Proposals and Implementations of MDS Diffusion Layer Dynamic Algorithms for AES Block Cipher

Tran Thi Luong
Academy of Cryptography Techniques, No.141 Chien Thang road, Tan Trieu, Thanh Tri, Hanoi, Vietnam

Correspondence: Tran Thi Luong, luongtranhong@gmail.com
Communication: received 31 Jan 2023, revised 04 Apr 2023, accepted 26 Jun 2023
DOI: 10.32913/mic-ict-research.v2023.n2.1194

***Abstract*: Block ciphers are an interesting cryptographic topic that is widely used today. Therefore, the issue of improving the security of block ciphers is increasingly concerned today. There have been many research directions to animate block ciphers to improve their security against well-known strong attacks. In [1], we proposed two diffusion layer dynamic algorithms for SPN block ciphers. In this paper, we will execute these two algorithms with recursive MDS matrices of size 4, 8, and 16, then evaluate the cryptographic properties of the obtained dynamic MDS matrices including branch number, number of fixed points, coefficient of fixed points, number of XOR and Xtime operations. Then, we apply these two dynamic algorithms to dynamically modify the AES block cipher and evaluate the execution speed of the dynamic AES block ciphers. We propose two new diffusion layer dynamic algorithms, implement these two algorithms and modify dynamically AES block cipher according to these two algorithms, and evaluate the speed of dynamic AES block ciphers. We also evaluate the security and implementation resources for dynamically modified AES block ciphers. The proposed diffusion layer dynamic algorithms contribute to improving the security of the AES block cipher against many current strong attacks.**

Keywords: *MDS matrix, AES, dynamic diffusion layer.*

I. INTRODUCTION

Block ciphers are an area of special interest in cryptography. Proof of this is that the US opened two block cipher standard selection contests in two consecutive periods. The Russian Federation developed their own block cipher standard very early, the second time recently is the GOST R34.12-2015 block cipher standard [2-4]. Since NIST's AES [5-7] selection competition in 2000, the SPN block cipher [8-10] has received more and more research attention.

Along with the research work on SPN block ciphers, there have been many works focused on the highly secure direction for these block ciphers. That's why methods for making block ciphers dynamic were born. The most basic way of animating a block cipher is usually based on a given secret key. With the SPN block cipher, there are many ways of animating its components: substitution layer dynamics, diffusion layer dynamics, both substitution and diffusion layer dynamics, and several other types of dynamics. In the SPN block cipher dynamic at the substitution layer, dynamic substitution boxes (S-boxes) are made depending on a secret key used by the block cipher. The first studies of this type must include the studies of B. Schneier et al. in [11, 12]. Only the improvement of AES in this direction has had a lot of research work, for example in [13-20]. These works focused on constructing dynamic S-boxes that depend on a given secret key, then the authors used the found dynamic S-boxes to replace the original AES S-box, generating dynamic AES block ciphers.

The inclusion of dynamic S-boxes in block ciphers has been applied to many block ciphers today. However, the generation of a dynamic MDS matrix to create a dynamic diffusion layer for block ciphers is still an open research problem and attracts many cryptographers today. In this dynamic method, the SPN block cipher is animated at the diffusion layer. With this research direction, there are a number of related studies, such as in [1, 21, 22, 23, 24]. In [21], the authors used a dynamic MDS matrix based on scalar multiplication in the rows of the MDS matrix of AES and a complement key of m-bit. However, in [21] only studied introducing a 4×4 MDS matrix in to the Mixcolumn transformation of AES, but not studied for larger MDS matrices. In [22], the authors proposed a dynamic SPN block cipher based on AES, denoted DRAES, where the animated transformation is a rotation whose amount of rotation depends on the data (plaintext and ciphertext) in AddRoundKey and depends on the key in

the AES key scheme. However, in [22], the authors did not focus on MDS matrix and making AES dynamic based on Mixcolumn transformation. In [23], the authors proposed to use a dynamic MixColumn transformation based on key-dependent DNA structures and processes. However, the authors only studied to create dynamic 4×4 MDS matrices from the original AES matrix, but not for the larger MDS matrices. In [24], the authors created a private XOR table that depends on the 3D chaos mapping and an original secret key. The authors also used the dynamic MixColumn transformation for AES with a 4×4 dynamic MDS matrix generated using a 3D chaos mapping to permute the AES 4×4 MDS matrix. Unfortunately, their dynamic MDS matrix is not an MDS matrix.

In [1], we proposed two diffusion layer dynamic algorithms for SPN block ciphers in general, and these algorithms can be applied to MDS matrices of any size. However, we have not yet implemented these algorithms and have not applied them to a specific SPN block cipher. In [25], we studied the use of MDS matrices of the original SPN block cipher to efficiently compute the outputs of dynamic MDS matrices without directly calculating the dynamic MDS matrices. Dynamic MDS matrices are generated from the original MDS matrix using direct exponent and scalar multiplication transformations. However, we just came up with the idea of taking advantage of the original MDS matrices without any specific implementation. In this paper, we will execute the two dynamic diffusion layer algorithms proposed in [1] with recursive MDS matrices of size 4, 8, and 16 then evaluate the cryptographic properties of the obtained dynamic MDS matrices including branch number, number of fixed points, coefficient of fixed points, number of XOR and Xtime operations. Then, we apply these two dynamic algorithms to dynamically modify the AES block cipher and evaluate the execution speed of the dynamic AES block ciphers. We propose two new diffusion layer dynamic algorithms, implement these two algorithms and modify dynamically AES block cipher according to these two algorithms, and evaluate the speed of dynamic AES block ciphers. We also evaluate the security and implementation resources for dynamically modified AES block ciphers.

Thus, in [1], we just introduced the algorithms but have not implemented these algorithms to obtain specific dynamic MDS matrices, so it is impossible to evaluate the cryptographic properties of obtained dynamic MDS matrices. This is important for the security as well as the execution speed of future block ciphers. In addition, in [1] and [25], we have not applied the proposed algorithms to the AES block cipher modification. Modifying the AES block cipher in a way that animates the diffusion layer is a meaningful way to improve the security of AES against many current attacks such as linear attacks, differential attacks.

This article is organized as follows. In Section 2, we present some related works. Section 3 proposes new diffusion layer dynamic algorithms. Section 4 implements dynamic algorithms and applies them to the dynamic modification of AES. Section 5 gives security analysis and implementation resources evaluation for modified AES block ciphers. Section 6 is the conclusion.

II. RELATED WORKS

1. Direct exponentiation and scalar multiplication transformation

Direct exponentiation and scalar multiplication of MDS matrices were first given in [26]. However, in [27, 28, 29, 30], we presented many useful results about the two transformations. Furthermore, we also extend the concept of scalar multiplication to the MDS matrix.

Direct exponentiation with exponent e of a matrix $A = \left[a_{i,j}\right]_{m\times m}$, $a_{i,j} \in GF\left(p^r\right)$ where p is prime, r>=1, r is an integer, denoted A_{d^e} , where each of its elements is the result of the power e of the corresponding elements in A.

Scalar multiplication is defined as follows [28]: For $E = (e_1, e_2, \ldots, e_m)$, $F = (f_1, f_2, \ldots, f_m)$, (for $e_i f_i \in GF^*\left(p^r\right)$) and for $A = \left[a_{i,j}\right]_{m\times m}$, $a_{i,j} \in GF\left(p^r\right)$. We say that the product of (E, F) and A is a matrix denoted by $(E, F)(A) = \left[b_{i,j}\right]_{m\times m}$ determined by the following formula: $b_{i,j} = e_i f_j a_{i,j}$.

2. Take advantage of MDS matrices of the original SPN block cipher [26]

With dynamic MDS matrices generated from direct exponentiation and scalar multiplication, in [26], we studied an efficient method to compute the outputs of dynamic MDS matrices based on the original MDS matrices when its inputs are known, as well as the efficient calculation of the inputs of the dynamic MDS matrices based on the original MDS matrices when the outputs are known. That is, we can take advantage of the MDS matrix of the original SPN block cipher to compute the input and output of the dynamic MDS matrices of the dynamic SPN block cipher without having to specifically compute the dynamic MDS matrix.

