

Constructing dynamic XOR charts for block ciphers using hadamard matrices

Truong Minh Phuong¹, Tran Thi Luong²

¹Faculty of Cryptography, Academy of Cryptography Techniques, Hanoi, Vietnam

²Faculty of Information Security, Academy of Cryptography Techniques, Hanoi, Vietnam

Article Info

Article history:

Received Aug 7, 2024

Revised Mar 30, 2025

Accepted Jul 2, 2025

Keywords:

AddRoundKey

AES block cipher

Hadamard matrices

Key-dependent

NIST statistical criteria

ABSTRACT

Block ciphers are vital for modern encryption, ensuring the security of digital communications. Currently, powerful attacks target block ciphers, prompting researchers to propose ideas to enhance their cryptographic strength. One notable concept involves making components dynamic and dependent on a secret key, with limited attention given to the dynamic AddRoundKey operation. In this article, we introduce the definitions of some Hadamard matrix forms like B_{had} , N_{had} , and NB_{had} matrices. Subsequently, we present an algorithm for generating key-dependent XOR charts to create a key-dependent AddRoundKey operation based on these matrices. We then construct a dynamic AES block cipher by applying the proposed AddRoundKey operation to AES. We implement the dynamic AES algorithm, assess its security, and evaluate AES and the advanced AES using NIST's statistical standards. The dynamic AES algorithm exhibits improved resistance against strong block cipher attacks compared to conventional AES.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Tran Thi Luong

Faculty of Information Security, Academy of Cryptography Techniques

141 Chien Thang, Tan Trieu, Thanh Tri, Hanoi, Vietnam

Email: luongtran@actvn.edu.vn

1. INTRODUCTION

Block ciphers operate on fixed-size data blocks, typically in multiples of 64 or 128 bits. Utilizing a secret key, block ciphers transform fixed-size input data blocks into encrypted output blocks of the same size through a series of mathematical operations. Permutation-substitution networks (SPNs) [1] represent a common architectural pattern in block ciphers. An SPN block cipher's round function includes three operations: substitution, key addition, and permutation.

NIST officially introduced the AES block cipher on November 26, 2001, as detailed in FIPS 197. Since its adoption, AES has become the predominant algorithm for ensuring data confidentiality across various applications. AES [2], [3] uses an SPN design. Several cryptographic attacks have been directed at the AES block cipher, but as of the present moment, AES continues to exhibit security against known attacks. Nevertheless, researchers still point out that powerful cryptographic analysis attacks, such as differential cryptanalysis [1], [4], linear cryptanalysis [1], [5]-[7], algebraic cryptanalysis, [8], [9], and several others, may pose potential security risks for the AES block cipher. Therefore, research efforts have been conducted to improve the resilience of the AES through the implementation of published dynamic methods.

To bolster the effectiveness of SPN block ciphers, particularly AES, numerous research efforts have introduced dynamism through various methodologies. Presently, various research endeavors can be rendered dynamic within the substitution transformation, the diffusion transformation, or both. Regarding SPNs in a

broader sense and AES, in particular, some studies have delved into the dynamism introduced within the S-box layer, as evidenced by references [9]-[12]. Additional research has concentrated on the implementation of dynamic features within the diffusion layer, as evidenced by references [13]-[16], while some works have inquired into the dynamic integration of both substitution and diffusion transformations, as exemplified by references [17]-[20].

James and Priya [9] introduced a method to secure communication between IoT platforms by developing a strong, key-dependent algorithm incorporating nonlinear S-Boxes. These dynamic algorithms are constructed using the PRESENT lightweight block cipher as a foundation. To assess its suitability for cold chain security applications, the security analysis includes evaluations of linear and differential cryptanalysis, non-linearity, as well as avalanche characteristics, in addition to IoT data encryption. Al-Dweik *et al.* [10] proposed a method for enhancing SPN block ciphers by creating dynamic, key-dependent S-boxes that maintain the original algebraic properties, including the strict avalanche criterion (SAC), non-linearity, and bit independence criteria (BIC). Similarly, Assafli and Hashim [11] introduced an algorithm to generate time-variant S-boxes for the AES block cipher, allowing ciphertext variations even with a fixed key, thereby guaranteeing varied encryption results for the same data. Additionally, the authors conducted a comprehensive evaluation of the new S-box's robustness and effectiveness, employing both SAC and the avalanche criterion (AC). Ejaz *et al.* [12] crafted a dynamic S-box reliant on the encryption key, employing dynamic permutations to create a dynamic block cipher with maximum security. The approach introduced for generating this dynamic S-box underwent empirical assessment utilizing various metrics, including Hamming distance, non-linearity, BIC, balanced output, and SAC.

In their research, the authors proposed methods to introduce dynamism into the diffusion layers of SPN block ciphers. They developed parameterized binary matrices of size $m \times m$, where $m = 4t$, enabling the integration of dynamic diffusion components with minimal software overhead. Additionally, they presented new sets of maximum distance separable (MDS) matrices designed for dynamic diffusion processes, enhancing the cipher's resistance to attacks. Furthermore, they introduced a technique to incorporate dynamism into AES by employing a key-dependent MixColumn operation, thereby enhancing the security of the cipher. This innovative MixColumn operation incorporates specific DNA processes. Meanwhile Ismail *et al.* [15] presented a dynamic variant of the AES block cipher, which incorporates rotations. The extent of rotation varies according to the key and the data within the key scheme. Chen *et al.* [16] proposed an innovative lightweight block cipher that utilizes ARX-based dynamic techniques. They merged the secret key with the extended two-dimensional cat transformation to create an ever-changing rearrangement stratum, referred to as P1. This augmentation bolsters the unpredictability across successive iterations of the dynamic block cipher process. The non-linear aspect of the cycle function employs an alternating combination of the AND gate and the NAND gate to enhance the intricacy of potential attacks.

