Research Article

Two-Key Dependent Permutation for Use in Symmetric Cryptographic System

Arulmani Kuppusamy,¹ Swaminathan Pitchai Iyer,² and Kannan Krithivasan³

¹ Department of Computer Science Engineering, SASTRA University, Srinivasa Ramanujan Centre, Kumbakonam 612 001, India

² School of Computing, SASTRA University, Thanjavur 613 401, India

³ School of Humanities and Sciences, SASTRA University, Thanjavur 613 401, India

Correspondence should be addressed to Arulmani Kuppusamy; arulmani.k@src.sastra.edu

Received 23 June 2014; Revised 30 August 2014; Accepted 30 August 2014; Published 2 October 2014

Academic Editor: Cheng Shao

Copyright © 2014 Arulmani Kuppusamy et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper deals with a two-key based novel approach for generating a permutation table that can be used in a symmetric cryptographic system to cause diffusion. It also discusses how the permutation table generated using the approach can be applied to character based encryption and binary data block produced at intermediate stages by symmetric cipher algorithms. It also describes the effect of our approach on characters of intermediate text as well as on bits of binary data block along with the impact of a single bit change in key information on producing permutation sequences applied to plaintexts to produce ciphertexts. The results are satisfactory and the proposed approach can be employed in any symmetric block cipher algorithm that uses the predefined permutation tables.

1. Introduction

The substitution and permutation processes play a very important role in causing confusion and diffusion in the text blocks being generated in various rounds of Feistel or SPN based ciphers. Many of the symmetric block ciphers use identical rounds of substitution and permutation processes in each encryption round with a subkey generation function to generate a different round key for use in each round. The algorithms like DES [1] use predetermined substitution and permutation boxes and others like Blowfish block cipher [2], Khufu algorithm [3], and Twofish [4] utilize the dynamic substitution and permutation boxes. To develop better insight of the work, the terminology and concepts related to cryptography with respect to the work and the description of our algorithm for generating permutation table with a required size are presented in the following section.

2. Materials and Methods

2.1. Functions. A function f from set P to set C is denoted by $f : P \rightarrow C$. If $c \in C$, then a preimage c is an element $p \in P$

for which $f(p) = c$. The set of all elements in C which have at least one preimage is called the image of f . The various types of functions are [5].

2.1.1. Permutation. A permutation p on S , finite set of elements, is a bijection from S to itself. It is denoted by $p : S \rightarrow S$. We will have $n!$ permutations on a set S of n elements.

2.1.2. Bijection Function. If a function $f : P \rightarrow C$ is one-to-one and onto, then f is called a bijection or one-to-one correspondence.

2.1.3. One-to-One Function. A function $f : P \rightarrow C$ is one-to-one (injective) function if each element in P is related to a different element in C .

2.1.4. Onto Function. A function $f : P \rightarrow C$ is said to be onto (surjective) if, for every element c in C , there is an element p in P such that $f(p) = c$. The definition can be restated as a function is onto when its image equals its range.

2.1.5. Inverse Function. If f is a bijection from P to C then there exists a bijection g from C to P such that, for each $c \in C$, $g(c) = p$ holds where $p \in P$ and $f(p) = c$. The function g derived from f is called the inverse function of f and is denoted by $g = f^{-1}$.

2.2. Symmetric Cryptographic Schemes. Cryptography is the study of schemes used for encryption [6] and mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication [5]. The scheme defines a cryptographic system or a cipher that can be used to enable two people, usually referred to as Alice and Bob, to communicate over an insecure channel that could be a telephone line or computer network, without giving chances to understand the scrambled text of what is being sent or guess the key used for transformation of the original text.

The message Alice wants to send is called the plaintext and the scrambled message which Bob receives is called the ciphertext. The plaintext and ciphertext are written in some alphabet consisting of a certain number N of letters including numerals, blanks, punctuation marks, or any other symbols that we use when writing the messages [7].

Alice uses a precomputed key to encrypt the plaintext and sends the ciphertext to Bob over the channel. Only Bob, who knows the encryption key, can decrypt the ciphertext and reconstruct the plaintext.

If Alice and Bob use the same key, the system is referred to as symmetric or conventional encryption. The system is called asymmetric when they use different keys.

Block ciphers and stream ciphers are the two kinds of symmetric-key encryption schemes. A Block cipher processes a block of plaintext at a time to produce a ciphertext block of equal length. On the other hand a stream cipher processes a digital data stream one bit/byte at a time. Many symmetric block encryption algorithms are based on Feistel structure. Such a structure consists of a number of identical rounds of processing involving substitution followed by a permutation [6]. Block ciphers are categorized into substitution ciphers and transposition ciphers. Since their results are in the set of all permutations on the alphabet set A , we present the following known definitions to facilitate better understanding of the role of permutations in the substitution and transposition ciphers.

2.2.1. Substitution Ciphers. In substitution, the letters of plaintext are replaced by other letters or by numbers or symbols. If the plain text is a binary block, substitution involves replacing plaintext bit patterns with cipher bit patterns one bit or a binary block as whole at a time.

Let A be an alphabet of n symbols and P be the set of all strings of length t over A . Let K be the set of all permutations on the set A . for each $e \in K$ an encryption transformation E_e is defined as [5]

$$E_e(p) = (e(p_1)e(p_2)\cdots e(p_t)) = (c_1c_2\cdots c_t) = c, \quad (1)$$

where $p = (p_1p_2\cdots p_t) \in P$.

The inverse permutation $d = e^{-1}$ and $D_d(c) = (d(c_1)d(c_2)\cdots d(c_t)) = (p_1p_2\cdots p_t) = P$ are computed to decrypt $c = (c_1c_2\cdots c_t)$. E_e is called a simple substitution cipher.

Some examples of substitution ciphers are Caesar cipher, monoalphabetic substitution cipher, play fair cipher, hill cipher, and polyalphabetic substitution ciphers.

