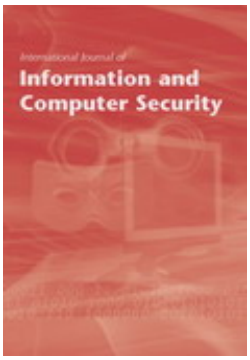


International Journal of Information and Computer Security

ISSN online: 1744-1773 - ISSN print: 1744-1765
https://www.inderscience.com/ijics

On generating new key dependent XOR tables to improve AES security and evaluating the randomness of the output of block ciphers

Tran Thi Luong, Hoang Dinh Linh

DOI: 10.1504/IJICS.2023.10055730

Article History:

Received:	12 October 2022
Last revised:	09 March 2023
Accepted:	20 March 2023
Published online:	19 February 2024

On generating new key dependent XOR tables to improve AES security and evaluating the randomness of the output of block ciphers

Tran Thi Luong*

Academy of Cryptography Techniques,
No. 141 Chien Thang Road, Tan Trieu, Thanh Tri, Hanoi, Vietnam
Email: luongtranhong@gmail.com
*Corresponding author

Hoang Dinh Linh

Institute of Cryptographic Science and Technology,
No. 105, Nguyen Chi Thanh Road, Lang Ha, Dong Da, Hanoi, Vietnam
Email: hoangdinhlinh@bcy.gov.vn

Abstract: Although block ciphers are widely used and are quite secure, there are still many types of attacks against components of block ciphers, and the Advanced Encryption Standard (AES) block cipher is no exception. To improve the security of AES, there have been many studies in the literature on methods of making this block cipher dynamic. There have been many works focused on the methods of making dynamic at the S-box and the MixColumn transformation of AES. In this paper, we propose a method to make dynamic at the Addroundkey transformation of AES using new key dependent XOR tables. We also propose a procedure to evaluate the randomness of the output of a block cipher and apply this procedure to evaluate the randomness of the modified AES block cipher using new XOR tables. The proposed dynamic method based on new XOR tables can help improve the security of the AES block cipher against many of today's strong attacks on block ciphers.

Keywords: XOR table; Advanced Encryption Standard; AES; modified AES; randomness assessment.

Reference to this paper should be made as follows: Luong, T.T. and Linh, H.D. (2024) 'On generating new key dependent XOR tables to improve AES security and evaluating the randomness of the output of block ciphers', *Int. J. Information and Computer Security*, Vol. 23, No. 1, pp.16–39.

Biographical notes: Tran Thi Luong received her Bachelor's in Mathematics and Informatics at the Ha Noi University of Sicence in 2006, Master's in Cryptographic Technique at Academy of Cryptographic Techniques in 2012, and PhD in Cryptographic Technique at Academy of Cryptographic Techniques in 2019. His recent research directions are cryptogaphy, coding theory and information security.

Hoang Dinh Linh graduated from the Advanced Math Program at the Ha Noi University of Science in 2014, and Master's in Applied Mathematics in 2021. His recent research directions are evaluation of random generators, and testing for statistical randomness.

1 Introduction

In the Advanced Encryption Standard (AES) selection project (Daemen and Rijmen, 2022), there were five finalists, but only Rijndael was the winning algorithm. AES has been selected as the block cipher standard for the US Government by the National Institute of Standards and Technology (NIST) released on November 26, 2001 and is specified in the Federal Information Processing Standard (FIPS 197). The AES algorithm is a block cipher with substitution – permutation network (SPN) structure (Keliher, 2003; Li et al., 2011), so it consists of three layers: substitution layer, diffusion layer and key addition layer.

To improve the security of block ciphers, people find ways to make them dynamic. To make an SPN block cipher dynamic, you can make one of its components dynamic. Currently, there are many works in these directions, which can be made dynamic at the diffusion layer or the substitution layer, or both. For SPN in general and AES in particular, there are some works in the direction of animating in the diffusion layer such as Al-Wattar et al. (2015), Ismail et al. (2012), Murtaza et al. (2011) and Shamsabad and Dehnavi (2020). Some studies in the direction of making dynamic at the S-box such as Agarwal et al. (2018), Al-Dweik et al. (2022), Assafli and Hashim (2020), Ejaz et al. (2021), Juremi et al. (2017) and Murphy and Robshaw (2002), and some works investigated the way of making dynamic at both diffusion and substitution layers as in Hambouz (2022), Manoj Kumar and Karthigaikumar (2020), Xu et al. (2018) and Yousif (2019).

For the direction of making the SPN block cipher dynamic at the diffusion layer, Al-Wattar et al. (2015) proposed a method for animating the AES block cipher using a dynamic MixColumn transformation. This new MixColumn transformation uses key-dependent MDS matrices based on key-dependent DNA structures and processes. The authors analysed the security of the new MixColumn and tested the randomness through NIST tests. Ismail et al. (2012) proposed a dynamic AES (DRAES) block cipher using a rotation, the amount of rotating depends on the data and the key in the AES key scheme. Murtaza et al. (2011) used a dynamic MixColumn transformation for AES. The new MDS matrices are generated for the MixColumn by scalar multiplication of the rows of the MixColumn matrix and depend on a secret key. Shamsabad and Dehnavi (2020) presented a family of $n \times n$ binary matrices that satisfy several properties, from which it is possible to animate cyclic AES-like matrices and some recursive MDS matrices with less overhead in software.

For the direction of animating the SPN block cipher in the S-box, Agarwal et al. (2018) proposed an algorithm to generate a key-dependent dynamic S-box using an irreducible polynomial and an affine constant. The idea is that whenever a bit of the key is added, the algorithm chooses an irreducible polynomial from the 30 available irreducible polynomials, an affine constant chosen from all the affine values from 0 to 255, and an XOR value of all the bytes of the key. The S-box's values depend on these three parameters. Assafli and Hashim (2020) proposed a dynamic S-box generation algorithm for the timestamp-dependent AES block cipher. The main strength of the proposed method is that the ciphertext changes while keeping the constant encryption key guaranteeing different encryption results for the same data. The authors also studied and analysed the strength and quality of the new S-box using the avalanche criterion and the strict avalanche criterion (SAC). Juremi et al. (2017) proposed a method generating AES dynamic S-boxes using a DeterminantRotation transformation. Each dynamic S-box is

generated for each round by performing a determinant matrix calculation by rotating the positions of the AES S-box. The authors tested the randomness and the avalanche criterion performed on the output to prove the security of the newly proposed algorithm. Murphy and Robshaw (2002) studied how to perform cryptanalysis with key-dependent S-boxes. They developed a framework for differential cryptanalysis of key-dependent S-boxes. Al-Dweik et al. (2022) provided an algorithm schema for generating key-dependent dynamic S-boxes having the same algebraic properties as those of the original S-box, including nonlinearity, SAC and bit independence criteria (BIC). Ejaz et al. (2021) designed a key-dependent dynamic S-box with dynamic permutations to generate symmetric block ciphers with optimal security. The proposed method of creating a dynamic S-box was experimentally evaluated through some measures such as BIC, nonlinearity, hamming distance, balanced output, SAC, and differential and linear approximation probabilities.

For the direction of animating SPN block ciphers at both substitution and diffusion layers in a block cipher, Yousif (2019) proposed a dynamic SPN block cipher based on the Serpent block cipher. The Serpent is one of the candidates for AES block cipher. Yousif (2019) proposed some dynamic methods for permutations, substitutions, and key generation based on chaotic mappings for added security. Manoj Kumar and Karthigaikumar (2020) proposed a key-dependent AES algorithm to secure data over the internet. This proposed algorithm has a better avalanche effect and SAC when compared to AES. Hambouz (2022) proposed a lightweight encryption algorithm based on AES named DLL-AES. DLL-AES animated S-box, ShiftRows, and MixColumn in AES depending on a master key. Instead of a single S-box, DLL-AES has four small-sized S-boxes. Xu et al. (2018) proposed an AES-like cipher by replacing AES's S-box and MixColumn matrix with key-dependent transformations that retain good cryptographic properties.