To explore the dynamics of SPN block ciphers, both in the substitution and diffusion transformations, the authors presented a dynamic cipher, Kumar and Karthigaikumar [17] provided a dynamic AES algorithm aimed at safeguarding data transmitted over the Internet. Their algorithm demonstrated good SAC and avalanche effects. Xu *et al.* [18] building upon the Serpent cipher. They introduced techniques using chaotic mappings for key generation, substitutions, and permutations to enhance security. Hambouz [19] provided a dynamic cipher that replaces the MixColumn matrix and the S-box in AES with other ones dependent on a key while preserving strong cryptographic attributes. Salih *et al.* [20] a lightweight algorithm was introduced, rooted in AES but modifying ShiftRows, S-box, and MixColumn operations based on a secret key.

Apart from approaches that introduce dynamism to block ciphers through substitution and diffusion transformations in [21], [22], the authors proposed approaches for making AES dynamic using dynamic XOR charts dependent on pre-shared secret values. They use chaos maps and Chebyshev mapping to generate one or two new XOR charts to replace the XOR chart used in AES. Moreover, the authors in [22] created a changing MDS matrix based on the chaos maps to substitute for AES's MDS matrix. However, unfortunately, the results in [21], [22] have several limitations. For instance, the algorithms presented are not clearly defined, and the dynamically generated MDS matrix from the chaos mapping in [22] does not satisfy the properties of an MDS matrix, and so on.

In this paper, we establish the definitions for the matrices B_{had} , N_{had} , and NB_{had} . Following that, we introduce an algorithm to produce key-dependent XOR charts for the creation of a key-dependent AddRoundKey procedure, which relies on these matrices. Incorporating the proposed key-dependent AddRoundKey operation into AES, we develop an advanced AES cipher. We implement this dynamic AES algorithm, perform an in-depth security evaluation and evaluate both the standard AES and our dynamic AES block cipher using NIST's statistical test suite. The dynamic AES algorithm displays enhanced resilience against formidable block cipher attacks compared to conventional AES.

This paper is divided into the following sections: Section 2 provides the basic principles and background information. In section 3, we introduce an algorithm designed to construct a key-dependent AddRoundKey operation utilizing Hadamard matrices, specifically B_{had} , N_{had} , and NB_{had} . Section 4

details the implementation of the advanced AES cipher and presents a comprehensive security analysis. Finally, section 5 offers summary remarks and summarizes our findings.

2. PRELIMINARIES

The AES algorithm's round function consists of four operations: AddRoundKey, SubByte, ShiftRow, and MixColumn. Among these, the AddRoundKey operation involves a bit-level XOR combination of the round key and the input state array for that round. This XOR operation is a standard bitwise XOR. The 4-bit XOR chart for AES is presented in Table 1.

2.1. Definition of the Hadamard matrix

In [23] and [24], A Hadamard matrix is characterized by the following definition:

Definition 1. [23], [24]: Given l elements t_0, t_1, \dots, t_{l-1} , a matrix $V = \text{had}(t_0, t_1, \dots, t_{l-1}) = [h_{i,j}]$ is referred to as a Hadamard matrix when its entries are defined by the relation: $h_{i,j} = t_{i \oplus j}$, for $0 \leq i, j \leq l-1$ and all elements of V are drawn from the field \mathbb{F}_2^s .

A 4×4 Hadamard matrix has the following form:

$$V = \text{had}(t_0, t_1, t_2, t_3) = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 \\ t_1 & t_0 & t_3 & t_2 \\ t_2 & t_3 & t_0 & t_1 \\ t_3 & t_2 & t_1 & t_0 \end{bmatrix} \quad (1)$$

where $h_{i,j} = t_{i \oplus j}$.

2.2. Some new definitions

In this paper, we introduce several new definitions of N_had , B_had , NB_had matrices to facilitate the construction of the key-dependent dynamic XOR chart.

2.2.1. Definition of the N_Had matrix

The symbol for the NXOR operation is \odot , which is the negation operation of the bitwise XOR operation. The output value is 1 if both input values are the same, and the output value is 0 if the input values are different. Based on the NXOR operation, we introduce the concept of an $N_Hadamard$ matrix as follows.

Definition 2. Given l elements $\gamma_0, \gamma_1, \dots, \gamma_{l-1}$, a matrix $A = N_had(\gamma_0, \gamma_1, \dots, \gamma_{l-1}) = [a_{i,j}]$ is called an N_had matrix if each cell in this matrix is formulated as follows: $a_{i,j} = \gamma_{i \odot j}$, $0 \leq i, j \leq l-1$, where the values of A belong to \mathbb{F}_2^s .

A 4×4 N_had matrix has the following form:

$$A = N_had(\gamma_0, \gamma_1, \gamma_2, \gamma_3) = \begin{bmatrix} \gamma_3 & \gamma_2 & \gamma_1 & \gamma_0 \\ \gamma_2 & \gamma_3 & \gamma_0 & \gamma_1 \\ \gamma_1 & \gamma_0 & \gamma_3 & \gamma_2 \\ \gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \end{bmatrix} \quad (2)$$

where $a_{i,j} = \gamma_{i \odot j}$.

