext \( C^0 \) is as follows.
\[
01100101011100100010000111000011011001101000
01110000001111111000011001100000110001000
001000000110101010000000110010110100
0011000110010000010111001000000000110101
01110010100001000110100110000001110011101
0101001011111111010000001100101110101001
01100101011010011010100011101100110110010
01110011010111111010000001100101111001100
0000110101111110100000011001011110011001
010100101100011.
\]
\((4.11)\)
Similarly, by completing the remaining rounds we get the final cipher text \( C^1 \) as follows.
\[
0100100101100010101011001101001100100000
111011011111011110001000000000110111010101010
11000110100110011110111101110011001101101100
000000110101101101011110011000111001101111
0000110101111111111101000000011001011101101
0101001011000111000.
\]
\((4.12)\)
In the process of decryption we use the same round function but the SBoxes are reverse permuted as explained in algorithm (3.5). Thus we obtain the original plain text from the given cipher text.

5. Cryptanalysis

The various cryptanalytic attacks available in the literature depend upon the facts that, the cipher text is...
known or pairs of plaintext and cipher text are known or they are chosen in a special manner.

When the cipher text only is known, the breaking of the cipher depends upon the size of the key space, and this is carried out by the brute force attack. When the pairs of plaintext and cipher text are known, the cipher can be broken if the key can be determined and this is done by known plain text attack.

Thus, we examine the brute force attack and the known plaintext attack on our cipher to assess the strength of our cipher. Here we show that the brute force attack is formidable and the known plaintext attack leads to a system of equations from which the key cannot be determined due to the unknown term \( d_i \).

**Brute Force Attack**

As the key matrix is of the order 16 x 8, the size of the key space is
\[
2^{128} \approx (2^{10})^{13} \approx (10)^{39} \quad (5.1)
\]
Thus, one cannot break the cipher by applying brute force attack.

**Known Plaintext Attack**

Let us consider the known plaintext attack. In this case, we have as many plaintext and cipher text pairs as we require. Through this paper, it is worth noticing the unknown term \( d_i \) induced in the equations for permutation and substitution in the repetitive process of feistel cipher.

Firstly, we have used different permutations in different rounds through the equation that we have derived (see (2.6)). Secondly, the substitution boxes, which used to be static in feistel cipher, are now made random (see 7.3).

Now first let us see the known plaintext then we will prove how our algorithm tackles this attack.

According to classical feistel network, the linear relation between plaintext and cipher text is as follows.\[
c_1 \leftarrow F( (c_0), kr_1),d_0 \quad (5.9)
\]
\[
c_2 \leftarrow F( (c_1), kr_2),d_1 \quad (5.10)
\]
\[
c_2 \leftarrow F( (F(c_0), kr_1), kr_2),d_1 \quad (5.11)
\]
\[
c_3 \leftarrow F( (c_2), kr_3),d_3 \quad (5.12)
\]
\[
c_3 \leftarrow F( (F(F(c_0), kr_1), kr_2), kr_3),d_3 \quad (5.13)
\]
similarly, \( c_{15} \leftarrow F(c_{14}, kr_{15}, d_{14}) \quad (5.14) \)
\[
c_{16} \leftarrow F( (c_{15}, kr_{16}, d_{15}) \quad (5.15)
\]

**Brute Force Attack on classical feistel ciphers and**

**Key based permutations & Key based substitutions.**

In this case, even if the plaintext-ciphertext pair \( c_0 \) and \( c_{16} \) are known, any number of such pairs is of no use mainly because of two reasons. Firstly, the key based permutations are not same in all the rounds. They vary...
from one round to another round based on permutations, then he should first know the values of \(d_0, d_1, \ldots, d_{15}\). And the values \(d_0, d_1, \ldots, d_{15}\) are unknown as long as the key is unknown. Secondly, the order of the key based substitution boxes is not same in all the rounds. They are permuted in each round based on the value of \(d_i\). Therefore, to know which substitution box is used for what bits, one must actually know the value of \(d_i\). And we know \(d_i\) is unknown as long as the key secret. Thus the known plaintext attack is also not possible.

