Regular Paper

Communications of the CCISA
Vol. 26 No. 2 May. 2020

Efficient schemes with diverse of a pair of circulant matrices for AES MixColumns-InvMixcolumns transformation

Jeng-Jung Wang[1], Yan-Haw Chen[2*], Guan-Hsiung Liaw[3], Jack Chang[4], Cheng-Chih Lee[5]
[1,2,3,5]*Dept. of Information Engineering, I-Shou University, Kaohsiung, Taiwan 84008.*
[4]*Intellectual Property Group, Davis, Wright, & Tremaine, Seattle, Washington, USA*
[1,2,3,5]yanchen@isu.edu.tw, [4]JackChang@dwt.com

Abstract

Recently, AES is a commonly used encryption-decryption algorithm applied to wireless communication protocols. However, confidentiality and speed both associated with Cipher-InvCipher that are a very important issue in many current communication systems. In this paper, the key idea here is to propose a method with more variations in circulant matrix for enhancing security in AES MixColumns-InvMixColumns step. The paper is also to propose a method minimizes the number of multiplications for matrix multiplication theoretically based on two-point cyclic convolution properties of circulant matrix. The conventional 4×4 matrix multiplication typically needs 16 multiplications and 12 additions; however, the proposed method, described herein as Scheme 3, can reduce the matrix multiplications into 5 multiplications and 15 additions, which is used for encryption and decryption. Using Scheme 3 and Horner's rule-based multiplication running on Intel CPU, the computational cost of the matrix multiplication can be reduced by ~63%. Furthermore, experiments using Scheme 3 along with Horner's rule-based multiplication by means of AES keys lengths with 128, 192, 256 bits were tested by running on STM32L476VG MCU, result leads to the reduction of encryption and decryption time respectively by ~60%. Finally, the proposed procedure enables found many a pair of the circulant matrices for AES Cipher-InvCipher so that diverse of a pair of the circulant matrices can enhance security of the data transmission.

Keywords: **AES; Circulant; Lookup Table; Finite Field; Multiplication**

*Corresponding author. Email: yanchen@isu.edu.tw, Fax: (886-7)-657-8944.

1. Introduction

New features are being introduced and protecting data transmission is now more important than ever. Thus, an improvement to efficiently apply the Advanced Encryption Standard (AES) to communication systems, and cloud computing in healthcare systems [18] are important. The MixColumns-InvMixColumns transformation [13] is one of the functions in the Cipher-InvCipher. In AES, MixColumns transformation is a computationally expensive operation where the input matrix is multiplied with the MDS matrix. This transformation plays an important role with respect to the wide trail strategy in the cipher. In the early, the MDS matrix is also using in error correction code which authors by Lacan [14] and Macwillanms [9] have performed cyclic convolution of complex values with a hybrid transformation over finite fields. There exists several new research directions suggested by searching methods for finding MDS matrices in [7][8][16][17]. Moreover, in [10] has shown that the method can generate a random MDS matrix, and those techniques can be enhanced by dynamic MDS matrices. The diversity circulant matrices are used in the modern cryptographic method in AES. The computation of MDS matrix might be used in the encryption and decryption such as Rijndael method and Twofish method in [5]. However, these articles fail to mention to get inverse MDS matrices method.

Furthermore, due to attacks [1] on AES-128 using known-key distinguishing attack with a computation complexity 2 method, this leads to opportunities to enhance security of data transmission. We propose using different coefficients of the polynomial $A(x)$ and the inverse polynomial $A(x)$, namely $A^{-1}(x)$. They are used in AES MixColumns-InvColumns by using some of the bits from the AES key as an index to find the variations of the coefficients of the polynomial. The method would be more difficult for attackers to locate and thus less prone to attacks in general. This paper also proposes an efficient method to find pairs consisting of the polynomial $A(x)$ and $A^{-1}(x)$ by the Find_inv_matrix() procedure. Scheme 3, as descried in this paper, may be designed as a circuit in VLSI, see [2][4][6][15][11][20], which can be used to decrease logic gates. The matrix product operation can be used with distinct method of the multiplication in finite field see [3][12]. The method also can provide the security of the data transfer to the health monitoring system on ARM-based microcontrollers [18].

The remaining portion of this paper is organized as follows: Section 2 introduces enhanced security in AES MixColumns step. Section 3 discusses the multiplication in finite field concepts necessary for further developments, and also proposes methods to reduce the multiplication in matrix products for the AES encryption-decryption which these methods are called Scheme 1, Scheme 2, and Scheme 3, respectively. Section 4 proposes an efficient method that can be found in the entries at the first row vectors of the matrix A and the first

row vectors of the inverse matrix *A* for using in AES MixColumns-InvMixColumns step. Section 5 presents a performance analysis of AES Cipher-InvCipher on Intel CPU and STM32L476VG ARM-based MCU. Section 6 concludes the paper.

2. Enhanced security in AES MixColumns step

This paper mainly is not focused on fix polynomial a(x) in AES MixColumns transformation. We aim to enhance security of this AES algorithm with diversity MixColumns of the coefficients of polynomial that can be for increasing security. Since, if data is given in both plaintext and ciphertext, the determining the key would require an exhaustive search. However, Encrypting and decrypting data is must to know the Table A and Table B as shown in Figure 1. In other words, the key cannot be known from the plaintext and the ciphertext because the ciphertext and plaintext are obtained from AES standard MixColumns (02, 03, 01, 01) and InvMixColumns (oe, ob, od, 09) transformation. Furthermore, it might be sent the coefficient of the polynomial a(*x*) by elliptic curve cryptography of the ECDH algorithm to receiver. Receiver got the polynomial a(x) must to compute inverse the polynomial a(x) for decryption. So that it does not need to the Tabe A and Tabe B.

