A NOVEL APPROACH TO INFORMATION SECURITY USING FOUR DIMENSIONAL (4D) PLAYFAIR CIPHER FUSED WITH LINEAR FEEDBACK SHIFT REGISTER

Krishnaraj Bhat*
Department of Computer Applications
National Institute of Technology, Jamshedpur, Jharkhand, India
krrishvaradha08@gmail.com

Dindayal Mahto
Department of Computer Applications
National Institute of Technology, Jamshedpur, Jharkhand, India
dindayal.mahto@gmail.com

Dilip Kumar Yadav
Department of Computer Applications
National Institute of Technology, Jamshedpur, Jharkhand, India
dkyadav1@gmail.com

Abstract

Information that has to be kept confidential can be in any form: text, image, audio, video, encoded, compressed etc. Therefore, the main objective of this research is to develop a cryptographic cipher which is strong enough to secure all kinds of information. It was ascertained from survey that the existing versions of Playfair cipher work with digrams and trigrams, and support securing circumscribed number of characters. So, their application is confined only to security of text messages. Also, most of them are ambiguous, and lack high avalanche effect and confusion rate. Hence, we propose a cipher called 4D Playfair Cipher which is an extension of the Classical Playfair cipher that can be used to secure all sorts of information more effectively without any ambiguity. It works with quartets and on the principle of a range of values that can be stored in a byte (8 bits) memory. This 4D Playfair cipher is amalgamated with Linear Feedback Shift Register (LFSR) to produce high avalanche effect and confusion rate. The proposed cipher is compared with the existing variations of Playfair cipher with respect to number of characters supported, possible number of keys, ambiguity, confusion rate and avalanche effect. It was found from the comparison analysis that the purported method is strong and more applicable when compared to other variants of Playfair cipher. The security analysis shows that it is highly resistant to five major types of cryptanalytic attacks. The 4D Playfair cipher can be extended to multi dimensional Playfair cipher to increase the security.

***Keywords*:** Security; Cryptography; Playfair cipher; Symmetric cipher; Linear Feedback Shift Register.

1. Introduction

In information age, Internet is the backbone to communicate information between users/systems. It is a network of computer networks and of open architecture type. Due to the openness of the architecture, information being transmitted may be attacked. Hence, they have to be made unintelligible in order to be secure. The important goal of this research is to transfer all kinds of information securely in the network. The proposed cipher uses novel extension of Classical Playfair cipher (or Wheatstone-Playfair cipher). Classical Playfair cipher is one of the best multiple letter, symmetric encryption ciphers invented by a British scientist named Sir Charles Wheatstone in 1854. But, it holds the name of his friend Baron Playfair of St. Andrews who vigorously supported it at the British foreign office. It has also played a vital role in World War I and World War II [Stallings, (2014)]. But, it supports securing only 26 uppercase English letters in which I and J are treated same, uses X as filler letter and works with digram (groups of 2 letters) [Schneier, (1996)]. Since a digram and its reverse is transformed in a similar way (i.e. if GH is a digram in plain text and KL is it's corresponding digram

in cipher text then for digram HG in plain text, it's corresponding digram in cipher text will be LK), cryptanalysis is easier [Alfred *et al*., (1996)], [Buchmann, (2001)]. Other demerits are: it has less confusion rate (complexity in relationship between statistics of cipher message and encryption key), less avalanche effect (changes in bits of cipher message produced by a change in one bit of the plain message or one bit of the key) [Stallings, (2014)] and it is ambiguous (inability of the cipher to determine whether a filler letter in decrypted message is a part of original message). There came many improvements over the Playfair cipher later on which are discussed in chronological order in literature survey in Section 2. It was found from survey that none of the existing variations of Playfair cipher could secure all kinds of information without ambiguity, with high confusion rate and with high avalanche effect. The variations which had high confusion rate and avalanche effect did not support securing all kinds of information. Therefore, we have proposed a variation of Playfair cipher whose working is discussed in Section 3 to Section 6, which can secure all kinds of information with no ambiguity and with high confusion and avalanche effect. Section 7 describes the security analysis of the proposed cipher in which it is found that the proposed cipher is strong against the major cryptanalytic attacks. Section 8 depicts the comparison analysis of the proposed cipher with the existing variants which shows that the purported cipher provides better security and is more utile when compared to other variants.

2. Literature Survey

A modified version of Playfair cipher was developed which maps random numbers generated using LFSR to secret key and transmits numbers instead of cipher text [Murali and Senthilkumar, (2009)]. This increased the confusion rate by one more level. But still cipher remains ambiguous, has less avalanche effect with respect to one bit change in plain text and supports only 26 characters. Umakanta et al [Sastry *et al*., (2009)] developed a variation of Playfair cipher which supports 128 ASCII characters from code 0 to 127. It uses a key of 64 unrepeatable characters, uses no filler character and supports encryption of plaintext of even length by performing interweaving and substitution on plain text 16 times which produces high confusion rate and avalanche effect. It is unambiguous. But, it won't support encryption of plaintext of odd length. Then came another variation which supports 64 characters where ^ is used as filler character and | is used to represent a space character [Srivastava and Gupta, (2011)]. LFSR is used to permute the bits in the bit representation of the characters which increased confusion rate, and avalanche effect is good when a bit is altered except at the end of plaintext. But, this variation is still ambiguous. Sanjay & Utpal proposed a variation which supports 90 characters and ^ is used as filler character [Basu and Ray, (2012)]. The only merit of this version is that it supports more characters than the Classical one. All other demerits of Classical Playfair cipher are still present in this cipher. Dhenakaran & Ilayaraja purported an extension of Playfair cipher supporting all 256 ASCII characters with no filler character for replacing repeating character in a pair and null character is appended to make plain text length even [Dhenakaran and Ilayaraja, (2012)]. It is unambiguous except at the end of decrypted plain text which can be easily resolved. But, this version has less confusion rate and avalanche effect. Amandeep et al [Kaur *et al*., (2012a)] suggested a variation that supported 36 characters and maps random numbers generated using LFSR to secret key and transmits numbers instead of cipher text. This just enhanced the confusion rate by one more degree. But still cipher is ambiguous and has less avalanche effect with respect to one bit change in plain text. Again, Amandeep et al [Kaur *et al*., (2012b)] proposed a new variation called 3D Playfair cipher which supports 64 characters, uses X and Z characters as fillers and works with trigram (group of 3 characters) which increased the confusion rate. But, it is still ambiguous and has less avalanche effect but more when compared to those which work with just digrams. Aftab et al [Alam *et al*., (2013)] used * and # characters as fillers instead of X for the messages encrypted using Classical Playfair cipher making it just unambiguous from being ambiguous. Then came an extension of 3D Playfair cipher which maps random numbers generated using LFSR to secret key and transmits numbers instead of cipher text [Kaur *et al*., (2013)]. This increased the confusion rate but still cipher is ambiguous and has less avalanche effect with respect to one bit change in plain text. Then another version of 3D Playfair cipher was proposed in which bits in bit representation of cipher text characters are rotated in circular fashion before transmission based on random numbers generated using LFSR enhancing the confusion rate [Singh *et al*., (2013)]. Still, cipher has ambiguity and has less avalanche effect with respect to one bit change in plain text. There was one more extension of 3D Playfair cipher where cipher text characters are XORed/XNORed with random numbers generated using LFSR to enhance the confusion rate [Verma *et al*., (2013)]. But, cipher is still ambiguous and has less avalanche effect with respect to one bit change in plain text. Nisarga & Subhajit suggested a new variation to Classical Playfair cipher which supports 36 characters and plain text is encrypted four times using four different keys before getting converted to cipher text which raised the confusion rate [Chand and Bhattacharyya, (2014)]. Still, this version is ambiguous and has less avalanche effect. Afterwards there came another variation of Classical Playfair cipher in which I and J are not treated as same, key matrix is rotated after processing each digram and rows and columns are swapped based on randomly generated swap patterns which increased the confusion rate [Hans *et al*., (2014)]. Still, this version is ambiguous and has less avalanche effect with respect to one bit change in plain text. At the end came an extension of 3D Playfair cipher in which key matrix is rotated after processing