2.2.2. Definition of the B_had matrix

We provide a definition for a Hadamard block matrix as follows:

Definition 3. Given 2^d ($d \geq 1$) Hadamard matrices denoted as $had_0, had_1, \dots, had_{2^d-1}$, a matrix:

$$H = B_had(had_0, had_1, \dots, had_{2^d-1}) = \begin{bmatrix} had_0 & had_1 & had_2 & \cdots & had_{2^d-2} & had_{2^d-1} \\ had_1 & had_0 & had_3 & \cdots & had_{2^d-1} & had_{2^d-2} \\ had_2 & had_3 & had_0 & \cdots & had_{2^d-4} & had_{2^d-3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ had_{2^d-2} & had_{2^d-1} & had_{2^d-4} & \cdots & had_0 & had_1 \\ had_{2^d-1} & had_{2^d-2} & had_{2^d-3} & \cdots & had_1 & had_0 \end{bmatrix}$$

is referred to as a Hadamard block matrix, where the elements of H belong to \mathbb{F}_2^s .

A 16×16 B_had matrix has the following form.

$$H = B_had(had_0, had_1, had_2, had_3) = \begin{bmatrix} had_0 & had_1 & had_2 & had_3 \\ had_1 & had_0 & had_3 & had_2 \\ had_2 & had_3 & had_0 & had_1 \\ had_3 & had_2 & had_1 & had_0 \end{bmatrix} \quad (3)$$

where had_i ($0 \leq i \leq 3$) are 4×4 hadamard matrices.

2.2.3. Definition of the NB_had matrix

Similarly, we provide a definition for an N_Hadamard block matrix as follows.

Definition 4. Given 2^d ($d \geq 1$) N_had matrices denoted as $N_had_0, N_had_1, \dots, N_had_{2^d-1}$, a matrix: $H = NB_{had(N_had_0, N_had_1, \dots, N_had_{2^d-1})} =$

$$\begin{bmatrix} N_had_0 & N_had_1 & N_had_2 & \dots & N_had_{2^d-2} & N_had_{2^d-1} \\ N_had_1 & N_had_0 & N_had_3 & \dots & N_had_{2^d-1} & N_had_{2^d-2} \\ N_had_2 & N_had_3 & N_had_0 & \dots & N_had_{2^d-4} & N_had_{2^d-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ N_had_{2^d-2} & N_had_{2^d-1} & N_had_{2^d-4} & \dots & N_had_0 & N_had_1 \\ N_had_{2^d-1} & N_had_{2^d-2} & N_had_{2^d-3} & \dots & N_had_1 & N_had_0 \end{bmatrix}$$

is referred to as a N_Hadamard block matrix, where the elements of H belong to \mathbb{F}_2^s .

A 16×16 NB_had matrix has the following form.

$$H = NB_{had(N_had_0, N_had_1, N_had_2, N_had_3)} = \begin{bmatrix} N_had_0 & N_had_1 & N_had_2 & N_had_3 \\ N_had_1 & N_had_0 & N_had_3 & N_had_2 \\ N_had_2 & N_had_3 & N_had_0 & N_had_1 \\ N_had_3 & N_had_2 & N_had_1 & N_had_0 \end{bmatrix} \quad (4)$$

where N_had_i ($0 \leq i \leq 3$) are 4×4 N_Hadamard matrices.

3. CONSTRUCTING A KEY-DEPENDENT ADDROUNDKEY OPERATION BASED ON HADAMARD MATRICES, B_HAD, N_HAD, AND NB_HAD

First, based on a predefined secret key and leveraging the elegant structure of Hadamard matrices, we introduce the following algorithm to build a new XOR chart with multiple choices from Hadamard matrices, B_had, N_had, and NB_had matrices as defined in Section 2.

Let's consider a block cipher denoted as *CipherA* with a block length of n ($n \geq 128$) bits and a key length of l ($l \geq 128$) bits. Algorithm 1 is generating key-dependent XOR chart based on hadamard matrices. The diagram for Algorithm 1 is showed in Figure 1.

Algorithm 1. Generating key-dependent XOR chart based on hadamard matrices, B_had, N_had, and NB_had

INPUT: A random key K with a length of l ($l \geq 128$) bits.

OUTPUT: A new XOR chart.

Step 1: Extract the initial 16 bits from the key K , if all 16 key bits are zero, then shift right one bit until we get a sequence of 16 key bits with non-zero bits. Let the obtained key bits be: $K_1 = k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15}$

Step 2: Build a 16×16 Hadamard matrix from K_1 . Let this matrix be matrix A , then:

$$A = Had(k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15})$$

Step 3: Let A_{row_i} ($0 \leq i \leq 15$) be row i of matrix A ; Let b_i ($0 \leq i \leq 15$) be the decimal value corresponding to row i of matrix A . Due to the nature of Hadamard matrix, all 16 of these values will be different. We get set $B = \{b_0, b_1, \dots, b_{15}\}$.

Step 4: Arrange the set $B = \{b_0, b_1, \dots, b_{15}\}$ in sequential arrangement of the values of its elements. Take the indices i of the elements b_i in the set B from left to right, obtaining 16 distinct indices. Then, create a new set from these 16 distinct indices, denoted as $\mathcal{R} = \{r_0, r_1, \dots, r_{15}\}$ where $0 \leq r_i \leq 15$.