Suppose both Alice and Bob have two initial MDS matrices, $A = \left[a_{i,j}\right]_{m\times m}$, $a_{i,j} \in GF\left(p^r\right)$ and $B = \left[b_{i,j}\right]_{m\times m}$, $b_{i,j} \in GF\left(p^r\right)$ $\left(B = A^{-1}\right)$. In [26], we make the following important remarks:

For direct exponentiation

With the secret parameter $k\ (1 \le k \le r)$, two dynamic MDS matrices are generated for encryption and decryption at the dynamic diffusion layer, denoted A_p and B_p (where $A_p = A_{dp^k}$ and $B_p = A_{p^{-1}}$).

Remark 1 [26]. Given any input vector $X \in (GF(p^r))^m$, Alice does not need to compute the matrix A_p and does not need to compute the output according to the formula $Y = A_p X$. Alice just needs to do presently:

- *Step 1: Calculate* $Z = X^{p^{r-k}}$.
- *Step 2: Calculate* $Y_1 = AZ$.
- *Step 3: Calculate* $Y = Y_1^{p^k}$ *and send* Y *to Bob.*

Remark 2 [26]. When Bob receives an encrypted vector $Y \in (GF(p^r))^m$, Bob does not need to compute the matrices A_p and B_p and also does not need to compute the plaintext using the formula $X = B_p Y$. Bob just need to do:

- *Step 1: Calculate* $Z' = Y^{p^{r-k}}$.
- *Step 2: Calculate* $Z = BZ'$.
- *Step 3: Calculate* $X = Z^{p^k}$.

For scalar multiplication

Alice and Bob agree on secret parameters $E = (e_0, e_1, \dots e_{m-1})$ and $F = (1, 1, \dots, 1)$. Then, two new dynamic MDS matrices are created for the encryption and decryption process at the dynamic diffusion layer, denoted A_M and B_M where $B_M = {A_M}^{-1}$.

Remark 3 [26]. Given any input vecto $X \in (GF(p^r))^m$, Alice does not need to compute the matrix A_M and does not need to compute the output according to the formula $Y = A_M X$. Alice just needs to do presently:

- *Step 1: Calculate* $Z = AX$.
- *Step 2: Calculate* $Y = (e_1 Z_1, e_2 Z_2, \dots, e_m Z_m)$ *and send* Y *to Bob.*

Remark 4 [26]. When Bob receives an encrypted vector $Y \in (GF(p^r))^m$, Bob does not need to compute the matrices A_M and B_M and also does not need to compute the plaintext using the formula $X = B_M Y$. Bob just need to do:

- *Step 1: Calculate* $Z = (Z_1, Z_2, \dots, Z_m)$ *where* $Z_i = {e_i}^{-1} Y_i$, $i = 1, 2, \dots, m$.
- *Step 2: Calculate* $X = BZ$.

Thus, in either case using direct exponentiation or scalar multiplication, neither Alice nor Bob need to compute the dynamic MDS matrices directly for encryption and decryption, but can take advantage of the original MDS matrices (A and B).

3. Diffusion layer dynamic algorithms [1]

In [1], we proposed two diffusion layer dynamic algorithms (Algorithm 1 and Algorithm 2) for SPN block ciphers based on two transformations namely direct exponentiation and scalar multiplication. Algorithm 1 generates a dynamic MDS matrix from an initial MDS matrix and a secret key. This dynamic matrix will be used for every round of the SPN block cipher. Algorithm 2 generates two dynamic MDS matrices from two initial MDS matrices and a secret key. These two dynamic matrices will be used alternately in the rounds of the SPN block cipher. In [1], we also provide a detailed analysis of improving the security of dynamic SPN block ciphers when using these dynamic algorithms. For more details see [1].

III. PROPOSING NEW DIFFUSION LAYER DYNAMIC ALGORITHMS

From the two diffusion layer dynamic algorithms proposed in [1], combined with the results (Remarks 1, 2, 3, 4) given in [26] about being able to take advantage of initial MDS matrices of the original SPN block cipher without having to compute the dynamic MDS matrices, in this section we propose two new diffusion layer dynamic algorithms (Algorithm 1*, Algorithm 2*) based on the idea of two Algorithms 1 and Algorithm 2 but apply the above results.

Algorithm 1* assumes that there are a given MDS matrix M and a secret key Key, the dynamic MDS matrix can be generated from the original matrix M , a secret key *Key*, and the direct exponential transformation (T_D) and the scalar multiplication transformation (T_S) . This dynamic MDS matrix will be used in the diffusion layer for every round of the block cipher. However, instead of directly calculating the dynamic MDS matrix, the algorithm below takes advantage of the original MDS matrix M as shown in [26]. Therefore, the dynamic algorithm below will only need to compute the secret parameters for the dynamic MDS matrix without having to specifically compute this dynamic MDS matrix.

algorithm 1*

Input: A secret key *Key*, a MDS matrix M of size $m \times m$ over $GF(2^r)$.

Output: A parameter $l\ (0 \le l \le r-1)$ or a secret vector $E = (e_0, e_1, \dots, e_{m-1})$ for $e_i \in GF(2^r)$.

Step 1: Take the first bit of the key *Key* to determine which matrix transformation is used in this algorithm:

- If it is 1, choose the transformation as.
- If it is 0, choose the transformation as .

Step 2:

If the result of step 1 is T_D *then do*:

Step 2.1. Take the next r bits of Key (and a is the corresponding decimal number). Calculate l=a mod r.

If the result of step 1 is T_S then do:

Step 2.2. Take consecutive non-zero bit segments (each segment with bits) from the key Key (i.e. if there is a segment where r bits are all zeros, shift right one bit until you get the segment with non-zero bits . Take m segments like that. Convert those *r-bit* strings to a non-zero element $e_i \in GF(2^r)$, $0 \le i \le m-1$.

Choose two vectors for scalar multiplication as $E = (e_0, e_1, \ldots, e_{m-1})$ and $F = (1, 1, \ldots, 1)$.

Encryption and decryption process

Both parties involved in the secret communication process will make the algorithm's parameters public and private as follows:

Public: Encryption algorithm, initial MDS matrix M (of size), and the dynamic diffusion algorithm (Algorithm 1*).

Private: The secret key Key.

The two communication parties need to implement Algorithm 1* and agree on a method to utilize the MDS matrices of the original SPN block cipher to find the secret parameters or for , and they don't need to calculate the dynamic MDS matrix M_K (where $M_K = M_{dp^k}$ hoặc $M_K = (E, F)\, M$). They can take advantage of the original MDS matrices M and M^{-1} to perform encryption and decryption at the diffusion layer (see Remarks 1, 2, 3, 4).

Algorithm 2*. In this dynamic algorithm, it is assumed that there are two given initial MDS matrices A and B and a secret key *Key*. Dynamic MDS matrices can be generated from such matrices by direct exponentiation or scalar multiplication and secret key *Key*. These dynamic MDS matrices will be used for the diffusion layer alternately in the rounds of the block cipher. However, instead of correctly calculating new key-dependent dynamic MDS matrices, we will make use of the original MDS matrices (A , A^{-1} and B , B^{-1}) for the encryption and decryption process of the diffusion layer.

Algorithm 2*

Input: A secret key *Key*, two MDS matrixs A, B of size $m \times m$ over $GF(2^r)$.

Output: Secret parameters l_1, l_2 $(0 \le l \le r-1)$ or secret vector $E = (e_0, e_1, \ldots, e_{m-1})$ and $E' = (e'_0, e'_1, ..., e'_{m-1})$ for $e_i, e'_i \in GF(2^r)$; Or secret parameters l $(0 \le l \le r-1)$ and $E = (e_0, e_1, \ldots, e_{m-1})$ for $e_i \in GF(2^r)$

Step 1: Find the first two bits of the key *Key* to determine which matrix transformation is used in this algorithm.