The above works mainly focus on animating methods for MDS matrices and S-boxes. Recently, Salih et al. (2019, 2020) proposed some dynamic methods for AES based on XOR tables. Specifically, in Salih et al. (2019), the authors used 3D logistic maps to create a private XOR table that can be used to replace the original AES XOR table. In Salih et al. (2020), the authors used Chebyshev polynomial mapping to generate two new XOR tables used for AES rounds alternatively. Furthermore, in Salih et al. (2020), the authors also generated a dynamic MDS matrix from this mapping to replace the MDS matrix in AES's MixColumn. In Salih et al. (2019, 2020), the authors evaluated the security of modified AES block cipher by some different criteria such as NIST test, diehard test, correlation coefficient, ENTROPY and histogram.

It can be seen that the methods proposed in Salih et al. (2019, 2020) are quite new and interesting, which can increase the security of the AES algorithm. However, by carefully studying the results of Salih et al. (2019, 2020), we find that there are many inaccuracies in the proposals of Salih et al. (2019, 2020). In addition, the method of evaluating the randomness of AES and modified AES used by the authors also has many unreasonable problems. We will mention more detail on these in Section 2.

For block ciphers in particular and cryptographic primitives in general, the assessment of randomness is an indispensable requirement in evaluating the security of those cryptographic primitives. Evaluation of the randomness of block ciphers often uses statistical tests. Since the approximation and asymptotic approaches used in the

distribution functions of statistical tests force the users to use long sequences, a general approach to solve this problem is concatenating the outputs of block ciphers or hash functions to form long sequences. In the selection of the AES finalists, NIST used this method (Soto 1999) that concatenates the output strings of block ciphers to generate long sequences of length 220 bit and then used the NIST SP 800-22 (Bassham et al., 2010) test suite to evaluate the randomness. However, the nature of block ciphers requires that the proposed tests and test parameters focus specifically on 'sequences of short length', which are derived directly from the outputs of these block ciphers. Doğanaksoy et al. (2010) also used the approach of concatenating the outputs of the block cipher into a long sequence, then used four tests: the SAC test, linear span test, collision test, and coverage test to evaluate some properties of the block ciphers.

Sulak (2011) proposed a method to evaluate the randomness of block ciphers and hash functions by using some non-random input datasets. The corresponding outputs of the cryptographic primitives will be evaluated for randomness by some statistical tests.

In this paper, we propose a method to create new key-dependent XOR tables to improve the security of the AES block cipher. We also propose a procedure to evaluate the randomness of the output of a block cipher and apply this procedure to evaluate the randomness of AES and the modified AES using these new XOR tables.

The article is organised as follows. In Section 2, we present related works and some comments on the results in Salih et al. (2019, 2020). Section 3 presents the proposed algorithm to create new key-dependent XOR tables. Section 4 proposes a procedure for evaluating the randomness of the output of a block cipher. Section 5 analyses the security of the modified AES block ciphers. Section 6 is the conclusions.

2 Related works

2.1 Transformations in AES

The round function of AES (Daemen and Rijmen, 2002) consists of four transformations:

- *SubByte:* Performs byte substitution of the state array using a (S-box) substitution table.
- *ShiftRow:* Performs left-rotation of the last three rows of the state array, specifically, left-rotation 1 byte in row number 2, left-rotation 2 byte in row number 3, left-rotation 3 byte in row number 4. The first row of the state array is unchanged.
- *MixColumn:* Multiplies the state array by a 4 × 4 circulant MDS matrix.
- *AddRoundKey:* Performs an exclusive OR operation between a round key and the state array. In practice, one will create a pre-stored XOR table (Table 1) for this transformation.

2.2 Remarks on the results in Salih et al. (2019, 2020)

Through studying the methods proposed in Salih et al. (2019, 2020), we make some comments as follows:

1 Remarks on Salih et al. (2019):
   - In Salih et al. (2019), the authors showed three properties of the new XOR table, but these three properties are not enough to ensure the correct decryption process. After creating the private XOR table, the authors did not specify the correctness of this XOR table.
   - The examples given in Salih et al. (2019) are not accurate compared to the actual values in the private XOR table suggested by the authors [see examples and Figure 2 in Salih et al. (2019)].

2 Remarks on Salih et al. (2020):
   - In the proposed algorithm of Salih et al. (2020) for creating two dynamic XOR tables, in step *e* creating two arrays *X*, *Y* but also based on the properties indicated in Salih et al. (2019), so it is not enough to ensure the decryption is correct.
   - In Salih et al. (2020), the authors used the *Z* array to generate a key matrix, and this key matrix is used to perform some permutations on the AES MixColumn matrix to generate a dynamic MDS matrix. However, this dynamic MDS matrix is not an MDS matrix because it has at least two singular square submatrices (see Figure 1).

Figure 1 The dynamic MDS matrix generated by Salih et al. (2020) (see online version for colours)

Dynamic MDS			
1	1	2	3
3	1	2	1
1	1	3	2
1	3	1	2

Dynamic MDS			
1	1	2	3
3	1	2	1
1	1	3	2
1	3	1	2

In addition to the above comments, it can be seen that Salih et al. (2019, 2020) used 3D logistic and Chebyshev mappings to generate secret number keys. These methods are not really effective.

On the other hand, the common point of these two papers is that the evaluation of proposed block ciphers used several methods such as: NIST test, diehard test, histogram, correlation coefficient and ENTROPY. However, there are many unreasonable and inaccurate points in these assessments. Details will be analysed as below:

- The data generation for evaluation in Salih et al. (2019, 2020) is not clear. Specifically, the authors did not specify how to encrypt 10^6 bits of plaintext with 128 different keys to obtain 128 ciphertexts of length 10^6. If only encrypting each block a time, the length of the ciphertext must be divisible by 128 (since each block has 128 bit), however 10^6 is not divisible by 128. Also, when performing block-by-block encryption will result in the same input blocks will produce the same output blocks (ciphertexts). Thus, the output data is no longer random. Furthermore, as recommended in the NIST SP 800-22 standard, the sequence length is 10^6 and the number of sequences needed is 1,000. Therefore, the author's approach is not reasonable when using only 128 sequences for evaluation.

- There is a mistake in presenting the results of the statistical tests of Salih et al. (2020). Specifically, on page 1578 of Salih et al. (2020), the authors made a comment that the AES algorithm does not pass the FFT standard. But the result table [Table 2 in Salih et al. (2020)] showed that AES still passes the *FFT* standard, the criteria it fails must be *overlapping template*.
- According to the NIST SP 800-22 standard, a data file is considered random only when the success rate is in the range [0.9805607, 0.9994392] (Bassham et al., 2010). As such, if the pass rate is 1 then the data is not random. Therefore, if the above requirements are applied, according to Table 2 in Salih et al. (2020), the AES does not pass the criterion of *non-overlapping template, long run, rank, linear complexity*; according to Table 1 in Salih et al. (2020), the algorithm proposed by the authors does not pass the criterion of *FFT, serial 1, linear complexity*. However, in these tables, the authors still marked this criterion as passed. This is not exactly as required by NIST SP 800-22.
- Besides, in NIST SP 800-22, the *non-overlapping template* test has 148 p-values, *Random excursions* has 8 p-values, *random excursions variants* has 18 p-values. But in Salih et al. (2019, 2020), only one p-value is given for each criteria without specifying how it is the p-value in the case of a specific parameter.

Table 1 The original XOR table in AES

XOR	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

3 Proposed algorithm for creating new key-dependent XOR tables

From the remarks in Section 2.2, we first give the necessary properties of an XOR table to be satisfied to ensure that the decryption process is performed correctly. Those properties include:

- The elements on each column and each row of the XOR table belong to [0, 15] and are distinct elements.
- The XOR table is symmetric by the main diagonal.
- If *a XOR b* = *c*, then *b XOR c* = *a* and *a XOR c* = *b*, for every element *a*, *b*, *c* in the XOR table.