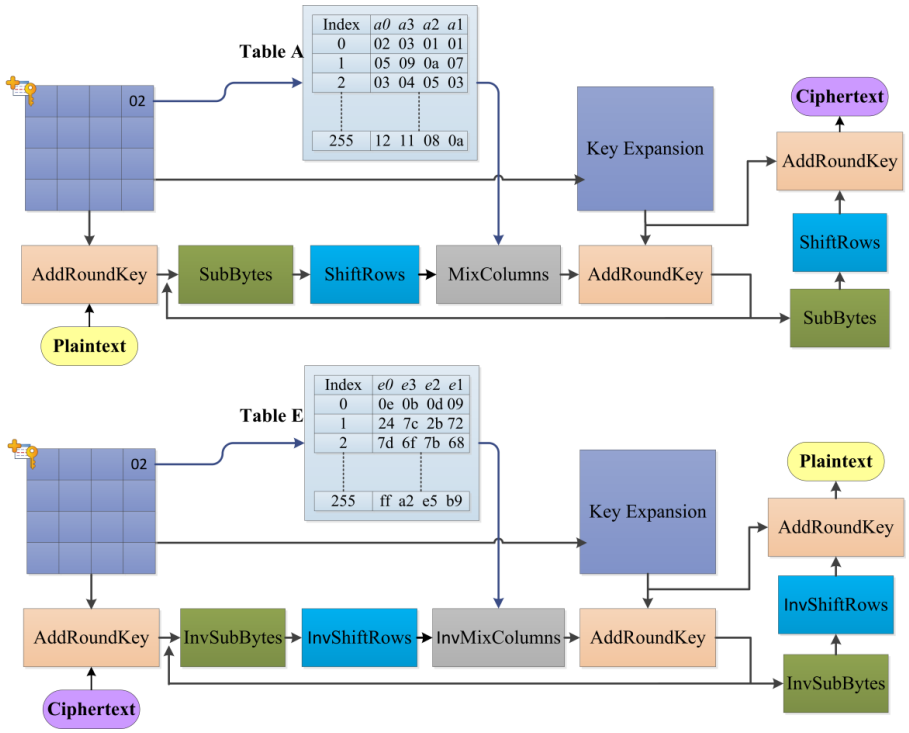

Figure 1: Some bits of a key as index of coefficients

3. Fast matrix multiplication in AES Mixcolumns step

A new method for computing of circulant matrix is described herein that is based on the 2-point cyclic convolution matrix. This section consists of three subsections, in the first subsection describes different method of the multiplication over finite field for matrix multiplication that can be also applied to matrix operation. Besides, Scheme 1, which uses a two point cyclic matrix for reducing multiplication of the matrix product, and Scheme 2 uses 2 multiplied by any element in $GF(2^m)$ which is zero for reducing a multiplication. The coefficients of the polynomial $A(x)$ has the property $(a_0 + a_1 + a_2 + a_3) = r_2$, where a_j is over $GF(2^m)$, which can use lookup table method for reducing 4 multiplications. Lastly, Scheme 3 uses sum of the coefficients of the polynomial $A(x)$ that has the properties $(a_0 + a_1 + a_2 + a_3) = 1$, which reduce 4 multiplications in Scheme 3.

3.1 Multiplication over finite field

Let $a(x) = \sum_{i=0}^{m-1} a_i x^i$ and $b(x) = \sum_{i=0}^{m-1} b_i x^i$ be polynomial equation of degree m-1 in $GF(2^m)$, where $a_i, b_i \in \{0, 1\}$. It is well know that finite field addition is defined as:

$$c(x) = a(x) + b(x), \tag{1}$$

Note that the symbol of "+" is XOR bitwise operation so it does not need extra defined function in C programming. Finite field multiplication is defined as:

$$c(x) \equiv a(x) \cdot b(x) \bmod f(x), \tag{2}$$

where the AES algorithm with multiplication is irreducible polynomial $f(x) = x^8 + x^4 + x^3 + x + 1$. In (2), the Russian Peasant method can be written as a function in C programming as follows:

Russian Peasant method

```
unsigned char GFM(unsigned char a, unsigned char b){
unsigned char c = 0;
for( int i = 0; i < 8; i++){
      if (b & 1)
          c ^= a;
      if (a & 0x80)
          a = (a << 1) ^ 0x11b;
      else
          a <<= 1;
   b >>= 1;
   }
return p;
}
```

In (2), the proposed multiplication can be evaluated by using Horner's rule, according to the following recursive formula, $c(x) \equiv (((a_7Bx \bmod f(x) + a_6B)x^2 \bmod f(x) + a_5Bx \bmod f(x) + a_4B)$ $x^2 \bmod f(x) + \cdots + a_1Bx \bmod f(x) + a_0B$, where B is represented as the polynomial $b(x)$. Thus, an expression $(a_iBx \bmod f(x) + a_jB)$ can be represented as a lookup table as following $\mathrm{Bt}[a_i, a_j] = (a_iBx \bmod f(x) + a_jB)$, where $a_i, a_j \in GF(2)$. Let $\mathrm{Bt}[a_i, a_j]$ be c, the $c \leftarrow cx^2 \bmod f(x)$ can be represented as $c \leftarrow cx^2 + \mathrm{f}[c_{m-1}, c_{m-2}]$, where $\mathrm{f}[c_i, c_j] = c_i re(x)x + c_j re(x)$ and $re(x) \equiv x^m \bmod f(x)$ is a remainder polynomial (*e.g.*, $re(x) = x^4 + x^3 + x + 1$, binary 11001, Hex 0x1b). Horner's rule method is rewritten in C programming as shown below:

Horner's rule

```
unsigned char f[4]; unsigned char Bt[4];
unsigned char GFM(unsigned char a, unsigned char b){
   unsigned char c; f[0] = 0; f[1] = 0x1b; f[2] = 0x36; f[3] = 0x2d; Bt[0] = 0; Bt[1] = b;
   if (b & 0x80)
      Bt[2] = (b << 1) ^ 0x1b;
   else
      Bt[2] = (b << 1);
   Bt[3] = Bt[2] ^ b;
   c= Bt[(a >> 6) & 0x3];
   c=(c << 2) ^ f[c >> 6] ^ Bt[(a >> 4) & 0x3];
   c=(c << 2) ^ f[c >> 6] ^ Bt[(a >> 2) & 0x3];
   c=(c << 2) ^ f[c >> 6] ^ Bt[a & 0x3];
  return c;
}
```

As mentioned above, the two methods of multiplication can be used for making an 2D array GFMT[][] for lookup table method (*i.e.*, GFMT[*i*][*j*]=GFM(*i*,*j*) where $0 \le i, j \le 255$). An array

GFMT[][] needs 256*256=64K bytes for saving data. The lookup table method is shown as below:

Lookup table method

```
unsigned char GFM(unsigned char a, unsigned char b)
{
  unsigned char c=0;
  c=GTMT[a][b];
  return c;
}
```