each trigram to enhance the confusion rate [Singh *et al*., (2015)]. But, this variation is also ambiguous and has less avalanche effect with respect to one bit change in plain text.
From the above survey details it can be seen that most variations of Playfair cipher lack good avalanche effect and are ambiguous. Also, applications of these variations are confined to security of limited types of messages. The variations which are ambiguous cannot be used for securing passwords. But, information to be secured can be in any form: text, audio, video, compressed, encoded etc. So, there is a need for implementing a variation which can support security of all kinds of information without any ambiguity and with high confusion rate and avalanche effect. Therefore, we introduce a new variation called 4D Playfair cipher which works with quartets (group of 4 values) and supports encryption of all kinds of information. Least amount of memory that is used to store information in a computer is a byte (8 bits). The range of values that can be stored in a byte is 0 to 255. If the cipher supports securing this range of values then it is sufficient to secure all sorts of information. Therefore, 4D Playfair cipher supports encrypting/decrypting 256 values along with four more values for being unambiguous in which three are used as fillers and fourth is used only during substitution. Since 4D Playfair cipher has less confusion rate and avalanche effect, it is amalgamated with LFSR to increase them, which is our research proposal.

3. 4D Playfair Cipher

4D Playfair cipher is a symmetric encryption cipher which works with a group of 4 values called a quartet at a time during encryption and decryption. For this, it uses a $2 \times 2 \times 13 \times 5$ key matrix formed using the symmetric key to hold 260 values from 0 to 259. Here, the values from 0 to 255 represent the values that can be stored in a byte. The values from 256 to 258 are used as filler values. Value 259 is used only during substitution. The key features of key matrix are: key matrix has 2 directions, each direction has 2 planes, each plane has 13 rows and each row has 5 columns. 4D Playfair cipher has mainly three steps for encryption. They are Key matrix formation, Message pre-processing and Message encryption. There are three steps for decryption. They are Key matrix formation, Message decryption and Message post-processing. Each step is discussed in detail in following sub sections. The way of forming key matrix at both encryption and decryption sides is same.

3.1. *Key matrix formation*

Table 1. 4D Playfair matrix for the key (109 111 110 97 114 99 104 121)

D1_P1					**D1_P2**				
109	111	110	97	114	57	58	59	60	61
99	104	121	0	1	62	63	64	65	66
2	3	4	5	6	67	68	69	70	71
7	8	9	10	11	72	73	74	75	76
12	13	14	15	16	77	78	79	80	81
17	18	19	20	21	82	83	84	85	86
22	23	24	25	26	87	88	89	90	91
27	28	29	30	31	92	93	94	95	96
32	33	34	35	36	98	100	101	102	103
37	38	39	40	41	105	106	107	108	112
42	43	44	45	46	113	115	116	117	118
47	48	49	50	51	119	120	122	123	124
52	53	54	55	56	125	126	127	128	129
D2_P1					**D2_P2**				
130	131	132	133	134	195	196	197	198	199
135	136	137	138	139	200	201	202	203	204
140	141	142	143	144	205	206	207	208	209
145	146	147	148	149	210	211	212	213	214
150	151	152	153	154	215	216	217	218	219
155	156	157	158	159	220	221	222	223	224
160	161	162	163	164	225	226	227	228	229
165	166	167	168	169	230	231	232	233	234
170	171	172	173	174	235	236	237	238	239
175	176	177	178	179	240	241	242	243	244
180	181	182	183	184	245	246	247	248	249
185	186	187	188	189	250	251	252	253	254
190	191	192	193	194	255	256	257	258	259

There are mainly 4 steps. They are:
(1) Take a secret key which is a sequence of values in the range 0 to 259. For example, (10 25 10 48 64 79 100 101 48).

(2) Remove the duplicate values if present. For example, after removing duplicates from (10 25 10 48 64 79 100 101 48), the key will become (10 25 48 64 79 100 101).
(3) Now, the key without any duplicates will be used to form the matrix. Position the values in the key in 2 × 2 × 13 × 5 key matrix direction by direction (from direction 1 to 2), plane by plane (from plane 1 to 2), row by row (from 1 (top) to 13 (bottom)) and column by column (1 (left) to 5 (right)).
(4) Fill the left out slots in the matrix with those values that are not part of the key starting from value 0 to 259 following the rule given in step 3.

For the key (109 111 110 97 114 99 104 121), the key matrix is shown in Table 1. In Table 1, each sub tables have a header of the form Di_Pj where 1 <= i, j <= 2. Here, Di_Pj is a part of key matrix containing the values in j^{th} plane in i^{th} direction. For example, D1_P2 represents 2^{nd} plane in 1^{st} direction.

3.2. *Message pre-processing*

In message pre-processing, message is divided into quartets.

While forming quartets, 3 fillers values 256, 257 and 258 are used based on requirement. There are some rules for forming a quartet which are as follows:

(1) In a quartet, if a value at a position is same as the value at next position, filler value 256 is filled in between them. For example, message (10 20 30 30 40 50 60) is divided into quartets (10 20 30 256) and (30 40 50 60).
(2) In a quartet, if a value at a position is same as the value at next position and next to next position, 256 is filled after first value and 257 is filled after second value. For example, message (10 10 10 20 30 40) is divided into quartets (10 256 10 257) and (10 20 30 40).
(3) After forming the quartets with the above rules, if the message length is not a multiple of 4 then it is made by appending the consecutive fillers at the end until message length becomes a multiple of 4. For example, message (10 20 30) will become (10 20 30 256), message (10 20) will become (10 20 256 257) and message (10) will become (10 256 257 258).

So, a message (5 10 20 30 30 30 30 40 50) will be divided into quartets (5 10 20 30), (30 256 30 257) and (30 40 50 256) by following above rules.

3.3. *Message encryption*

After forming the quartets, pre-processed message is ready for encryption. A value in a quartet is substituted by the value having the same row number as that of value getting encrypted, column number of next value, dimension number of next to next value and plane number of next to next to next value in circular fashion. This concept is illustrated in Table 2.

Table 2. Encryption procedure of 4D Playfair cipher

Message Quartet	**Message Quartet**				**Cipher Message Quartet**
	1^{st}	2^{nd}	3^{rd}	4^{th}	
1^{st}	R	C	D	P	1^{st}
2^{nd}	P	R	C	D	2^{nd}
3^{rd}	D	P	R	C	3^{rd}
4^{th}	C	D	P	R	4^{th}

In Table 2, 1^{st}, 2^{nd}, 3^{rd} and 4^{th} mean first, second, third and fourth value in a quartet respectively. R, C, D and P mean row, column, direction and plane numbers respectively. Same convention is followed for Table 3 also. Here, circular fashion means, for encrypting first value, second value will be next value, third value will be next to next value and fourth value will be next to next to next value. For second value encryption, third value will be next value, fourth value will be next to next value and first value will be next to next to next value. For third value encryption, fourth value will be next value, first value will be next to next value and second value will be next to next to next value. For fourth value encryption, first value will be next value, second value will be next to next value and third value will be next to next to next value.

3.4. *Message decryption*

Table 3. Decryption procedure of 4D Playfair cipher

Cipher Message Quartet	**Cipher Message Quartet**				**Message Quartet**
	1^{st}	2^{nd}	3^{rd}	4^{th}	
1^{st}	R	P	D	C	1^{st}
2^{nd}	C	R	P	D	2^{nd}
3^{rd}	D	C	R	P	3^{rd}
4^{th}	P	D	C	R	4^{th}

While decrypting, a value in a quartet is substituted by the value having the same row number as that of the value getting decrypted, plane number of next value, direction number of next to next value and column number of next to next to next value in circular fashion. This concept is illustrated in Table 3.

3.5. *Message post-processing*

Since a byte value cannot exceed 255, all the values greater than 255 cannot be a part of original message. Hence, they are discarded from the decrypted message in order to get back the original message. This makes 4D Playfair Cipher unambiguous. For example, after post-processing decrypted message (10 256 10 257), original message will be (10 10).