Algorithm 1 Creating 02 new XOR tables

Input: a secret key x_0 consists of n bit; the original XOR table of AES (Table 1);	
Output: two new XOR tables A and B.	
Step 1	Use a pseudo-random number generator G with the key seed x_0. The output is then a pseudo-random bit sequence. Take two strings, each consists of 4 bits of the sequence, starting with the $\left\lceil \frac{n}{4} \right\rceil^{\text{th}}$ and $\left\lceil \frac{n}{2} \right\rceil^{\text{th}}$ bits respectively, and then convert them to decimal, yielding two numbers in [0, 15], denoted a and b.
Step 2	• Add all the cells of the original AES XOR table (Table 1) with the number a, then divide the results modulo by 16, to get a new XOR table, denoted A_1. • Add all cells of the original AES XOR table (Table 1) with the number b, then divide the results modulo by 16, to get a new XOR table, denoted B_1.
Step 3	• Shift the positions of rows and columns in two tables A_1 and B_1 so that the first row and column in these two tables is an ascending sequence from 0 to 15. Then, two new XOR tables are obtained, respectively A and B.

In the following, Algorithm 1 is proposed to generate new XOR tables depending on a given secret key.

We should choose the pseudo-random number generator G in Algorithm 1 as a cryptographically secure pseudo-random number generator (Bassham et al., 2010; Saarinen, 2022).

Remark 1: This remark is about the correctness of the new XOR tables. The original AES XOR table satisfies the necessary properties of an XOR table. Algorithm 1 simply adds all the cells of that original XOR table with a number of [0, 15] and then modulo by 16. Thus, the new XOR table can preserve the required properties of an XOR table. We also conduct an experimental check that the new XOR tables satisfy all the three required properties of an XOR table.

Remark 2: The numbers *a*, *b* in Algorithm 1 belong to [0, 15], so there are all 16 possible dynamic XOR tables according to this algorithm.

3.1 Application of new XOR tables to improve AES block cipher

Executing Algorithm 1 yields two new key-dependent XOR tables, denoted A and B. XOR table A will be used for odd rounds, and XOR table B will be used for even rounds of AES.

Table 2 XOR table A_1

XOR	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*0*
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
2	2	1	4	3	6	5	8	7	10	9	12	11	14	13	0	15
3	3	4	1	2	7	8	5	6	11	12	9	10	15	0	13	14
4	4	3	2	1	8	7	6	5	12	11	10	9	0	15	14	13
5	5	6	7	8	1	2	3	4	13	14	15	0	9	10	11	12
6	6	5	8	7	2	1	4	3	14	13	0	15	10	9	12	11
7	7	8	5	6	3	4	1	2	15	0	13	14	11	12	9	10
8	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9
9	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8
10	10	9	12	11	14	13	0	15	2	1	4	3	6	5	8	7
11	11	12	9	10	15	0	13	14	3	4	1	2	7	8	5	6
12	12	11	10	9	0	15	14	13	4	3	2	1	8	7	6	5
13	13	14	15	0	9	10	11	12	5	6	7	9	1	2	3	4
14	14	13	0	15	10	9	12	11	6	5	8	7	2	1	4	3
15	15	0	13	14	11	12	9	10	7	8	5	6	3	4	1	2
0	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Table 3 XOR table B_1

XOR	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8

To apply Algorithm 1 to improve the security of the AES block cipher, we find a true random key derived from a true random number generator source (https://www.random.org/bytes/). This key is used as the input to Algorithm 1. Executing Algorithm 1 with this key as input, two values are obtained:

$a = 1$

$b = 8$

From this, two new XOR tables A_1 and B_1 are obtained as shown in Table 2 and Table 3.

Table 4 XOR table *A* for odd rounds

XOR	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	15	2	1	4	3	6	5	8	7	10	9	12	11	14	13	0
3	14	3	4	1	2	7	8	5	6	11	12	9	10	15	0	13
4	13	4	3	2	1	8	7	6	5	12	11	10	9	0	15	14
5	12	5	6	7	8	1	2	3	4	13	14	15	0	9	10	11
6	11	6	5	8	7	2	1	4	3	14	13	0	15	10	9	12
7	10	7	8	5	6	3	4	1	2	15	0	13	14	11	12	9
8	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10
9	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
10	7	10	9	12	11	14	13	0	15	2	1	4	3	6	5	8
11	6	11	12	9	10	15	0	13	14	3	4	1	2	7	8	5
12	5	12	11	10	9	0	15	14	13	4	3	2	1	8	7	6
13	4	13	14	15	0	9	10	11	12	5	6	7	9	1	2	3
14	3	14	13	0	15	10	9	12	11	6	5	8	7	2	1	4
15	2	15	0	13	14	11	12	9	10	7	8	5	6	3	4	1

Table 5 XOR table *B* for even rounds

XOR	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
1	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
2	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
3	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
4	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
5	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
6	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
7	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
9	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
10	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
11	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
12	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
13	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
14	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
15	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8

Next, shift the positions of rows and columns in two tables A_1 and B_1 so that the first row and column in these two tables is an ascending sequence from 0 to 15, yielding two new XOR tables, denoted A and B, as shown in Table 4 and Table 5.

Then, we use the XOR table A for the odd rounds, the XOR table B for the even rounds of AES to perform encryption and decryption.

4 Proposing a procedure to evaluate the randomness of the output of block ciphers

In this section, we propose an efficient procedure as shown in Figure 2 to evaluate the output randomness for a block cipher. Our process does not need to concatenate the outputs of the block cipher into a long sequence through a mode of operation, but can efficiently evaluate the randomness of the block cipher by statistical tests for sequences of short length. Specifically, our process is as follows:

Step 1 Generate a non-random dataset as input for the block cipher as detailed below. Here, we consider four types of input datasets: low weight (LW) plaintext datasets, high weight (HW) plaintext datasets, 1-bit plaintext avalanche datasets, and plaintext rotation datasets.

Step 2 For each round-reduced version of the block cipher, compute the output dataset corresponding to the input dataset generated in Step 1 and the arbitrary key (we can consider the worst case as non-random key). For example, if the input dataset is LW128, then we need to compute ten output datasets corresponding to the modified block cipher versions with round number from 1 to 10. Each output dataset consists of 349,632 corresponding to 349,632 output sequences of 128-bit input sequences in the LW128 input dataset.

Step 3 For each output dataset, use the NIST two-level test approach with some correction for statistical tests for short sequences to calculate the corresponding p-values for each sequence in the output dataset.

Step 4 Summarise the results and conclusions. A block cipher version with round number r is considered to be random if all the corresponding output datasets pass all the below statistical tests.

Note: The number of rounds r here corresponds to the number of rounds of the block cipher.

4.1 Non-random input plaintext datasets

We use special types as follows:

1 *LW plaintext:* A LW plaintext dataset is made by LW binary sequences. The plaintext length corresponds to the proposed block cipher algorithms with different block size versions. In the 128-bit case, the dataset consists of 128-bit binary sequences whose weight does not exceed 3. Specifically, the number of plaintexts with length of m bits and each plaintext with Hamming weight less than or equal to

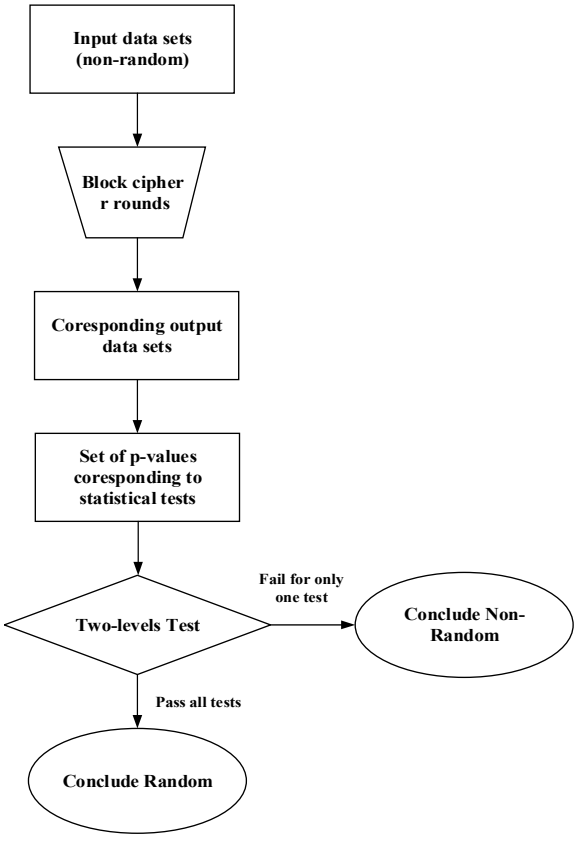
k, denoted by N_n^k, is calculated as $N_n^k = \sum_{i=1}^{k} \binom{n}{i}$. Then, the number of plaintexts for different lengths is 349,632 for 128-bit block length.