When a random substitution is performed over the 26-letter English alphabet, dramatic increase in key space is obtained by permuting the elements of the alphabet randomly. The encrypted text will be any permutation of the 26 alphabetic characters from the set of $n!$ permutations making it difficult for cryptanalysis [6].

The ciphers are easy to break since they are monoalphabet based and they reflect the frequency details of the original alphabet. Multiple substitutes for a single letter can be a counter measure in this case. Some multiletter substitution ciphers are Playfair and Hill ciphers, *Vigenère* cipher, autokey system, and so forth.

Even *Vigenère* cipher is vulnerable to cryptanalysis. Because the key and the plaintext share the same frequency distribution of letters, a statistical technique can be applied.

The Playfair cipher is relatively easy to break, because it still leaves much of the structure of the plaintext language intact. A few hundred letters of ciphertext are generally sufficient. Although the Hill cipher is strong against a ciphertext-only attack, it is easily broken with a known plaintext attack [6].

2.2.2. Transposition Ciphers. In this, a very different kind of mapping is achieved by performing some sort of permutation on the plaintext letters. Let t be the block length and K be the set of all permutations on the set $\{1, 2, \dots, t\}$. For each $e \in K$ an encryption transformation E_e is defined as [5]

$$E_e(p) = (p_{e(1)}p_{e(2)}\cdots p_{e(t)}), \quad \text{where } p = (p_1p_2\cdots p_t) \in P. \quad (2)$$

The set of all such transformations is called a simple transposition cipher. The decryption key corresponding to e is the inverse permutation $d = e^{-1}$. $D_d(c) = (c_{d(1)}c_{d(2)}\cdots c_{d(t)})$ is computed to decrypt $c = (c_1c_2\cdots c_t)$.

2.3. Substitution and Permutation Related Works. The modern design of block ciphers is based on the concept of an iterated product cipher. They improve security effectively by combining substitutions and permutations [8]. Both substitution and permutation are added to the encryption in a round [5] to cause confusion and diffusion, respectively, and represent the two basic building blocks for any cryptographic system [8] to thwart statistical cryptanalysis.

In diffusion, the statistical structure of the plaintext is so distributed across the ciphertext that each plaintext element affects the value of many ciphertext elements or vice versa. Diffusion is achieved by repeatedly performing some permutation on the elements of the plain text so as to prevent the cryptanalyst to infer the key. But in confusion relationship between the ciphertext and the key value is hidden to determine the key [6].

Substitution operation is constructed as a box with input bits and output bits. One noticeable characteristic of the S-box is its size. Larger S-boxes are generally more immune to differential and linear cryptanalysis [6] but require more lookup overhead and careful design.

S-boxes can be a precomputed one or built with a predetermined key. The static S-boxes are vulnerable to differential cryptanalysis [9] and linear cryptanalysis [10] and cryptographically insignificant but key dependent S-boxes are insusceptible to these attacks.

Block ciphers like DES [1] and CAST-128 [11] use the precomputed S-Boxes. Blowfish block cipher [2], Khufu algorithm [3], and Twofish [4] employ key dependent S-Boxes. While some others, like GOST [12], utilize variable S-boxes. Permuting bits of plaintext using key will make the cryptanalysis difficult.

A substitution and diffusion based new image 128-bit key encryption scheme proposed in [13] divides an image into several key based dynamic blocks and each block passes through the eight rounds of diffusion for rearranging sequences of block pixels and substitution process to replace block pixels with that of another using row-column difference computation.

A fast permutation algorithm to generate key-dependent S-box and key-dependent P-Box has been discussed in [14] to protect its internal structure so as to resist the linear and differential cryptanalysis in a few round encryptions. In [15], the modified Rijndael algorithm that constructs the S-boxes defined over GF (2^4) has been discussed. Each one of the small S-boxes has different equation and each equation is extracted by using one of the three irreducible polynomials existing in GF (2^4).

Dynamic S-box generation using RC4 algorithm, well known stream cipher, for Advanced Encryption Standard and for AES-128 has been described in [16] and [17], respectively. An S-box generation method through compositions of permutations from some fixed set using random bijective methods has been discussed in [18]. Pseudorandom S-boxes generated using a secret key for AES Block cipher has been described in [19]. Cryptographically key-dependent S-box design using a four-dimensional hyperchaotic Chen system and its suitability for designing block cipher with dynamic S-boxes satisfying the S-box design criteria is suggested in [20].

In all the works above, permutations have been used to prevent cryptanalysis.

2.4. Two-Key Dependent Permutation Algorithm. The algorithm consists of the following major steps.

Step 1 (Initialization). This step initializes the variables y and z with 1 and p which is the product of K_1 and K_2 . We initialize set $I = \{1, 2, 3, \dots, \text{size}\}$ where size denotes the number of letters/bits in the given text block. Initially, C is empty.

Step 2 (Generation). It generates $C_1, C_2, C_3, \dots, C_{\text{size}}$ and adds them to C . During each iteration new values for w, x, y, z , and p are computed.

Step 3 (Replacement). It locates for duplicate elements in the set C generated in the previous step and replaces them with -1 .

Step 4 (Unique Set Construction). It constructs a unique set U of positive integers.

Step 5 (Set Difference Operation). This step produces a set M consisting of elements that are in I but not in U .

Step 6 (Sorting). It sorts the elements of the set M in descending order.

Step 7 (Placement). As the elements of C are scanned from left to right, -1 s and 0 s are replaced by successive elements of M . This process is repeated until the last occurrence of -1 or 0 is replaced with the last element of the set M . The set C is, now consisting of elements from 1 through size but permuted in some order which depends on K_1 and K_2 values.

