Using Cipher Key to Generate Dynamic S-Box in AES Cipher System

Razi Hosseinkhani *r.hosseinkhani@nioc.ir*
Computer Engineering Department Science and Research Branch
Islamic Azad University
Tehran, Iran

H. Haj Seyyed Javadi *h.s.javadi@shahed.ac.ir*
Department of Mathematics and Computer Science
Shahed University
Tehran, Iran

Abstract

The Advanced Encryption Standard (AES) is using in a large scale of applications that need to protect their data and information. The S-Box component that used in AES is fixed, and not changeable. If we can generate this S-Box dynamically, we increase the cryptographic strength of AES cipher system. In this paper we intend to introduce new algorithm that generate S-Box dynamically from cipher key. We describe how S-Box can be generated dynamically from cipher key and finally analyze the results and experiments.

Keywords: Advanced Encryption Standard, AES, Dynamic S-Box Generation, S-Box

1. INTRODUCTION

Encryption has an important role in data protection. The importance of encryption realized with increasing communication. Encryption makes sense when data packets using open channels, which they can be reached by other devices or people, to transfer their contents.

Encryption is knowledge of changing data with cipher key by using cipher algorithms, so that someone who knows the cipher key and cipher algorithm can export the plain text from cipher text. The meaning of Encryption is not only hiding information, but also it means sending information with another form, so that ensure security of data.

An Encryption system contains set of transformations that convert plain text into cipher text. In the block cipher system, plain text converts into blocks that cipher algorithm applies on them to create cipher text.

The block cipher systems divided into two general principles: Diffusion and Confusion. In Diffusion principle, each bit of plain text converts into many bits. However, in Confusion principle, number of bits doesn't change and only transformations apply to plain text, hence in Confusion principle, size of plain text and cipher text is equal. Usually in both principles, using round repetition to create cipher text. Repeating a single round contributes to cipher's simplicity [1].

Cipher algorithms have the two general categories: Private Key algorithms and public key algorithms. Private Key algorithms using single key to encrypt plain text and decrypt cipher text in sender and receiver side. Private Key algorithm samples are: DES, 3DES and Advanced Encryption Standard (AES). Public Key algorithms, such as the Rivest-Shamir-Adleman (RSA), using two different key for encrypt plain text and decrypt cipher text in sender and receiver sides.
Block cipher systems depend on the S-Boxes, which are fixed and no relation with a cipher key. So only changeable parameter is cipher key. Since the only nonlinear component of AES is S-Boxes, they are an important source of cryptographic strength. So we intend use cipher key to

generate dynamic S-Box that is changed with every changing of cipher key. That cause increasing the cryptographic strength of AES algorithm. Other systems using key-dependent S-Boxes have been proposed in the past, the most well-known is Blowfish and Khufu [2], [3]. Each of these two systems uses the cryptosystem itself to generate the S-Boxes.

In section 2, we briefly introduce the AES algorithm. In section 3, we study about the S-Box that used in AES. In section 4, we show that how S-Box will be generated from key and in the final section we analyze experiments and investigate about results.

2. ADVANCED ENCRYPTION STANDARD (AES)

This standard specifies the Rijndael algorithm, asymmetric clock cipher can process data blocks of 128 bits, using cipher key with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key length, however they are not adopted in this standard [4].

2.1 Definitions

- **Cipher**: Series of transformations that converts plaintext to cipher text using the Cipher Key.
- **Cipher Key**: Secret, cryptographic key that is used by the Key Expansion routine to generate a set o f Round Keys; can be pictured as a rectangular array of bytes, having four rows and Nk columns.
- **Ciphertext**: Data output from the Cipher or input to the Inverse cipher.
- **Plaintext**: Data input to the Cipher or output from the Inverse Cipher.
- **S-Box**: Non-linear substitution table used in several byte substitution transformations and in the Key Expansion routine to perform a one-for-one substitution of a byte value.