2 *HW plaintext:* A HW plaintext dataset is formed by selecting HW plaintexts and they are formed similar to the case of LW plaintext. In other words, the high density inputs are the bitwise complement of the low density inputs.

3 *1-bit plaintext avalanche:* In order to form 1-bit plaintext avalanche (Av1) dataset, firstly a random plaintext R of length m is chosen. Then, each time by flipping another bit of R, a set of m plaintexts is formed and the corresponding ciphertexts are obtained. The same procedure is applied to k different plaintexts to get a set of mk sequences. The values of k for 128-bit input are 1,048,576.

4 *Plaintext rotation:* A random plaintext R of length m is chosen to form plaintext rotation (Rot) dataset, and a set of m plaintexts is formed by consecutive 1-bit rotations of R and the corresponding ciphertexts are obtained. The same procedure is applied to k different plaintexts to get a set of mk sequences.

Figure 2 Procedure of randomness evaluation for block ciphers



4.2 Two-level test

In Step 3, we use two-level test approach in the NIST SP 800-22 test suite (Bassham et al., 2010): Level 1 is checking the proportion of sequences with p-values greater than a threshold; the second is checking the distribution of the p-values.

For Level 1, we count the number of sequences in the sample that have *P-value* $\geq \alpha$ and are denoted by m_p. Then, under the assumption of randomness, m_p follows the binomial distribution $\mathcal{B}(m, 1-\alpha)$ that is approximate to the normal distribution $\mathcal{N}(m(1-\alpha), m\alpha(1-\alpha))$ when n is large enough, where m is the number of test sequences. Therefore, the percentage of sequences that pass a test (= m_p / m) is approximate to $\mathcal{N}\left((1-\alpha), \frac{\alpha(1-\alpha)}{m}\right)$. The acceptable interval of m_p / m is determined using the following level of significance (with $\alpha = 0.01$):

$$1-\alpha-3\sqrt{\frac{\alpha(1-\alpha)}{m}} < \frac{m_p}{m} < 1-\alpha+3\sqrt{\frac{\alpha(1-\alpha)}{m}}.$$

If the pass rate is outside the range above, there is evidence that the data is not random.

For Level 2, we have a set of p-values corresponding to the output dataset for each statistical test. Then, apply the good of fitness test to check if the p-values have a theoretically consistent distribution on the [0, 1] segment, by dividing the [0, 1] segment into ten subintervals [0.0, 0.1], (0.1, 0.2], …, (0.9, 1.0]. Let m be the number of test sequences, and the number of sequences with p-values in the i^{th} interval for i = 1, 2, ⋯, 10. Then, the statistic X follows a χ^2 distribution with nine degrees of freedom and the *p-value* of Level 2 is calculated as follows:

$$X = \sum_{i=1}^{10} \frac{(F_i - m \cdot p_i)^2}{m \cdot p_i}$$

$$\textbf{\textit{p-value}} = igamc\left(\frac{9}{2}, \frac{\chi^2}{2}\right)$$

where *igamc* is an incomplete gamma function. The values p_i, $1 \leq i \leq 10$ have been determined in detail in the below.

If ***p-value*** ≥ 0.0001 then the test result is considered to pass, i.e., the distribution of the p-values is not far from the theoretical values.

In this paper, we just consider 128-bit sequences for AES-128 and the modified AES-128. The exact distributions for six NIST statistics are in Table 6.

In five basic test in Menezes et al. (2018), autocorrelation test checks for correlations between the sequence s and (non-cyclic) shifted versions of it. Let d be a fixed integer, $1 \leq d \leq \lfloor n / 2 \rfloor$. The number of bits in s not equal to their d-shifts is $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$, where $\oplus$ denotes the XOR operator.

The statistic used is

$$X = \frac{2\left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}}$$

which approximately follows an $\mathcal{N}(0,1)$ distribution if $n - d \geq 10$.

Table 6 Theoretical distributions of test results for 128-bit sequence

Sub-interval	*Frequency test*	*Runs test*	*Test for longest run of ones*	*Serial test*	*Approximate entropy test*	*CuSum test*
[0.0, 0.1]	0.092690	0.100679	0.095011	0.101931	0.105734	0.083277
(0.1, 0.2]	0.091993	0.101395	0.104278	0.095514	0.095735	0.102103
(0.2, 0.3]	0.146253	0.113361	0.101632	0.101915	0.092841	0.079860
(0.3, 0.4]	0.095504	0.102226	0.106461	0.112556	0.110089	0.104008
(0.4, 0.5]	0.109829	0.095862	0.102993	0.082076	0.091082	0.130600
(0.5, 0.6]	0.122433	0.104063	0.120047	0.108904	0.119882	0.078669
(0.6, 0.7]	0.000000	0.071873	0.077501	0.089985	0.077519	0.076607
(0.7, 0.8]	0.132306	0.110294	0.109233	0.119499	0.119499	0.086252
(0.8, 0.9]	0.138606	0.114290	0.089208	0.084588	0.084588	0.153731
(0.9, 1.0]	0.070386	0.085956	0.093636	0.103031	0.103031	0.104892

Source: Menezes et al. (2018)

In this paper, we compute exact distributions for autocorrelation test with $d = 1$ (correlation in bits) and $d = 8$ (correlation in bytes).

For $d = 1$, note that the *p-value* of an array depends only on the value of S_{n-1}. For a given value $S_{n-1} = k$, there are exactly $2C_{n-1}^{k}$ n-bit sequences.

First, we have the following lemma.

Lemma 1: Equation $x_1 + x_2 + \cdots + x_a = b$, $x_i \in \{0, 1\}$, $1 \leq i \leq a$ with integers $a \geq b \geq 0$ have exactly C_a^b solutions.

Proof: It is easy to see that for $b = 0$, the equation has a unique solution, $1 = C_a^0$, $x_1 = x_2 = \cdots = x_a = 0$.

For any $b \leq a$, the number of solutions to the equation is the number of ways to choose b values out of a and assign them equal to 1. Therefore, the number of solutions to the equation is C_a^b. ■

Now, we have the following proposition.

Proposition 1: In the space of n-bit sequences $s_0 s_1 \cdots s_{n-1}$, $s_i \in \{0, 1\}$, $0 \leq i \leq n - 1$, there are exactly $2C_{n-1}^{k}$ sequences satisfying $S_{n-1} = \sum_{i=0}^{n-2} s_i \oplus s_{i+1} = k$.

Proof: Let $\delta_j = s_j \oplus s_{j+1}$, $0 \leq j \leq n - 2$ then $\delta_j \in \{0, 1\}$. Apply Lemma 1, the equation $S_{n-1} = \sum_{i=0}^{n-2} \delta_j = k$ have C_{n-1}^{k} solutions. For a set of solutions $(\delta_0, \delta_1, \cdots, \delta_{n-2})$, there are exactly two satisfying sequences $s_0 \cdots s_{n-1}$ because for each value δ_i, $0 \leq i \leq n - 2$ there is (s_i, s_{i+1}) satisfying $s_i \oplus s_{i+1} = \delta_i$. Thus, there is a total $2C_{n-1}^{k}$ of sequences that satisfy $S_{n-1} = \sum_{i=0}^{n-2} s_i \oplus s_{i+1} = \delta_i$. ■

Then, the probability $Pr(S_{n-1} = k) = \frac{2C_{n-1}^{k}}{2^{n}}$.