3.2 Reducing multiplications in matrix multiplication

The AES MixColumns transformation, the modular product of $A(x)$ and $B(x)$, is presented as the four-term polynomial $D(x)$, defined as

$$D(x) \equiv A(x)B(x) \bmod T(x) \tag{3}$$

where $T(x) = x^4 + 1$, $A(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ and $B(x) = b_3x^3 + b_2x^2 + b_1x + b_0$, for $a_i, b_i \in GF(2^m)$. By (3), there is a circulant matrix form as:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{4}$$

In (4), the matrix D is a product of matrices A and B, which requires 16 multiplications and 12 additions (16M, 12A) listed below:

(16M, 12A)

$$d_0 = a_0b_0 + a_3b_1 + a_2b_2 + a_1b_3$$
$$d_1 = a_1b_0 + a_0b_1 + a_3b_2 + a_2b_3$$
$$d_2 = a_2b_0 + a_1b_1 + a_0b_2 + a_3b_3$$
$$d_3 = a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3$$

Using the two-point cyclic convolution matrix property for 2×2 matrices multiplication is given by:

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} a_0(b_0+b_1)+(a_0+a_1)b_1 \\ a_0(b_0+b_1)+(a_0+a_1)b_0 \end{bmatrix}. \tag{5}$$

Hence, the method only requires 3 multiplications and 4 additions (3M, 3A) as shown in Table 1.

Table 1: The two-point cyclic convolution method with (3M, 3A).

$s_0 = a_0(b_0+b_1)$	$s_1 = a_0 + a_1$
$y_0 = s_0 + s_1 b_1$	$y_1 = s_0 + s_1 b_0$

In Table 1, two entries a_0 and a_1 are fix data, the item $s_1 = a_0 + a_1$ can be precomputed in the program. Thus, the 2-point cyclic matrix method only uses 3 multiplications and 3 additions. If the matrices $A = \begin{bmatrix} a_0 & a_3 \\ a_1 & a_0 \end{bmatrix}$ is not 2-point cyclic matrix, that product of the matrix *A* and *B* is given by

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 \\ a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} a_0 b_0 + a_3 b_1 \\ a_1 b_0 + a_0 b_1 \end{bmatrix}. \tag{6}$$

Theorem 1 Let *A* be any *n*×*n* cyclic matrix, where $n = n_1 \times n_2$ and $\mathrm{GCD}(n_1, n_2) \neq 1$, then the matrix *A* can be partitioned into a cyclic $n_1 \times n_1$ matrix, in which entries are $n_2 \times n_2$ submatrix. It is similar to the proof by Winograd (1978). Using (4), by Theorem 1, the four-point cyclic matrix can be partitioned as,

$$\begin{bmatrix} d_0 \\ d_1 \\ \hline d_2 \\ d_3 \end{bmatrix} = \left[\begin{array}{cc|cc} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ \hline a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{array}\right] \begin{bmatrix} b_0 \\ b_1 \\ \hline b_2 \\ b_3 \end{bmatrix}. \tag{7}$$

From (7), it can be rewritten as

$$\begin{bmatrix} D_0 \\ D_1 \end{bmatrix} = \begin{bmatrix} A_0 & A_1 \\ A_1 & A_0 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \end{bmatrix}, \tag{8}$$

where $D_0 = \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}, D_1 = \begin{bmatrix} d_2 \\ d_3 \end{bmatrix}, A_0 = \begin{bmatrix} a_0 & a_3 \\ a_1 & a_0 \end{bmatrix}, A_1 = \begin{bmatrix} a_2 & a_1 \\ a_3 & a_2 \end{bmatrix}, B_0 = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$, and $B_1 = \begin{bmatrix} b_2 \\ b_3 \end{bmatrix}$. In (8), it can be used to reduce the multiplications by (5) form as follows:

$$\begin{bmatrix} D_0 \\ D_1 \end{bmatrix} = \begin{bmatrix} A_0(B_0 + B_1) + (A_0 + A_1)B_1 \\ A_0(B_0 + B_1) + (A_0 + A_1)B_0 \end{bmatrix} = \begin{bmatrix} F + G \\ F + H \end{bmatrix}, \tag{9}$$

where $F = A_0(B_0 + B_1) = \begin{bmatrix} a_0 & a_3 \\ a_1 & a_0 \end{bmatrix}\begin{bmatrix} b_0 + b_2 \\ b_1 + b_3 \end{bmatrix}$ $G = (A_0 + A_1)B_1 = \begin{bmatrix} a_0 + a_2 & a_3 + a_1 \\ a_1 + a_3 & a_0 + a_2 \end{bmatrix}\begin{bmatrix} b_2 \\ b_3 \end{bmatrix}$, and $H = (A_0 + A_1)B_0 = \begin{bmatrix} a_0 + a_2 & a_3 + a_1 \\ a_1 + a_3 & a_0 + a_2 \end{bmatrix}\begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$. The matrix F can be form by (6), and matrix G and matrix H are form by (5) yields:

$$\begin{aligned} F &= \begin{bmatrix} a_0(b_0 + b_2) + a_3(b_1 + b_3) \\ a_1(b_0 + b_2) + a_0(b_1 + b_3) \end{bmatrix} \\ G &= \begin{bmatrix} (a_0 + a_2)(b_2 + b_3) + ((a_0 + a_2) + (a_3 + a_1))b_3 \\ (a_0 + a_2)(b_2 + b_3) + ((a_0 + a_2) + (a_3 + a_1))b_2 \end{bmatrix} \\ H &= \begin{bmatrix} (a_0 + a_2)(b_0 + b_1) + ((a_0 + a_2) + (a_3 + a_1))b_1 \\ (a_0 + a_2)(b_0 + b_1) + ((a_0 + a_2) + (a_3 + a_1))b_0 \end{bmatrix} \end{aligned} \tag{10}$$

Obviously, the matrices F, G, and H are combination of the sets with element b_i. Rewrite the terms in $s_0 = b_0 + b_2$, $s_1 = b_1 + b_3$, $s_2 = a_0 s_0 + a_3 s_1$, $s_3 = a_1 s_0 + a_0 s_1$, $s_4 = b_2 + b_3$, and $s_5 = b_0 + b_1$ as follows:

$$F = \begin{bmatrix} s_2 \\ s_3 \end{bmatrix}, \quad G = \begin{bmatrix} (a_0 + a_2)s_4 + ((a_0 + a_2) + (a_3 + a_1))b_3 \\ (a_0 + a_2)s_4 + ((a_0 + a_2) + (a_3 + a_1))b_2 \end{bmatrix}, \text{ and}$$

$$H = \begin{bmatrix} (a_0 + a_2)s_5 + ((a_0 + a_2) + (a_3 + a_1))b_1 \\ (a_0 + a_2)s_5 + ((a_0 + a_2) + (a_3 + a_1))b_0 \end{bmatrix}.$$