- If they are 01, choose the transformation as T_D .

- If they are 10, choose the transformation as T_S .

- If they are 00 or 11 then use both transformations as T_D, T_S.

Step 2:

If the result of step 1 is T_D then execute:

Step 2.1. Get the next r bits of Key (and a is the corresponding number). Calculate $l_1 = a \bmod r$. Take the last r bits of *Key* (and b is the corresponding number) and determine $l_2 = b \bmod r$.

If the result of step 1 is T_S then execute:

Step 2.2. Take the next r non-zero bits of *Key* and convert that sequence of r bits to the element $e_1 \in GF(2^r)$. Thus $e_1 \in GF^*(2^r)$. If r bits are all 0, then we shift right by bit of *Key* until we get a sequence of r bits such that $e_1 \ne 0 \in GF(2^r)$. Take the last r bits of *Key* and convert this string of r bits to the element $e_2 \in GF(2^r)$. If $e_2 = 0 \in GF(2^r)$ then we shift left 1 bit until we get the element $e_2 \ne 0 \in GF(2^r)$.

Step 2.3. Get the next r bits at the beginning of *Key* (after step 2.2). Then convert this string to an integer, denoted a, taking $i = a \bmod m$, so $0 \le i \le m-1$. Get the next r bits (in the right-to-left direction) at the end of the *Key* bit sequence (after step 2.2). Then convert this string to an integer, denoted by b, taking $j = b \bmod m$, so $0 \le j \le m-1$.

We specify that the rows and columns of the matrix A, B will be numbered from 0 to $m-1$.

Choose the vectors for multiplying by scalars as $E = (1, 1, \ldots, 1, e_1, 1, \ldots, 1)$, $F = (1, 1, \ldots, 1)$ where e_1 is the $(i+1)$th element of the vectors E and $E' = (1, 1, \ldots, 1, e'_2, 1, \ldots, 1)$, $F' = (1, 1, \ldots, 1)$ where e_2 is the $(j+1)$th element of the vector E'.

If the result of step 1 is both T_D and T_S then exxecute (apply T_D to and T_S to B):

Step 2.4. Get the next r bits of *Key* (and a is the corresponding number). Calculate $l = a \bmod r$. Take the last r bits of *Key* and convert this string of r bits to the element $e \in GF(2^r)$. If $e = 0 \in GF(2^r)$ then we shift left 1 bit until we get the element $e \ne 0 \in GF(2^r)$.

Step 2.5. Get the next r bits at the beginning of *Key* (after step 2.4). Then convert this string to an integer, denoted by c, taking $i = c \bmod m$, so $0 \le i \le m-1$.

Take the vectors multiplied by the scalar as $E = (1, 1, \ldots, 1, e, 1, \ldots, 1)$, $F = (1, 1, \ldots, 1)$ where e is the $(i+1)$th element of the vector E.

Note. In the case of initial MDS matrix of size ≥ 16 , in order for **Algorithms 1**, **Algorithms 2** to work, it is necessary to choose a secret key Key with bit length greater than 128, for example, with $m = 16$, it should be chosen the secret key *Key* has bit length of $n \ge 192$.

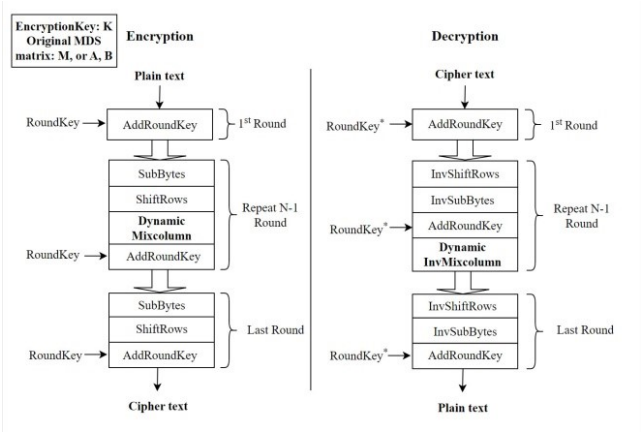

Figure 1. Structure of diffusion layer of dynamic modified AES Algorithms.

Encryption and decryption process

Both parties involved in the secret communication process will make the algorithm's parameters public and private as follows:

Public: Encryption algorithm, two initial MDS matrices A,B (of size), and the dynamic diffusion algorithm (Algorithm 2).

Private: The secret key Key.

The two parties need to implement Algorithm 2 and agree on a method to take advantage of the MDS matrices of the original SPN block cipher to find the secret parameters and they do not need to calculate the exact dynamic MDS matrices A_K, B_K. They can take advantage of the original MDS matrices A, A^{-1} and B, B^{-1} to perform encryption and decryption at the dynamic diffusion layer (see Remarks 1, 2 , 3, 4).

The **Algorithm 1*** and **Algorithm 2*** are very simple and the outputs are just parameters so the complexity is negligible (approximately $O(r)$).

IV. IMPLEMENTING DYNAMIC ALGORITHMS AND APPLICATION TO DYNAMICALLY MODIFIED AES

In this section, we implement the proposed diffusion dynamic algorithms including: Algorithm 1, Algorithm 2 in [1], and Algorithm 1*, Algorithm 2* in this paper. These dynamic algorithms are then applied to the dynamically modified AES block cipher, generating the corresponding dynamic versions of the AES block cipher.

Denote two dynamically modified AES block ciphers corresponding to two dynamic algorithms Algorithm 1 and Algorithm 2 AESD1 and AESD2 respectively. Two dynamic modified AES block ciphers correspond to two dynamic algorithms **Algorithm 1***, and **Algorithm 2***are **AESD1*** and **AESD2*** respectively.

Figure 1 shows the structure of dynamically modified AES block ciphers animated at the diffusion layer, specifically in the Mixcolumn transformation of AES.

The original MDS matrices in the original SPN are selected as recursive MDS matrices of sizes 4, 8, and 16 in Table I and Table II. These matrices are selected based on the following attributes: having small number of fixed points and small coefficient of fixed points ($D(A)$) and belongs to the class of matrices with a small number of *XORs* and *Xtimes*, in order to be efficient in performance.

1. Executing Algorithm 1 and Algorithm 2 [1]

In order to perform two dynamically modified AES block ciphers AESD1 and AESD2, we first implement two dynamic algorithms: Algorithm 1 and Algorithm 2 in [1] to find key-dependent MDS matrices or dynamic MDS matrices from the original MDS matrices. These dynamic MDS matrices are then fed into the AES Mixcolumn transformation (how these matrices are included in the Mixcolumn depends on Algorithm 1 and Algorithm 2).

a) Executing Algorithm 1

Conduct test implementation in C/C++ programming language, compiled on Visual Studio 2015 environment. To illustrate this algorithm, program execution is built with input matrices and keys as follows. The key pair corresponding to the **scalar multiplication** is:

+ 05060708090a0b0c0d0e0f1011121314 (for MDS matrices of size 4 and 8).

+05cb174fc9d78021321ad90348f5dabca23c45acb9b4c6da (for MDS matrices of size 16).

The result of Algorithm 1 implementation is that the corresponding key-dependent MDS matrices are generated. These matrices are also integrated into the diffusion layer of AES (AESD1) to analyze several cryptographic properties such as number of fixed points and coefficient $D(A)$, number of XOR and Xtime operations. The analysis results with the above two key pairs are described in Table III.

Remark 5. Analysis of the results in Table III shows that, the cryptographic properties as Branch number, number of fixed points and coefficient $D(A)$ of dynamically modified AES diffusion layer (AESD1) with the MDS matrix obtained from Algorithm 1 is completely equivalent to the diffusion layer's the same properties with the original MDS matrix selected. For Table 3, there exists an element $(e_i f_j)$ that is a generator element of the finite field, so the cycle of the transformation also reaches a maximum of 255 (see more in [28]).