The pseudocode of the probability interval computing algorithm is presented in Algorithm 2.

Algorithm 2 Computation probability intervals for autocorrelation test in bit (d = 1)

```
Prop_BitAutoCorr(n){
    Initialise array of probability values: Pr[10] = {0, ⋯, 0}.
    for S ← 0 to n – 1 do {
      Compute p-value via S, n
      if p-value in j interval then {
            Pr[j] = Pr[j] + Pr[S_{n–1} = S]
      }
    }
}
```

Table 7 shows the theoretical distributions of BitAutoCorr test results for individual sequences.

Table 7 Theoretical distributions of BitAutoCorr test results for individual sequences

Sub-interval	*Autocorrelation test (d = 1)*			
	128	*160*	*256*	*512*
[0.0, 0.1]	0.1098495026	0.1124301194	0.1032997668	0.0926615712
(0.1, 0.2]	0.1041156515	0.0919094698	0.1070224747	0.1227748137
(0.2, 0.3]	0.0729494532	0.0624856743	0.1060414054	0.0729268419
(0.3, 0.4]	0.0880424435	0.1610145475	0.0643152170	0.0879465019
(0.4, 0.5]	0.1029648915	0.0980846952	0.1505713250	0.1027967555
(0.5, 0.6]	0.1166935437	0.1084094000	0.0852223753	0.1164603740
(0.6, 0.7]	0.1281715972	0.1168568857	0.0907205931	0.1278855543
(0.7, 0.8]	0.1364407325	0.0000000000	0.0950751815	0.0672612360
(0.8, 0.9]	0.0000000000	0.1228495465	0.0980934413	0.1387970805
(0.9, 1.0]	0.1407721843	0.1259596616	0.0996382199	0.0704892710

For d = 8, applying the same argument in the proof of Proposition 1, we have the following proposition.

Proposition 2: In the space of n-bit sequences $s_0 s_1 \cdots s_{n-1}$, $s_i \in \{0, 1\}$, $0 \le i \le n-1$, there are exactly $2^d C_{n-d}^{k}$ sequences satisfying $S_{n-d} = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d} = k$. Then, we have

$$Pr(S_{n-d} = k) = \frac{C_{n-d}^{k}}{2^{n-d}}.$$

For d = 8, we have:

$$Pr(S_{n-8} = k) = \frac{C_{n-8}^{k}}{2^{n-8}}.$$

The pseudocode of the probability interval computing algorithm is given in Algorithm 3 as follows.

Algorithm 3 Computation probability intervals for autocorrelation test in byte ($d = 8$)

```
Prop_ByteAutoCorr(n){
    Initialise array of probability values: Pr[10] = {0, ⋯, 0}.
      for S ← 0 to n – 1 do {
      Compute p-value via S, n
       if p-value in j interval then {
          Pr[j] = Pr[j] + Pr[S_{n–8} = S]
       }
    }
}
```

Table 8 shows the theoretical distributions of ByteAutoCorr test results for individual sequences.

Table 8 Theoretical distributions of ByteAutoCorr test results for individual sequences

Sub interval	*Autocorrelation test (d = 8)*			
	128	*160*	*256*	*512*
[0.0, 0.1]	0.0824074794	0.0881746480	0.1122155681	0.0992339139
(0.1, 0.2]	0.0882377338	0.1354384538	0.0700378602	0.0971560850
(0.2, 0.3]	0.1446580877	0.0680413496	0.0980850582	0.1092047340
(0.3, 0.4]	0.0961199194	0.0806776002	0.1287998752	0.0917978299
(0.4, 0.5]	0.1115677636	0.0931769467	0.0758017644	0.1066825418
(0.5, 0.6]	0.1252690679	0.1048240651	0.0828086502	0.1201174264
(0.6, 0.7]	0.0000000000	0.1148756877	0.0890192990	0.0643483118
(0.7, 0.8]	0.1360681255	0.1226375585	0.1921977440	0.1352417030
(0.8, 0.9]	0.1429868438	0.1275430609	0.1004195089	0.0699293336
(0.9, 1.0]	0.0726849789	0.0646106295	0.0506146718	0.1062881206

5 Security analysis of modified AES algorithm

5.1 Security analysis

Currently, there are many types of attacks on SPN block ciphers, of which the two most strong attacks are linear attacks and differential attacks. The linear attack is a kind of attack introduced by Matsui (1994). The linear attack (Heys and Tavares, 1996; Matsui, 1994) is a known plaintext attack that requires the collection of a large number of plaintext and ciphertext pairs corresponding to an unknown key to be searched. The differential attack (Lai et al., 1991; Matsui, 1994) is a form of selected plaintext attack

that requires the collection of a large number of ciphertexts generated by pre-selected plaintexts.

For linear attacks, the strongest known version of the attack is Matsui's (1994) Algorithm 2. The data complexity (number of plaintext/ciphertext pairs) of Matsui's Algorithm 2 against linear attack is given by the following formula (Keliher, 2003):

$$N_L = \frac{c}{ELP^{[1...T]}(a, b)} \tag{1}$$

The data complexity (number of plaintext/ciphertext pairs) against differential attack is (Keliher, 2003):

$$N_D = \frac{1}{EDP^{[1...T]}(\Delta X, \Delta Y)} \tag{2}$$

where $ELP^{[1...T]}(a, b)$ is the average linear probability over $1 \ldots T$ rounds with input and output mask are a and b, respectively. Similarly, $EDP^{[1...T]}(\Delta X, \Delta Y)$ is the probability of the average differential probability over $1 \ldots T$ rounds with input and output differences are ΔX and ΔY, respectively (see more in Keliher, 2003).

Thus, to successfully perform these two types of linear and differential attacks, one must collect an enormous number of plaintext/ciphertext pairs. For example, with the DES algorithm, if a cryptanalyst expects successfully perform the linear attack, he must collect about 2^{47} plaintext/ciphertext pairs (Matsui, 1994). This number is a large one.

However, there is a crucial point to note here, to perform these two types of attacks, the cryptanalyst must know each component in the structure of the block cipher. For example, with the AES algorithm, cryptanalysts need to know the exact S-box, MixColumn matrix, ShiftRow operation, and XOR operation used in AES before they can attack AES by linear or differential ones. But when we animate any component of AES, it means that the cryptanalyst does not know exactly what that component is. So they meet more difficulties finding the exact part made dynamic in AES and then proceed with the regular cryptanalysis.

Thus, if a cryptanalyst wants to perform a linear or differential attack on a dynamic block cipher, they must perform the following steps:

Step 1 Try (by exhausting) the animated component in the block cipher. Assuming the candidate for this component is X_i (corresponding to the i^{th} attempt).

Step 2 Carry out linear/differential attacks with a dynamic block cipher associated with a dynamic component X_i (in Step 1). The cryptanalyst then needs to collect T plaintext/ciphertext pairs. Then perform the cryptanalysis steps as usual.

Step 3 If the cryptanalysis in Step 2 is unsuccessful, return to Step 1.

The above three-step process ends when the cryptanalyst successfully executes for cryptanalysis the block cipher with a candidate of the dynamic component.

Thus clearly, in terms of data complexity, instead of T plaintext/ciphertext pairs, the cryptanalyst would have to collect nT plaintext/ciphertext pairs (with n attempts). In terms of time complexity, it also increases accordingly. Therefore, we can assert that animating block ciphers will greatly increase security of block ciphers.

In this paper, for the modified AES algorithm, instead of using an XOR table like AES's, we use two new XOR tables (A and B), which are used alternately in AES rounds.

The new XOR tables are generated from a given secret key and the original AES XOR table. The modified AES block cipher (also known as key-dependent DRAES block cipher) will be much more difficult than AES for cryptanalysis. In practice, the algorithm generating new XOR tables (Algorithm 1) can be public but the cryptanalysts do not know the secret key. Therefore, they can not know which XOR table used for AES. Furthermore, the alternatively use of new XOR tables for AES rounds also makes it much more difficult for cryptanalysts because they do not know which XOR table used for which round of AES.