Step 5: Select the first 2 bits of the key K and denote them as K_2 .

Step 5.1. If they are $K_2 = 00$, then construct the Hadamard matrix as follows

$$\mathcal{H} = had(r_0, r_1, \dots, r_{15})$$

Step 5.2. If they are $K_2 = 01$, then construct the N_Hadamard matrix as follows

$$\mathcal{H} = N_had(r_0, r_1, \dots, r_{15})$$

Step 5.3. If they are $K_2 = 10$, then construct the Hadamard block matrix as follows

$$\mathcal{H} = B_had(H_1, H_0, H_3, H_2)$$

where $H_0 = \text{had}(r_0, r_1, r_2, r_3)$, $H_1 = \text{had}(r_4, r_5, r_6, r_7)$, $H_2 = \text{had}(r_8, r_9, r_{10}, r_{11})$, $H_3 = \text{had}(r_{12}, r_{13}, r_{14}, r_{15})$

Step 5.4. If they are $K_2 = 11$, then construct the $N_Hadamard$ block matrix as follows

$$\mathcal{H} = NB_had(N_H_1, N_H_0, N_H_3, N_H_2)$$

where $N_H_0 = N_had(r_0, r_1, r_2, r_3)$, $N_H_1 = N_had(r_4, r_5, r_6, r_7)$, $N_H_2 = N_had(r_8, r_9, r_{10}, r_{11})$, $N_H_3 = N_had(r_{12}, r_{13}, r_{14}, r_{15})$

Step 6: Create a new XOR chart using the \mathcal{H} matrix derived in step 5 with the following approach:

+ The first row and first column of the XOR chart (labeled as row 0 and column 0) contain entries that match those in the initial row of the \mathcal{H} matrix.

+ The remaining cells within the XOR chart are populated with the corresponding elements from the \mathcal{H} matrix.

Step 7: Perform permutations of the columns and rows of the XOR chart in step 6 so that column 0 and row 0 of this XOR chart have elements in progressive order 0, 1, 2, ..., 15.

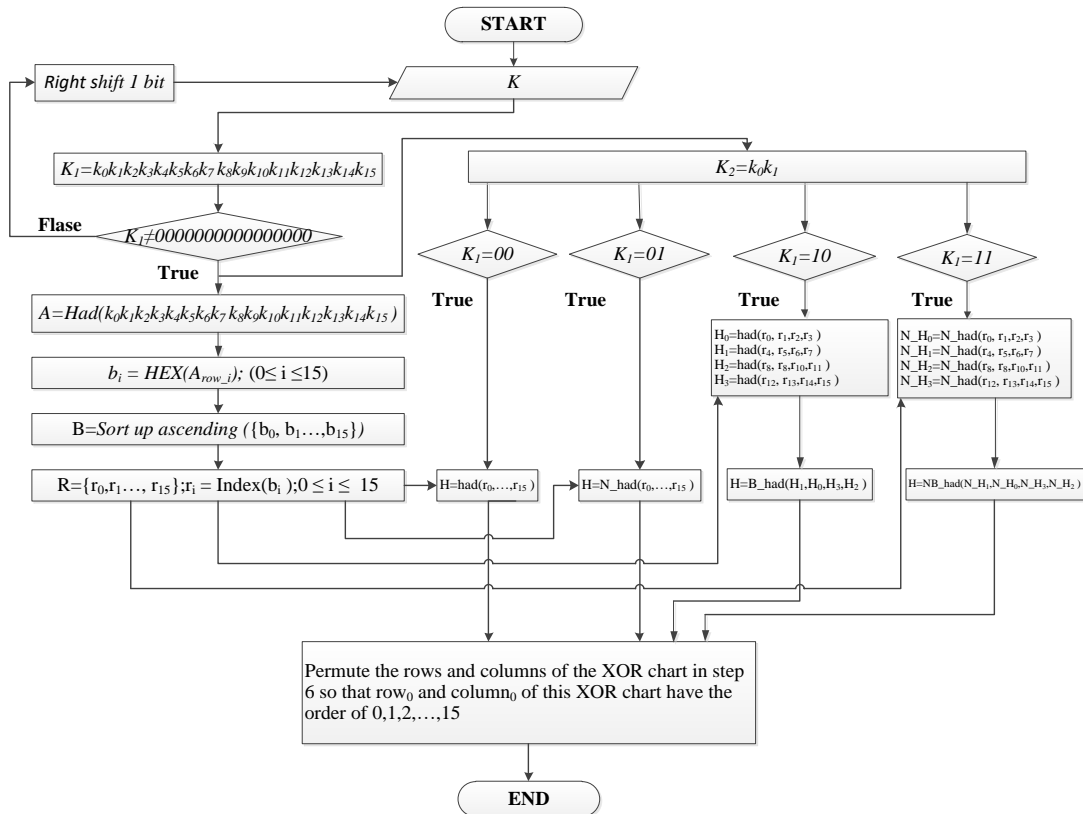


Figure 1. The flowchart of Algorithm 1

Remark 1. Since the construction of the binary Hadamard matrix in step 2 will determine the number of binary Hadamard matrices used to generate the 16 values of the XOR chart. Therefore, with 16 input key bits, it is possible to generate 2^{16} states and correspondingly 2^{16} matrices A and generate 2^{16} sets \mathcal{R} . On the other hand, with four cases of constructing the Hadamard matrix from the set \mathcal{R} in step 5 to create the XOR chart, it is possible to generate: $4 \times 2^{16} = 2^{18}$ different XOR charts from only 16 key bits. **Prove the validity of the XOR chart.**

Derived from the initial XOR chart in AES (as depicted in Table 1), we can make an observation concerning three fundamental Characteristics that an XOR chart should possess.