TABLE I
ORIGINAL MATRIX M FOR ALGORITHM 1, ALGORITHM 1*; MATRIX A FOR ALGORITHM 2, ALGORITHM 2*

Matrix type	Original matrix representation (M or A)	Root element, cycle of T_p	Branch number, number of XOR, Xtime	Number of fixed points $F\_(A\_0)$	Coefficient $D(A)$
44 recursive MDS matrix	//0x169 0x29 0x46 0x5A 0x85 0x4F 0xD2 0x3D 0xA5 0xCB 0x87 0xA1 0xED 0x8B 0x87 0xE6 0xCA	$0x29, 8$	5, 48, 28	2^0	2.93874e-039 $(2^{-127.9999})$
88 recursive MDS matrix	//0x169 0xA3 0x2B 0x09 0x8A 0x19 0x2B 0xD3 0x75 0x8A 0x63 0x4D 0xA9 0xCA 0xD9 0x53 0xE9 0xF1 0x1E 0xB4 0x93 0x02 0x5E 0x35 0xB8 0x07 0xCD 0xC2 0x96 0x87 0x3E 0xF3 0x15 0xEC 0xE2 0x70 0xE1 0x12 0x62 0x7D 0x1D 0x50 0x38 0x17 0xCE 0xAD 0xC6 0xA6 0x80 0x88 0xFB 0x75 0x5D 0x70 0x07 0x45 0xFF 0x91 0x84 0x8A 0x7F 0x59 0x7C 0xB5 0xDF	$0x2b, 8$	9, 232, 56	2^0	2.93874e-039 $(2^{-127.9999})$
16×16 recursive MDS matrix	//0x169 4E A9 54 75 C7 61 62 D0 45 3D D6 1C 26 16 90 4E DB F9 61 88 43 67 13 FA 7B 8F 68 08 8B C6 3A 4B C4 B3 5C 75 DE 6F 96 A0 69 78 EF DA 21 25 E8 FE AD E5 68 B3 33 17 CD 9D E0 FD A5 4F C6 92 D3 45 AB 64 A3 38 82 83 68 D3 03 EC 14 BF EB AC 2A 78 61 EB 78 42 9C 1C 95 3B 41 FA C6 10 43 64 9C 4B C4 09 4E 6C 14 B0 ED 26 A8 42 9A 74 39 ED 4A 58 3B 8F B4 E3 54 94 D8 10 E7 87 06 A5 85 A4 80 71 8D 3E B5 33 A7 18 4B C4 3D A2 D4 FE 26 AC 49 0D BD BE 21 6E 74 98 30 FF 06 75 38 58 19 D8 E3 F4 93 4B BF 37 E8 CC 55 6D CF 69 C2 40 51 36 2A 70 C3 3F 25 49 B4 C5 71 99 05 B7 EC 26 E5 6E 4B E9 03 B7 C5 CE C2 3C 1F CF F6 49 AD B1 7A 3D 49 48 16 F9 EE 4E B8 4D 6B B5 93 B2 3A 3B F2 EE CA 5F B8 B9 81 61 F1 76 62 64 C6 0F 06 51 38 0D C8 72 5F 2F 7F 99 05 1E 0F 67 96 C4 4F DA B8 2B 39 97	$0x4e, 8$	17, 912, 112	2^0	2.93874e-039 $(2^{-127.9999})$

TABLE II
ORIGINAL MATRIX B FOR ALGORITHM 2, ALGORITHM 2*

Matrix type	Original matrix representation (B)	Root element, cycle of T_p	Branch number, number of XOR, Xtime	Number of fixed points $F_{(A_0)}$	Coefficient $D(A)$
44 recursive MDS matrix	//0x169 0x14 0xe6 0xa2 0xf6 0x86 0x20 0x9e 0x73 0xe3 0xda 0xa8 0x84 0x71 0x6d 0x88 0x95	$0x2a, 8$	5, 60, 28	2^0	2.93874e-039 $(2^{-127.9999})$
88 recursive MDS matrix	//0x169 0x50 0xbf 0xef 0x2b 0x57 0xbe 0x94 0x97 0xf9 0x0a 0x1c 0xf7 0x8c 0x9a 0xfc 0x06 0x89 0xc0 0xba 0xe6 0x6c 0x83 0x59 0x35 0xfc 0xb9 0x0c 0x9b 0x91 0x69 0xa2 0x17 0x7d 0xc2 0xfa 0xbf 0x83 0xb8 0xe4 0x16 0x2d 0xfc 0x6e 0x62 0xf0 0x14 0xa1 0xc7 0x1f 0xfd 0x33 0xf5 0xfa 0xe7 0x5f 0xca 0x34 0x02 0xd4 0x1e 0x65 0x2d 0xd7 0x58	$0xef, 8$	9, 304, 56	2^0	2.93874e-039 $(2^{-127.9999})$
16×16 recursive MDS matrix	//0x169 3a ce 17 d6 3a 0a 54 26 e1 ca aa c3 75 ec 24 27 90 05 99 89 46 25 6e ab c2 10 4c 2d ad 7b 5d fc c8 5b 34 81 41 41 56 3e 5a 42 67 a6 28 7d ba f1 a3 21 cb b6 22 34 2d e1 6f cc 11 fe 78 09 41 fc c8 68 42 d3 7e 25 47 7d 10 ef bb fb fb a8 c8 ed e9 13 05 bb 3a d3 e6 83 91 c4 b3 9c ad eb df e1 b8 de 9a b0 03 ef 5b e3 2e 99 16 24 64 a0 45 3b da e7 74 7f 6a c4 24 d7 ba 9d 10 2f c2 5b 5a f2 ed 08 4e e5 92 01 54 f9 cc 1b 59 a5 6e be 0b 75 93 68 0c fe 76 6b 02 4f 01 f4 f7 07 9f 4d 4f 65 88 c8 75 8b 76 2f 8c ab 4b a7 99 a7 1b 90 2e 83 96 5c 39 b7 1d cc 85 d2 77 41 f5 c5 61 4f a1 f3 d7 8a e2 7e 60 7c 08 7e 28 1c 2f 83 f1 f1 3b a9 6d 4c df 24 13 b7 ad 4e e6 c5 ec 87 f0 5a 8c d4 47 1c a5 06 63 6d 17 a9 50 7c 2d 1d dd 6e 9f 5c 26 b6 85 02 20 e9 e9 7e 5f b8 a5 ab 0d 5c bf aa	$0x3a, 8$	17, 976, 112	2^0	2.93874e-039 $(2^{-127.9999})$

b) Executing Algorithm 2

For **Algorithm 2**, an implementation is carried out using C/C++ programming language, compiled on Visual Studio 2015. To illustrate this algorithm, the test program is built