Algorithm 1. 2KDP (size, K_1, K_2)

(1) Initialize:

$$\begin{aligned} y &= z = 1, \\ p &= K_1 \times K_2, \\ I &= \{1, 2, 3, \dots, \text{size}\}. \end{aligned} \quad (3)$$

(2) Generate a set C of elements C_i for $i = 1$ to size:

$$\begin{aligned} w &= y + (K_2 + i), \\ x &= z + (K_1 + i), \\ y &= y + w, \\ z &= x + y + z, \\ p &= (p + y + z) \% \text{size}, \\ C_i &= \lfloor p \rfloor. \end{aligned} \quad (4)$$

(3) Replace duplicate elements in C by -1 .

(4) Let K denote the number of elements in C where $C_i \leq 0$.

(5) Create a unique set U with size $-K$ elements that satisfy $C_i > 0$:

$$U = \{C_i \mid C_i > 0\} \quad (5)$$

(6) Construct a missing element set M with K elements using set minus operation:

$$M = I - U \quad (6)$$

(7) Sort the elements of set M in descending order.

(8) Replace all $C_i \leq 0$ with successive elements of M .

TABLE 1: Initial position of letters in the message M .

I	1	2	3	4	5	6	7	8
P_i	a	b	c	d	e	f	g	h

Example 2. The algorithm is illustrated with an example.

Inputs

$$\begin{aligned} \text{size} &= 10, \\ K_1 &= 57, \quad K_2 = 89. \end{aligned} \quad (7)$$

(1) Initialization as follows:

$$\begin{aligned} y &= z = 1, \\ p &= 5073, \\ I &= 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10. \end{aligned} \quad (8)$$

(2) Generating the set C using the second step as follows:

$$C = 4 \ 6 \ 5 \ 7 \ 2 \ 6 \ 3 \ 5 \ 4 \ 0 \quad (9)$$

(3) Replacing duplicates by -1 in C as follows:

$$C = 4 \ 6 \ 5 \ 7 \ 2 \ -1 \ 3 \ -1 \ -1 \ 0. \quad (10)$$

(4) Building a unique set U with elements > 0 as follows:

$$U = 4 \ 6 \ 5 \ 7 \ 2 \ 3. \quad (11)$$

(5) Sorting the elements of missing element set M as follows:

$$M = 10 \ 9 \ 8 \ 1. \quad (12)$$

(6) Final permuted set C as follows:

$$C = 4 \ 6 \ 5 \ 7 \ 2 \ 10 \ 3 \ 9 \ 8 \ 1. \quad (13)$$

3. Results and Discussion

This section discusses how the results of the algorithm can be applied to text composed of characters (Character Based Permutation) and bits of binary data block (Binary Based Permutation).

3.1. Character Based Permutation. Let M be the text of length (in characters), $size$. Let I be set of P_i values representing positions of letters in M . There are $size!$ ways in which letters of M can be permuted. Given that text $M = abcdefgh$, the positional set I consists of values in the range from 1 through $size$. Table 1 shows the positions of letters in the text M .

The 2KDP algorithm with input $size = 8$ generates the permutations for various values of K_1 and K_2 , as shown in Table 2 and Figure 1 shows their deviations from the original one.

FIGURE 1: Permutations graph for various values of K_1 and K_2 .

How the permutations $P1$ through $P5$ applied to message M permute elements in M can be understood from Table 3.

By choosing the appropriate values for K_1 and K_2 the algorithm generates permutations that affect positions of all the characters making up the message M .

For example, for a set of 4 different values of K_1 and K_2 , Table 4 shows permutations generated by the algorithm and it can be seen from Table 5 that the permutations have affected all the characters' positions in the message M with $size = 8$.

3.2. Binary Based Permutation. In the binary based permutation, characters in the message M are represented as their 8-bit binary equivalents as shown in Figure 2.

Since the message M consists of 8 characters, the size of the entire message M is 64 bits. The binary representation of the message M is 01100001011000 10011000110110010001-1001010110011001100110110101000 and denoted by the letter B .

A permutation table can be generated by using the 2KDP algorithm with a size equivalent to number of bits of the message M . The permutation table thus generated will consist of 64 entries that indicate how the bits of the binary block B related to the message M should be permuted.

Table 6 shows the initial position of bits of binary data block for the message M . Four 64-bit permutation tables constructed using different values of K_1 and K_2 are shown in Tables 7 through 10. The permutations graphs shown in Figures 3, 4, 5, and 6 help understanding the amount of deviations they have with respect to one another.

By using the permutations tables (see Tables 7, 8, 9, and 10) produced for four different values of K_1 and K_2 , the bits of the text are rearranged and the results are shown in hexadecimal representation in Tables 11, 12, 13, and 14.

The effectiveness of the algorithm on binary bits of the text and how it affects the various binary bits of the text can be understood from Table 15. The bits of the given text are XORed with permuted output to count the number of bits affected due to the permutation algorithm. If the i^{th} bit in the text matches with the i^{th} bit in the permuted output, the i^{th} bit in the text is not affected. Otherwise the i^{th} bit in the text is affected. Table 16 shows the number of bits affected after XORing the permuted output obtained by applying

TABLE 2: Permutations generated for various values of K_1 and K_2 with size: 8.

P. number	K_1	K_2	Permutation sequence							
P1	57	123	8	6	3	7	2	4	5	1
P2	123	57	6	2	5	7	8	4	3	1
P3	-123	57	6	4	5	8	1	2	7	3
P4	57	57	2	8	5	1	4	6	7	3
P5	123	123	8	4	7	1	2	6	5	3

TABLE 3: Impact of permutations on positions of characters of the message M .

P. number	Permuted text								Number of positions affected	
P1	h	f	c	g	b	d	e	a	7	
P2	f	b	e	g	h	d	c	a	7	
P3	f	d	e	h	a	b	g	c	7	
P4	b	h	e	a	d	f	g	c	6	
P5	h	d	g	a	b	f	e	c	7	

TABLE 4: Permutations generated for another set of K_1 and K_2 values with size: 8.