Next, the matrix G and matrix H are replaced with $w_0 = a_0 + a_2$ and $w_1 = a_3 + a_1$. Thus, the matrix G and H matrix can be given as

$$G = \begin{bmatrix} r_0 + r_2 b_3 \\ r_0 + r_2 b_2 \end{bmatrix} \text{ and } H = \begin{bmatrix} r_1 + r_2 b_1 \\ r_1 + r_2 b_0 \end{bmatrix},$$

where $r_0 = w_0 s_4$, $r_1 = w_0 s_5$, and $r_2 = w_0 + w_1$. Finally, the four-point cyclic matrix method can be obtained as a new matrix form

$$\begin{bmatrix} D_0 \\ D_1 \end{bmatrix} = \begin{bmatrix} F+G \\ F+H \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} s_2 + r_0 + r_2 b_3 \\ s_3 + r_0 + r_2 b_2 \\ s_2 + r_1 + r_2 b_1 \\ s_3 + r_1 + r_2 b_0 \end{bmatrix}.$$

In the simplified case, the MixColumns transformation can be performed by 10 multiplications and 17 additions. Two items $w_0 = a_0 + a_2, w_1 = a_3 + a_1$ and $r_2 = w_0 + w_1$ are known because the value a_i of the coefficients of polynomial $A(x)$, can be precomputed in the program. So that the method only uses 10 multiplications and 14 additions, that is remarked as **(10M, 14A)**.

Scheme 1. (10M, 14A)
$s_0 = b_0 + b_2,\ s_1 = b_1 + b_3$
$s_2 = a_0 s_0 + a_3 s_1,\ s_3 = a_1 s_0 + a_0 s_1, w_0 = a_0 + a_2,\ w_1 = a_3 + a_1$
$r_0 = w_0(b_2 + b_3)\ \ r_1 = w_0(b_0 + b_1),\ \ r_2 = w_0 + w_1$
$d_0 = s_2 + r_0 + r_2 b_3$
$d_1 = s_3 + r_0 + r_2 b_2$
$d_2 = s_2 + r_1 + r_2 b_1$
$d_3 = s_3 + r_1 + r_2 b_0$

3.3 Reducing multiplications by multiply 2

The matrix product $A_0 B_0 = \begin{bmatrix} a_0 & a_3 \\ a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$ can be further simplified by properties of addition over $GF(2^m)$. Adding two entries of $2a_0 b_1 = 0$ and $2a_0 b_0 = 0$ are into matrix product $A_0 B_0$ as follows:

$$A_0 B_0 = \begin{bmatrix} a_0 b_0 + a_3 b_1 + 2a_0 b_1 \\ a_1 b_0 + a_0 b_1 + 2a_0 b_0 \end{bmatrix} = \begin{bmatrix} a_0(b_0 + b_1) + (a_0 + a_3) b_1 \\ a_0(b_0 + b_1) + (a_0 + a_1) b_0 \end{bmatrix}. \quad (11)$$

In Scheme 2, the matrix F was replaced by (6). Now, the matrix F is replaced by (11) to obtain the following matrix

$$F = \begin{bmatrix} a_0 & a_3 \\ a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 + b_2 \\ b_1 + b_3 \end{bmatrix} = \begin{bmatrix} a_0((b_0 + b_2) + (b_1 + b_3)) + (a_0 + a_3)(b_1 + b_3) \\ a_0((b_0 + b_2) + (b_1 + b_3)) + (a_0 + a_1)(b_0 + b_2) \end{bmatrix}.$$

$$F = \begin{bmatrix} a_0(s_0 + s_1) + t_0 s_1 \\ a_0(s_0 + s_1) + t_1 s_0 \end{bmatrix} = \begin{bmatrix} t_2 + t_0 s_1 \\ t_2 + t_1 s_0 \end{bmatrix},$$

where $s_0 = b_0 + b_2$, $s_1 = b_1 + b_3$, $t_0 = a_0 + a_3$, $t_1 = a_0 + a_1$, and $t_2 = a_0(s_0 + s_1)$.

In Scheme 1, the two items $s_2 = a_0 s_0 + a_3 s_1$ and $s_3 = a_1 s_0 + a_0 s_1$ can be replaced as $s_2 = t_2 + t_0 s_1$, $s_3 = t_2 + t_1 s_0$, $t_0 = a_0 + a_3$, $t_1 = a_0 + a_1$, and $t_2 = a_0(s_0 + s_1)$ for computing MixColumns transformation. Consequently, in Scheme 1, each $r_2 b_i$ term can be replaced with lookup table method of $tc[b_i] = r_2 b_i$, namely, constant multiplication doesn't require computing multiplications as it did. It needs 256 bytes of memory, which is called Scheme 2. Scheme 1 can further be rewritten as follows:

Scheme 2. (5M, 15A) (It needs 256 bytes as lookup table)

$s_0 = b_0 + b_2$, $s_1 = b_1 + b_3$
$t_0 = a_0 + a_3$, $t_1 = a_0 + a_1$, $t_2 = a_0(s_0 + s_1)$
$s_2 = t_2 + t_0 s_1$, $s_3 = t_2 + t_1 s_0$, $w_0 = a_0 + a_2$
$r_0 = w_0(b_2 + b_3)$, $r_1 = w_0(b_0 + b_1)$
$d_0 = s_2 + r_0 + tc[b_3]$
$d_1 = s_3 + r_0 + tc[b_2]$
$d_2 = s_2 + r_1 + tc[b_1]$
$d_3 = s_3 + r_1 + tc[b_0]$

In Scheme 2, it uses only 5 multiplications and 18 additions with 256 bytes of memory for matrix multiplication. Obviously, if the coefficients of the polynomial $A(x)$ have the equality $a_0+a_3+a_2+a_1=1$ in AES standard, then the property would make $r_2=w_0+w_1=1$, based on Scheme 2. Consequently, the $r_2 = 1$ doesn't require lookup table computing as it did in Scheme 2 (*e.g.*, $tc[b_i] = r_2 b_i = 1 \cdot b_i$), does not need memory used in embedded system, so that the method can be rewritten as Scheme 3. In Scheme 3, there are three items $w_0 = a_0 + a_2$, $t_0 = a_0 + a_3$, and $t_1 = a_0 + a_1$, which can be precomputed in the program, so that the method only used 5 multiplications and 15 additions, namely, **(5M, 15A)**.