TABLE III
KEY-DEPENDENT MDS MATRICES - RESULTS OF ALGORITHM 1 AND AND THE SPEED OF AESD1

Original MDS Matrix (M)	The key-dependent MDS matrix (M_K) according to Algorithm 1 with the key pair is: *05060708090a0b0c0d0e0f1011121314* and *05cb174fc9d78021321ad90348f5dabca23 c45acb9b4c6da*	The transformation, Cycle	Branch number, number of XOR, Xtime	Number of fixed points $F_{(A_0)}$	Coefficient $D(A)$	Encryption/ Decryption speed of the AESD1 (Mb/sec)
44 recursive MDS matrix in Table 1	//0x169 0x29 0x29 0x7a 0x8e 0xfb 0xd2 0x1b 0x3f 0xcd 0xf6 0xa1 0x10 0x1f 0xaa 0x47 0xca	T_M, 255	5, 64, 28	2^0	2.93874e-039 $(2^{-127.9999})$	58.382
88 recursive MDS matrix in Table 1	//0x169 0xa3 0xb2 0x44 0xb9 0xa3 0x2c 0x28 0xd0 0xec 0x63 0xd1 0x78 0x84 0xe2 0x5e 0xa5 0x36 0xc4 0xb4 0x78 0x8d 0x6 0xae 0x85 0x19 0x3f 0x95 0x96 0x1e 0xe9 0xf0 0x37 0x84 0x1a 0xeb 0x85 0x12 0x78 0x37 0x23 0xb9 0x47 0x82 0x6f 0x87 0xc6 0x49 0xb0 0x24 0x9c 0xea 0xda 0xe9 0xa1 0x45 0x1f 0xe4 0x4e 0xd2 0xfb 0x1f 0x8d 0x6b 0xdf	T_M, 255	9, 246, 56	2^0	2.93874e-039 $(2^{-127.9999})$	55.323
1616 recursive MDS matrix in Table 1	//0x169 4E A9 54 75 C7 61 62 D0 45 3D D6 1C 26 16 90 4E DB F9 61 88 43 67 13 FA 7B 8F 68 08 8B C6 3A 4B C4 B3 5C 75 DE 6F 96 A0 69 78 EF DA 21 25 E8 FE AD E5 68 B3 33 17 CD 9D E0 FD A5 4F C6 92 D3 45 AB 64 A3 38 82 83 68 D3 03 EC 14 BF EB AC 2A 78 61 EB 78 42 9C 1C 95 3B 41 FA C6 10 43 64 9C 4B C4 09 4E 6C 14 B0 ED 26 A8 42 9A 74 39 ED 4A 58 3B 8F B4 E3 54 94 D8 10 E7 87 06 A5 85 A4 80 71 8D 3E B5 33 A7 18 4B C4 3D A2 D4 FE 26 AC 49 0D BD BE 21 6E 74 98 30 FF 06 75 38 58 19 D8 E3 F4 93 4B BF 37 E8 CC 55 6D CF 69 C2 40 51 36 2A 70 C3 3F 25 49 B4 C5 71 99 05 B7 EC 26 E5 6E 4B E9 03 B7 C5 CE C2 3C 1F CF F6 49 AD B1 7A 3D 49 48 16 F9 EE 4E B8 4D 6B B5 93 B2 3A 3B F2 EE CA 5F B8 B9 81 61 F1 76 62 64 C6 0F 06 51 38 0D C8 72	T_M, 255	17, 1028, 112	2^0	2.93874e-039 $(2^{-127.9999})$	47.894

with matrices and input key as follows:

- The input matrices of size 4×4,8×8,16×16 are all recursive MDS matrices. In which, the first input matrices (A) is the recursive matrices proposed in Table 1; The second input matrices (B) is also a recursive matrices, as listed in Table II.

- 02 encryption keys with length 128 bits respectively for matrices of size, and 01 key of length 192 bits for matrices of size 16×16 are:

0405060708090a0b0c0d0e0f10111213,

0708090a0b0c0d0e0f10111213141516,

0f101112131415161718191a1b1c1d1e1f20212223242526

The test result of Algorithm 2 is that the corresponding key-dependent MDS matrices are generated. These matrices are integrated into the diffusion layer of AES (AESD2) to analyze several cryptographic properties such as: number of fixed points, coefficient $D(A)$ number of XOR and Xtime operations. The results of the analysis are described in Table IV. And the speed of AESD2 is given here.

Remark 6. The analysis of the results in Table V shows that the cryptographic properties such as: Branch number, number of fixed points and coefficient $D(A)$ of dynamically modified AES diffusion layer (AESD2) with key-dependent MDS matrices A_K and B_K obtained from Algorithm 2 are completely equivalent to the diffusion layer with the original MDS matrices (A, B) selected.

2. Dynamic modification of the AES block cipher based on Algorithm 1 and Algorithm 2 [1]

This section applies two dynamic algorithms Algorithm 1 and Algorithm 2 to modify the AES block cipher, using the dynamic MDS matrices generated from these two algorithms to feed the Mixcolumn transformation of AES. Dynamically modified AES based on Algorithm 1 uses a dynamic matrix for Mixcolumn in every round of AES. Dynamically modified AES based on Algorithm 2 uses two dynamic matrices for Mixcolumn in alternating rounds of the AES block cipher.

After applying the above two algorithms to AES, we evaluate the performance speed of the dynamically modified AES block ciphers AESD1 and AESD2 when implementing Algorithms 1 and Algorithm 2.

AESD1 and AESD2 use new MDS matrices created by Algorithm 1 and Algorithm 2 from the original MDS matrices selected with different sizes as shown in Tables 1 and 2. AESD1 and AESD2 are implemented according to the component functions corresponding to the operations - transformations of the AES algorithm; Multiplications