2.2 Algorithm Parameters, Symbols, and Functions

The following algorithm parameters, symbols, and functions are used throughout this standard:

- **AddRoundKey():** Transformation in the Cipher and Inverse Cipher in which a Round Key is added to the State using an XOR operation. The length of a Round Key equals the size of the State.
- **MixCloumns():** Transformation in the Cipher that takes all of the columns of the State and mixes their data (independently of one other) to produce new columns.
- **Nb:** Number of columns (32-bit words) comprising the State.
- **Nk:** Number of 32-bit words comprising the Cipher Key.
- **Nr:** Number of rounds, which is a function of Nk and Nb (which is fixed).
- **RotWord():** Function used in a Key Expansion routine that takes a four-byte word and performs a cyclic permutation.
- **ShiftRows ():** Function is the Cipher that processes the state by cyclically shifting the last three rows of the State by different offsets.
- **SubBytes()**: Transformation in the Cipher that processes the State using a non-linear byte substitution table (S-Box) that operates on each of State bytes independently.
- **SubWord():** Function used in the Key Expansion routine that takes a four-byte input word and applies an S-Box to each of the four bytes to produce an output word.

2.3 Cipher Algorithm

From the beginning of the Cipher, the input is copied to State array. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times(depending of key length), with the final round differing slightly from the first Nr-1 rounds. The final State is then copied to the output. The Cipher is described in the pseudo code in algorithm 1.

```
public word[] Cipher(byte[] plainText, byte[] cipherKey)
{
        state = new word[4];
        sBox = new newSbox(cipherKey);
        ks = new KeySchedule(cipherKey);
```

```
        for (int i = 0; i < 4; i++)
        {
                for (int j = 0; j < 4; j++)
                {
                        if (state[j] == null)
                        state[j] = new word();
                        state[j].w[i] = plainText[i * 4 + j];
                }
        }
        AddRoundKey(0);
        for (int i = 1; i < Nr; i++)
        {
                SubBytes();
                ShiftRows();
                MixColumn();
                AddRoundKey(i);
        }
        SubBytes();
        ShiftRows();
        AddRoundKey(Nr);
        return state;
}
```

ALGORITHM 1: Pseudo Code for Cipher

3. THE SUBSTITUTION BOX (S-BOX)

Substitution is a nonlinear transformation which performs confusion of bits. A nonlinear transformation is essential for every modern encryption algorithm and is proved to be a strong cryptographic primitive against linear and differencial cryptanalysis. Nonlinear transformations are implemented as lookup tables (S-Boxes). An S-Box with p input bits and q output bits is denoted p * q. The DES uses eight 6 * 4 S_boxes. S-Boxes are designed for software implementation on 8-bit processors. The block ciphers with 8 * 8 S-Boxes are SAFER, SHARK, and AES.

For processors with 32-bit or 64-bit words, S-Boxes with more output bits provide high efficiency. The Snefru, Blowfish, CAST, and SQUARE use 8 * 32 S-Boxes. The S-Boxes can be selected at random as in Snefru, can be computed using a chaotic map, or have some mathematical structure over a finite Galois field. Examples of the last approach are SAFER, SHARK, and AES. S-Boxes that depend on key values are slower but more secure than key independent ones (Schneier,1996). Use of key independent chaotic S-Boxes are analyzed in which the S-Box is constructed with a transformation F((X + K)modM),where K is the key [5].

4. DYNAMIC S-BOX GENERATION FROM CIPHER KEY ALGORITHM

4.1 First Step

We need primary S-Box to generate dynamic S-Box , that should has 16 rows and columns. We use S-Box generation algorithm that introduced in AES, to create primary S-Box as follows[4].
Take the mulltiplicative inverse in the finite feild GF(28); the element {00} is mapped itself.
Apply the following affine transformation (over GF(2)) that represent in following equation.

$$b_i' = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

EQUATION 1: S-Box transformation

For $0 \leq i < 8$,where b_i is the i^{th} bit of the byte, and c_i is the i^{th} of a bytec with the value {63} or (01100011}. Here and elsewhere, a prime on a variable (e.g., b^i) indicate that the variable is to be uodated with the value on the right. In matrix form, the affine transformation element of the S-Box can be expressed as:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

FIGURE 1: Step 2 in S-Box generation in AES

4.2 Second Step

In this step, rows swapped with columns of primary S-Box in ***GenerateDynamicSbox(cipherKey)*** function. This function guarantees new S-Box remain one-for-one. This routin get cipher key as input and generate dynamic S-Box from cipher key. Note that in this paper if cipher key has 192 or 256 bits size, we use only first 128 bits of cipher key.