4. An Example of 4D Playfair Cipher

The byte values of a message of length 5 bytes which has to be secured are 97, 98, 98, 99 and 100. The key used is (109 111 110 97 114 99 104 121). The encryption and decryption processes are explained below.

4.1. *Encryption process*

Key matrix as shown in Table 1 is formed initially using the key.

4.1.1. *Message pre-processing*

Message (97 98 98 99 100) is divided into quartets (97 98 256 98) and (99 100 256 257).

4.1.2. *Message encryption*

Table 4. Encryption of (97 98 256 98)

Message Quartet	Message Quartet				Cipher Message Quartet
	97	98	256	98	
97	1	1	2	2	195
98	1	9	2	1	33
256	1	2	13	1	125
98	4	1	2	9	102

Encryptions of quartets (97 98 256 98) and (99 100 256 257) are illustrated in Table 4 and Table 5 respectively. By using Table 1 and Table 2 as reference, Table 4 shows that value 97 is substituted by the value with row number 1, column number 1, direction number 2 and plane number 2 which is 195, value 98 is substituted by the value with row number 9, column number 2, direction number 1 and plane number 1 which is 33, value 256 is substituted by the value with row number 13, column number 1, direction number 1 and plane number 2 which is 125 and value 98 is substituted by the value with row number 9, column number 4, direction number 1 and plane number 2 which is 102. In a similar way, (99 100 256 257) is encrypted as (201 171 127 125). Thus, the cipher message formed is (195 33 125 102 201 171 127 125).

Table 5. Encryption of (99 100 256 257)

Message Quartet	Message Quartet				Cipher Message Quartet
	99	100	256	257	
99	2	2	2	2	201
100	1	9	2	2	171
256	1	2	13	3	127
257	1	1	2	13	125

4.2. *Decryption process*

Key matrix as shown in Table 1 is formed initially using the key. The cipher message (195 33 125 102 201 171 127 125) is divided into quartets (195 33 125 102) and (201 171 127 125).

4.2.1. *Message decryption*

Table 6. Decryption of (195 33 125 102)

Cipher Message Quartet	Cipher Message Quartet				Message Quartet
	195	33	125	102	
195	1	1	1	4	97
33	1	9	2	1	98
125	2	2	13	2	256
102	2	1	1	9	98

Decryptions of quartets (195 33 125 102) and (201 171 127 125) are illustrated in Table 6 and Table 7 respectively. By using Table 1 and Table 3 as reference, Table 6 shows that value 195 is substituted by the value with row number 1, plane number 1, direction number 1 and column number 4 which is 97, value 33 is substituted by the value with row number 9, plane number 2, direction number 1 and column number 1 which is 98, value 125 is substituted by the value with row number 13, plane number 2, direction number 2 and column number 2 which is 256 and value 102 is substituted by the value with row number 9, plane number 2, direction number 1 and column number 1 which is 98. Similarly, quartet (201 171 127 125) is decrypted.

Table 7. Decryption of (201 171 127 125)

Cipher Message Quartet	**Cipher Message Quartet**				**Message Quartet**
	201	171	127	125	
201	2	1	1	1	99
171	2	9	2	1	100
127	2	2	13	2	256
125	2	2	3	13	257

4.2.2. *Message Post-processing*

Here, all the values present in decrypted message greater than 255 are discarded. So, decrypted message (97 98 256 98 99 100 256 257) becomes (97 98 98 99 100) which is the original message.

5. Proposed Method

In order to increase the confusion rate and avalanche effect, 4D Playfair cipher is amalgamated with LFSR. This proposed method has mainly six steps for encryption. They are Key matrix formation, Message pre-processing, Block matrix formation, Circular shifting of bits of values in Block matrix, Shuffling of values in Block matrix, Block matrix encryption by key matrix shifting. There are six steps for decryption. They are Key matrix formation, Block matrix formation, Block matrix decryption by key matrix shifting, Reverse shuffling of values in Block matrix, Reverse circular shifting of bits of values in Block matrix, Message post-processing. Pseudocodes for encryption process and decryption process are given below.

```
EncryptionProcess ()
{
    Form key matrix from the key
    Pre-process the message got from user or read from file
    Form block matrices from pre-processed message

    for each block matrix formed
        Circularly shift bits of values in block matrix
        Shuffle values in block matrix
        Perform block matrix encryption by key matrix shifting
    end for
}

DecryptionProcess ()
{
    Form key matrix from the key
    Form block matrices from encrypted message got from sender or read from encrypted file

    for each block matrix formed
        Perform block matrix decryption by key matrix shifting
        Reverse shuffle values in block matrix
        Reverse circularly shift bits of values in block matrix
    end for

    Post-process message
}
```

The way of forming key matrix and block matrix at both encryption and decryption sides is same. Key matrix formation, Message pre-processing and Message post-processing are same as that of 4D Playfair cipher. The Circular shifting of bits, Reverse circular shifting of bits, Block matrix encryption and Block matrix decryption steps in the proposed approach use LFSR. So, before going to elaborate each step, LFSR implementation is discussed for the proposed method.

5.1. *Linear Feedback Shift Register (LFSR)*

A LFSR is a shift register whose input state is a linear function of its immediate former state. The only linear functions of single bits are XOR and XNOR. Thus, it is a shift register whose input bit is determined by the XOR or XNOR of some bits of the overall shift register value. The initial value of LFSR is called seed, the stream values produced by the register is completely driven by immediate former state. The bit position that affects next state is called tap. LFSR can generate different random sequences by changing the seed and taps. A x flip flops LFSR can generate at most a sequence of $2^x - 1$ random numbers. When the period is precisely $2^x - 1$, the sequence is called an x-sequence [Haykin, (2008)], [Wayne and Tomasi, (2008)]. For the proposed method, LFSR design is shown in Fig. 1.

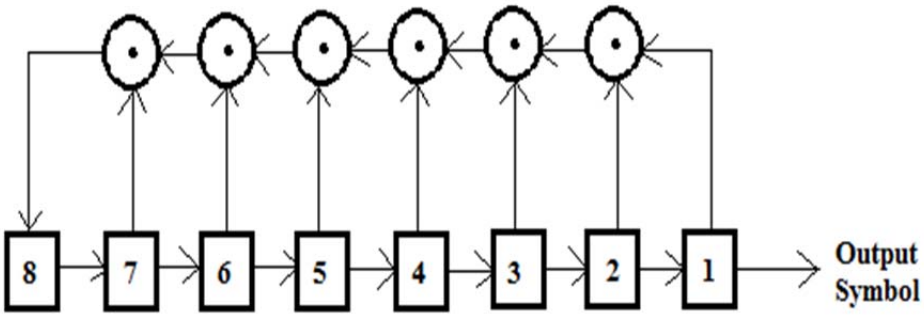

Fig. 1. 8 flip flops LFSR with XNOR operation

LFSR consists of 8 flip flops. So, a maximum of $2^8 - 1 = 255$ random numbers can be generated. Tap bits are 1, 2, 3, 4, 5, 6 and 7. The LFSR pseudocode is given below. The procedure LFSR takes as input a Seed array of size 8 and an empty OutputSymbol array.

```
LFSR (Seed[8], OutputSymbol[ ])
{
    k = 0;
    Copy Seed array contents to lfsr array;
    do
        y = 0;

        for i = 0 to 6
            y = y XNOR lfsr[i];
        end for

        k = k + 1;
        OutputSymbol[k] = lfsr[0];

        for i = 1 to 7
            lfsr[i-1] = lfsr[i];
        end for

        lfsr[7] = y;
   until (lfsr array contents != Seed array contents and no consecutive 8 values in OutputSymbols array is repeated)
 }
```

In the above pseudocode, Seed array holds the initial 8 bit sequence which will be the initial state of LFSR. At the termination of above pseudocode, OutputSymbols array consists of a sequence of binary values using which random numbers are generated. For the propsed method, Seed value is computed by performing XNOR operation on least 8 bits of the values in the key. So, for the key (109 111 110 97 114 99 104 121), Seed value will be 242. So, Seed array contents are 1, 1, 1, 1, 0, 0, 1, 0. OutputSymbols array contents generated using Seed value 242 is:

0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,

0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1.