TABLE IV
KEY-DEPENDENT MDS MATRICES - RESULTS OF ALGORITHM 2 AND THE SPEED OF AESD2

Original MDS Matrix (A, B)	The key-dependent MDS matrix (A_K adn B_K) with keys corresponding to sizes of 4×4,8×8,16×16 are: *0405060708090a0b0c0d0e0f10111213 0708090a0b0c0d0e0f10111213141516 0f101112131415161718191a1b1c1d1e1 f20212223242526*	The transformation, Cycle	Branch number, number of XOR, Xtime	Number of fixed points $F_{(A_0)}$	Coefficient $D(A)$	Encryption/ Decryption speed of the AESD2 (Mb/sec)
Recursive Matrices A,B of size 4×4 in Table 1, Table 2	A_K 0x8c 0x9b 0xa2 0xff 0xda 0x0c 0xf5 0x32 0x24 0xfb 0x22 0xfd 0xab 0xfb 0xb8 0x25	$T_P, 8$ $\left(A_K = A_{d^{2^1}}\right)$	5, 68, 28	2^0	2.93874e-039 $(2^{-127.9999})$	56.287
	B_K 0x14 0xcf 0x8e 0x96 0x49 0x20 0x9e 0x73 0xdb 0xda 0xa8 0x84 0xec 0x6d 0x88 0x95 (Multiply row 1 column 1)	$T_M, 15$ $(e = 0x13, f = 0x6a)$	5, 58, 28	2^0	2.93874e-039 $(2^{-127.9999})$	56.287
Recursive Matrices A,B of size 8×8 in Table 1, Table 2	A_K 0x26 0x88 0x41 0xaa 0x28 0x88 0x0d 0x3a 0xaa 0x47 0xde 0x62 0x25 0x49 0xe3 0xed 0xc4 0x3d 0x5a 0x82 0x04 0xb2 0xb5 0x0a 0x15 0x30 0x65 0x93 0xfb 0xf0 0xc0 0x78 0xfc 0xa8 0x2b 0xad 0x6d 0x46 0x7a 0x38 0xe6 0xe4 0x7c 0x35 0x72 0x75 0x37 0xee 0xae 0x80 0x3a 0xb7 0x2b 0x15 0x9e 0x90 0x86 0xfe 0xaa 0x7e 0xa7 0x7b 0x5b 0x5d	$T_P, 8$ $\left(A_K = A_{d^{2^1}}\right)$	9, 247, 56	2^0	2.93874e-039 $(2^{-127.9999})$	42.778
	B_K 0x50 0x81 0xac 0x98 0x4f 0x97 0x19 0x23 0xda 0x0a 0x1c 0xf7 0x8c 0x9a 0xfc 0x06 0xfd 0xc0 0xba 0xe6 0x6c 0x83 0x59 0x35 0xa9 0xb9 0x0c 0x9b 0x91 0x69 0xa2 0x17 0x16 0xc2 0xfa 0xbf 0x83 0xb8 0xe4 0x16 0x50 0xfc 0x6e 0x62 0xf0 0x14 0xa1 0xc7 0x11 0xfd 0x33 0xf5 0xfa 0xe7 0x5f 0xca 0xc4 0x02 0xd4 0x1e 0x65 0x2d 0xd7 0x58 (Multiply row 1 column 1)	$T_M, 85$ $(e = 0x16, f = 0x52)$	9, 248, 56	2^0	2.93874e-039 $(2^{-127.9999})$	42.778
Recursive Matrices A,B of size 16×16 in Table 1, Table 2	A_K de 9b 09 ac e1 aa c2 8f 0d 18 e5 bd ce 6f 80 de 5c 91 aa 3e 67 c0 6d f9 7d 55 10 bb 56 e0 73 dc 89 4c b2 ac 5e 7b ea 21 11 15 fe 5d a5 a6 95 fa 98 2c 10 4c c9 6e 33 39 2e 92 23 df e0 e9 e7 0d f2 a8 49 1a ec ed 10 e7 68 96 06 f4 fd 99 76 15 aa fd 15 66 38 bd 82 72 0e f9 e0 05 67 a8 38 dc 89 ba de 13 06 24 97 ce 9a 66 52 ad 1b 97 dd b1 72 55 27 46 09 83 34 05 45 ee 6a 23 87 22 85 af 3c 70 26 c9 4a be dc 89 18 48 8c fa ce 99 b5 b9 9d f5 a5 7a ad 3b a1 fb 6a ac 1a b1 bf 34 46 28 e8 dc f4 ca 95 32 08 12 5a 11 e3 0f 0b cb 76 ae e2 71 a6 b5 27 88 af 3a 02 4f 96 ce 2c 7a dc 94 68 4f 88 5b e3 19 d5 5a 41 b5 98 25 7c 18 b5 b4 6f 91 ff de 9f b6 78 26 e8 4d 73 72 42 ff 58 da 9f 9e 84 aa 2a c4 c2 a8 e0 d0 6a 0b 1a b9 31 c7 da 74 7e 3a 02 d4 d0 c0 ea 89 df 5d 9f 77 1b eb	$T_P, 8$ $\left(A_K = A_{d^{2^1}}\right)$	17, 1021, 112	2^0	2.93874e-039 $(2^{-127.9999})$	38.843
	B_K 3a a3 40 48 aa 15 30 d9 05 3b 2c 44 1b e2 95 ff c5 05 99 89 46 25 6e ab c2 10 4c 2d ad 7b 5d fc f8 5b 34 81 41 41 56 3e 5a 42 67 a6 28 7d ba f1 ce 21 cb b6 22 34 2d e1 6f cc 11 fe 78 09 41 fc f8 68 42 d3 7e 25 47 7d 10 ef bb fb fb a8 c8 ed 72 13 05 bb 3a d3 e6 83 91 c4 b3 9c ad eb df e1 bb de 9a b0 03 ef 5b e3 2e 99 16 24 64 a0 45 3b 79 e7 74 7f 6a c4 24 d7 ba 9d 10 2f c2 5b 5a f2 a6 08 4e e5 92 01 54 f9 cc 1b 59 a5 6e be 0b 75 9a 68 0c fe 76 6b 02 4f 01 f4 f7 07 9f 4d 4f 65 ef c8 75 8b 76 2f 8c ab 4b a7 99 a7 1b 90 2e 83 7b 5c 39 b7 1d cc 85 d2 77 41 f5 c5 61 4f a1 f3 59 8a e2 7e 60 7c 08 7e 28 1c 2f 83 f1 f1 3b a9 88 4c df 24 13 b7 ad 4e e6 c5 ec 87 f0 5a 8c d4 9c 1c a5 06 63 6d 17 a9 50 7c 2d 1d dd 6e 9f 5c 01 b6 85 02 20 e9 e9 7e 5f b8 a5 ab 0d 5c bf aa (Multiply row 1 column 1)	$T_M, 17$ $(e = 0x26, f = 0x35)$	17, 1029, 112	2^0	2.93874e-039 $(2^{-127.9999})$	38.843

by the elements of the MDS matrix are performed by looking up the finite field multiplication table as the usual implementation of Vincent Rijmen in [5]. The test results of AESD1 and AESD2 encryption/decryption speed evaluation with 128-bit key are described in Table 3 and Table 4 respectively.

To be able to compare the speed of dynamically modified AES block ciphers with AES, we also evaluate the execution speed of the AES block cipher in Table V.

Remark 7. Analysis of the results in Table 3, Table 4 and comparison with the results in Table 5 shows that the performance speed of dynamically modified AES algorithm

TABLE V
PERFORMANCE EVALUATION OF AES BLOCK CIPHER

		Optimal implementation according to Brian Gladman		Normal implementation			
	Matrix representation 4×4	Key length (bits)	Encryption/ Decryption Speed (Mb/sec)	Key length (bits)	Encryption/ Decryption Speed of AES_4_4 (Mb/sec)	Number of multiplications (look up table)/1 round	Number of additions/1 round
Original MDS Matrix	//0x11B 0x02 0x03 0x01 0x01 0x01 0x02 0x03 0x01 0x01 0x01 0x02 0x03 0x03 0x01 0x01 0x02	128	886.512	128	59.998	24	48
		192	753.311	192	59.820	24	48
		256	669.231	256	58.812	24	48

TABLE VI
PERFORMANCE RESULTS FOR ALGORITHM 1* AND SPEED OF AESD1*

Original Matrix	Key value	First two bits	Transformation	Set of secret parameters for transformation	Encryption/ Decryption Speed of AESD1* (Mb/s)
4×4 recursive MDS matrix in Table 1	27F12130405060708090A0B0C0D0E0F0 (128 bit)	01	T_P	$l = 4, r - l = 4$	38.898
	71F12130405060708090A0B0C0D0E0F0 (128 bit)	11	T_M	$(e_0, e_1, e_2, e_3) = (c5, 87, c4, 80)$	45.028
8×8 recursive MDS matrix in Table 1	EFFF0130405060708090A0B0C0D0E0F0 (128 bit)	01	T_P	$l = 7, r - l = 1$	31.215
	19FF0130405060708090A0B0C0D0E0F0 (128 bit)	10	T_M	$(e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7) = (c6, 37, c4, 80, 81, c1, 01, 42)$	36.305
16×16 recursive MDS matrix in Table 1	E0FF2130405060708090A0B0C0D0E0F 00111213241516171 (192 bit)	01	T_P	$l = 3, r - l = 5$	23.113
	71FF2130405060708090A0B0C0D0E0F 00111213241516171 (192 bit)	11	T_M	$(e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}) = (c5, bf, c4, 80, 81, c1, 01, 42, 82, c2, 02, 43, 83, c3, 03, 44)$	20.878

AESD1 (Algorithm 1) is equivalent to AES algorithm, while the AESD2 algorithm (Algorithm 2) is not significantly slower. While the security of AESD1 and AESD2 is significantly increased compared to AES.

3. Executing Algorithm 1* and Algorithm 2* and dynamically modifying the AES block cipher

In order to perform two dynamically modified AES block ciphers AESD1* and AESD2*, we first perform two dynamic algorithms Algorithm 1* and Algorithm 2* to find out the secret parameters (number of direct exponent; scalar multiplier vectors) to include the encryption and decryption of AESD1* and AESD2* in a way that utilizes the MDS matrices of the original SPN block cipher.

In this section, the test results of AESD1* by Algorithm 1* and AESD2* by Algorithm 2* with the idea of taking advantage of the original MDS matrices are described as follows:

• The original matrices of size 4 , 8, 16 are the selected as recursive MDS matrices in Table I, Table II.

• Test results with different keys (length of 128 bits for matrices of size 4 and 8 , and length of 192 bits for matrices of size) according to Algorithm 1* and the speed of AESD1* are described in Table VI.

• Test results with different keys (length of 128 bits for matrices of size 4 and 8 , and length of 192 bits for matrices of size 16) according to Algorithm 2* and the speed of AESD2* are described in Table VII.

Remark 8. With AESD1* and AESD2* using Algorithm 1* and Algorithm 2* respectively, the encryption/decryption speeds of these dynamic block ciphers are shown in Table 6 and Table 7. It can be seen that the speed of these block ciphers is not significantly slower than the AESD1 and AESD2 using Algorithms 1 and Algorithm 2.