4.2.1 GenerateDynamicSbox Algorithm

```
1: void GenerateDynamicSbox(byte[16] key)
2: {
3:      byte rowIndex, columnIndex;
4:      byte shiftCount = GetShiftCount(key);

5:      byte[,] sBox = GeneratePrimarySbox();

6:      for(int i = 0; i < 16;i++)
7:      {
8:          GetProperIndex(key[i], out rowIndex, out columnIndex);
9:          ShiftRow(rowIndex, shiftCount, sBox);
10:         ShiftColumn(columnIndex, shiftCount, sBox);
11:         Swap(rowIndex, columnIndex, sBox);
12:     }
13: }
```

ALGORITHM 2: The ***GenerateDynamicSbox*** function generate dynamic S-Box form cipher key.

In line 4, ***GetShiftCount(cipherKey)*** get cipherKey as input and return number of shift that should be applied to rows and columns before replacing with each other.
In line 5, ***GeneratePrimarySbox ()*** generate primary S-Box according 4.1.
In line 6, start loop for 16 times (foreach byte of cipher key, only first 16 byte of cipher key is used).

In line 8, ***GetProperIndex(cipherKey[i], out rowIndex, out columnIndex)*** get byte of cipher key and return indexes of row and column that should be replaced with each other.

In line 9, ***ShiftRow(rowIndex, shiftCount, sBox)*** get row index of S-Box and shift each element of given row cyclically. It means if rowIndex = 0 and shiftCount = 1, first element of S-Box,

sBox[0,1] replace with sBox[0,0] and sBox[0,2] replace with sBox[0,1] ... and sBox[0,0] replace with sBox[0,15]. (The first index of sBox determine rowIndex and second one determine columnIndex).

In line 10, ***ShiftColumn(columnIndex, shiftCount, sBox)*** get column index of S-Box and shift each element of given column cyclically. It means if columnIndex = 0 and shiftCount = 1, first element of S-Box, sBox[1,0] replace with sBox[0,0] and sBox[2,0] replace with sBox[1,0] ... and sBox[0,0] replace with sBox[15,0].

In line 11, ***Swap(rowIndex, columnIndex, sBox)*** get row and column index and then swapped them with each other. For example if rowIndex = 5 and columnIndex = 4 the Swap function swapping element at sBox[0,5] with sBox[4,0] and sBox[1,5] with sBox[4,1] and ... and finaly sBox[15,5] swap with sBox[4,15].

4.2.2 GetShiftCount Algorithm

This function get cipher key as input and return number of shift count as output. If cipher key larger than 128 bit, only first 128 should be used.

```
1: byte GetShiftCount(byte[16] cipherKey)
2: {
3:      byte customizingFactor = 0x00;
4:      byte shiftCount = 0;

5:      for(int i = 0 ; i < 16 ;i++)
6:      {
7:              shiftCount ^= (byte)((key[i] * (i + 1)) % (0xFF + 1));
8:      }

9:      return shiftCount ^ customizingFactor;
10:}
```

ALGORITHM 3: The ***GetShiftCount*** function used to get shift count before swapping rows with columns.

In line 4, ***customizingFactor*** value is in [0-255] range. This variable can customize the ***GetShiftCount*** return value and then customize ***GenerateDynamicSbox***.
In line 5, start loop for 16 times (foreach byte of cipher key, only first 16 byte of cipher key is used).

In line 6, sign ^ means XOR operation and sign % means modulo in C#. This equation guarantees that changing only one bit of Cipher key cause changing the value of ***shiftCount***.
In line 9, ***shiftCount*** XOR with ***customizingFactor*** that cause generate 256 different customizing states for ***shiftCount*** value.