### Avalanche Effect

Consider the plaintext “lets pray together for all of us” (see (4.3)), we have obtained the cipher text given by (4.12). on changing the first character of the above plaintext from ‘l’ to ‘k’ (as we know the ASCII codes of ‘l’ and ‘k’ differ in one bit), keeping the key constant, we obtain the corresponding cipher as

\[
\begin{align*}
0001011101001111101101000100110011111100 \\
010101010111000010011111111111101 \\
110001100000110011110111111111110 \\
010000111001111111111111111111101 \\
1110101101001100000111111111111100 \\
1101011111100110110011001111111100 \\
00101101111111.
\end{align*}
\]

Comparing (4.12) and (5.16), we notice that the two cipher texts differ in 121 bits out of 256 bits. This shows that the algorithm exhibits strong avalanche effect. the value \(d_i\). And if one has to guess the

Now let us change the key in one bit and keep the plaintext as it is. This is achieved by changing the first number ‘1’ to ‘0’ (since ‘1’ and ‘0’ differ in one bit) in the key vector \(K\) given by (4.1). Then we obtain the corresponding cipher as

\[
\begin{align*}
1001010001000101111010100000111100000010 \\
1010100001011110110100110011111101 \\
110010011100010111010011111011111101 \\
001000111001111111111111111111101 \\
1111011010110011000011111111111101 \\
1101011111100110110011001111111100 \\
00101101111111.
\end{align*}
\]

Comparing (4.12) and (5.17), we readily notice that the two ciphers differ in 130 bits out of 256 bits. This shows that, in our encryption algorithm, the permutations and substitution boxes that we have derived from the key exhibit strong avalanche effect.

### 6. Illustration of permutations

To get required key based permutation \(P^i\) in the \(i^{th}\) round, use

\[
d_i = K_i \mod 4 \\
s = (r + n/2 + d_i) \mod n.
\]

Let the bits to be permuted are as follows.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(6.1)

Since these are the 8 bits.

**Case 1:**

In the \(1^{st}\) round, Let \(d_1 = 0\).

Then according to the equation

\[
s = (r + n/2 + d_i) \mod n
\]

\[
r = 0 \text{ to } r < n/2 = \{ 0, 1, 2, 3 \}
\]

Then according to (2.6) & (2.7), The corresponding ‘S’ values will be \(S = \{ 4, 5, 6, 7 \}\)

Now interchange the corresponding bits of the \(r^{th}\) and \(S^{th}\) positions in the following order.

\[
\begin{align*}
0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 1
\end{align*}
\]

**Case 2:**

Consider the bits of (6.1)

In the \(1^{st}\) round, Let \(d_1 = 2\).

Thus, \(n = 8\), \(r = \{ 0, 1, 2, 3 \}\) and corresponding ‘S’ values will be

\[
S = \{ 6, 7, 0, 1 \}
\]

Interchange the bits of \(r^{th}\) and \(S^{th}\) positions in the following order.
Case 3:
Let the number of bits be 4. Thus, \( n = 4 \).
Let the bits be as follows.
\[
\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
\]

In the 1\(^{st}\) round,
Let \( d_i = 1 \) and 
\( r = 0 \) to \( ( r < n/2 ) = \{ 0 , 1 \} \). Then, 
according to (2.6) & (2.7), the corresponding \('S'\) values will be.
\[
S = \{ 3, 0 \}.
\]
Now interchange the corresponding bits of the \('r^{th}\) and \('s^{th}\)' positions in the following order.

\[
\begin{array}{cccc}
1 & 0 & 1 & 0 \\
\end{array}
\]

Bits after permutation are
\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
\]

7. Generating key based substitution boxes.

Generation of key based substitution boxes:
We need 16 Substitution boxes, each containing 8 rows and 32 columns. Let us expand the elements of key vector \( K \) from 16 to 32 numbers. This is done by picking the 16 numbers of \( K \) in the reverse order and placing them in between every two consecutive elements of vector \( K \) itself. Due to this process, each row of a substitution box will contain a number which is repeated exactly twice. Hence, we get a balanced substitution box.