Now, by combining the consecutive 8 bits, like bits in positions from 1 to 8, 2 to 9, 3 to 10 and so on, from above binary sequence, random numbers sequence generated is:

79, 159, 62, 124, 248, 240, 225, 194, 132, 9, 18, 37, 75, 150, 45, 90, 181, 107, 215, 174, 92, 185, 115, 231, 206, 156, 57, 114, 229, 203, 151, 47, 95, 190, 125, 250, 245, 234, 212, 169, 82, 164, 72, 145, 35, 71, 142, 29, 58, 117, 235, 214, 172, 89, 178, 101, 202, 149, 42, 84, 168, 80, 161, 67, 135, 14, 28, 56, 112, 224, 192, 129, 2, 4, 8, 16, 32, 64, 128, 0, 1, 3, 6, 13, 27, 54, 109, 219, 182, 108, 217, 179, 103, 207, 158, 60, 121, 243, 230, 204, 153, 50, 100, 200, 144, 33, 66, 133, 11, 23, 46, 93, 187, 118, 236, 216, 177, 98, 196, 136, 17, 34, 69, 139, 22, 44, 88, 176, 96, 193, 131, 7, 15, 30, 61, 123, 246, 237, 218, 180, 105, 210, 165, 74, 148, 40, 81, 163, 70, 140, 24, 49, 99, 198, 141, 26, 52, 104, 208, 160, 65, 130, 5, 10, 21, 43, 86, 173, 91, 183, 110, 220, 184, 113, 226, 197, 138, 20, 41, 83, 166, 77, 154, 53, 106, 213, 171, 87, 175, 94, 188, 120, 241, 227, 199, 143, 31, 63, 126, 253, 251, 247, 239, 223, 191, 127, 255, 254, 252, 249, 242, 228, 201, 146, 36, 73, 147, 38, 76, 152, 48, 97, 195, 134, 12, 25, 51, 102, 205, 155, 55, 111, 222, 189, 122, 244, 232, 209, 162, 68, 137, 19, 39, 78, 157, 59, 119, 238, 221, 186, 116, 233, 211, 167.

5.2. *Block matrix formation*

For both encryption and decryption, this step is common. Here, a pre-processed/encrypted message is divided into block matrices. Each block matrix is a matrix of size R × 4 with R rows and 4 columns which will hold R × 4 values of pre-processed/encrypted message. Value of R depends on the length of pre-processed/encrypted message and its maximum value is 256. If L is the pre-processed/encrypted message length than R can be calculated using pseudocode given below. The procedure FindBlockMatrixRowCount takes pre-processed/encrypted message length L as input. It can be seen from pseudocode that value of L is decremented after forming each block matrix. Until L becomes zero, new block matrices are formed using the same formula.

```
FindBlockMatrixRowCount (L)
{
    R = L / 4;

    if 256 < R
        R = 256;
    end if

    L = L – (R * 4);
}
```

For example, If L is 1064 then first a block matrix of size 256 × 4 is formed to hold first 1024 values in pre-processed/encrypted message and another block matrix of size 10 × 4 is formed to hold rest 40 values. In general, after processing first block matrix, second block matrix is processed than third and so on.

5.3. *Circular shifting of bits of values in a block matrix*

This step is a part of encryption process and it comes after block matrix formation. Here, random numbers generated using LFSR are used to perform circular bits shifting of values less than 256 in a block matrix. The pseudocode is given below. The procedure CicularShiftBits accepts as input a block matrix Block[R][4], row count R, array of random numbers RandNum[N] with N random numbers and an index to a particular random number in RandNum array which has to be used next. From the pseudocode it can be seen that bits of each value less than 256 in a block matrix, read row wise from index 0 to R-1 and column wise from index 0 to 3 are circularly shifted to either right or left based on the parity of random number. If parity is even shift left, else right in circular fashion.

```
CicularShiftBits (Block[R][4], R, RandNum[N], N, Bitshiftindex)
{
    for r = 0 to R-1
        for c = 0 to 3
```

```
            if 256 > Block[r][c]

                if RandNum[Bitshiftindex] is even
                    Circular left shift bits of Block[r][c] by ((RandNum[Bitshiftindex] mod 7) + 1);
                else
                    Circular right shift bits of Block[r][c] by ((RandNum[Bitshiftindex] mod 7) + 1);
                end if

                Bitshiftindex = Bitshiftindex + 1;
                if Bitshiftindex > N
                    Bitshiftindex = 1;
                endif

            end if

        end for
    end for
}
```

For example, a pre-processed message of length 8 is taken and its corresponding block matrix with values 10, 20, 30, 40, 50, 60, 70 and 80 is shown below whose binary values are 00001010, 00010100, 00011110, 00101000, 00110010, 00111100, 01000110 and 01010000 respectively.

$$\begin{pmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \end{pmatrix}$$

Taking (109 111 110 97 114 99 104 121) as key, random numbers generated are shown in Section 5.1. So, the first 8 random numbers in RandNum array will be 79, 159, 62, 124, 248, 240, 225 and 194 which are used to circularly shift the bits of 10, 20, 30, 40, 50, 60, 70 and 80 respectively. The process of conversion is illustrated in Table 8. It can be seen from Table 8 that 10 is converted to 65, 20 is converted to 80, 30 is converted to 15, 40 is converted to 10, 50 is converted to 35, 60 is converted to 225, 70 is converted to 145 and 80 is converted to 20.

Table 8. Circular shifting of bits of 10, 20, 30, 40, 50, 60, 70 and 80

Value	Binary	Number of bit shifts	Direction	Resultant Binary	Resultant Value
10	00001010	(79 mod 7) + 1 = 3	Right	01000001	65
20	00010100	(159 mod 7) + 1 = 6	Right	01010000	80
30	00011110	(62 mod 7) + 1 = 7	Left	00001111	15
40	00101000	(124 mod 7) + 1 = 6	Left	00001010	10
50	00110010	(248 mod 7) + 1 = 4	Left	00100011	35
60	00111100	(240 mod 7) + 1 = 3	Left	11100001	225
70	01000110	(225 mod 7) + 1 = 2	Right	10010001	145
80	01010000	(194 mod 7) + 1 = 6	Left	00010100	20

The resultant block matrix formed after circular shifting is shown below.

$$\begin{pmatrix} 65 & 80 & 15 & 10 \\ 35 & 225 & 145 & 20 \end{pmatrix}$$

5.4. *Shuffling of values in a block matrix*

This step is a part of encryption process. After circular shifting of bits of values, values in each column are shuffled based on the value of XOR operation performed on all values less than 256 in that column. The pseudocode for shuffling is given below. The procedure ShuffleValues takes as input a block matrix Block[R][4], row count R. It can be seen that shuffling is not done when R in a Block is 1. Also, Shuffling of odd indexed columns and even indexed columns are different. Values in even indexed columns replace the values that are in rows after them in circular fashion and values in odd indexed columns replace the values that are in rows before them in circular fashion. For the odd indexed columns, there may or may not be shuffling of values if row count is less than 256. But, if the number of rows in a block matrix is 256 then there will always be a shuffling since XOR value lies in the range 0 to 255.

```
ShuffleValues (Block[R][4], R)
{
```

```
    if R > 1
        Column[R] = {0}; //Temporary array to hold values in a column after shuffling

        for c = 0 to 3
            x = 0;

            // Calculate XOR value of all values in a column
            for r = 0 to R-1
                if 256 > Block[r][c]
                    x = x XOR Block[r][c]
                end if
            end for

            if c == 0 or c == 2
                // Store the Shuffled column values in a temporary array
                for r = 0 to R-1
                    Column[r] = Block[(r + (x mod (R - 1)) + 1) mod R][c];
                end for

                // Copy the temporary array elements to Block matrix
                for r = 0 to R-1
                    Block[r][c] = Column[r];
                end for
            else
                // Store the Shuffled column values in a temporary array
                for r = R-1 to 0
                    Column[r] = Block[(r + R - (x mod R)) mod R][c];
                end for

                // Copy the temporary array elements to Block matrix
                for r = R-1 to 0
                    Block[r][c] = Column[r];
                end for
            end if
        end for
    end if
}
```