4.2.3 GetProperIndex Algorithm

This function gets byte of cipher key and then return rowIndex and columnIndex as output. This function using Shuffle exchange algorithm that used in designing parallel algorithms [6].

```
1:void GetProperIndex (byte key, out byte rowIndex, out byte columnIndex)
2:{
3:      int[] rowUsedArray, columnUsedArray;

4:      rowIndex = key & 0x0F;
5:      columnIndex = key >> 4;

6:      rowIndex = Shuffle (rowUsedArray, rowIndex);
7:      columnIndex = Shuffle (columnUsedArray, columnIndex);
```

```
8:      rowUsedArray.Add(rowIndex);
9:      columnUsedArray.Add(columnIndex);

10:}
```

ALGORITHM 4: The ***GetProperIndex*** function pseudo code

In line 3, ***rowUsedArray*** and ***columnUsedArray*** variables are using for saving index that used in previous steps.
In line 4, sign & means AND operation in C#.
In line 5, sign >> means shift right n-times in C#.
In line 6, ***Shuffle*** function get ***rowIndex*** number and return next available ***rowIndex*** number if given ***rowIndex*** is in ***rowUsedArray***.
In line 7, ***Shuffle*** function get ***columnIndex*** number and return next available ***columnIndex*** number if given ***columnIndex*** is in ***columnUsedArray***.
In line 8, current ***rowIndex*** add to ***rowUsedArray*** array.
In line 9, current ***columnIndex*** add to ***columnUsedArray*** array.
This causes that every row and column only one time returns with this function thus every row and column is used for one time in ***GenerateDynamicSbox***.

5. EXPERIMENTAL RESULTS

In general, S-Box is substitution table that get number and return another number. This action is nonlinear. In S-Box, *n* input bits are represented as one of 2^n different characters. The set of 2^n characters are then transposed to one of the others in the set. For example possible output 3 * 3 S-Boxes are shown in figure 2.

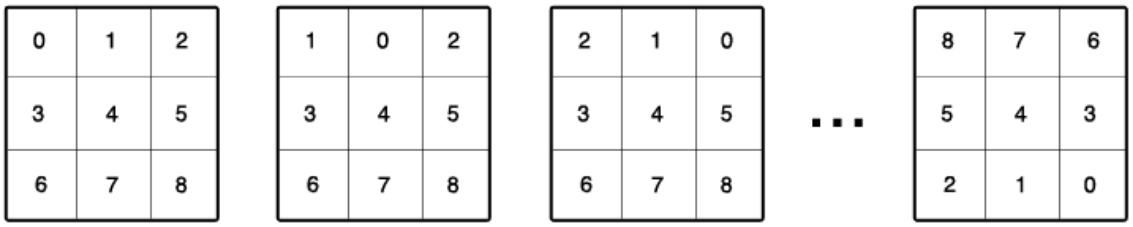

FIGURE 2: Possible 3 * 3 S-Boxes.

The character is then converted back to an n-bit output. It can be easily shown that there are $(2^n)!$ different substitution or connection patterns possible. Thus if *n* is large then the possible S-Boxes that can be generate is large. If the cryptanalyst want to decode AES algorithm he should try to generate possible S-Box and use them in *SubBytes* function in AES cipher system. The cryptanalyst's task becomes computionally unfeasible as *n* gets large, say n = 128; then $2^n = 10^{38}$, and $(10^{38})!$ possible S-Box can be generate which is an <u>*astronomical*</u> number.

$$(10^{38})! = \infty$$

EQUATION 2: Possible S-Box can be generate when n = 128

Experiment 1: We experimentally checked the difference measure of S-Box elements that only different between two bits depend on interval length. For generation random keys we use function that change only two bits of random byte of cipher key. The difference table illustrated in table 1.

Interval length	Avg. of difference	Std. of difference	Perentage of difference
8	229.5	20.4310687784211	89.6484375
16	225.0625	19.9280999261512	87.9150390625
32	227.9375	20.2244462422696	89.0380859375
64	223.96875	21.5037602839412	87.48779296875
128	225.4453125	21.9149424426299	88.0645751953125
256	228.4296875	20.8423338400948	89.2303466796875
512	227.072265625	21.7735331073363	88.7001037597656
1024	226.482421875	21.3572160638917	88.4696960449219
2048	227.00830078125	21.1134241069894	88.6751174926758
4096	228.115966796875	20.8932978239743	89.1077995300293

TABLE 1: The difference table between S-Box elements by changing 2 bits of cipher key's random byte.

.