Scheme 3. (5M, 15A)
$s_0 = b_0 + b_2,\ s_1 = b_1 + b_3$
$t_0 = a_0 + a_3,\ t_1 = a_0 + a_1,\ t_2 = a_0(s_0 + s_1)$
$s_2 = t_2 + t_0 s_1,\ s_3 = t_2 + t_1 s_0,\ w_0 = a_0 + a_2$
$r_0 = w_0(b_2 + b_3),\ r_1 = w_0(b_0 + b_1)$
$d_0 = s_2 + r_0 + b_3$
$d_1 = s_3 + r_0 + b_2$
$d_2 = s_2 + r_1 + b_1$
$d_3 = s_3 + r_1 + b_0$

Note that the traditional matrix multiplication (16M, 12A), Scheme 1 and Scheme 2 can be used for evaluating any diverse 4×4 circulant matrices in AES MixColumns-InvMixColumns transformation. However, Scheme 3 is only for the polynomial $A(x)$ and $A^{-1}(x)$ that each of the sum of the coefficients is 1.

4. Finding a pair of the circulant matrices

In this section, the inverse polynomial $A(x)$ is called polynomial $E(x)$ for AES InvMixColumns transformation. The modular product of $E(x)$ and $D(x)$ is presented as the four-term polynomial $B(x)$, defined as $B(x) \equiv D(x)E(x) \bmod x^4 - 1$, where the polynomial $E(x) = e_3x^3 + e_2x^2 + e_1x + e_0$ is the $A^{-1}(x)$ that might be expressed the cyclic matrix E form as follows:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} e_0 & e_3 & e_2 & e_1 \\ e_1 & e_0 & e_3 & e_2 \\ e_2 & e_1 & e_0 & e_3 \\ e_3 & e_2 & e_1 & e_0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}. \tag{12}$$

Now, if matrix E is inverse matrix A, the product AE is equal to an identity matrix I as below:

$$\begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \times \begin{bmatrix} e_0 & e_3 & e_2 & e_1 \\ e_1 & e_0 & e_3 & e_2 \\ e_2 & e_1 & e_0 & e_3 \\ e_3 & e_2 & e_1 & e_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{13}$$

Therefore, an efficient way to find the inverse of the matrix A is need, regardless of whether the matrix A has a matrix inverse or not. The finding inverse of a 4 by 4 matrix E procedure is called Find_inv_matrix().

Find_inv_matrix()	
1	Initial value a_0=0, a_3=0, a_2=0, a_1=0.
2	$a_0 \leftarrow a_0+1$, if $a_0 > 255$, go stop.
3	$a_3 \leftarrow a_3+1$, if $a_1 > 255$, a_3=1, go to step 2.
4	$a_2 \leftarrow a_2+1$, if $a_2 > 255$, a_2=1, go to step 3.
5	$a_1 \leftarrow a_1+1$, if $a_3 > 255$, a_1=1, go to step 4.
6	If $(a_0+a_3+a_2+a_1)$ is not equal to r_2, go to step 5.
7	Calculating the coefficients of the polynomial E. $(e_0 = M_{0,0}/r_2^4, e_3 = M_{0,1}/r_2^4, e_2 = M_{0,2}/r_2^4, e_1 = M_{0,3}/r_2^4)$
8	Save a_0, a_3, a_2, a_1 and e_0, e_3, e_2, e_1. Go to step 5.

Here the sum of coefficients of the polynomial $A(x)$ is the value of r_2 range from 1 to 255. The det(A) is determinant of the matrix A, and the adj(A) is the adjugate of matrix A. The procedure to obtain all the coefficients of polynomial $A(x)$ by computing XOR is equal to r_2 and it needs to evaluate the determinant of the matrix A is not equal to zero. In above procedure using the property $\sum_{i=1}^{4} a_i = r_2$ for testing all det(A) has property of det(A)=r_2^4, the meaning is not required to compute det(A). The det(A)=r_2^4 can also be proved as below:

$$\det(A) = a_0 M_{0,0} + a_3 M_{0,1} + a_2 M_{0,2} + a_1 M_{0,3},$$

where

$$a_0 M_{0,0} = a_0 a_0 a_0 a_0 + a_0 a_1 a_1 a_2 + a_0 a_2 a_3 a_3 + a_0 a_2 a_0 a_2 + a_0 a_3 a_1 a_0 + a_0 a_0 a_3 a_1,$$
$$a_3 M_{0,1} = a_3 a_1 a_0 a_0 + a_3 a_2 a_1 a_2 + a_3 a_3 a_3 a_3 + a_3 a_2 a_0 a_3 + a_3 a_3 a_2 a_0 + a_3 a_1 a_1 a_3,$$
$$a_2 M_{0,2} = a_2 a_1 a_1 a_0 + a_2 a_2 a_2 a_2 + a_2 a_3 a_3 a_0 + a_2 a_2 a_1 a_3 + a_2 a_0 a_2 a_0 + a_2 a_1 a_2 a_3,$$
$$a_1 M_{0,3} = a_1 a_1 a_1 a_1 + a_1 a_2 a_2 a_3 + a_1 a_3 a_0 a_0 + a_1 a_3 a_1 a_3 + a_1 a_0 a_2 a_1 + a_1 a_1 a_2 a_0.$$

Now, the determinant of the matrix A is given as

$$\det(A) = a_0^4 + a_3^4 + a_2^4 + a_1^4. \tag{14}$$

Lemma 1 Let $a_0, a_1, a_2, \ldots, a_i$ be elements in the finite field $GF(p^m)$, $(a_0 + a_1 + \cdots + a_i)^{P^j} = a_0^{P^j} + a_1^{P^j} + \cdots + a_i^{P^j}$.