Table 8 gives a speed comparison between dynamically modified AES algorithms. Although the AESD1* and AESD2* algorithms are slower than the AESD1 and AESD2 algorithms, the advantage here is: when utilizing the original MDS matrices of the original SPN block cipher according to Algorithm 1* and Algorithm 2*, we do not need to compute, store and protect for dynamic MDS matrices. Especially in systems where many people are

TABLE VII
PERFORMANCE RESULTS FOR ALGORITHM 1* AND SPEED OF AESD2*

Original Matrix	Key value	First two bits	Trans formation	Set of secret parameters for transformation	Encryption/ Decryption Speed of AESD1* (Mb/s)
4×4 recursive MDS matrix in Table 1	17F12130405060708090A0B0C0D0E0F0 (128 bit)	10	T_P	$l_1 = 4, r - l_1 = 4$ và $l_2 = 7, r - l_2 = 1$	37.683
	2FF12130405060708090A0B0C0D0E0F0 (128 bit)	01	T_M	$(e_{10}, e_{11}, e_{12}, e_{13}) = (04, 01, 01, 01)$ $(e_{20}, e_{21}, e_{22}, e_{23}) = (0f, 01, 0f, 01)$	44.215
	3DF12130405060708090A0B0C0D0E0BE (128 bit)	11	T_P, T_M	$l_1 = 2, r - l_1 = 6(e_{10}, e_{11}, e_{12}, e_{13}) = (01, 01, eb, 01)$	40.231
8×8 recursive MDS matrix in Table 1	1EFE2130405060708090A0B0C0D0E0F0 (128 bit)	10	T_P	$l_1 = 0, r - l_1 = 8 v l_2 = 7, r - l_2 = 1$	30.124
	2EFE2130405060708090A0B0C0D0E0FC (128 bit)	01	T_M	$(e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17} = (01, 01, 01, f8, 01, 01, 01, 01)$ $(e_{20}, e_{21}, e_{22}, e_{23}, e_{24}, e_{25}, e_{26}, e_{27} = (01, 01, 01, 01, 01, 01, cf, 01)$	35.415
	3DFE2130405060708090A0B0C0D0E0FC (128 bit)	11	T_P, T_M	$l_1 = 4, r - l_1 = 4(e_10, e_11, e_12, e_13, e_14, e_15, e_16, e_17 = (01, 01, 01, 01, 01, 01, 01, cf)$	31.895
16×16 recursive MDS matrix in Table 1	1BFF2130405060708090A0B0C0D0E 0F00111213241516171 (192 bit)	10	T_P	$l_1 = 4, r - l_1 = 4$ và $l_2 = 7, r - l_2 = 1$	17.905
	2DFF2130405060708090A0B0C0D0 E0F00111213241516171 (192 bit)	01	T_M	$(e_0, \ldots, e_2, \ldots, e_{15}) = (01, \ldots, \ldots, f4)(e_0, \ldots, e_14, e_{15}) = (01, \ldots, 0f, 01)$	20.625
	71FF2130405060708090A0B0C0D 0E0F00111213241516171 (192 bit)	11	T_P, T_M	$l_1 = 5, r - l_1 = 3(e_0, \ldots, e_12, \ldots, e_{15}) = (01, \ldots, 0f, \ldots, 01)$	18.830

TABLE VIII
SPEED COMPARISON OF DYNAMICALLY MODIFIED AES BLOCK CIPHERS

Original Matrix	Key value	First two bits	Trans formation	Set of secret parameters for transformation
4×4 recursive MDS matrix	58.382	56.287	38.898	37.683
8×8 recursive MDS matrix	55.323	42.778	31.215	30.124
16×16 recursive MDS matrix	47.898	38.843	23.113	17.905

secretly communicating with each other - the number of dynamic MDS matrices is very large, this method will be very useful. We can save storage space for dynamic MDS matrices, not having to calculate and protect these matrices anymore.

V. SECURITY ANALYSIS AND IMPLEMENTATION RESOURCES FOR DYNAMICALLY MODIFIED AES BLOCK CIPHERS

Regarding the security of dynamic AES block ciphers, we analyzed it in detail in [1]. Animating the diffusion layer makes cryptanalysis much more difficult because it is not known which MDS matrix is used in the Mixcolumn transformation. Shannon [31] showed that the unicity distance of the cryptosystem is calculated by the following formula: $U = \frac{H(K)}{D}$

where $H(K)$ is the entropy of the key, and D is the redundancy of the source in bits per character. Another definition of U is the average number of plaintext characters required for the redundancy (in bits) to exceed the key length. If the length of the encrypted message is less than this unicity distance, it is impossible to decide the unique key or plaintext from knowledge of the ciphertext alone (a lot of dummy keys will be found, without knowing which is the real key). It is quite possible that cryptosystems have an infinite value of U, and Shannon calls such cryptosystems ideal ones.

In the case of a static AES block cipher, the primary keyspace is the space of the original secret key used to encrypt the data. However, with dynamic AES block ciphers, the key space includes not only the secret key, but also the dynamic diffusion layer (or dynamic MDS matrix). Thus, the key space in the case of dynamic AES has been increased by the space of dynamic MDS matrices. In fact, if we use direct exponentiation, the number of dynamic matrices in $GF\left(2^8\right)$ is 8, if we use scalar multiplication,

the number of dynamic MDS matrices is about 255. Thus, the number of dynamic MDS matrices for the diffusion layer when combining these two transformations can be up to approximately 2^{11}. This number is not too large in theory, but it is very significant in practice.

For example, in order to perform strong attacks on block ciphers such as linear attacks, differential attacks, cryptanalysts must collect a lot of plaintext/ciphertext pairs, like DES algorithm, they must obtain 2^{47} clear code pair. Then if using dynamic block cipher algorithm, they have to collect about 2^{58} plaintext/ciphertext pairs. This is a very large number in practice, which will make it much more difficult for cryptanalysts to attack dynamic block ciphers.

Even cryptanalysts don't know when to use which dynamic algorithm: Algorithm 1, Algorithm 2, Algorithm 1* or Algorithm 2*. They also don't know which dynamic matrix will be used in which round. Besides, in order to perform strong attacks on block ciphers such as linear attacks, differential attacks, cryptanalysts must collect a lot of plain text/cipher text pairs, such as the DES algorithm, which they must obtain 2^{43} pairs of plain text/cipher text. In short, when we use dynamic algorithms, cryptanalysis will be many times more difficult than normal AES cryptanalysis. That will contribute to improving the security of the AES block cipher.

However, it should be recalled here that, when utilizing the MDS matrices of the original SPN block cipher according to Algorithm 1* and Algorithm 2*, there is no need to compute, store and protect the dynamic MDS matrices. Especially in systems where many people secretly communicate with each other - the number of dynamic MDS matrices is very large, this method is very useful.

The advantage of Algorithms 1*, 2* over Algorithms 1, 2 is: cryptographic parties do not need to specifically compute dynamic MDS matrices, nor do they have to store and protect the matrices. In particular, in a multi-user encryption system, if the static block cipher's version that utilizes MDS matrices is applied, the user does not have to compute and store and protect multiple dynamic MDS matrices at the same time. Also Algorithm 1* and Algorithm 2* are very simple, only need to calculate output as parameters, not dynamic MDS matrices, so Algorithm 1* and Algorithm 2* are faster than Algorithm 1 and Algorithm 2. However, their limitation is that for each round of encryption, we have to apply additional transformations on the input and output of the diffusion layer (in order to take advantage of the MDS matrices of static block ciphers), which will reduce the software execution of dynamic block ciphers, for example when applying them to AES modifications (AESD1*, AESD2*).

About implementation resources for dynamically modified AES block ciphers:

- For the optimal implementation method by reducing the table lookups, it is completely equivalent to AES (all uses 04 lookup tables of the same size).