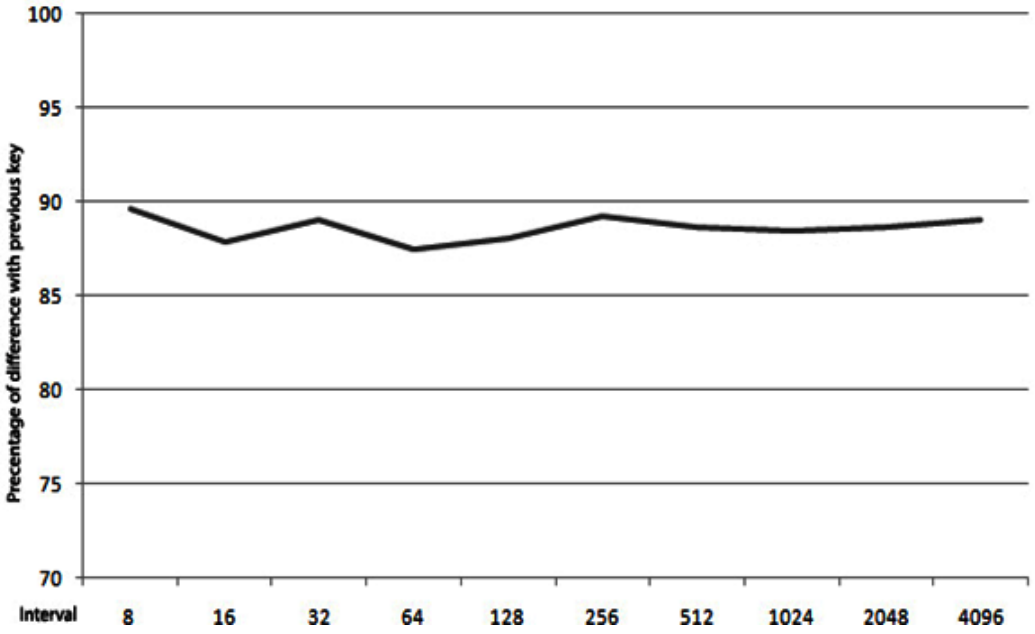

FIGURE 2: Plot of the difference of the S-Box elements with previous key

Experiment 2: The purpose is to verify GenerateDynamicSbox algorithm. Consider the 128 bit length secret key in the hexadecimal form. The key and S-Box2 is represent in table 2.

key_hex$_1$={5e,d3,f1,b4,7c,18,51,9a,ae,81,42,57,42,78,dc,8f}

67	1E	77	7B	4B	4F	6F	46	1F	F8	6A	87	B9	71	60	CD
AD	99	85	AF	48	D5	81	0E	17	82	9C	C9	35	13	47	0C
E5	7C	B8	A4	E1	55	E3	DF	02	01	31	70	B0	B7	BD	C1
80	D4	8F	E2	BF	54	B2	68	3B	A1	18	0F	9A	03	DC	EC
D6	E6	AC	B3	3F	7A	22	C5	5E	1D	EB	CA	D2	A6	5A	A0
D0	EF	AA	FB	43	16	AE	07	7D	F9	0A	33	93	26	E9	44
9B	4D	53	D1	00	04	20	5B	5D	51	76	CB	DB	6E	4A	6B
25	AB	D9	E7	73	BA	9D	DA	59	12	74	C0	C4	F7	11	63
94	C7	ED	6C	27	61	CE	39	30	FA	08	C3	36	EA	7F	E8
19	A2	A8	05	52	D7	9F	8C	4C	BE	9E	96	3E	FE	8E	A7
49	D3	8B	38	F1	1C	B6	41	1A	E4	FD	F5	89	D8	24	CF
5F	72	37	6D	92	95	4E	FC	83	0B	65	45	0D	A5	14	3D
E0	2D	C2	3A	B1	06	B4	C6	2E	32	10	62	8D	34	DE	BC
CC	C8	B5	66	2F	58	F6	98	79	56	86	F0	2A	2B	57	3C
88	78	84	90	91	97	EE	5C	BB	DD	69	15	42	40	28	64
75	A3	F2	23	2C	50	7E	1B	29	F4	8A	21	A9	FF	F3	09

TABLE 2: The dynamic S-Box generated with key_hex1. (S-Box1)

We change only two bit of key_hex1, for example, $5e \rightarrow 5d$. The key and S-Box2 is represent in Table 3.
We find 225 different between S-Box1 and S-Box2 by changing only 2 bit of key, thus approximately %87 of second S-Box is changed. The difference of S-Box1 and S-Box2 elements is illustrated in figure 3.