Three fundamental Characteristics of an 4-bit XOR chart

Characteristic 1: In the case of any u, v , and w elements in the XOR chart satisfying $u \text{ XOR } v = w$, the following assertion is true: $u \text{ XOR } w = v$ and $v \text{ XOR } w = u$.

Characteristic 2: The XOR chart should manifest reflection along the primary diagonal, signifying that $u \text{ XOR } v = v \text{ XOR } u$.

Characteristic 3: Every row and column in the XOR chart includes unique values spanning from 0 to 15. Based on the three characteristics for an XOR chart, we can establish the validity of the newly created XOR chart by Algorithm 1 through Proposition 1.

Proposition 1. The XOR chart recently produced through Algorithm 1 conforms to the fundamental characteristics of an XOR chart.

Proof. From (2), it can be seen that the matrix N_had has the form of a Hadamard matrix, therefore, from (4), it follows that the matrix NB_had is also a Hadamard matrix. From (3), we can see that the matrix B_had is also a Hadamard matrix. Therefore, the matrix \mathcal{H} generated in Step 5 of Algorithm 1 is a Hadamard matrix.

From the construction of the XOR chart in step 6 of Algorithm 1 and the form (1) of the Hadamard matrix, we have:

- The updated XOR chart exhibits symmetry along its main diagonal, so *characteristic 2* of the updated XOR diagram is fulfilled.
- The elements in matrix \mathcal{H} consist of distinct elements r_i satisfying $0 \leq r_i \leq 15$. Therefore, *characteristic 3* of the new XOR chart is satisfied.
- Because column 0 and row 0 of the new XOR chart consist of elements identical to those in the first row of matrix \mathcal{H} , while the elements inside the XOR chart are elements of the Hadamard matrix \mathcal{H} , essentially, the new XOR chart is created by replacing elements from the original XOR chart (Table 1) with elements from matrix \mathcal{H} .

Assuming $\mathcal{H} = had(h_0, h_1, \dots, h_{15})$, the new XOR chart is created by replacing 0 with h_0 , 1 with h_1 , ..., and 15 with h_{15} for the entire original XOR chart, including both column 0 and row 0 of the original XOR chart. Therefore, for the original XOR chart, we have:

If $u \text{ XOR } v = w$ then $u \text{ XOR } w = v$ and $v \text{ XOR } w = u$.

In that case, we also have:

If $h_u \text{ XOR } h_v = h_w$ then $h_u \text{ XOR } h_w = h_v$ and $h_v \text{ XOR } h_w = h_u$.

Thus, the updated XOR chart maintains the essential properties required for an XOR chart.

On the other hand, rearranging the columns and rows of the XOR chart in step 7 has no impact on the values within the XOR chart. As a result, the XOR chart produced by Algorithm 1 adheres to all three characteristics of an XOR chart. Example: assuming the confidential key K is 128 bits long in hexadecimal format as follows :

$K = 0x2B7E151628AED2A6ABF7158809CF4F3C$

Performing Algorithm 1 with the key K , we derive the new key-dependent XOR chart, as illustrated in Figure 1.

Based on the results of Algorithm 1, we integrate it into the AES to dynamically modify the AddRoundKey operation by utilizing the new XOR chart created by Algorithm 1. This procedure is detailed in Algorithm 2. In practical use, the sending and receiving parties must share a confidential key, K , beforehand. Table 2 shows the newly generated key-dependent XOR chart produced by Algorithm 1.

Algorithm 2. Generating key-dependent AES with dynamic AddRoundKey transformation

INPUT: A random key K with a length of l ($l \geq 128$) bits, AES block cipher with the original XOR chart A (Figure 1).

OUTPUT: A key-dependent AES block cipher ($AESHD$).

Step 1: Apply Algorithm 1 to obtain a new XOR chart \hat{A} .

Step 2: Substitute the original XOR chart A in AES with the updated XOR chart \hat{A} .

Step 3: Replace the original AddRoundKey operation of AES with the key-dependent AddRoundKey using the new XOR chart \hat{A} . All other operations in AES remain unchanged.

Return $AESHD$;

Table 1. The original XOR chart of 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

Table 2. The newly generated key-dependent XOR chart produced by Algorithm 1

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

4. IMPLEMENTING THE DYNAMIC AESHD BLOCK CIPHER AND CONDUCTING A SECURITY ANALYSIS

We implement the dynamic AESHD algorithm using the newly generated key-dependent XOR chart, following the steps outlined in Algorithm 1 and Algorithm 2. Following this, we analyze the safety considerations of the AESHD algorithm and examine the statistical benchmarks of this adaptive block cipher. We execute the AESHD algorithm on a Windows 10 64-bit system equipped with an Intel Core i5 2430M CPU (Operating at 2500MHz, 2x2.4GHz), Intel HD Graphics 3000, 2GB DDR3 1333MHz RAM.