For example, after circular shifting of bits of values, a block matrix of size 3 × 4 will produce resultant values 90, 78, 45, 110, 26, 220, 194, 156, 176, 166, 256 and 257 as shown below.

$$\begin{pmatrix} 90 & 78 & 45 & 110 \\ 26 & 220 & 194 & 156 \\ 176 & 166 & 256 & 257 \end{pmatrix}$$

Values of XOR operations performed on all values of individual columns 1, 2, 3 and 4 are 240, 52, 239 and 242 respectively. After passing this block matrix via above procedure, value in each row of 1st column replaces the value in the next second row in circular fashion; value in each row of 2nd column replaces the value in the previous second row in circular fashion; value in each row of 3rd column replaces the value in the next row in circular fashion and value in each row of 4th column replaces the value in the previous row in circular fashion. Resultant block matrix formed is shown below.

$$\begin{pmatrix} 26 & 166 & 256 & 156 \\ 176 & 78 & 45 & 257 \\ 90 & 220 & 194 & 110 \end{pmatrix}$$

5.5. *Block matrix encryption by key matrix shifting*

This step is a part of encryption process and comes after shuffling of values. Since each row in a block matrix consists of 4 values, each row values form a quartet. Now, each quartet is encrypted using the encryption procedure of 4D Playfair cipher. After encrypting each quartet, the key matrix is shifted based on the parity of

random numbers generated using LFSR. If random number N is odd than shift right by N positions, else shift left by N positions in circular fashion. New key matrix formed is used for encrypting next quartet. This key matrix shifting is done after encrypting each quartet in the message except after the last quartet in last block matrix formed.

For example, a pre-processed message of length 12 is taken and its corresponding block matrix whose values after shuffling of values is shown below.

$$\begin{pmatrix} 97 & 98 & 99 & 100 \\ 101 & 102 & 103 & 104 \\ 105 & 106 & 107 & 108 \end{pmatrix}$$

The key used for encryption is (109 111 110 97 114 99 104 121). The random numbers generated for this key is shown in Section 5.1. The first two random numbers among them are 79 and 159. The key matrix for this key is shown in Table 1. Using this key matrix shown in Table 1, the quartet (97 98 99 100) is encrypted as (183 13 45 163). Now, the random number 79 is used for shifting key matrix shown in Table 1. Since number 79 is odd, key matrix is shifted right by 79 positions in circular fashion and the resultant key matrix is shown in Table 9.

Table 9. Key matrix after shifting key matrix shown in Table 1 to right by 79 positions

D1_P1					D1_P2				
181	182	183	184	185	246	247	248	249	250
186	187	188	189	190	251	252	253	254	255
191	192	193	194	195	256	257	258	259	109
196	197	198	199	200	111	110	97	114	99
201	202	203	204	205	104	121	0	1	2
206	207	208	209	210	3	4	5	6	7
211	212	213	214	215	8	9	10	11	12
216	217	218	219	220	13	14	15	16	17
221	222	223	224	225	18	19	20	21	22
226	227	228	229	230	23	24	25	26	27
231	232	233	234	235	28	29	30	31	32
236	237	238	239	240	33	34	35	36	37
241	242	243	244	245	38	39	40	41	42

D2_P1					D2_P2				
43	44	45	46	47	115	116	117	118	119
48	49	50	51	52	120	122	123	124	125
53	54	55	56	57	126	127	128	129	130
58	59	60	61	62	131	132	133	134	135
63	64	65	66	67	136	137	138	139	140
68	69	70	71	72	141	142	143	144	145
73	74	75	76	77	146	147	148	149	150
78	79	80	81	82	151	152	153	154	155
83	84	85	86	87	156	157	158	159	160
88	89	90	91	92	161	162	163	164	165
93	94	95	96	98	166	167	168	169	170
100	101	102	103	105	171	172	173	174	175
106	107	108	112	113	176	177	178	179	180

Using key matrix shown in Table 9 as reference, the next quartet (101 102 103 104) is encrypted as (18 18 219 92). Now, the random number 159 is used for shifting key matrix shown in Table 9. Since number 159 is odd, key matrix is shifted right by 159 positions in circular fashion to form a new key matrix. Using the newly formed key matrix as reference, the next quartet (105 106 107 108) is encrypted as (244 78 171 138). The resultant encrypted message in the block matrix is shown below.

$$\begin{pmatrix} 183 & 13 & 45 & 163 \\ 18 & 18 & 219 & 92 \\ 244 & 78 & 171 & 138 \end{pmatrix}$$

5.6. *Block matrix decryption by key matrix shifting*

It is the reverse process of block matrix encryption and the first step in decryption after forming the block matrix from encrypted message. Since each row in a block matrix consists of 4 values, each row values form a quartet. Now, each quartet is decrypted using the decryption procedure of 4D Playfair cipher. After decrypting each quartet, the key matrix is shifted based on the parity of random numbers generated using LFSR. If random

number N is odd than shift right by N positions, else shift left by N positions in circular fashion. New key matrix formed is used for decrypting next quartet. This key matrix shifting is done after decrypting each quartet in the message except after the last quartet in last block matrix formed.
For example, an encrypted message of length 12 is taken and its corresponding block matrix is shown below.

$$\begin{pmatrix} 183 & 13 & 45 & 163 \\ 18 & 18 & 219 & 92 \\ 244 & 78 & 171 & 138 \end{pmatrix}$$

The key used for decryption is (109 111 110 97 114 99 104 121). The random numbers generated for this key is shown in Section 5.1. The first two random numbers among them are 79 and 159. The key matrix for this key is shown in Table 1. Using this key matrix shown in Table 1, the quartet (183 13 45 163) is decrypted as (97 98 99 100). Now, the random number 79 is used for shifting key matrix shown in Table 1. Since number 79 is odd, key matrix is shifted right by 79 positions in circular fashion and the resultant key matrix is shown in Table 9. Using key matrix shown in Table 9 as reference, the next quartet (18 18 219 92) is decrypted as (101 102 103 104). Now, the random number 159 is used for shifting key matrix shown in Table 9. Since number 159 is odd, key matrix is shifted right by 159 positions in circular fashion to form a new key matrix. Using the newly formed key matrix as reference, the next quartet (244 78 171 138) is decrypted as (105 106 107 108). The resultant decrypted message in a block matrix is shown below.

$$\begin{pmatrix} 97 & 98 & 99 & 100 \\ 101 & 102 & 103 & 104 \\ 105 & 106 & 107 & 108 \end{pmatrix}$$

5.7. *Reverse shuffling of values in a block matrix*

This step is a part of decryption process which comes after block matrix decryption and is the reverse process of shuffling of values during encryption. After decrypting each quartet in a block matrix, values in each column are shuffled based on the value of XOR operation performed on all values less than 256 in that column. Here, values in even indexed columns replace the values that are in rows before them in circular fashion and values in odd indexed columns replace the values that are in rows after them in circular fashion. For the odd indexed columns, there may or may not be shuffling of values if row count is less than 256. But, if the number of rows in a block matrix is 256 then there will always be a reverse shuffling since XOR value lies in the range 0 to 255.
For example, a block matrix of size 4 × 4 is taken with values 26, 166, 256, 156, 176, 78, 45, 257, 90, 220, 194 and 110 as shown below which is the resultant matrix obtained after performing shuffling of values during encryption on the example taken in Section 5.4.

$$\begin{pmatrix} 26 & 166 & 256 & 156 \\ 176 & 78 & 45 & 257 \\ 90 & 220 & 194 & 110 \end{pmatrix}$$