Another aspect is that the number of dynamic XOR tables generated by Algorithm 1 is 16, which is not too large, but in practice it will make it much more difficult for cryptanalysts to collect many pairs of plaintexts and ciphertexts. Generally, if the cryptanalysts do the exploration by the exhaustive method, it is very difficult for them to do. Thus, it can be seen that the modified AES block cipher with new key-dependent XOR tables can improve the security of the AES block cipher.

Compare our method with methods in Salih et al. (2019, 2020).

Since our method is to generate dynamic XOR tables to animate the AES block cipher at the Addroundkey layer, so in this paper we only compare our method with those proposed in Salih et al. (2019, 2020). Other research directions focus on animating AES at the substitution and diffusion layers, so we will not compare them with our method in this paper.

Table 9 Compare our method with methods in Salih et al. (2019, 2020)

Criteria	*Our method*	*Methods in Salih et al. (2019, 2020)*
Dynamic by key	Yes	Yes
Number of new XOR tables created	2	1
Dynamic XOR table space	16	1
Random bit generation algorithm	Pseudo-random bit generator (more efficient)	Chaotic mapping
Necessary properties for the XOR table	Fully indicated	Incompletely indicated
Security of modified AES algorithm	More secure because dynamic XOR tablespace is larger	The dynamic MDS matrix in Salih et al. (2019) is not an MDS matrix, the number of XOR tables is small, which may affect the security of the modified AES algorithm

5.2 Evaluation of randomness via NIST two-level approach

We have applied our procedure to evaluate the randomness of outputs of AES and the modified block cipher. The obtained results show that the modified block cipher achieve the output randomness equivalent to AES block cipher. Figure 3 shows the successful proportion of statistical tests by round for AES and the modified AES. Table 10 shows randomness assessment results for original AES.

Experimental results show that the original AES block cipher needs three rounds to achieve randomness for datasets AV1, HW, LW and two rounds to achieve randomness for dataset Rot. In summary, the AES block cipher requires three rounds to achieve randomness for the datasets considered in this paper. Table 11 presents the randomness assessment results for the modified AES.

Figure 3 The successful proportion of statistical tests by round for AES and the modified AES (see online version for colours)

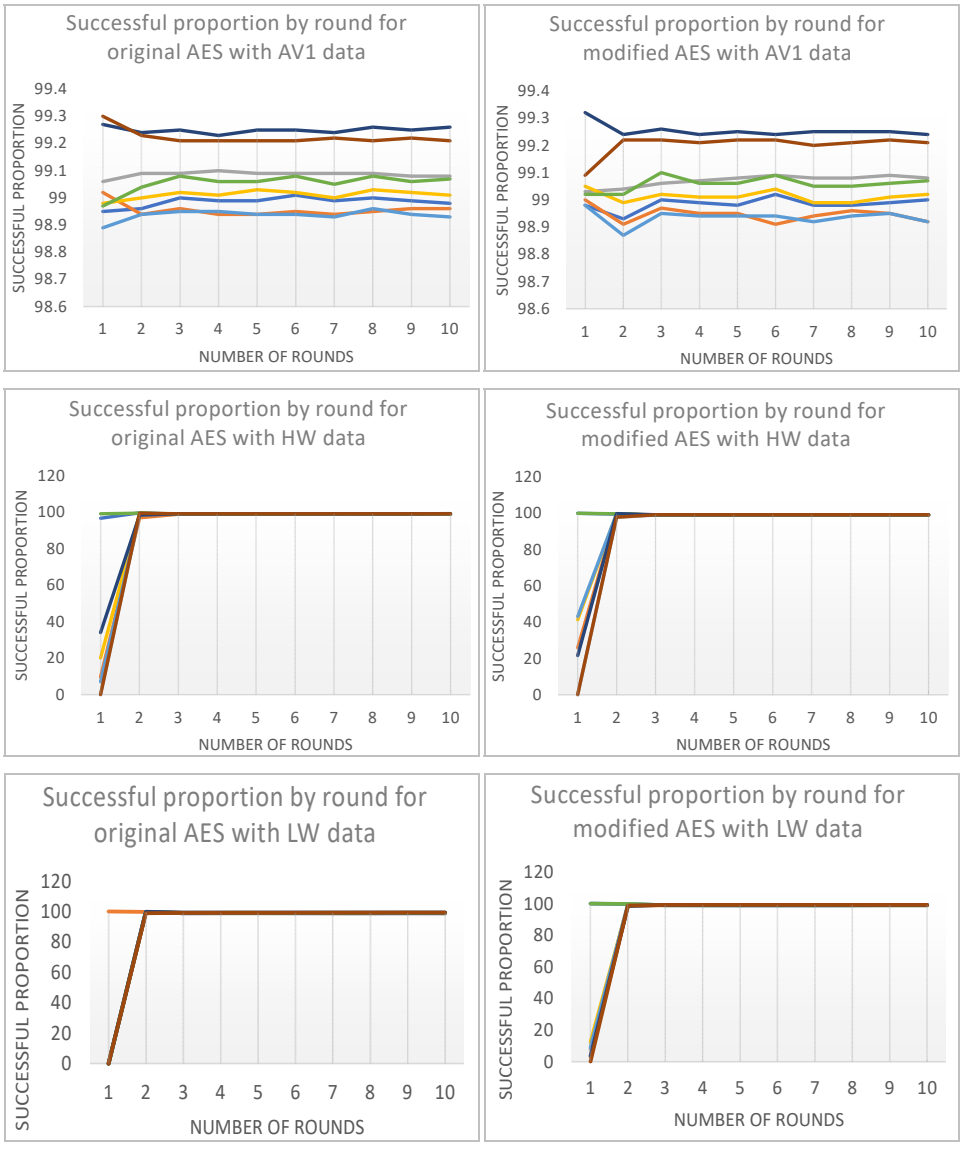


Figure 3 The successful proportion of statistical tests by round for AES and the modified AES (continued) (see online version for colours)