4.1. Security analysis

At present, there exist various categories of assaults against block ciphers, with the twin formidable assaults being differential and linear cryptanalysis. As for the differential cryptanalysis [1], [4] it falls within the category of preferred plaintext attacks, necessitating the procurement of an extensive assemblage of cipher text results originating from specifically chosen input data. The linear assault is a breed of offensive strategy unveiled by Matsui [6]. The linear cryptanalysis [1], [5], [6] is a recognized entry maneuver, demanding the acquisition of an extensive assortment of plaintext-ciphertext pairings, aligning with a concealed key under investigation.

We observe that, with the AddRoundKey operation using the regular XOR chart of AES, this XOR operation is simply a bitwise XOR of 4-bit groups in the binary field. Since the AddRoundKey operation with this XOR chart is public, and all components of AES are public, when attackers can collect a substantial number of plaintext/ciphertext pairs (for instance, as seen in the DES algorithm, where at least 2^{47} pairs are needed), they can perform linear cryptanalysis and differential cryptanalysis attacks on the AES block cipher. However, when we make the AddRoundKey transformation dynamic based on a secret key using Algorithm 1, cryptanalysts will not be able to discern how we perform the XOR operations within the AESHD algorithm. This is because the new XOR chart depends on the given secret key. For the AESHD block cipher, if cryptanalysts aim to carry out differential or linear cryptanalysis, they are compelled to identify the dynamic XOR chart in use. As per Remark 1, the number of new XOR charts created by Algorithm 1 can be as high as 2^{18} , which means that cryptanalysts would have to try any one of these XOR charts. With a candidate XOR chart, attackers must have a substantial number of plaintext/ciphertext pairs before proceeding to uncover differential or linear patterns to find the secret key.

Hence, by introducing dynamism into the AddRoundKey operation and the XOR charts, we have notably augmented the challenge faced by cryptanalysts in their analytical endeavors when compared to the conventional AES method. To illustrate, when cryptanalysts engage in linear/differential cryptanalysis on the AES block cipher, they require 2^{95} pairs of plaintext-ciphertext. On the other hand, conducting cryptanalysis on AESHD necessitates an astounding $2^{95} \times 2^{18}$ pairs of plaintext-ciphertext. This vast number is undeniably insurmountable, rendering it virtually unattainable for cryptanalysts to amass such an extensive dataset. Consequently, AESHD substantially escalates both the temporal and data complexities in cryptographic analysis relative to the standard AES. Hence, the integration of the key-dependent AddRoundKey transformation can notably fortify the strength of the AES.

Thus, it is evident that by integrating the key-reliant dynamic AddRoundKey layer, the AESHD block cipher gains a higher level of unpredictability, significantly strengthening the resilience of the AES. Compare our results with those in [21], [22]. In Table 3, we provide some comparisons of our key-dependent XOR chart construction method with the approaches presented in [21], [22].

Table 3. Comparison of our proposal and results in [21], [22]

Criterion	Our method	Methods in [21], [22].
Dynamic method Process for designing a key-reliant XOR representation	Secret key dependence - Derived from permutation and the form of the Hadamard matrix, Present clearly	Secret parameter dependence - Based on Chaotic mappings - Unclear presentation, still many gaps.
The number of XOR chart	2^{18}	1 or 2
Demonstration of the correctness of the XOR chart	Yes	No
Security Analysis	Yes	No
Evaluate statistical randomness criteria	Yes	Yes

4.2. Randomness evaluation according to NIST statistical standards

In this section, we assess the NIST statistical benchmarks [25] for the two AES and the adaptive AESHD cipher. The results for AESHD are presented in Tables 3. Where Table 3 represents the P-value according to the NIST standards, and Table 4 shows the percentage of sequences meeting the requirements of the tests of NIST.

Table 4. Evaluation outcomes of Level-2 p-values for the AESHD using examinations for brief sequences