Values of XOR operations performed on all values of individual columns 1, 2, 3 and 4 are 240, 52, 239 and 242 respectively. After passing this block matrix via above procedure, value in each row of 1st column replaces the value in the previous second row in circular fashion; value in each row of 2nd column replaces the value in the next second row in circular fashion; value in each row of 3rd column replaces the value in the previous row in circular fashion and value in each row of 4th column replaces the value in the next row in circular fashion. Resultant block matrix formed is shown below.

$$\begin{pmatrix} 90 & 78 & 45 & 110 \\ 26 & 220 & 194 & 156 \\ 176 & 166 & 256 & 257 \end{pmatrix}$$

5.8. *Reverse circular shifting of bits of values in a block matrix*

This step is a part of decryption process which comes after reverse shuffling of values and is the reverse process of circular shifting of bits during encryption. Here, random numbers generated using LFSR are used to perform reverse circular bits shifting of values less than 256 in a block matrix. Here, bits of the values are circularly shifted to either right or left based on the parity of random number. If parity is odd shift left, else right in circular fashion.
For example, an encrypted message of length 8 is taken and its corresponding block matrix after reverse shuffling of values with resultant values 65, 80, 15, 10, 35, 225, 145 and 20 is shown below whose binary values are 01000001, 01010000, 00001111, 00001010, 00100011, 11100001, 10010001 and 00010100 respectively.

$$\begin{pmatrix} 65 & 80 & 15 & 10 \\ 35 & 225 & 145 & 20 \end{pmatrix}$$

Taking (109 111 110 97 114 99 104 121) as key, random numbers generated are shown in Section 5.1. So, the first 8 random numbers will be 79, 159, 62, 124, 248, 240, 225 and 194 which are used to reversely circular shift the bits of 65, 80, 15, 10, 35, 225, 145 and 20 respectively. The process of conversion is illustrated in Table 10. It can be seen from Table 10 that 65 is converted to 10, 80 is converted to 20, 15 is converted to 30, 10 is converted to 40, 35 is converted to 50, 225 is converted to 60, 145 is converted to 70 and 20 is converted to 80.

Table 10. Reverse circular shifting of bits of 65, 80, 15, 10, 35, 225, 145 and 20

Value	Binary	Number of bit shifts	Direction	Resultant Binary	Resultant Value
65	01000001	(79 mod 7) + 1 = 3	Left	00001010	10
80	01010000	(159 mod 7) + 1 = 6	Left	00010100	20
15	00001111	(62 mod 7) + 1 = 7	Right	00011110	30
10	00001010	(124 mod 7) + 1 = 6	Right	00101000	40
35	00100011	(248 mod 7) + 1 = 4	Right	00110010	50
225	11100001	(240 mod 7) + 1 = 3	Right	00111100	60
145	10010001	(225 mod 7) + 1 = 2	Left	01000110	70
20	00010100	(194 mod 7) + 1 = 6	Right	01010000	80

The resultant block matrix formed after reverse circular shifting is shown below.

$$\begin{pmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \end{pmatrix}$$

6. An Example of Proposed Method

The byte values of a message of length 11 bytes which has to be secured are 97, 98, 99, 100, 101, 102, 103, 104, 105, 106 and 107. The key used is (109 111 110 97 114 99 104 121). The encryption and decryption processes are explained below.

6.1. *Encryption process*

At first, key matrix as shown in Table 1 is formed from the key.

6.1.1. *Message pre-processing*

Message (97 98 99 100 101 102 103 104 105 106 107) is divided into quartets (97 98 99 100), (101 102 103 104) and (105 106 107 256). Therefore, pre-processed message is (97 98 99 100 101 102 103 104 105 106 107 256).

6.1.2. *Block matrix formation*

There is only one block matrix formed from the pre-processed message (97 98 99 100 101 102 103 104 105 106 107 256) which is shown below.

$$\begin{pmatrix} 97 & 98 & 99 & 100 \\ 101 & 102 & 103 & 104 \\ 105 & 106 & 107 & 256 \end{pmatrix}$$

6.1.3. *Circular shifting of bits of values*

The bits of the values 97, 98, 99, 100, 101, 102, 103, 104, 105, 106 and 107 (256 is not considered) in the block matrix are circularly shifted based on the random numbers 79, 159, 62, 124, 248, 240, 225, 194, 132, 9 and 18 respectively that are generated using LFSR as shown in Section 5.1. After circularly shifting the bits of values, 97 becomes 44, 98 becomes 137, 99 becomes 177, 100 becomes 25, 101 becomes 86, 102 becomes 51, 103 becomes 217, 104 becomes 26, 105 becomes 180, 106 becomes 77 and 107 becomes 109. The resultant block matrix formed is shown below.

$$\begin{pmatrix} 44 & 137 & 177 & 25 \\ 86 & 51 & 217 & 26 \\ 180 & 77 & 109 & 256 \end{pmatrix}$$

6.1.4. *Shuffling of values*

Value of XOR operation performed on 1st column values 44, 86 and 180 is 206 and value in each row of 1st column replaces the value in next second row in circular fashion. Value of XOR operation performed on 2nd

column values 137, 51 and 77 is 247 and value in each row of 2nd column replaces the value in previous second row in circular fashion. Value of XOR operation performed on 3rd column values 177, 217 and 109 is 5 and value in each row of 3rd column replaces the value in next row in circular fashion. Value of XOR operation performed on 4th column values 25 and 26 (256 is not considered) is 3 and 4th column values are not shuffled. The resultant block matrix is shown below.

$$\begin{pmatrix} 86 & 77 & 109 & 25 \\ 180 & 137 & 177 & 26 \\ 44 & 51 & 217 & 256 \end{pmatrix}$$

6.1.5. *Block matrix encryption*

Now, each row values that form a quartet in the block matrix formed after shuffling of values is encrypted. By using key matrix shown in Table 1 as reference, the quartet (86 77 109 25) is encrypted as (17 77 60 26). Now, key matrix shown in Table 1 is shifted 79 times to the right forming key matrix shown in Table 9. Using key matrix shown in Table 9 as reference, the next quartet (180 137 177 26) is encrypted as (177 121 179 165). Now, key matrix shown in Table 9 is shifted 159 times to the right forming a new key matrix. Using the newly formed key matrix as reference, the last quartet (44 51 217 256) is encrypted as (249 187 18 122). So, the cipher message formed is (17 77 60 26 177 121 179 165 249 187 18 122).

6.2. ***Decryption process***

At first, key matrix as shown in Table 1 is formed from the key.

6.2.1. *Block matrix formation*

There is only one block matrix formed from the encrypted message (17 77 60 26 177 121 179 165 249 187 18 122) as shown below.

$$\begin{pmatrix} 17 & 77 & 60 & 26 \\ 177 & 121 & 179 & 165 \\ 249 & 187 & 18 & 122 \end{pmatrix}$$

6.2.2. *Block matrix decryption*

Now, each row values that form a quartet in the formed block matrix are decrypted. By using key matrix shown in Table 1 as reference, the quartet (17 77 60 26) is decrypted as (86 77 109 25). Now, key matrix shown in Table 1 is shifted 79 times to the right forming key matrix shown in Table 9. Using key matrix shown in Table 9 as reference, the next quartet (177 121 179 165) is decrypted as (180 137 177 26). Now, key matrix shown in Table 9 is shifted 159 times to the right forming a new key matrix. Using the newly formed key matrix as reference, the last quartet (249 187 18 122) is decrypted as (44 51 217 256). The resultant block matrix formed is shown below.

$$\begin{pmatrix} 86 & 77 & 109 & 25 \\ 180 & 137 & 177 & 26 \\ 44 & 51 & 217 & 256 \end{pmatrix}$$