- With the usual implementation method: For the application of the MDS matrix, the resource usage is completely equivalent to AES. For and MDS matrices, the amount of memory required is slightly larger because the size of the matrices is larger.

VI. CONCLUSION

In this paper, we execute the two dynamic diffusion layer algorithms proposed in [1] with recursive MDS matrices of size 4, 8, and 16 then evaluate the cryptographic properties of the obtained dynamic MDS matrices including branch number, number of fixed points, coefficient of fixed points, number of XOR and Xtime operations. Then, we apply these two dynamic algorithms to dynamically modify the AES block cipher and evaluate the execution speed of the dynamic AES block ciphers. We propose two new diffusion layer dynamic algorithms, implement these two algorithms and modify dynamically AES block cipher according to these two algorithms, and evaluate the speed of dynamic AES block ciphers. We also evaluate the security and implementation resources for dynamically modified AES block ciphers. The proposed diffusion layer dynamic algorithms contribute to improving the security of the AES block cipher against many current strong attacks.

REFERENCES

[1] Luong, T. T, "Building the dynamic diffusion layer for SPN block ciphers based on direct exponent and scalar multiplication", *Journal of Science and Technology on Information security*, 1(15), 38-45, 2022.

[2] Ishukova, E.A., Krasovki, A.V, Babenko L. K, "Security assessment of Kuznhyechik block cipher using key-related method", *basic research* (11-4) 698-703, 2016 (Russian).

[3] Tomanenko E. A, "Results of testing the possibility of using hybrid cryptography on the basis of symmetric algorithms", 22, 2022 (Russian).

[4] Dolmatov, V, "GOST R", *Block Cipher*" Kuznyechik", 34-12, 2015.

[5] Deamen, J., & Rijmen, "The design of rijndael", *In AES-The Advanced Encryption Standard*, Springer-Verlag, 2002.

[6] Daemen, J., & Rijmen, Aes proposal: Rijndael (version 2), nist aes website, 1999

[7] Zenzin O.X. Ivanov M.A, AES Encryption Standard. *Finite field. M.: Kudits-Obraz* 176, 15, 2002 (Russian)

[8] Ayoub.F, "Probabilistic completeness of substitution-permutation encryption networks". *IEE Proceedings E (Computers and Digital Techniques)*, 129(5), 195-199, 1982.

[9] Heys, H. M., & Tavares, S. E, "Avalanche characteristics of substitution-permutation encryption networks", *IEEE Transactions on Computers*, 44(9), 1131-1139, 1995.

[10] Heys. H. M., & Tavares. S. E, Substitution-permutation networks resistant to differential and linear cryptanalysis. *Journal of cryptology*, 9(1), 1-19, 1996.
[11] Schneier B., Kelsey J., Whiting D., Wagner D., Hall C. and Ferguson N, “Twofish: a 128-bit block cipher”, *NIST AES Proposal*, vol. 15, 1998.
[12] Schneier B., Kelsey J., Whiting D., Wagner D., Hall C. and Ferguson N, “The twofish encryption algorithm”, *Wiley*, 1999.
[13] Abd-ElGhafar, I., Rohiem, A., Diaa, A., & Mohammed. F, "Generation of AES key dependent S-boxes using RC4 algorithm", In *International Conference on Aerospace Sciences and Aviation Technology* (Vol. 13, No. AEROSPACE SCIENCES & AVIATION TECHNOLOGY, ASAT-13, May 26–28, 2009.
[14] Agarwal, P., Singh, A., & Kilicman. A, " Development of key-dependent dynamic S-boxes with dynamic irreducible polynomial and affine constant", *Advances in Mechanical Engineering*, 10(7), 2018.
[15] Assafli, H. T., & Hashim, I. A, "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*, 978, 1, 2020.
[16] Hosseinkhani, R., & Javadi, H. H. S, "Using cipher key to generate dynamic S-box in AES cipher system", *International Journal of Computer Science and Security (IJCSS)*, 6(1), 19-28, 2012.
[17] Juremi, J., Mahmod, R., Zukarnain, Z. A., & Yasin.S.M, "Modified AES s-box based on determinant matrix algorithm", *Int. J. Adv. Res. Comput. Sci. Softw. Eng*., 7(1), 110-116, 2017.
[18] Kazlauskas, K., & Kazlauskas. J, "Key-dependent S-box generation in AES block cipher system", *Informatica*, 20(1), 23-34, 2009.
[19] KHAMLICH. E, "Implementation of stronger AES by using Dynamic S-Box dependent of Master Key", *Journal of Theoretical and Applied Information Technology*, 53(2), 2013.
[20] Mahmoud, E. M., Abd, A., El Hafez, T. A. E., & El Hafez, T. A, Dynamic AES-128 with key-dependent S-box, 2013.
[21] Murtaza G., Khan A.A., Alam S.W. and Farooqi A, “Fortification of aes with dynamic mix-column transformation,” *IACR Cryptology ePrint Archive*, 2011.
[22] I.A. Ismil, Galal H. Galal - Edeen, Sherif Khattab and Mohamed Abd ElHamid I. Moustafa El Bahtity, "Performance examination of AES encrytion algorithm with constant and dynamic rotation", *International Journal of Reviews in Computing*, ISSN: 2076-3328, 31st December 2012. Vol. 12, 2012.
[23] Auday H. Al-Wattar, Ramlan Mahmod, Zuriati Ahmad Zukarnain and NurIzura Udzir, "A new DNA based approach of generating key dependent Mixcolumns transformation", *International Journal of Computer Networks & Communications (IJCNC)* Vol.7, No.2, 2015.
[24] Adnan Ibrahem Salih, Ashwak Alabaich, Ammar Yaseen 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, 2020.
[25] Luong, T. T., Cuong, N. N., & Tho, H. D, "On the calculation of input and output for dynamic MDS matrices in diffusion layer of SPN block ciphers", *In 2017 International Conference on Information and Communications (ICIC)* (pp. 281-287), 2017.
[26] Luong, T. T., Cuong, N. N., & Tho, H. D, "On the calculation of input and output for dynamic MDS matrices in diffusion layer of SPN block ciphers", *In 2017 International Conference on Information and Communications (ICIC)*, 2017
[27] Luong, T. T., & Cuong, N. N, "The preservation of good cryptographyic properties of mds matrix under direct exponent transformation", *Journal of Computer Science and Cybernetics*, 31(4), 291-291, 2015.
[28] Luong, T. T., & Cuong, N. N, "DIRECT EXPONENT AND SCALAR MULTIPLICATION TRANSFORMATIONS OF MDS MATRICES: SOME GOOD CRYPTOGRAPHIC RESULTS FOR DYNAMIC DIFFUSION LAYERS OF BLOCK CIPHERS", *Journal of Computer Science and Cybernetics*, 32(1), 1-17, 2016.
[29] Luong, T. T., Cuong, N. N., & Dung, L. T, "The preservation of the coefficient of fixed points of an MDS matrix under direct exponent transformation", *In 2015 International Conference on Advanced Technologies for Communications (ATC)* (pp. 111-116), 2015.
[30] Luong, T. T., Cuong, N. N., & Dung, L. T, "A new statement about direct exponent of an MDS matrix in block ciphers", *In 2015 Seventh International Conference on Knowledge and Systems Engineering (KSE)* (pp. 340-343), 2015
[31] Shannon C.E, “Communication theory of secrecy systems,” *Bell System Technical Journal*, vol. 28, no. 4, pp. 656-715, 1949.

Tran Thi Luong received Bachelor of Mathematics and Informatics of Ha Noi university of Sicence in 2006; Master degreee in cryptographic technique at Academy of Cryptographic Techniques in 2012; Ph.D degree in cryptographic technique at Academy of Cryptographic Techniques in 2019;
Recent research direction: Cryptogaphy, Coding theory and Information Security.
Email: luongtranhong@gmail.com