Let this new vector be \( K'_1 \) which is as follows.
\[
K'_1 = \{ K_0, K_15, K_1, K_14, K_2, K_13, K_12, K_4, K_11, K_3, K_10, K_6, K_9, K_7, K_8, K_5, K_16, K_1, K_15, K_4, K_14, K_2, K_13, K_12, K_3, K_15, K_2, K_14, K_11, K_10, K_8, K_7, K_6, K_9, K_5, K_16, K_1, K_15 \} \quad (7.1)
\]

Now compute the 8 bit binary equivalent of each element in \( K'_1 \) and then convert the 6\(^{th}\), 7\(^{th}\) and 8\(^{th}\) bits from 1's to 0's and 0's to 1's respectively.

Next Compute the decimal equivalent of respective 8 bits and place them in a new vector called \( K'^{dl} \). Such that
\[
K'^{dl} = \{ K'^{dl}_0, K'^{dl}_1, \ldots, K'^{dl}_{31} \} \quad (7.2)
\]

Left shift vector \( K'^{dl} \) twice and place these 32 numbers as the first row in matrix \( B_{16\times32} \). Again left shift vector \( K'^{dl} \) twice and place those 32 numbers as the second row in the matrix \( B_{16\times32} \). Continue this process for 14 more times and we get the remaining 14 rows of matrix \( B_{16\times32} \).

Now we will take the first row of matrix \( B_{16\times32} \) and permute those 32 numbers by using the following equation.
\[
t = ( j + n/2 + ( i \mod 2 ) ) \mod n \quad (7.3)
\]

Here \( n = 32 \), since we are permuting 32 numbers. Let \( j = 0 \) to \( ( j < n/2 ) \). Such that, ‘t’ and ‘j’ indicates the positions at which the elements are to be interchanged.

Let \( i = 0 \) to 7. So that, ‘i’ indicates the row that is to be generated in a substitution box. Such that, For each value of ‘i’, \( j = 0 \) to \( ( j < n/2 ) \). Similarly, the other elements of the SBox can be demonstrated.
7.1 Illustration of generating key based substitution boxes

Consider the following code for required permutations in a substitution box.

7.2 Illustration of generating substitution boxes:

*Consider the following code for required permutations in a substitution box.*
for b = 0 to 15                                // 'b' indicates the substitution box
{
    for i = 0 to 7                               // 'i' indicates the row in b^th substitution box
        { for j = 0 to 15
            { t = ( j + n/2 + ( i mod 2 ) ) mod n     //Permute // 't' and 'j' indicates the interchange ( B_{i,j} , B_{i,t} )
            positions at which elements are to be interchanged.
                for r = 0 to 31
                    { Sbox^{b}_{i,r} = B_{i,r} 
                    }
                }
        }
    }
}

7.3 Illustration of permuting substitution boxes:

Let the initial order of substitution boxes be bx[16] = { 0, 1, 2, 3,..14, 15 }
After permuting the substitution boxes in the first round,
Let bx = { 4, 7, 2, 0,........15, 9, 1, 11 }. And
After permuting the substitution boxes in the second round,
Let bx = { 1, 12, 6, 15,........8, 11, 5, 0 }.
8. Computational experiments and Conclusions

In this paper, we have developed a block cipher for a block of size 256 bits. The key contains 32 numbers and it is represented as a block of 256 bits by using a matrix. The plaintext which is of 32 characters is also represented as a matrix of 256 binary bits. The development of cipher is essentially represented by feistel network and we have used 16 rounds for encryption. The algorithms given for the encryption- decryption, permutation and substitution are all written in C language.

From the cryptanalysis presented, we have found that the cipher cannot be broken by the brute force attack or known plaintext attack. Moreover since we have used the random substitution boxes and since the permutations

Permutation of Substitution boxes during encryption.
Similarly, during decryption, we perform reverse permutation of substitution boxes.
and substitutions are based on key, the cryptanalysis is very difficult. Keeping all the above aspects in view, we conclude that the cipher is a very interesting one and it cannot be broken by any cryptanalytic attack.

Acknowledgment:
The authors are very thankful to Prof. Depanwita Roy Chaudhury, IIT Kharagpur, for giving her valuable inputs while writing this paper. We are also thankful to the management of SreeNidhi Institute of Science and Technology, for their support and encouragement given during this research work.

References