P. number	K_1	K_2	Permutation sequence							
P1	723	142	7	3	6	8	1	5	4	2
P2	-11	7	4	6	7	8	3	2	1	5
P3	57	-123	2	4	7	3	8	6	5	1
P4	123	-57	4	8	5	3	2	7	6	1

TABLE 5: Permutations affecting positions of all characters of the message M .

P. number	Permuted text								Number of positions affected	
P1	g	c	f	h	a	e	d	b	8	
P2	d	f	g	h	c	b	a	e	8	
P3	b	d	g	c	h	f	e	a	8	
P4	d	h	e	c	b	g	f	a	8

FIGURE 2: Binary representation of the message M .TABLE 6: Initial position of bits of binary block for the message M .

B ₁	0	B ₂	1	B ₃	1	B ₄	0	B ₅	0	B ₆	0	B ₇	0	B ₈	1
B ₉	0	B ₁₀	1	B ₁₁	1	B ₁₂	0	B ₁₃	0	B ₁₄	0	B ₁₅	1	B ₁₆	0
B ₁₇	0	B ₁₈	1	B ₁₉	1	B ₂₀	0	B ₂₁	0	B ₂₂	0	B ₂₃	1	B ₂₄	1
B ₂₅	0	B ₂₆	1	B ₂₇	1	B ₂₈	0	B ₂₉	0	B ₃₀	1	B ₃₁	0	B ₃₂	0
B ₃₃	0	B ₃₄	1	B ₃₅	1	B ₃₆	0	B ₃₇	0	B ₃₈	1	B ₃₉	0	B ₄₀	1
B ₄₁	0	B ₄₂	1	B ₄₃	1	B ₄₄	0	B ₄₅	0	B ₄₆	1	B ₄₇	1	B ₄₈	0
B ₄₉	0	B ₅₀	1	B ₅₁	1	B ₅₂	0	B ₅₃	0	B ₅₄	1	B ₅₅	1	B ₅₆	1
B ₅₇	0	B ₅₈	1	B ₅₉	1	B ₆₀	0	B ₆₁	1	B ₆₂	0	B ₆₃	0	B ₆₄	0

TABLE 7: Permutation table PT1 for size: 64; K_1 : 123; K_2 : 57.

22	34	13	15	48	64	47	45	42	38	33	27	20
12	3	57	46	55	21	7	56	40	23	5	50	30
9	51	28	4	43	17	54	26	61	31	35	32	63
29	58	16	49	11	36	60	19	41	62	18	37	10
8	24	25	53	2	14	39	6	44	52	59	1	

TABLE 8: Permutation table PT2 for size: 64; K_1 : 57; K_2 : 123.

24	46	59	31	2	4	5	64	56	39	63	37	54
48	41	33	36	14	3	55	42	28	13	61	44	26
7	51	30	8	49	25	34	38	11	47	18	52	21
53	20	50	15	43	6	32	57	17	40	62	45	29
58	27	35	19	60	10	23	22	16	12	1	9	

TABLE 9: Permutation table PT3 for size: 64; K_1 : 57; K_2 : 57.

34	40	5	41	12	14	15	64	49	47	9	46	44
58	51	43	37	24	13	1	52	38	23	7	54	36
17	61	33	18	59	35	10	48	21	57	28	62	31
63	30	60	25	53	16	42	3	27	50	8	29	32
26	22	39	55	6	20	4	19	56	2	11	45	

TABLE 10: Permutation table PT4 for size: 64; K_1 : 123; K_2 : 123.

16	44	7	9	42	64	41	39	36	32	27	21	14
6	61	51	40	28	15	1	50	34	17	63	60	24
3	45	22	62	37	11	48	20	55	25	58	26	57
23	52	59	43	5	30	54	13	35	56	12	33	49
2	46	31	47	4	8	19	29	38	18	53	10

FIGURE 3: Permutation graph for PT1 with K_1 : 123, K_2 : 57. FIGURE 5: Permutation graph for PT3 with K_1 : 57, K_2 : 57. FIGURE 4: Permutation graph for PT2 with K_1 : 57, K_2 : 123. FIGURE 6: Permutation graph for PT4 with K_1 : 123, K_2 : 123.

the permutation table tables (see Tables 7, 8, 9, and 10) on bits in the given text with original bit stream of the given text.

3.3. Cryptanalysis of the Algorithm. In this section, we discuss the worthiness of our algorithm after having applied different K_1 and K_2 values and tested its results on texts. When

TABLE 11: PB1-permuted bits of binary data block using permutation table PT1.

01010010	11010010	11001110	11010010	11101000	10010010	01011100	10000010
52	D2	CE	D2	E8	92	5C	82

TABLE 12: PB2-permuted bits of binary data block using permutation table PT2.

11101000	10001000	00111001	01011100	11111000	01110000	10001111	01100000
E8	88	39	5C	F8	70	8F	60

TABLE 13: PB3-permuted bits of binary data block using permutation table PT3.

11000010	01010111	01000110	10010111	10000000	10000111	11001001	00011110
C2	57	46	97	80	87	C9	1E

TABLE 14: PB4-permuted bits of binary data block using permutation table PT4.

00001000	00100011	10101100	01100001	00101101	01101101	10001101	01101101
08	23	AC	61	2D	6D	8D	6D

TABLE 15: Permuted bits of binary data for various values of K_1 and K_2 .

Permuted bits of binary data block of the message M with size: 64 bits	K_1	K_2
PB1: 0101001011010010110011101101001011101000100100100101110010000010	123	57
PB2: 111010001000100000111001010111001111000011100001000111101100000	57	123
PB3: 110000100101011101000110100101111000000100001111100100100011110	57	57
PB4: 000010000010001110101100011000010010110101101100011010110101	123	123

TABLE 16: XORing B with PB1, PB2, PB3, and PB4.