The proof of Lemma 1 is given by Wicker (1995). It immediately follows from Lemma 1 that (14) becomes

$$\det(A) = (a_0 + a_1 + a_2 + a_3)^4. \tag{15}$$

Therefore, $a_0 + a_1 + a_2 + a_3 = r_2$ is submitted into (15) then $\det(A) = r_2^4$ can be obtained. In Find_inv_matrix() the calculation of det(*A*) and adj(*A*) are needed. In fact, det(*A*) and adj(*A*) require more multiplications for evaluating M_{ij} over $GF(2^m)$. The adjugate of *A* is defined as follows:

$$\text{adj}\left(\begin{bmatrix} a_{0(0,0)} & a_{3(0,1)} & a_{2(0,2)} & a_{1(0,3)} \\ a_{1(1,0)} & a_{0(1,1)} & a_{3(1,2)} & a_{2(1,3)} \\ a_{2(2,0)} & a_{1(2,1)} & a_{0(2,2)} & a_{3(2,3)} \\ a_{3(3,0)} & a_{2(3,1)} & a_{1(3,2)} & a_{0(3,3)} \end{bmatrix}\right) = C^T, \quad \text{where } C = \begin{bmatrix} M_{(0,0)} & M_{(0,1)} & M_{(0,2)} & M_{(0,3)} \\ M_{(1,0)} & M_{(1,1)} & M_{(1,2)} & M_{(1,3)} \\ M_{(2,0)} & M_{(2,1)} & M_{(2,2)} & M_{(2,3)} \\ M_{(3,0)} & M_{(3,1)} & M_{(3,2)} & M_{(3,3)} \end{bmatrix}.$$

The minor of entry a_{ij} is denoted by M_{ij}, where *C* is the matrix of cofactors, and C^T is transpose of the matrix *C*.

$$E = A^{-1} = \frac{C^T}{\det(A)}, \tag{16}$$

where det(*A*) is determinant of the matrix *A* and

$$C^T = \begin{bmatrix} M_{(0,0)} & M_{(1,0)} & M_{(2,0)} & M_{(3,0)} \\ M_{(0,1)} & M_{(1,1)} & M_{(2,1)} & M_{(3,1)} \\ M_{(0,2)} & M_{(1,2)} & M_{(2,2)} & M_{(3,2)} \\ M_{(0,3)} & M_{(1,3)} & M_{(2,3)} & M_{(3,3)} \end{bmatrix}.$$

The matrix C^T using first rows entries are transpose of the cofactor matrix *A*. the matrix *E* of first row entries are given as

$$e_0 = (a_0a_0a_0 + a_1a_1a_2 + a_2a_3a_3 + a_2a_0a_2 + a_3a_1a_0 + a_0a_1a_3)/r_2^4,$$
$$e_3 = (a_3a_0a_0 + a_1a_1a_1 + a_2a_3a_2 + a_1a_0a_2 + a_2a_1a_0 + a_3a_1a_3)/r_2^4,$$
$$e_2 = (a_3a_3a_0 + a_0a_1a_1 + a_2a_2a_2 + a_1a_3a_2 + a_2a_0a_0 + a_3a_1a_2)/r_2^4,$$
$$e_1 = (a_3a_3a_3 + a_0a_0a_1 + a_1a_2a_2 + a_1a_3a_1 + a_2a_0a_3 + a_3a_0a_2)/r_2^4.$$

When the sum of coefficients of polynomial $A(x)$ has a property $a_0 + a_1 + a_2 + a_3 \neq 0$, the sum of coefficients of polynomial $E(x)$ has a property $e_0 + e_1 + e_2 + e_3 \neq 0$. So, if the sum of first rows entries of the matrix *A* is r_2, (*i.e.*, $a_0 + a_1 + a_2 + a_3 = r_2$), then det(*A*)=$r_2^4$ and the sum of first rows entries in the matrix A^{-1} (or the matrix *E*) is r_2^3/r_2^4. Since the sum of the coefficients of the polynomial $A(x)$ is r_2 (*i.e.*, $a_0 + a_1 + a_2 + a_3 = r_2$), the sum of the coefficients of the polynomial $E(x)$ is r_2^3/r_2^4. The matrix $E = C^T/r_2^4$ also has a cyclic matrix property. Therefore, finding the inverse matrix procedure can obtain the Find_inv_matrix() function. Now, the proposed procedure can cause an exhausting search for the sum of the coefficients of the polynomial $A(x)$ that has the property $a_0+a_3+a_2+a_1=r_2=1$ and the coefficients of the polynomial $A(x)^{-1} = E(x)$ also has the sum $e_0+e_3+e_2+e_1=1$. There are 16,516,604 a pair of entries that can be chosen to save into both of Table A and Table E. The sizes of each table are 2^m, where *m*-bit taken from a given key of size *n*-bit. For example, taken *m*=4 bits from key length as direct addressing Table A and Table B listing in Table 2, which the sizes of each table requires $2^m \times 4$=66 Bytes to store in memory.

Table 2: The sum of the coefficients is just the value of the polynomial in r_2=1.

Items	Table A $a_0+a_3+a_2+a_1=r_2=1$				Table B $e_0+e_3+e_2+e_1=r_2^3/r_2^4=1$			
	a_0	a_3	a_2	a_1	e_0	e_3	e_2	e_1
1	02	03	01	01	0e	0b	0d	09
2	02	0b	04	0c	7c	60	7a	67
3	03	02	0f	0f	e2	b2	ee	bf
4	03	04	05	03	7d	6f	7b	68
5	04	06	0d	0e	72	30	7b	38
6	04	07	03	01	68	7f	6f	79
7	05	09	0a	07	24	7c	2b	72
8	05	0a	01	0f	41	5f	45	5a
9	06	02	08	0d	7d	2c	73	23
10	06	03	08	0c	7d	2d	73	22
11	07	02	0d	09	93	d3	99	d8
12	07	03	0b	0e	e6	b3	ea	be
13	08	08	11	10	7a	21	63	39

14	08	08	13	12	a0	ff	bb	e5
15	09	11	0a	13	05	19	06	1b
16	09	12	11	0b	38	79	20	60

If the range of r_2 is between 1 and 255, then a pair of the entries can be found 255×16,516,604=4,211,734,020 pairs, which the test run **Find_inv_matrix()** function. For example, the value of r_2 = 2, 3, 4, 5, 6, 7, 8 some of a pair of the entries are listed in Table 3.

Table 3: The value of r_2=2, 3, 4, 5, 6, 7, 8.