key_hex$_2$={5d,d3,f1,b4,7c,18,51,9a,ae,81,42,57,42,78,dc,8f}

4D	CA	D7	59	3F	50	DE	5B	BC	21	16	2D	3A	55	28	A9
82	7D	EB	FA	9E	FE	F0	51	E2	0C	72	68	A2	B9	99	73
E3	C5	B6	01	46	AA	5D	DA	42	F2	BE	B5	77	B7	52	F9
0D	9C	27	B2	B3	86	C7	FD	29	C0	6D	9A	7B	81	DC	EC
BF	AD	17	22	45	89	83	31	5E	35	2F	E6	64	A4	3B	69
D0	A1	7C	FB	43	65	AE	38	47	D9	24	87	67	26	02	E7
80	E1	BD	D4	00	7F	20	8F	61	23	11	05	6F	6E	4A	6B
2C	94	40	88	75	2E	D1	10	07	A0	F3	C9	E5	F7	19	CF
6C	04	66	25	62	30	54	39	C3	5F	09	ED	1F	EA	18	CB
48	A3	7A	12	92	FC	9F	F5	4C	53	76	8B	B8	06	AC	A7
A8	91	0B	1E	F1	C6	79	08	5A	E4	15	95	44	F8	D3	96
BA	70	37	0F	D8	8A	57	CC	1B	98	33	AF	D5	A5	85	8E
C2	B1	E8	DD	CE	41	32	71	4E	1A	E0	78	8C	A6	49	E9
13	C8	C1	1D	34	58	3E	6A	B4	56	2B	0E	9D	84	FF	3C
DB	7E	4F	EE	9B	97	4B	5C	BB	D6	14	74	60	0A	1C	D2
B0	DF	93	90	63	C4	EF	03	36	8D	2A	F4	CD	F6	AB	3D

TABLE 3: The dynamic S-Box generated with *key_hex2.(S-Box2)*

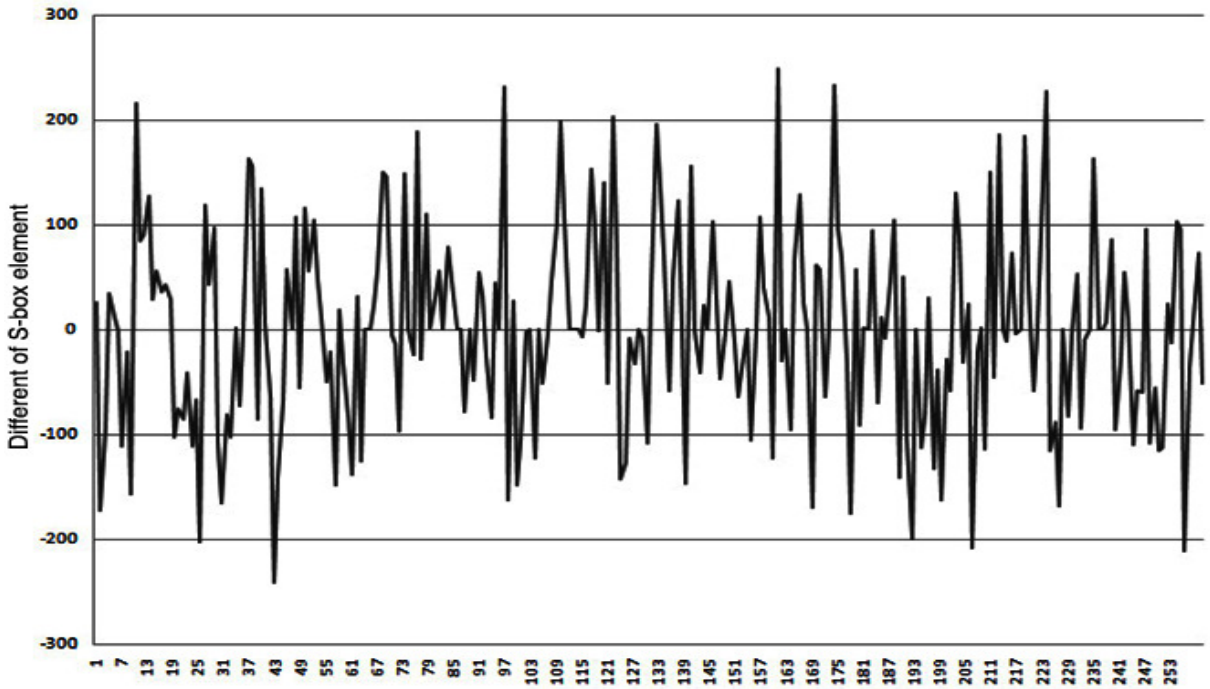

FIGURE 3: Plot of the difference of the S-Box elements (S-Box1 and S-Box2)