B	011000010110001001100011011001000110010101100110011001101101000	Number of bits affected
B + PB1	0101001011010010110011101101001011101000100100100101110010000010	36
B + PB2	111010001000100000111001010111001111000011100001000111101100000	28
B + PB3	110000100101011101000110100101111000000100001111100100100011110	36
B + PB4	000010000010001110101100011000010010110101101101100011010110101	26

TABLE 17: Impact of change in 5th bit on plain text: abcdefgh and its corresponding permuted text.

a	b	c	d	e	f	g	h
01100001	01100010	01100011	01100100	01100101	01100110	01100111	01101000
01010010	11010010	11001110	11010010	11101000	10010010	01011100	10000010
52	D2	CE	D2	E8	92	5C	82
i	b	c	d	e	f	g	h
01101001	01100010	01100011	01100100	01100101	01100110	01100111	01101000
01010010	11010010	11001111	11010010	11101000	10010010	01011100	10000010
52	D2	CF	D2	E8	92	5C	82

the algorithm is applied on two different plaintexts that differ in a single bit with same keys K_1 and K_2 , the cipher texts will also differ in a single bit. For example, for the two plain texts $P1$: abcdefgh and $P2$: ibcdefgh, the algorithm with $K_1 = 123$ and $K_2 = 57$ results in two different ciphertexts $C1$: 52d2ced2e8925c82 and $C2$: 52d2cfd2e8925c82, respectively, as shown in Table 17. Table 18 shows all the texts

varying in single bit and its corresponding permuted texts that also differ in a single bit.

Given two cipher texts of two different plaintexts varying in single bit or more make the cryptanalyst hard to deduce the keys. Since the key space is huge for an adversary to choose keys from, it makes the cryptanalysis somewhat difficult. We have found that it is possible to have the same permutation

TABLE 18: Texts with single bit change in the plain text: abcdefgh and corresponding permuted texts.

Bit position	Text (after a single bit change)	Permuted text	Bit position	Text (after a single bit change)	Permuted text
1	ábcdefgh	52d2ced2e8925c83	33	abcdáfgh	52f2ced2e8925c82
2	!bcdefgh	52d2ced2e8925c02	34	abcd%fgh	12d2ced2e8925c82
3	abcdefgh	52d0ced2e8925c82	35	abcdefgh	52d2ced2e0925c82
4	qbcdefgh	52d2ced6e8925c82	36	abcdufgh	52d2ced2e89a5c82
5	ibcdefgh	52d2cfd2e8925c82	37	abcdmfgh	52d2ced2e8927c82
6	ebcdefgh	52d2ced2e8925c92	38	abcdafgh	5292ced2e8925c82
7	cbcdefgh	52d2ded2e8925c82	39	abcdgfgh	52d2ced2e8925ca2
8	ˆbcdefgh	52d2ced2e8925482	40	abcddfgh	52d2cad2e8925c82
9	âcdefgh	52d2cef2e8925c82	41	abcdeægh	52d2ced2e8935c82
10	äcdefgh	52d2ced2e8924c82	42	abcde&gh	5252ced2e8925c82
11	abcdefgh	52d2ced2e8825c82	43	abcdefgh	52d2ced0e8925c82
12	arcdefgh	52d6ced2e8925c82	44	abcdevgh	52d2ced2e8925c8a
13	ajcdefgh	72d2ced2e8925c82	45	abcdengh	53d2ced2e8925c82
14	afcdefgh	52d2ced2e8925cc2	46	abcdebgh	52d24ed2e8925c82
15	äcdefgh	42d2ced2e8925c82	47	abcdedgh	50d2ced2e8925c82
16	accdefgh	52d2ced2e8d25c82	48	abcdeggh	5ad2ced2e8925c82
17	abâdefgh	52d2ced3e8925c82	49	abcdefçh	52d2ced2e8b25c82
18	ab#defgh	52d2ced2e8921c82	50	abcdef’h	52d2ce52e8925c82
19	abcdefgh	52d2ced2e8905c82	51	abcdefgh	52d2cec2e8925c82
20	absdefgh	52daced2e8925c82	52	abcdefwh	52d2ced2e8925c86
21	abkdefgh	52d2eed2e8925c82	53	abcdefoh	52d2ced2e8925d82
22	abgdefgh	d2d2ced2e8925c82	54	abcdefch	52d2ced268925c82
23	abadefgh	52d2ccd2e8925c82	55	abcdefeh	52d28ed2e8925c82
24	abbdefgh	52d2ced2e8925882	56	abcdeffh	52d2c6d2e8925c82
25	abcäefgh	52d2ced2e8925e82	57	abcdefgè	52d3ced2e8925c82
26	abc\$efgh	52d2ced2a8925c82	58	abcdefg(	52d2ced2e8125c82
27	abcdefgh	52c2ced2e8925c82	59	abcdefgh	52d2ced2e8925c80
28	abctefgh	52d2cedae8925c82	60	abcdefgx	52d2ced2e8965c82
29	abclefgh	52d2ced2e9925c82	61	abcdefg’	52d2ced2c8925c82
30	abc’efgh	52d2ce92e8925c82	62	abcdefgl	52d2ced2e892dc82
31	abcfefgh	52d2ced2f8925c82	63	abcdefgj	52d2ced2ea925c82
32	abceefgh	52d2ced2ec925c82	64	abcdefgi	56d2ced2e8925c82

TABLE 19: Single bit change in key K_1 : 123 without change in K_2 : 57 and corresponding permuted texts.