Items	$r_2=a_0+a_3+a_2+a_1=2$				$e_0+e_3+e_2+e_1=r_2^3/r_2^4=8d$			
	a_0	a_3	a_2	a_1	e_0	e_3	e_2	e_1
1	1e	1e	20	22	98	32	1a	3d
2	1e	1e	21	23	14	56	5d	92
3	1e	1e	22	20	3d	1a	32	98
4	1e	1e	23	21	92	5d	56	14
Items	$r_2=a_0+a_3+a_2+a_1=3$				$e_0+e_3+e_2+e_1=r_2^3/r_2^4=f6$			
	a_0	a_3	a_2	a_1	e_0	e_3	e_2	e_1
1	1e	1e	20	23	54	63	fc	3d
2	1e	1e	21	22	82	72	78	7e
3	1e	1e	22	21	7e	78	72	82
4	1e	1e	23	20	3d	fc	63	54
Items	$r_2=a_0+a_3+a_2+a_1=4$				$e_0+e_3+e_2+e_1=r_2^3/r_2^4=cb$			
	a_0	a_3	a_2	a_1	e_0	e_3	e_2	e_1
1	1e	1e	20	24	0b	6a	a6	0c
2	1e	1e	21	25	2c	50	f5	42
3	1e	1e	22	26	6c	79	29	f7
4	1e	1e	23	27	d5	dd	e4	27
Items	$r_2=a_0+a_3+a_2+a_1=5$				$e_0+e_3+e_2+e_1=r_2^3/r_2^4=52$			
	a_0	a_3	a_2	a_1	e_0	e_3	e_2	e_1
1	1e	1e	20	25	2c	fe	aa	2a
2	1e	1e	21	24	a1	57	93	37
3	1e	1e	22	27	c2	80	37	27
4	1e	1e	23	26	77	11	36	02
Items	$r_2=a_0+a_3+a_2+a_1=6$				$e_0+e_3+e_2+e_1=r_2^3/r_2^4=7b$			
	a_0	a_3	a_2	a_1	e_0	e_3	e_2	e_1
1	1e	1e	20	26	cc	47	E6	16
2	1e	1e	21	27	96	ab	25	63
3	1e	1e	22	24	43	26	40	5e
4	1e	1e	23	25	af	7c	35	9d

Items	$r_2=a_{0+}a_{3+}a_{2+}a_1=7$				$e_{0+}e_{3+}e_{2+}e_1=r_{,}^{3}/r_{,}^{4}$=d1			
	a_0	a_3	a_2	a_1	e_0	e_3	e_2	e_1
1	1e	1e	20	27	85	90	ad	69
2	1e	1e	21	26	ff	35	fc	e7
3	1e	1e	22	25	93	d7	ed	78
4	1e	1e	23	24	1f	84	4a	00
Items	$r_2=a_{0+}a_{3+}a_{2+}a_1=8$				$e_{0+}e_{3+}e_{2+}e_1=r_{,}^{3}/r_{,}^{4}$=e8			
	a_0	a_3	a_2	a_1	e_0	e_3	e_2	e_1
1	1e	1e	20	28	74	42	94	4a
2	1e	1e	21	29	f0	a0	0d	b5
3	1e	1e	22	2a	9c	29	46	1b
4	1e	1e	23	2b	bf	6c	78	43

5. Simulation results

The various methods of multiplication execute time running 10,000,000 times in Intel Core i5-5200 and the results are given in Table 4.

Table 4: Different methods of multiplication executes time.

Finite field Multipcation	**Execution time**	**Memory size**
Lookup table	0.031 s	64 Kbytes
Russian Peasant	0.327 s	0 bytes
Horner’s rule	0.098 s	8 bytes

In Figure 2, the symbol “M” represents the multiplications and the symbol “A” represents the additions. There are many methods for computing matrix multiplication, in which using the multiplication of lookup table method for Scheme 3 is very fast when compared other methods as shown in Table 5. However, lookup table method requires 64 Kbytes memory that is not suitable for many embedded systems with limited resources and Scheme 3 is only used to the sum of the coefficients of the polynomial $A(x)$ and the polynomial $E(x)$, that the both of results is 1.

Table 5: Computing time of MixColumns tansformation.

Multiplication algorithms	**(16M, 12A)**	**Scheme 1**	**Scheme 2**	**Scheme 3**	Reducing Percentage (16M, 12A)- Scheme 3 /(16M, 12A)×100%
Lookup table	1.93	1.42	0.96	0.92	52%
Horner's rule	6.35	4.21	2.41	2.3	63%
Russian Peasant	12.52	8.23	4.37	4.3	65%

Using the multiplication based on several algorithms in $GF(2^m)$ and Scheme 3 are for evaluating encryption and decryption procedure running 1,000,000 times state with different AES key lengths, where the state is 4×4 bytes. The **keys** with lengths **128, 192, 256** bits run cipher-InCipher average execution time as shown in Figure 2. The trade-off between memory size and speed performance for AES Cipher-InvCipher, would suggest that algorithms using Horner's rule method and Scheme 3 is better suitable for embedded system. The **keys** with lengths **128, 192, 256** bits can be reduced ~60%, ~60%, and ~59%, respectively, with Horner's rule-based multiplication, faster than (16M, 12A) method.

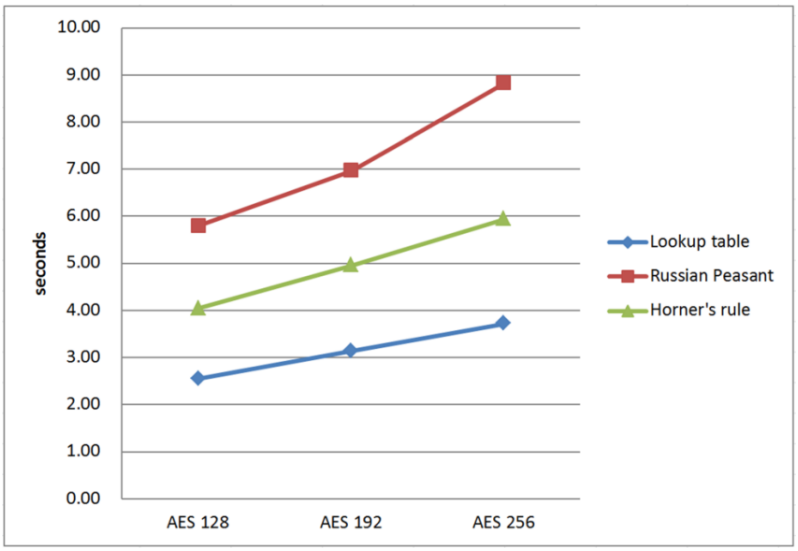

Figure 2: AES execution time with the different key lengths

Finally, using Scheme 3 and Horner's rule-based multiplication runs on STM32L476VG discovery board with FPU ARM Cortex-M4 MCU 80 MHz. The encryption and decryption procedure are running 10000 states, where a state is 4×4 bytes. The Cipher-InvCipher counts the elapsed CPU cycles as shown in Table 6. It has to divide running cycles times with the CPU clock frequency to obtain a value in seconds. As 32 bit value, this can overflow quite fast at higher clock frequencies (i.e. 53.68 seconds at 80 MHz). Fortunately, testing execution time for different keys lengths is not overflow 53.68 seconds. Using Horner's rule-based