6.2.3. *Reverse shuffling of values*

Value of XOR operation performed on 1st column values 86, 180 and 44 is 206 and value in each row of 1st column replaces the value in previous second row in circular fashion. Value of XOR operation performed on 2nd column values 77, 137 and 51 is 247 and value in each row of 2nd column replaces the value in next second row in circular fashion. Value of XOR operation performed on 3rd column values 109, 177 and 217 is 5 and value in each row of 3rd column replaces the value in previous row in circular fashion. Value of XOR operation performed on 4th column values 25 and 26 (256 is not considered) is 3 and 4th column values are not shuffled. The resultant block matrix is shown below.

$$\begin{pmatrix} 44 & 137 & 177 & 25 \\ 86 & 51 & 217 & 26 \\ 180 & 77 & 109 & 256 \end{pmatrix}$$

6.2.4. *Reverse circular shifting of bits of values*

The bits of the values 44, 137, 177, 25, 86, 51, 217, 26, 180, 77 and 109 (256 is not considered) in the block matrix are reverse circularly shifted based on the random numbers 79, 159, 62, 124, 248, 240, 225, 194, 132, 9

and 18 respectively that are generated using LFSR as shown in Section 5.1. After reverse circularly shifting the bits of values, 44 becomes 97, 137 becomes 98, 177 becomes 99, 25 becomes 100, 86 becomes 101, 51 becomes 102, 217 becomes 103, 26 becomes 104, 180 becomes 105, 77 becomes 106 and 109 becomes 107. The resultant block matrix formed is shown below.

$$\begin{pmatrix} 97 & 98 & 99 & 100 \\ 101 & 102 & 103 & 104 \\ 105 & 106 & 107 & 256 \end{pmatrix}$$

6.2.5. *Post-process message*

Now, all the values greater than 255 present in the decrypted message (97 98 99 100 101 102 103 104 105 106 107 256) are discarded to get back the original message (97 98 99 100 101 102 103 104 105 106 107).

The proposed cipher is implemented using C programming and executed in a computer having Ubuntu 16.0 Operating System, 2GB RAM and 64-bit processor with 2.16GHz speed. Times taken for encryptions and decryptions of different data sizes are shown in Table 11.

Table 11. Encryption and decryption times for different data sizes

Data size (in bytes)	Encryption time (in micro seconds)	Decryption time (in micro seconds)
5	14	12
50	58	57
500	508	505
5000	4994	4966
50000	50921	49748

7. Security Analysis

There are mainly five types of cryptanalytic attacks. They are ciphertext only attack, known plaintext attack, chosen plaintext attack, chosen ciphertext attack and chosen text attack [Stallings, (2014)].

Brute force attack is used in ciphertext only or known plaintext attack where every possible key is applied by an adversary to get the original message from cipher message [Forouzan and Mukhopadhyay, (2011)]. For Playfair ciphers, brute force attack is done on the basis of number of possible keys and number of possible digrams/trigrams/quartets. For 4D Playfair cipher, possible number of keys or key matrices is ${}^{260}P_{260}$ ($\approx 10^{516}$) which is very huge. Possible number of quartets with no repeating values and no fillers is ${}^{256}P_4$. Possible number of quartets with no fillers but repeating values only at location one and three is $256 \times {}^{255}P_2$ and only at location two and four is $256 \times {}^{255}P_2$. Possible number of quartets of the form (i 256 i 257) where 0 <= i <= 255 is 256. Possible number of quartets of the form (i j 256 j) where 0 <= i, j <= 255 and $i \neq j$ is 256 * 255. Possible number of quartets of the form (i 256 i j) where 0 <= i, j <= 255 and $i \neq j$ is 256×255. Possible number of quartets of the form (i j i 256) where 0 <= i, j <= 255 and $i \neq j$ is ${}^{256}P_3$. So, a quartet can be any one of ${}^{256}P_4 + 256 \times {}^{255}P_2 + 256 \times {}^{255}P_2 + 256 + 256 \times 255 + 256 \times 255 + {}^{256}P_3 = 49874176$ quartets which is a large set to search.

More is the confusion rate and avalanche effect, less prone is the cipher to chosen plaintext, chosen ciphertext and chosen text attacks [Forouzan and Mukhopadhyay, (2011)]. In the proposed cipher, initially circular shifting of bits are done based on random numbers generated using LFSR then shuffling is done. Afterwards, a quartet is encrypted then key matrix is shifted to form a new one which will be used to encrypt next quartet. So, a total of four levels of confusion are present leading to a high confusion rate. This confusion can be further increased by repeating all four steps multiple times for each quartet in each block matrix. If repetition is done 16 times, 64 levels of confusion are added. Based on the level of security required by the application, number of repetitions is chosen.

Avalanche effect with respect to one bit change in key is high because seed value to LFSR will change leading to change in the generated random number series which in turn causes change in circular bit shifting and key matrix shifting patterns.

For example, a message whose byte values are (97 98 99 100 101 102 103 104 105 106 107 108) is considered for encryption with the key (109 111 110 97 114 99 104 121). The corresponding cipher message generated is (17 215 58 6 177 121 178 165 184 49 16 89). The bit representation of the cipher message is given below where each value is represented using 9 bits since a value in cipher message will be in range 0 to 259.

000010001 011010111 000111010 000000110 0010110001 001111001 010110010 010100101 10111000 000110001 000010000 001011001

Now, for the message one bit change in the key is done resulting to the new key (109 111 110 96 114 99 104 121). The cipher message generated is (234 187 158 2 13 155 196 139 76 148 113 139). The corresponding bit representation is shown below.

011101010 010111011 010011110 000000010 000001101 010011011 011000100 010001011 001001100 010010100 001110001 010001011

From the above two bit representations it can be seen that by changing one bit value in the key there are changes in 49 bits in cipher messages.
Avalanche effect with respect to one bit change in plain message depends on the number of rows in a block matrix formed. If the number of rows is 256 then avalanche effect is high since by changing one bit of plain message, shuffling pattern in a column will vary and if number of rows is less than 256, avalanche effect can be either high or moderate. It will be high if shuffling pattern varies; else will be moderate if shuffling pattern remains the same.
For example, a message whose byte values are (97 98 99 100 101 102 103 104 105 106 107 108) is considered for encryption with the key (109 111 110 97 114 99 104 121). There will be block matrix formed with 3 rows. The corresponding cipher message generated is (17 215 58 6 177 121 178 165 184 49 16 89). The bit representation is already shown above. Now, using the same key one bit change in message is done forming a new message (97 98 97 100 101 102 103 104 105 106 107 108). The new cipher message formed is (155 217 78 71 39 2 128 165 184 53 36 24). The bit representation of the cipher message is shown below.

010011011 011011001 001001110 001000111 000100111 000000010 010000000 010100101 010111000 000110101 000100100 000011000

It can be seen that changing one bit in plain message is affecting 31 bit locations in cipher message. It is high due to change in shuffling patterns of both encryptions. It would have been moderate if shuffling patterns were same. Avalanche effect with respect to one bit change in the key and plain message can be further increased by repeating the four steps during encryption process multiple times for each quartet in each block matrix. From the analysis it can be inferred that proposed cipher is strong against all five major cryptanalytic attacks.