K_1	K_2	Hex text after permutation
251	57	52d2ced2e0d24d82
59	57	52d2ced2e8924c92
91	57	50928e92e97ae4e0
107	57	c3921f6d6092cca2
115	57	5614dad244d24bd2
127	57	1ac252c658d2d2d5
121	57	c25746978087c31e
122	57	008fc31f0e1f851d

sequence for different values of keys if the number of bits to be permuted is less. For example, the key pairs (251, 53) and (59, 53) applied to the algorithm with size = 8 will generate the same permutation sequence 2, 6, 1, 3, 4, 8, 7, 5 which is not desirable. So, the algorithm with less size value can be used to generate one time key, which is shorter, for authentication purpose. If the size is larger than at least

64 bits, the chances for generating the same permutation sequence are remote and therefore it requires tremendous efforts.

The algorithm can be iterated with different key pairs. Each iteration results in a permutation sequence. A permuted text obtained in the first iteration can be made secure by applying it to a permutation sequence generated in the successive rounds with different key pairs. Thus, the resultant permuted text is so secure that it increases the time required to decipher it. For example, the ciphertext CI : 52d2ced2e8925c82 formulated with permutation produced with keys $K_1 = 123$ and $K_2 = 53$ by the algorithm for the plaintext PI : abcdefgh can be made secure by subjecting it to any number of rounds with different values of K_1 and K_2 . Suppose the algorithm generates the following permutation sequence with $K_1 = 91$ and $K_2 = 131$:

56 52 47 17 18 50 64 63 12 40 3 29 54 14 37 59
16 36 55 9 26 42 57 7 20 32 43 53 62 6 13 19
24 28 31 33 34 60 58 51 49 46 45 44 41 39 38 35
30 27 25 23 22 21 15 11 10 48 8 5 4 2 1 61
(14)

TABLE 20: Single bit change in key K_1 : 123 without change in K_2 : 57.

Binary	Decimal	Permutation sequences															
11111011	251	22	34	13	15	48	64	47	45	42	38	33	27	20	12	3	57
		46	55	21	7	56	40	23	5	50	30	9	51	28	4	43	17
		54	26	61	31	53	32	63	29	58	35	49	11	36	60	19	41
		62	18	37	16	10	24	25	8	2	14	39	6	44	52	59	1
00111011	59	22	34	13	15	48	64	47	45	42	38	33	27	20	12	3	57
		46	55	21	7	56	40	23	5	50	30	9	51	28	4	43	17
		54	26	61	31	35	32	63	29	58	25	49	11	36	60	19	41
		62	18	37	16	10	24	39	53	8	14	6	2	44	52	59	1
01011011	91	22	2	13	47	48	16	64	60	42	6	33	59	20	44	3	25
		46	57	21	39	56	8	23	37	50	62	9	19	28	36	43	49
		54	58	61	63	55	53	52	51	45	40	38	35	34	32	30	29
		27	26	24	41	17	15	14	12	11	18	10	7	5	4	1	31
01101011	107	54	18	13	31	16	32	47	61	10	22	33	43	52	60	3	9
		14	64	21	23	24	58	56	55	53	51	50	48	46	42	41	40
		39	38	35	29	28	20	63	45	26	6	49	27	4	44	19	57
		30	2	37	17	15	8	25	5	34	62	11	7	12	36	59	1
01110011	115	6	58	45	55	32	40	47	53	64	62	1	3	4	56	39	36
		35	34	31	30	26	22	23	13	2	54	41	27	12	60	43	25
		21	50	29	7	48	24	63	37	10	46	17	51	20	52	19	49
		14	42	5	33	8	16	15	61	18	38	57	11	28	44	59	9
01111111	127	62	22	29	27	56	52	47	41	34	26	17	7	60	48	35	21
		6	54	37	19	64	44	23	1	42	18	57	31	4	40	11	45
		14	46	13	43	8	36	63	25	50	10	33	55	12	32	51	5
		59	38	53	61	39	28	30	49	58	2	20	15	16	24	9	3
01111001	121	34	40	5	41	12	14	15	64	49	47	9	46	44	58	51	43
		39	24	13	1	52	38	23	7	54	36	17	61	37	18	59	35
		10	48	21	57	28	62	31	63	30	60	25	53	16	42	3	27
		50	8	29	33	32	22	26	55	6	20	4	19	56	2	11	45
01111010	122	28	37	9	60	62	63	64	52	40	57	53	48	42	35	27	18
		8	61	49	36	22	7	55	38	20	1	45	24	2	43	19	58
		32	5	41	12	46	15	47	14	44	39	31	30	26	51	11	34
		56	13	33	29	21	23	25	54	4	17	16	10	50	59	6	3

TABLE 21: Single bit change in key K_2 : 57 without change in K_1 : 123 and corresponding permuted texts.

K_1	K_2	Hex text after permutation
123	185	52d2ced2e0d254a2
123	121	52d2ced2e0d25892
123	25	d2920e924592dce3
123	41	60929c92d2d2dc93
123	49	c89206d25e9267e2
123	61	d2da92e492889257
123	59	0823ace12d2d9d65
123	56	191c3f127e803f18

When C1 is applied to the above permutation sequence, it results in a secure ciphertext C2: ac692c218b43d3a5. The number of bits that differ in C2 as compared to C1 and P1 is 40 and 34, respectively. This leads to a good impact.

One of the known weaknesses of the algorithm is that if the same letter or bit (0/1) makes up a plaintext, the permutation sequence generated by the algorithm will not be useful because even after its application on the plaintext the permuted plaintext is the same as the original plaintext.

Suppose when the algorithm is used on plaintext of n characters/bits, the brute force attack requires $n!$ attempts to break the cipher text (if the algorithm is used for encryption). The more the size of bits, the greater the time required for a brute force attack by an opponent.

We have also applied $K_1 = 123$ and $K_2 = 57$ values varying in single bit to our algorithm. By applying a set of K_1 values (251, 59, 91, 107, 115, 127, 121, 122), all varying in single bit with no change in $K_2 = 57$ to the algorithm, it creates different permutation sequences that can be used for producing different permuted texts (see Tables 19 and 20). All the permuted texts are shown in hexadecimal representation.