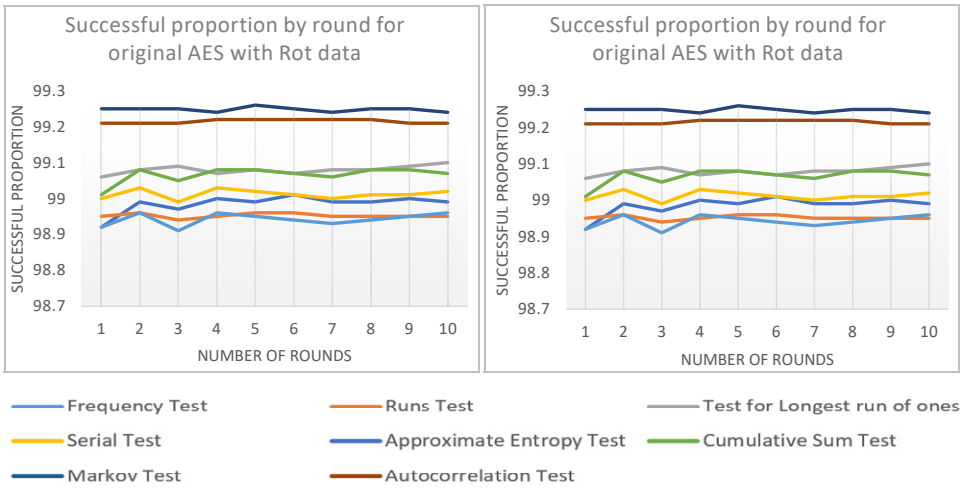


Table 10 Randomness assessment results for original AES

No. of rounds	*Freq. test*	*Runs test*	*Test for longest run of ones*	*Serial test*	*AppEn. test*	*CuSum. test*	*Bit AutoCorr. test*	*Byte AutoCorr. test*
				AV1 input data				
1	0	0	0	0	0	0	0	0
2	0	0	0.097352	0.018387	0.001756	0	0	0.000240
3	*0.674352*	*0.718134*	*0.543234*	*0.854342*	*0.938274*	*0.079594*	*0.833264*	*0.819625*
4	0.837992	0.780179	0.677498	0.520289	0.861955	0.660107	0.225586	0.515942
5	0.311486	0.477947	0.361286	0.176993	0.831995	0.972854	0.685608	0.241037
6	0.161481	0.277486	0.620682	0.444318	0.690645	0.067994	0.358080	0.987605
7	0.691699	0.182190	0.878501	0.859037	0.749850	0.545242	0.142185	0.963018
8	0.463798	0.573471	0.645756	0.833488	0.847496	0.814524	0.850207	0.751446
9	0.554941	0.494566	0.595929	0.468640	0.519504	0.238870	0.294216	0.719405
10	0.637994	0.085964	0.919557	0.073139	0.217203	0.664697	0.216509	0.246476
				HW input data				
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	*0.092147*	*0.376637*	*0.483114*	*0.468203*	*0.508598*	*0.047413*	*0.248938*	*0.650548*
4	0.187437	0.331880	0.430121	0.433326	0.371806	0.188852	0.399271	0.037653
5	0.729675	0.688300	0.680246	0.513069	0.663646	0.073020	0.172822	0.943102
6	0.909228	0.321444	0.081417	0.609881	0.810902	0.994181	0.417283	0.956882
7	0.387684	0.458585	0.281717	0.064036	0.095100	0.990907	0.039058	0.120226
8	0.429938	0.909851	0.535240	0.673103	0.283729	0.061184	0.599375	0.283946
9	0.363365	0.130117	0.296703	0.219469	0.061937	0.354226	0.385171	0.825214
10	0.703550	0.510023	0.195757	0.258551	0.052523	0.090337	0.691955	0.793155

Table 10 Randomness assessment results for original AES (continued)

No. of rounds	*Freq. test*	*Runs test*	*Test for longest run of ones*	*Serial test*	*AppEn. test*	*CuSum. test*	*Bit AutoCorr. test*	*Byte AutoCorr. test*
				LW input data				
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	*0.283404*	*0.499057*	*0.062722*	*0.662772*	*0.710793*	*0.409956*	*0.285987*	*0.977911*
4	0.602345	0.242995	0.059947	0.622687	0.575576	0.338580	0.305306	0.066553
5	0.227775	0.993694	0.618060	0.911933	0.911703	0.788742	0.782909	0.673211
6	0.264241	0.521711	0.018088	0.000463	0.110097	0.023398	0.616281	0.860850
7	0.291630	0.714159	0.039952	0.467268	0.351942	0.677446	0.796704	0.215079
8	0.188269	0.399319	0.115799	0.908694	0.976455	0.078170	0.033836	0.062437
9	0.980581	0.002436	0.281112	0.310044	0.144750	0.916318	0.551379	0.637450
10	0.228045	0.364696	0.543255	0.033877	0.030543	0.420899	0.847672	0.097713
				Rot input data				
1	0	0.377923	0.000022	0	0	0	0.123395	0.725779
2	*0.200353*	*0.746394*	*0.584519*	*0.096069*	*0.086802*	*0.719978*	*0.740125*	*0.521973*
3	0.731760	0.761428	0.239517	0.661889	0.628289	0.364076	0.782763	0.664107
4	0.174402	0.957485	0.632356	0.361589	0.087942	0.923035	0.561526	0.772368
5	0.217767	0.846416	0.440244	0.647179	0.861486	0.222809	0.540009	0.962706
6	0.948885	0.319307	0.717474	0.178828	0.150365	0.268130	0.739672	0.845436
7	0.733056	0.515920	0.243853	0.504955	0.266844	0.546180	0.573704	0.409911
8	0.724931	0.382191	0.917489	0.030084	0.582831	0.761451	0.741164	0.335243
9	0.803725	0.069296	0.265447	0.267188	0.267692	0.900419	0.064927	0.552041
10	0.773706	0.108572	0.186316	0.899298	0.888591	0.070233	0.958691	0.158013

Table 11 Randomness assessment results for the modified AES

No. of rounds	*Freq. test*	*Runs test*	*Test for longest run of ones*	*Serial test*	*AppEn. test*	*CuSum. test*	*Bit AutoCorr. test*	*Byte Autocor. test*
				AV1 input data				
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0.000022	0
3	*0.598894*	*0.018305*	*0.339122*	*0.114644*	*0.017900*	*0.401902*	*0.068564*	*0.783297*
4	0.543253	0.601551	0.597370	0.474953	0.471361	0.511647	0.962485	0.907979
5	0.704804	0.098806	0.752760	0.019669	0.009143	0.770070	0.349626	0.745146
6	0.590701	0.129585	0.685053	0.286265	0.313612	0.489214	0.106276	0.727202
7	0.936945	0.023026	0.582644	0.416645	0.402667	0.176081	0.005053	0.302354
8	0.128106	0.413249	0.187094	0.450533	0.667270	0.617485	0.098959	0.065653
9	0.752848	0.202312	0.891466	0.449087	0.144161	0.583684	0.141038	0.861803
10	0.528404	0.357128	0.749070	0.221138	0.430105	0.752828	0.624921	0.712713

Table 11 Randomness assessment results for the modified AES (continued)

No. of rounds	*Freq. test*	*Runs test*	*Test for longest run of ones*	*Serial test*	*AppEn. test*	*CuSum. test*	*Bit AutoCorr. test*	*Byte Autocor. test*
				HW input data				
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	*0.174302*	*0.610036*	*0.605127*	*0.056554*	*0.048234*	*0.293482*	*0.270106*	*0.802985*
4	0.494330	0.879917	0.057344	0.520374	0.848626	0.588860	0.960304	0.760260
5	0.233626	0.378885	0.837758	0.637098	0.344653	0.392646	0.048854	0.008479
6	0.013096	0.003210	0.186732	0.106056	0.077918	0.018103	0.069971	0.083227
7	0.150654	0.605880	0.316956	0.253551	0.158879	0.568887	0.281234	0.406410
8	0.416360	0.868021	0.242318	0.926386	0.825539	0.869072	0.649840	0.644573
9	0.018936	0.896625	0.644544	0.822869	0.779568	0.314417	0.486969	0.456262
10	0.449559	0.000391	0.487652	0.525436	0.157062	0.336376	0.383195	0.889668
				LW input data				
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	*0.043039*	*0.883915*	*0.526806*	*0.679981*	*0.508092*	*0.262151*	*0.951689*	*0.677082*
4	0.884328	0.503495	0.475520	0.932677	0.929271	0.400384	0.892766	0.825893
5	0.317119	0.818725	0.616227	0.182688	0.183484	0.050599	0.975772	0.353146
6	0.849597	0.001014	0.084951	0.859124	0.189599	0.038893	0.759658	0.326692
7	0.082559	0.191408	0.447837	0.014648	0.012876	0.479664	0.057522	0.011973
8	0.247371	0.121516	0.716245	0.629854	0.905881	0.206362	0.135625	0.075361
9	0.856573	0.172818	0.184829	0.802924	0.767476	0.650117	0.396199	0.474076
10	0.306695	0.015107	0.049570	0.560368	0.326504	0.505363	0.105989	0.239043
				Rot input data				
1	0	0.652409	0	0.000011	0.000007	0.277951	0.049359	0.040243
2	0.000167	0.000250	0.156069	0.133972	0.279440	0	0.764219	0.840597
3	*0.952683*	*0.742995*	*0.051011*	*0.980236*	*0.842737*	*0.094734*	*0.951273*	*0.859309*
4	0.492108	0.033846	0.238144	0.713371	0.830867	0.628785	0.767785	0.630855
5	0.858886	0.807524	0.594868	0.263077	0.291826	0.111183	0.376281	0.114936
6	0.303912	0.378293	0.567738	0.450278	0.429716	0.126456	0.479104	0.182860
7	0.634938	0.104833	0.272378	0.010423	0.453412	0.707197	0.229118	0.961944
8	0.589786	0.577970	0.340470	0.727833	0.669176	0.470889	0.233397	0.792162
9	0.960757	0.203889	0.392777	0.767143	0.561793	0.315061	0.489857	0.997386
10	0.379779	0.693256	0.006520	0.040628	0.026037	0.920812	0.386460	0.390020

Experimental results show that the modified AES block cipher requires three rounds to achieve randomness for all four datasets AV1, HW, LW and Rot. Although modified AES requires one more round than original AES to achieve randomness for Rot dataset,

the overall result still requires three rounds equivalent to original AES to achieve randomness for all four datasets.