Number of Rounds	Frequency Test	Runs Test	Evaluation of the maximum run of ones	Serial Test	AppEn. Test	CuSum. Test	Bit AutoCorr. Test	Byte Autocorr. Test
AV1 Input Data								
1	0.000000	0.000002	0.000000	0.000167	0.002642	0.000000	0.000000	0.000000
2	0.839118	0.516543	0.000000	0.010751	0.041082	0.000000	0.000000	0.005369
3	0.380491	0.177322	0.215230	0.894644	0.788543	0.069959	0.138532	0.546674
4	0.892658	0.990783	0.966658	0.739805	0.716736	0.195910	0.474348	0.609771
5	0.698068	0.662030	0.323675	0.678460	0.734859	0.072978	0.700746	0.637956
6	0.119203	0.445908	0.154656	0.834388	0.250477	0.403447	0.285786	0.733883
7	0.813397	0.727084	0.273645	0.968229	0.846125	0.486678	0.727771	0.883852
8	0.821386	0.663897	0.714845	0.130298	0.632307	0.794124	0.092452	0.894300
9	0.698068	0.662030	0.323675	0.678460	0.734859	0.072978	0.700746	0.637956
10	0.646300	0.629920	0.107248	0.061537	0.175298	0.577306	0.738451	0.224640
HW Input Data								
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.178163	0.688721	0.807376	0.203180	0.371836	0.440107	0.065749	0.836290
4	0.581894	0.011147	0.604585	0.681054	0.590476	0.679676	0.007101	0.881369
5	0.535947	0.179983	0.867622	0.446410	0.294630	0.384835	0.337331	0.328601
6	0.923017	0.253842	0.455873	0.585449	0.918323	0.787686	0.224398	0.373549
7	0.547715	0.215657	0.468932	0.657826	0.379454	0.618625	0.010979	0.958272
8	0.626316	0.545824	0.008063	0.902931	0.723317	0.216293	0.750984	0.838468
9	0.762578	0.344255	0.292304	0.261295	0.424222	0.847349	0.403913	0.545831
10	0.857050	0.144153	0.037715	0.197433	0.445099	0.700560	0.596591	0.484707
LW Input Data								
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.706153	0.809753	0.832735	0.371203	0.280577	0.989641	0.283016	0.771438
4	0.376333	0.940666	0.178621	0.634937	0.610670	0.421563	0.341414	0.694258
5	0.917682	0.970431	0.989177	0.881099	0.995119	0.966098	0.906470	0.986374
6	0.958790	0.795450	0.822314	0.790310	0.664333	0.134939	0.670924	0.775430
7	0.152501	0.760392	0.105816	0.924372	0.909478	0.990341	0.916565	0.753758
8	0.703553	0.088458	0.070614	0.162333	0.145893	0.135520	0.204071	0.997179
9	0.194626	0.339262	0.302658	0.054058	0.075663	0.040609	0.894864	0.301139
10	0.332898	0.360915	0.574232	0.598514	0.361494	0.650666	0.109974	0.586361
Rot Input Data								
1	0.000000	0.000003	0.007434	0.000000	0.000000	0.000002	0.000932	0.493142
2	0.141308	0.694600	0.999034	0.407917	0.598432	0.991088	0.242368	0.181808
3	0.013409	0.832160	0.719537	0.857320	0.697085	0.335150	0.836450	0.450856
4	0.136200	0.484708	0.497188	0.328082	0.239615	0.010704	0.800716	0.429596
5	0.251520	0.894110	0.995275	0.947471	0.945356	0.934738	0.581689	0.016082
6	0.968448	0.351803	0.837870	0.774738	0.558743	0.288553	0.684007	0.198765
7	0.821181	0.462576	0.529957	0.893864	0.694646	0.723061	0.715656	0.216957
8	0.037583	0.053786	0.924451	0.088295	0.231525	0.580492	0.076205	0.195261
9	0.052261	0.677056	0.236238	0.479375	0.387537	0.249608	0.317436	0.754972
10	0.111908	0.364044	0.848878	0.040609	0.019562	0.032572	0.816783	0.039949

Through Table 4, we observe that when using a random key with four input datasets : AV1, HW, LW, ROT, which are non-random datasets, after 3 rounds of encryption with the dynamic AESHD algorithm, all achieve the randomness standards. Specifically, for the ROT dataset, it only requires 2 rounds to meet the NIST randomness standards. In general, to achieve randomness when using a key created randomly, the AESHD block cipher demands a minimum of three rounds, similar to AES, as observed in our analysis. Subsequent to the evaluation, it is evident that the AESHD block accomplishes an output randomness equivalent to AES, with the added benefit of heightened security compared to AES.

5. CONCLUSION

In this paper, we provide the definitions for B_had, N_had, and NB_had matrices. Following that, we describe an approach to generate key-dependent XOR charts to establish a key-dependent AddRoundKey operation utilizing these matrices. Subsequently, we devise the dynamic AESHD block cipher by integrating the suggested key-dependent AddRoundKey operation into AES. We execute the AESHD algorithm, appraise its security, and gauge AES alongside the novel dynamic AESHD block cipher against the statistical benchmarks outlined by NIST. The dynamic AESHD algorithm demonstrates enhanced resilience against formidable block cipher attacks in comparison to traditional AES. For future research, we will persist in exploring diverse dynamic approaches and novel amalgamations to fortify the strength of block ciphers.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Truong Minh Phuong	✓	✓	✓			✓			✓					
Tran Thi Luong		✓			✓	✓		✓		✓	✓			

C : Conceptualization	I : Investigation	Vi : Visualization
M : Methodology	R : Resources	Su : Supervision
So : Software	D : Data Curation	P : Project administration
Va : Validation	O : Writing - Original Draft	Fu : Funding acquisition
Fo : Formal analysis	E : Writing - Review & Editing	

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

ETHICAL APPROVAL

The research related to human use has been complied with all the relevant national regulations and institutional policies in accordance with the tenets of the Helsinki Declaration and has been approved by the authors' institutional review board or equivalent committee

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

[1] H. M. Keys and S. E. Tavares, "Substitution-permutation networks resistant to differential and linear cryptanalysis," *Journal of Cryptology*, vol. 9, no. 1, pp. 1–19, Mar. 1996, doi: 10.1007/bf02254789.

[2] J. Daemen and V. Rijmen, "Rijndael/AES," in *Encyclopedia of Cryptography and Security*, Springer US, 2006, pp. 520–524.

[3] J. Daemen and V. Rijmen, *The design of Rijndael: {AES}-the Advanced Encryption Standard*. Springer Berlin Heidelberg, 2002.

[4] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptology*, vol. 4, no. 1, pp. 3–72, Jan. 1991, doi: 10.1007/BF00630563.




[5] S. Mister and C. Adams, "Practical S-box design, in workshop on selected areas in cryptography (SAC)," pp. 61–76, 1996.

[6] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 765 LNCS, 1994, pp. 386–397.