TABLE 22: Single bit change in key K_2 : 57 without change in K_1 : 123.

Binary	Decimal	Permutation sequences															
10111001	185	22	34	13	15	48	64	47	45	42	38	33	27	20	12	3	57
		46	35	21	7	56	40	23	5	50	30	9	51	28	4	43	17
		54	26	61	31	25	32	63	29	58	24	49	11	36	60	19	41
		62	18	37	55	16	10	39	53	2	14	8	6	44	52	59	1
01111001	121	22	34	13	15	48	64	47	45	42	38	33	27	20	12	3	57
		46	55	21	7	56	40	23	5	50	30	9	51	28	4	43	17
		54	26	61	31	37	32	63	29	58	35	49	11	36	60	19	41
		62	18	25	24	8	16	39	53	10	14	6	2	44	52	59	1
00011001	25	54	2	45	47	16	64	15	13	10	6	1	59	52	44	35	25
		14	57	53	39	24	8	55	37	18	62	41	19	60	36	11	49
		22	58	29	63	32	56	31	61	26	48	17	43	4	28	51	9
		30	50	5	23	42	40	7	21	38	46	34	33	12	20	27	3
00101001	41	6	18	61	63	32	64	31	29	26	22	17	11	4	60	51	41
		30	62	5	55	40	24	7	53	34	14	57	35	12	52	27	1
		38	10	45	15	48	16	47	13	42	58	33	59	20	44	3	25
		46	2	21	54	56	8	49	39	50	37	9	23	28	36	43	19
00110001	49	46	58	37	39	8	64	7	5	2	62	57	51	44	36	27	17
		6	63	45	31	16	59	47	29	10	54	33	11	52	28	3	41
		14	50	21	55	24	56	23	53	18	48	9	35	60	20	43	1
		22	42	61	49	32	40	34	30	38	26	15	25	4	12	19	13
00111101	61	10	54	1	3	36	64	35	33	30	26	21	15	8	62	55	45
		34	22	9	59	44	28	11	57	38	18	61	39	16	56	31	5
		42	14	49	19	52	20	51	17	46	60	37	63	24	48	7	29
		50	6	25	58	4	12	27	41	53	2	13	43	32	40	47	23
00111011	59	16	44	7	9	42	64	41	39	36	32	27	21	14	6	61	51
		40	28	15	1	50	34	17	63	59	24	3	45	22	62	37	11
		48	20	55	25	58	26	57	23	52	49	43	5	30	54	13	35
		56	12	31	47	38	18	33	10	60	8	19	29	4	46	53	2
00111000	56	25	29	48	50	51	64	62	58	45	41	36	30	23	15	6	60
		49	37	24	10	59	43	26	8	53	33	12	54	31	7	46	20
		57	55	47	34	3	35	2	32	61	28	52	14	39	63	22	44
		1	21	40	19	18	27	42	56	16	17	13	38	11	9	5	4

TABLE 23: Permuted texts for K_1 : 123 and K_2 : 57 with a single bit change (in keys).

K_1	K_2	Hex text after permutation
251	185	52d28ed2e8d21d82
59	121	52d2ced2e0d258a2
91	25	d0d24ed2476c0956
107	41	f09248cdaad24ce2
115	49	d8137a92cb92d412
127	61	c6d092bad692d218
121	59	e8c8395c7870aa70
122	56	0dc81f860f0f1f88

It can be noted that it produces a good avalanche effect. Similarly, we also tried different K_2 values (185, 121, 25, 41, 49, 61, 59, 56) that differ in single bit but with $K_1 = 123$ (refer to Table 22) to generate the permuted texts as shown in Table 21.

Table 23 shows permuted texts after having applied permutation sequences on $P1$ constructed by the algorithm for various values of K_1 (251, 59, 91, 107, 115, 127, 121, 122) which we obtained by changing one bit of K_1 . Similarly for K_2 (see Table 24).

The algorithm has also produced a good effect even after having swapped K_1 and K_2 values applied to it (see Table 25).

The power of the algorithm lies in constructing permutation/substitution tables of any size using key information derived from plain text/key blocks. Instead of predefined P -Boxes and S -Boxes used with some symmetric block ciphers to achieve an avalanche effect that makes the relationship between plaintext and ciphertext complex at each round during encryption process, the same effect can be realized if we generate these tables dynamically based on the information available during rounds of encryption at runtime. Making a single bit change in any of the key values K_1 , K_2 , or both has resulted in a permutation sequence that can be applied

TABLE 24: Permutation sequences for K_1 : 123 and K_2 : 57 with a single bit change (in keys).