8. **Comparison Analysis**

Table 12. Comparison table

Source	Number of characters/ values supported	Possible number of keys	Works with	Possible number of digrams/trigrams / quartets	Levels of confusion	Avalanche effect with respect to 1 bit change in		Ambiguity
						Key	Plain message	
[Stallings, (2014)]	26	$\approx 10^{25}$	digrams	650	1	low	low	ambiguous
[Murali and Senthilkumar, (2009)]	26	$\approx 10^{25}$	digrams	650	2	low	low	ambiguous
[Sastry *et al*., (2009)]	128	$\approx 10^{126}$	digrams	16384	33	high	high	unambiguous
[Srivastava and Gupta, (2011)]	64	$\approx 10^{89}$	digrams	4032	2	high	high	ambiguous
[Basu and Ray, (2012)]	90	$\approx 10^{138}$	digrams	8010	1	low	low	ambiguous
[Dhenakaran and Ilayaraja, (2012)]	256	$\approx 10^{506}$	digrams	65536	1	low	low	unambiguous
[Kaur *et al*., (2012a)]	36	$\approx 10^{41}$	digrams	1260	2	low	low	ambiguous
[Kaur *et al*., (2012b)]	64	$\approx 10^{89}$	trigrams	254016	1	low	low	ambiguous
[Alam *et al*., (2013)]	26	$\approx 10^{26}$	digrams	702	1	low	low	unambiguous
[Kaur *et al*., (2013)]	64	$\approx 10^{89}$	trigrams	254016	2	high	low	ambiguous
[Singh *et al*., (2013)]	64	$\approx 10^{89}$	trigrams	254016	2	high	low	ambiguous
[Verma *et al*., (2013)]	64	$\approx 10^{89}$	trigrams	254016	2	high	low	ambiguous
[Chand and Bhattacharyya, (2014)]	36	$\approx 10^{164}$	digrams	1260	4	low	low	ambiguous
[Hans *et al*., (2014)]	26	$\approx 10^{26}$	digrams	650	3	high	low	ambiguous
[Singh *et al*., (2015)]	64	$\approx 10^{89}$	trigrams	254016	2	high	low	ambiguous
This work	260	$\approx 10^{516}$	quartets	49874176	4	high	high	unambiguous

As shown in Table 12, the comparison between the proposed cipher and existing variants is done on the basis of number of characters/values supported, possible number of keys, possible number of digrams/trigrams/quartets, levels of confusion, avalanche effect with respect to 1 bit change in key and plain message, and ambiguity.
In Table 12, possible number of keys is calculated by using the number of characters/values supported. If N is the number of characters/values supported then possible number of keys is N factorial except for the Playfair variants proposed in [Stallings, (2014)], [Murali and Senthilkumar, (2009)], [Sastry *et al*., (2009)] and [Chand and Bhattacharyya, (2014)]. For variants proposed in [Stallings, (2014)] and [Murali and Senthilkumar, (2009)], it is N - 1 factorial. For variant proposed in [Sastry *et al*., (2009)], it is $^{128}P_{64}$ and in [Chand and Bhattacharyya, (2014)], it is N factorial to the power 4.
Possible number of digrams/trigrams/quartets is computed based on which group the variant works with. For all variants working with digrams except for those proposed in [Sastry *et al*., (2009)], [Dhenakaran and Ilayaraja, (2012)] and [Hans *et al*., (2014)], possible number of digrams is calculated as $N \times (N - 1)$ where N is number of characters supported. For variants proposed in [Sastry *et al*., (2009)] and [Dhenakaran and Ilayaraja, (2012)], it is $N \times N$ and in [Hans *et al*., (2014)], it is $N \times (N - 1) + 2 \times N$. For all variants working with trigrams, possible number of trigrams is calculated as $^{N}P_3 + N \times (N - 1)$ where N is number of characters supported.
Levels of confusion are based on the number of steps in encryption process of the variant. For instance, if only direct substitution is done as in case of Classical Playfair cipher then level is 1. If random numbers generated using LFSR are mapped to secret key and numbers with respect to cipher letters are transmitted as in case of variant proposed in [Murali and Senthilkumar, (2009)] then levels are 2.
It is found from Table 12 that the proposed cipher supports more values when compared to other variants which are sufficient to secure all kinds of data. Also, proposed cipher has large set of possible keys and possible quartets making it stronger against brute force attack when compared to other variants. It has the second highest levels of confusion. But, this can still be increased by repeating all four steps in encryption process as discussed in Section 7. If the repetition is 16 then levels of confusion will become 64 which will be greater than that of variant proposed in [Sastry *et al*., (2009)] where the number of repetitions is also 16. Avalanche effect is also good for the proposed cipher. Hence, the proposed cipher is better than existing variants in terms of security and applicability.

9. Conclusion

It can be concluded from security analysis and comparison analysis that 4D Playfair cipher when coalesces with LFSR becomes strong against all major types of cryptanalytic attacks and can be used to secure all sorts of information without any ambiguity which was not possible with previous versions of Playfair cipher. Hence, the proposed cipher provides better security and is more utile when compared to existing variants.

References

[1] Alam A.; Khalid S.; Salam M. (2013): A Modified Version of Playfair Cipher Using 7×4 Matrix, International Journal of Computer Theory and Engineering, **5** (4), pp. 626-628.
[2] Alfred J. M.; Paul C. V. O.; Scott A. V. (1996): Handbook of applied cryptography, CRC Press: Florida.
[3] Basu S.; Ray U. K. (2012): Modified Playfair Cipher using Rectangular Matrix, International Journal of Computer Applications, **46** (9), pp. 28-30.
[4] Buchmann J. A. (2001): Introduction to Cryptography, 2nd ed., Springer-Verlag: New York.
[5] Chand N.; Bhattacharyya S. (2014): A Novel Approach for Encryption of Text Messages Using PLAY-FAIR Cipher 6 by 6 Matrix with Four Iteration Steps, International Journal of Engineering Science and Innovative Technology, **3** (1), pp. 478-484.
[6] Dhenakaran S. S.; Ilayaraja M. (2012): Extension of Playfair Cipher using 16X16 Matrix, International Journal of Computer Applications, **48** (7), pp. 37-41.
[7] Forouzan B. A.; Mukhopadhyay D. (2011): Cryptography and Network Security, Special Indian Edition, 3rd ed., McGraw-Hill companies: New Delhi.
[8] Hans S.; Johari R.; Gautam V. (2014): An Extended PlayFair Cipher using Rotation and Random Swap patterns, Fifth International Conference on Computer and Communication Technology, pp. 157-160.
[9] Haykin S. (2008): Communication Systems, Wiley India Edition, 3rd ed., Sharda Offset Press: Delhi.
[10] Kaur A.; Verma H. K.; Singh R. K. (2012): 6 X 6 Playfair Cipher using LFSR based Unique Random Number Generator, International Journal of Computer Applications, **51** (2), pp. 30-35.
[11] Kaur A.; Verma H. K.; Singh R. K. (2012): 3D (4 X 4 X 4) - Playfair Cipher, International Journal of Computer Applications, **51** (2), pp. 36-38.
[12] Kaur A.; Verma H. K.; Singh R. K. (2013): 3D - Playfair Cipher using LFSR based Unique Random Number Generator, Sixth International Conference on Contemporary Computing (IC3), pp. 18-23.
[13] Murali P.; Senthilkumar G. (2009): Modified Version of Playfair Cipher using Linear Feedback Shift Register, International Conference on Information Management and Engineering, pp. 488-490.
[14] Sastry V. U.; Shankar N. R.; Bhavani S. D. (2009), A Modified Playfair Cipher Involving Interweaving and Iteration, International Journal of Computer Theory and Engineering, **5** (1), pp. 597-601.
[15] Schneier B. (1996): Applied cryptography: protocols, algorithms and source code in C, 2nd ed., Wiley Computer Publishing, John Wiley and sons, Inc: New York.
[16] Singh S.; Kaur A.; Singh R. K.; Kaur D. (2015): Developing 3D-Playfair Cipher Algorithm Using Structure Rotation, International Conference on Advances in Computer Engineering and Applications, pp. 1004-1008.
[17] Singh S.; Singh R.K.; Kaur A. (2013): 3D - Playfair Cipher using Linear Feedback Shift Register, Fourth International Conference on the Next Generation Information Technology, pp. 164-171.
[18] Srivastava S. S.; Gupta N. (2011): Optimization and Analysis of the Extended Playfair Cipher, International Conference on Emerging

Trends in Networks and Computer Communications, pp. 267-270.
[19] Stallings W. (2014): Cryptography and Network Security Principles and Practice, 6th ed., Pearson Education: United States.
[20] Verma V.; Kaur D.; Singh R. K.; Kaur A. (2013): 3D - Playfair Cipher with additional Bitwise Operation, International Conference on Control, Computing, Communication and Materials (ICCCCM), pp. 1-6.
[21] Wayne; Tomasi (2008): Electronic Communications Systems: Fundamentals Through Advanced, 5th ed., Pearson India.