- [7] L. R. Knudsen and C. V. Miolane, "Counting equations in algebraic attacks on block ciphers," *International Journal of Information Security*, vol. 9, no. 2, pp. 127–135, Dec. 2010, doi: 10.1007/s10207-009-0099-9.
- [8] M. Bros, "Algebraic cryptanalysis and contributions to post-quantum cryptography based on error-correcting codes in the rank-metric," *Doctoral dissertation, Université de Limoges*, 2022.
- [9] D. James and T. L. Priya, "An innovative approach for dynamic key dependent S-Box to enhance security of IoT systems," *Measurement: Sensors*, vol. 30, p. 100923, Dec. 2023, doi: 10.1016/j.measen.2023.100923.
- [10] A. Y. Al-Dweik, I. Hussain, M. Saleh, and M. T. Mustafa, "A novel method to generate key-dependent s-boxes with identical algebraic properties," *Journal of Information Security and Applications*, vol. 64, p. 103065, Feb. 2022, doi: 10.1016/j.jisa.2021.103065.
- [11] H. T. Assafl and I. A. Hashim, "Generation and evaluation of a new time-dependent dynamic S-Box algorithm for AES block cipher cryptosystems," *IOP Conference Series: Materials Science and Engineering*, vol. 978, no. 1, p. 12042, Dec. 2020, doi: 10.1088/1757-899X/978/1/012042.
- [12] A. Ejaz, I. A. Shoukat, U. Iqbal, A. Rauf, and A. Kanwal, "A secure key dependent dynamic substitution method for symmetric cryptosystems," *PeerJ Computer Science*, vol. 7, pp. 1–29, Jul. 2021, doi: 10.7717/PEERJ-CS.587.
- [13] M. R. M. Shamsabad and S. M. Dehnavi, "Dynamic MDS diffusion layers with efficient software implementation," *International Journal of Applied Cryptography*, vol. 4, no. 1, p. 36, 2020, doi: 10.1504/ijact.2020.10029198.
- [14] A. H. Al-Wattar, R. Mahmood, Z. A. Zukarnain, and N. Udzir, "A New DNA Based Approach of Generating Key Dependent Mix Columns Transformation," *International journal of Computer Networks and Communications*, vol. 7, no. 2, pp. 93–102, Mar. 2015, doi: 10.5121/ijcnc.2015.7208.
- [15] and M. M. I. Ismail, G. Galal-Edeen, S. Khattab, "Performance examination of AES encryption algorithm with constant and dynamic rotation," *International Journal of Reviews in Computing*, vol. 12, 2012.
- [16] W. Chen, L. Li, and Y. Guo, "DABC: A dynamic ARX-based lightweight block cipher with high diffusion," *KSII Transactions on Internet and Information Systems*, vol. 17, no. 1, pp. 165–184, Jan. 2023, doi: 10.3837/tiis.2023.01.009.
- [17] T. M. Kumar and P. Karthigaikumar, "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 (Singapore)*, vol. 12, no. 3, pp. 825–830, May 2020, doi: 10.1007/s41870-020-00465-1.
- [18] T. Xu, F. Liu, and C. Wu, "A white-box AES-like implementation based on key-dependent substitution-linear transformations," *Multimedia Tools and Applications*, vol. 77, no. 14, pp. 18117–18137, Mar. 2018, doi: 10.1007/s11042-017-4562-8.
- [19] A. A. Hambouz, "DLL-AES: dynamic layers lightweight AES Algorithm," *Doctoral dissertation, Princess Sumaya University for Technology (Jordan)*, 2022.
- [20] A. A. Hambouz, DLL-AES: Dynamic Layers Lightweight AES Algorithm (Doctoral dissertation, Princess Sumaya University for Technology (Jordan)), 2022.
- [21] A. I. Salih, A. Alabaichi, and A. S. Abbas, "A novel approach for enhancing security of advanced encryption standard using private XOR table and 3D chaotic regarding to software quality factor," *International Journal of Research and Surveys*, vol. 10, no. 9, pp. 823–832, 2019.
- [22] A. I. Salih, A. Alabaichi, and A. Y. Tuama, "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, Sep. 2020, doi: 10.11591/ijeecs.v19.i3.pp1574-1581.
- [23] R. Elumalai and A. R. Reddy, "Improving diffusion power of AES Rijndael with 8x8 MDS matrix," *International Journal of Scientific and Engineering Research*, vol. 2, no. 3, 2011.
- [24] M. Sajadieh, M. Dakhilalian, H. Mala, and B. Omoomi, "On construction of involutory MDS matrices from Vandermonde Matrices in GF(2^q)," *Designs, Codes, and Cryptography*, vol. 64, no. 3, pp. 287–308, Nov. 2012, doi: 10.1007/s10623-011-9578-x.
- [25] A. Rukhin, J. Soto, and J. Nechvatal, *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, vol. 22, no. April. 2010.

BIOGRAPHIES OF AUTHORS



Truong Minh Phuong    has master's degree in cryptography technique since 2018 and is currently a PhD student in this major at the Academy of Cryptography Techniques, Ha Noi, Viet Nam. His research direction is secret key cryptography, in which dynamic block ciphers are currently an important component to build his thesis. He can be contacted at email: minhphuongh19@gmail.com.



Tran Thi Luong    received her Bachelor's degree from Hanoi University of Science in 2006, and Master's (2012) and Ph.D. (2019) degrees in cryptographic techniques from the Academy of Cryptography Techniques, Hanoi, Vietnam. She has led and contributed to multiple research projects on cryptographic primitives and block cipher design. She served as a co-chair for KSE 2023 and a reviewer for PeerJ Computer Science. Her research interests include cryptography, coding theory, and information security. She can be contacted at email: luongtran@actvn.edu.vn.