K_1	K_2	Permutation sequences															
251	185	22	34	13	15	48	64	47	45	42	38	33	27	20	12	3	57
		46	53	21	7	56	40	23	5	50	30	9	51	28	4	43	17
		54	26	61	31	35	32	63	29	58	18	49	11	36	60	19	41
		62	16	37	55	10	24	25	8	2	14	39	6	44	52	59	1
59	121	22	34	13	15	48	64	47	45	42	38	33	27	20	12	3	57
		46	55	21	7	56	40	23	5	50	30	9	51	28	4	43	17
		54	26	61	31	37	32	63	29	58	35	49	11	36	60	19	41
		62	18	25	24	8	16	39	53	2	14	10	6	44	52	59	1
91	25	54	34	45	15	16	48	64	62	10	38	1	27	52	12	35	57
		14	61	53	7	24	40	55	5	18	30	41	51	60	4	11	17
		22	26	29	31	32	59	58	56	49	47	46	44	43	42	39	37
		36	33	28	25	23	21	20	19	13	50	9	8	6	3	2	63
107	41	38	2	61	15	64	16	31	45	58	6	17	27	36	44	51	57
		62	59	5	7	8	53	49	48	42	40	39	37	35	34	32	30
		26	25	23	22	19	13	47	29	10	54	33	11	52	28	3	41
		14	50	21	9	24	56	12	4	46	18	55	1	60	20	43	63
115	49	30	18	5	15	56	64	7	13	63	22	25	27	28	60	59	58
		57	55	54	50	46	45	47	37	26	14	1	51	36	20	3	49
		42	10	53	31	8	48	23	61	34	6	41	11	44	12	43	9
		38	2	29	40	16	24	39	33	32	62	21	35	52	4	19	17
127	61	2	58	33	31	60	56	51	45	38	30	21	11	64	52	39	25
		10	63	41	23	4	48	27	5	46	22	61	35	8	44	15	49
		18	50	17	47	12	40	3	29	54	14	37	59	16	36	55	9
		26	42	57	34	20	32	43	53	62	6	13	19	24	28	7	1
121	59	24	46	59	31	2	4	5	64	58	56	63	45	54	48	41	33
		39	14	3	55	42	28	13	61	44	26	7	51	30	8	49	25
		37	38	11	47	18	52	21	53	20	50	15	43	6	32	57	17
		40	62	19	36	34	12	35	29	60	10	23	27	22	16	1	9
122	56	32	33	45	64	2	3	63	35	34	61	57	52	46	39	31	22
		12	1	53	40	26	11	59	42	24	5	49	28	6	47	23	62
		36	9	29	16	50	19	51	18	48	13	41	4	30	55	15	38
		60	17	37	8	54	27	43	58	56	21	25	44	10	20	7	14

TABLE 25: Permuted texts for K_1 : 57 and K_2 : 123 with a single bit change (in keys) after swapping.

K_1	K_2	Hex text after permutation
185	251	e858395c7870c670
121	59	e8c8395c7870aa70
25	91	e8503954788b5372
41	107	a8707a90f650ea54
49	115	6871ab8cb8504b2d
61	127	307c69f8087cb039
59	121	52d2ced2e0d258a2
56	122	a554f0aca184e51d

to intermediate ciphertexts to achieve good confusion and diffusion. The results of the algorithm are satisfactory that it can be used with symmetric crypto systems like DES for generating substitution/permutation tables dynamically.

4. Conclusion

The two-key dependent permutation algorithm generates a permutation table based on two keys. The good choice of

key values enables algorithm to construct a permutation table that will disturb positions of all characters/bits making up a given message. The results show that the 2KDP algorithm can also be used with Feistel based symmetric block ciphers. We have also shown how a single bit change in any of the two keys or both will produce a permutation table that brings in drastic impact on complicating the relationship between plaintext and ciphertext after its application on plain text.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

- [1] National Bureau of Standards, "Data encryption standard," Tech. Rep. 46, U.S. Department of Commerce, FIPS Publication, 1977.
- [2] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," in *Proceedings of the Fast Software Encryption, Cambridge Security Workshop*, pp. 191–204, Springer, 1993.

- [3] R. C. Merkle, "Fast software encryption functions," in *Advances in Cryptology—CRYPTO '90 Proceedings*, vol. 537, pp. 476–501, Springer, 1991.
- [4] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, *The Twofish Encryption Algorithm: A 128-Bit Block Cipher*, John Wiley & Sons, New York, NY, USA, 1999.
- [5] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
- [6] W. Stallings, *Cryptography and Network Security*, Prentice Hall, New York, NY, USA, 2011.
- [7] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer, New York, NY, USA, 1994.
- [8] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.
- [9] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer, 1993.
- [10] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT '93)*, pp. 386–397, Springer, 1993.
- [11] C. M. Adams and S. E. Tavares, "Designing S-boxes for ciphers resistant to differential cryptanalysis," in *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*, pp. 181–190, Rome, Italy, February 1993.
- [12] GOST 28147-89, Cryptographic Protection for Data Processing Systems, Cryptographic Transformation Algorithm, Government Standard of the U.S.S.R., Inv. No. 3583, UDC 681.325.6:006.354. (Russian).
- [13] N. K. Pareek, V. Patidar, and K. K. Sud, "Substitution-diffusion based image cipher," *International Journal of Network Security & Its Applications*, vol. 3, no. 2, 2011.
- [14] R. Zhang and L. Chen, "A block cipher using key-dependent S-box and P-boxes," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '08)*, pp. 1463–1468, Cambridge, UK, July 2008.
- [15] H. M. El-Sheikh, O. A. El-Mohsen, T. Elgarf, and A. Zekry, "A new approach for designing key-dependent S-box defined over GF (2^4) in AES," *International Journal of Computer Theory and Engineering*, vol. 4, no. 2, pp. 158–164, 2012.
- [16] Abd-ElGhafar, A. Rohiem, A. Diaa, and F. Mohammed, "Generation of AES key dependent S-boxes using RC4 algorithm," in *Proceedings of the 13th International Conference on Aerospace Sciences & Aviation Technology (ASAT '13)*, May 2009.
- [17] E. M. Mahmoud, A. Abd El Hafez, T. A. Elgarf, and A. Zekry, "Dynamic AES-128 with key-dependent S-box," *International Journal of Engineering Research and Applications*, vol. 3, no. 1, pp. 1662–1670, 2013.
- [18] D. Lambić and M. Živković, "Comparison of random S-box generation methods," *Publications de l'Institut Mathématique*, vol. 93, no. 107, pp. 109–115, 2013.
- [19] K. Kazlauskas and J. Kazlauskas, "Key-dependent S-box generation in AES block cipher system," *Informatika*, vol. 20, no. 1, pp. 23–34, 2009.
- [20] J. Peng and S. Jin, "Designing key-dependent S-boxes using hyperchaotic chen system," in *Proceedings of the International Conference on Information Engineering and Applications (IEA '13)*, vol. 216 of *Lecture Notes in Electrical Engineering*, pp. 733–740, 2013.