Experiment 3: We comparison *GenerateDynamicSBox* function with the older algorithm that was named *KeyDependantSBox* [5]. The *KeyDependantSBox* generates dynamic S-Box using random parameters by using iterative loop. The main property of S-Box is one to one attribute. *KeyDependantSBox* algorithm check every new generated S-Box element with the other elements that is generated sooner to avoid generate duplicate elements and satisfy one to one property of S-Box. The number of comparison in *KeyDependantSBox* presents in figure 4. In our *GenerateDynamicSBox* algorithm no need to check duplicate elements because original S-Box that is used in AES is one to one and we only substitute and move the original S-Box elements. Thus generated S-Box remains one to one.

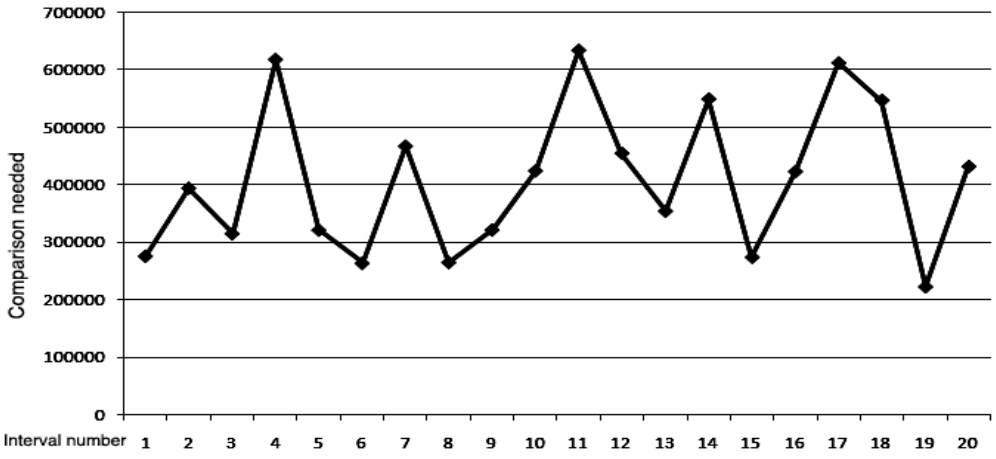

FIGURE 4: Number of comparison that needs to generate S-Box in KeyDependantSBox

6. CONCLUSIONS

We introduced a new algorithm to generate dynamic S-Box from cipher key. The quality of this algorithm tested by changing only two bits of cipher key to generate new S-Boxes. For that purpose we are testing difference of S-Box element by many intervals. This algorithm will lead to generate more secure block ciphers, solve the problem of the fixed structure S-Boxes and will increase the security level of the AES block cipher system. The main advantage of this algorithm is that many S-Boxes can be generated by changing Cipher key.

7. REFERENCES

[1] Masuda, N. Jakimovski, G. Jakimovski, K. Aihara and L. Kocarev . "Chaotic block ciphers: from theory to practical algorithms" IEEE Trans. on Circuits and Systems – I: Volume: 53 Issue: 6 – 2006

[2] B. Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C, New York: Wiley. 1996

[3] Merkle, R. Fast software encryption functions. In Advances in Cryptology: Proceedings of CRYPTO'90, Berlin: Springer-Verlag, 1991

[4] Federal Information Processing Standards, "Advanced Encryption Standard (AES)" Publication 197, November 26 - 2001

[5] Kazys KAZLAUSKAS, Janunius KAZLAUSKAS," Key-Dependent S-Box Generation in AES Block Cipher System" , Inoformatica Volume: 20 - 2009

[6] Michael J.Quinn, Designing efficient algorithms for parallel computers, University of New Hamoshire, 1987