6 Conclusions

Studying of dynamic methods and techniques to increase the security of block ciphers in general and AES in particular is very important today due to the strong development of cryptanalysis as well as the variety of new attacks on block ciphers. The more different dynamic methods of block ciphers are available, the more candidates can be chosen to improve the security of the block ciphers against today's strong attacks. In this paper, we have proposed a method to make the AES block cipher dynamic through new key-dependent XOR tables. We also propose a procedure to evaluate the randomness of the output of a block cipher and apply this procedure to evaluate the randomness of the modified AES block cipher. We also analyse the security of the modified AES block cipher and evaluate the randomness using NIST statistical criterion. The proposed method contributes to improve the security of the AES block cipher against many current strong attacks.

References

Agarwal, P., Singh, A. and Kilicman, A. (2018) 'Development of key-dependent dynamic S-boxes with dynamic irreducible polynomial and affine constant', *Advances in Mechanical Engineering*, Vol. 10, No. 7, pp.1–18.

Al-Dweik, A.Y., Hussain, I., Saleh, M. and Mustafa, M.T. (2022) 'A novel method to generate key-dependent s-boxes with identical algebraic properties', *Journal of Information Security and Applications*, Vol. 64, p.103065, ISSN: 2214-2126 [online] https://doi.org/10.1016/j.jisa.2021.103065; https://www.sciencedirect.com/science/article/pii/S2214212621002477.

Al-Wattar, A.H., Mahmod, R., Zukarnain, Z.A. and Udzir, N. (2015) 'A new DNA based approach of generating key dependent MixColumns transformation', *International Journal of Computer Networks & Communications (IJCNC)*, Vol. 7, No. 2, pp.93–102.

Assafli, H.T. and Hashim, I.A. (2020) 'Generation and evaluation of a new time-dependent dynamic S-box algorithm for AES block cipher cryptosystems', in *IOP Conference Series: Materials Science and Engineering*, Vol. 978, No. 1, p.012042.

Bassham III, L.E., Rukhin, A.L., Soto, J., Nechvatal, J.R., Smid, M.E., Barker, E.B., Vo, S. et al. (2010) *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publication No. 800-22.

Daemen, J. and Rijmen, V. (2002) *The Design of Rijndael*, Vol. 2, Springer-Verlag, Berlin, Heidelberg.

Doğanaksoy, A., Ege, B., Koçak, O. and Sulak, F. (2010) *Cryptographic Randomness Testing of Block Ciphers and Hash Functions*, Cryptology ePrint Archive.

Ejaz, A., Shoukat, I.A., Iqbal, U., Rauf, A. and Kanwal, A. (2021) 'A secure key dependent dynamic substitution method for symmetric cryptosystems', *PeerJ Computer Science*, Vol. 7, p.e587, PMID: 34395857, PMCID:PMC: 8323725 [online] https://doi.org/10.7717/peerj-cs.587.

Hambouz, A.A. (2022) *DLL-AES: Dynamic Layers Lightweight AES Algorithm*, Doctoral dissertation, Princess Sumaya University for Technology, Jordan.

Heys, H.M. and Tavares, S.E. (1996) 'The design of product ciphers resistant to differential and linear crypt-analysis', *Journal of Cryptography*, Vol. 9, No. 1, pp.1–19.

Ismail, I., Galal-Edeen, G., Khattab, S. and Moustafa, M. (2012) 'Performance examination of AES encryption algorithm with constant and dynamic rotation', *International Journal of Reviews in Computing*, Vol. 12.

Juremi, J., Mahmod, R., Zukarnain, Z.A. and Yasin, S.M. (2017) 'Modified AES s-box based on determinant matrix algorithm', *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 7, No. 1, pp.110–116.

Keliher, L.T. (2003) *Linear Cryptanalysis of Substitution-Permutation Networks*, Queen's University.

Lai, X., Massey, J.L. and Murphy, S. (1991) 'Markov ciphers and differential cryptanalysis', in *Proceedings of Advances in Cryptology, LNCS 473*, Springer, pp.389–404.

Li, R., Sun, B. and Li, C. (2011) 'Impossible differential cryptanalysis of SPN ciphers', *IET Information Security*, Vol. 5, No. 2, pp.111–120.

Manoj Kumar, T. and Karthigaikumar, P. (2020) 'A novel method of improvement in Advanced Encryption Standard algorithm with dynamic shift rows, sub byte and mixcolumn operations for the secure communication', *International Journal of Information Technology*, September, Vol. 12, pp.825–830, Springer, DOI [online] https://doi.org/10.1007/s41870-020-00465-1; https://link.springer.com/article/10.1007/s41870-020-00465-1#citeas.

Matsui, M. (1994) 'Linear cryptanalysis method for DES cipher', in *Advances in Cryptology – EUROCRYPT'93: Workshop on the Theory and Application of Cryptographic Techniques, 1993 Proceedings 12*, Springer, Berlin, Heidelberg, Lofthus, Norway, 23–27 May, pp.386–397.

Menezes, A.J., Van Oorschot, P.C. and Vanstone, S.A. (2018) *Handbook of Applied Cryptography*, CRC Press, DOI: 10.1201/9781439821916 [online] https://cacr.uwaterloo.ca/hac/.

Murphy, S. and Robshaw, M.J.B. (2002) 'Key-dependent S-boxes and differential cryptanalysis', *Designs, Codes and Cryptography*, Vol. 27, pp.229–255, DOI: 10.1023/A:1019991004496 [online] https://www.semanticscholar.org/paper/Key-Dependent-S-Boxes-and-Differential-Murphy-Robshaw/d7e4f403886fb704928d25e7b465c79fb3909a8e.

Murtaza, G., Khan, A.A., Alam, S.W. and Farooqi, A. (2011) *Fortification of AES with Dynamic Mix-column Transformation*, IACR Cryptology ePrint Archive.

Saarinen, M.J.O. (2022) 'SP 800-22 and GM/T 0005-2012 tests: clearly obsolete, possibly harmful', in *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp.31–37.

Salih, A.I., Alabaich, A. and Tuama, Y. (2020) 'Enhancing advance encryption standard security based on dual dynamic XOR table and mixcolumns transformation', *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 19, No. 3, pp.1574–1581.

Salih, A.I., Alabaichi, A. and Abbas, A.S. (2019) 'A novel approach for enhancing security of advance encryption standard using private XOR table and 3D chaotic regarding to software quality factor', *ICIC Express Letters Part B: Applications, An International Journal of Research and Surveys*, Vol. 10, No. 9, pp.823–832.

Shamsabad, M.R.M. and Dehnavi, S.M. (2020) 'Dynamic MDS diffusion layers with efficient software implementation', *International Journal of Applied Cryptography*, Vol. 4, No. 1, pp.36–44.

Soto, J. (1999) *Randomness Testing of the Advanced Encryption Standard Candidate Algorithms*, p.9, US Department of Commerce, Technology Administration, National Institute of Standards and Technology.

Sulak, F. (2011) *Statistical Analysis of Block Ciphers and Hash Functions*, PhD – Doctoral Program, Middle East Technical University [online] https://open.metu.edu.tr/handle/11511/20626.

Xu, T., Liu, F. and Wu, C. (2018) 'A white-box AES-like implementation based on key-dependent substitution-linear transformations', *Multimedia Tools and Applications*, July, 10 March, Vol. 77, DOI [online] https://doi.org/10.1007/s11042-017-4562-8; https://link.springer.com/article/10.1007/s11042-017-4562-8#citeaspp.18117–18137.

Yousif, I.A. (2019) 'Proposed a permutation and substitution methods of serpent block cipher', *Ibn AL-Haitham Journal for Pure and Applied Science*, Vol. 32, No. 2, pp.131–144.