multiplication and Scheme 3 for testing the **keys** with lengths **128, 192, 256** bits can be reduced ~59.61%, ~59.72%, and ~59.83% respectively, faster than (16M, 12A) method. AES execution time using Horner's rule-based multiplication and Scheme 3 are measure of the different key lengths as shown in Figure 3.

Table 6: Execution CPU cycles on STM32L476VG MCU.

AES	(16M,12A)		(Cyclic times)
keys lengths	**Cipher**	**InvCipher**	**Ave. Cycle times**
128 bits	423583003	422833181	423208092
192 bits	516154588	516009350	516081969
256 bits	609845413	608996752	609421083
AES	Scheme 3		(Cyclic times)
keys lengths	**Cipher**	**InvCipher**	**Average Cycle times**
128 bits	171098400	170750823	170924612
192 bits	207938492	207746120	207842306
256 bits	245008423	244576184	244792304

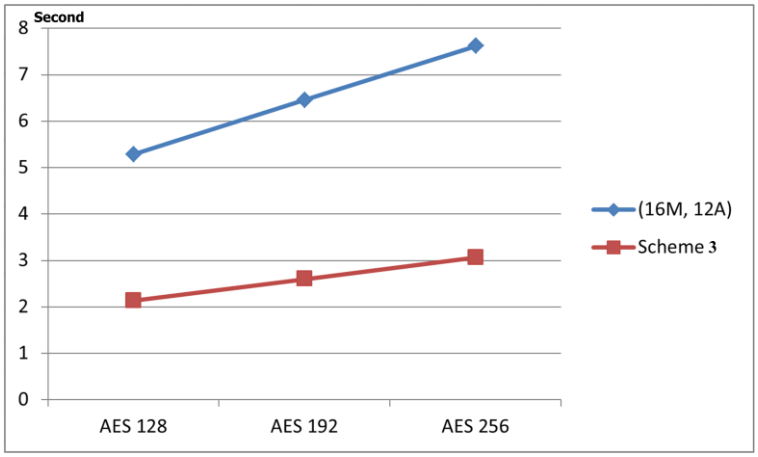

Figure 3: AES execution time on STM32L476VG MCU

6. Conclusion

In summary, it is demonstrated herein that the computational complexity matrix multiplication over GF(2^8) can be minimized by 2-point cyclic convolution property. In comparison for each of the schemes, Scheme 3 can be run on STM32L476VG discovery board with more reduced ~60% time than (16M, 12A) method using in

MixColumns-InvMixColumns step. In some instances, the sum of coefficients of the polynomial is not 1 which can use for Scheme 1 and Scheme 2 (Note: If the memory is enough on an embedded system). Scheme 1 and Scheme 2 have many a pair of the coefficients than Scheme 3 for encryption and decryption. Nevertheless, Scheme 3 still has 16,516,604 a pair of entries that can be chosen to save into both of Table A and Table E. The proposed method Figure 1 can be an extended procedure of AES with more variations in circulant matrix for enhanced security of data transmissions. In the future, Scheme 3 may also be used for designing VLSI circuits to save the amount of logic gates in diverse MixColumns-InvMixColumns transformation.

References

[1] A. Biryukov, D. Khovratovich, "Related-Key cryptanalysis of the full AES-192 and AES-256," In: Matsui, M. (ed.) ASIACRYPT 2009 LNCS, (5912): 1-18 https://eprint.iacr.org/2009/317.pdf

[2] A. Maximov, "AES MixColumn with 92 XOR gates," Cryptology ePrint Archive, Report 2019/833, https://eprint.iacr.org/2019/833, 2019.

[3] A. Stepanov, D. Rose, *From mathematics to generic programming.* Pearson Education, New York, 3st edn, 2015.

[4] B. Langenberg, H. Pham and R. Steinwandt, "Reducing the Cost of Implementing the Advanced Encryption Standard as a Quantum Circuit," in *IEEE Transactions on Quantum Engineering*, vol. 1, no. 2500112, pp. 1-12, 2020.

[5] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall, "*Twofish: a 128-Bit block cipher*" Available NIST's AES homepage, https://www.schneier.com/academic/paperfiles/paper-twofish-paper.pdf, 1998

[6] C. H. Yang and Y. S. Chien, "FPGA Implementation and Design of a Hybrid Chaos-AES Color Image Encryption Algorithm," *Symmetry*, vol. 12, no. 2, 187(pp. 1-17), 2020..

[7] D. Augot, M. Finiasz, "Exhaustive search for small dimension recursive MDS diffusion layers for block ciphers and hash functions," *IEEE Int. Conf. on Information Theory*, Turkey, July, pp. 1551-1555, 2013.

[8] D. Yin and Y. Gao, "A new construction of lightweight MDS matrices," *IEEE Int. Conf. on Computer and communication*, China, December, pp. 2560-2563, 2017.

[9] F. J. MacWilliams and N. J. Sloane, *The theory of error-correcting codes*: North-Holland, 1nd edn, 1978.

[10] G. Murtaza and N. Ikram, "Direct exponent and scalar multiplication classes of an MDS matrix," IACR, http://eprint.iacr.org/2011/151, 2011.

[11] G. Selimis, A. Fournaris, and O. Koufopavlou, "Applying low power techniques in AES MixColumn/InvMixColumn transformations," *IEEE Int. Conf, Electronics, Circuits and Systems ICECS'06*, France, December, pp. 10-13, 2006.

[12] I. Mahboob, "Lookup table based multiplication technique for $GF(2^m)$ with cryptographic significance," *IEE Proc. Commun*, vol. 152, no. 6, pp.965-974, 2005.

[13] J. Daemen, V. Rijmen, AES proposal: Rijndael document version 2, 1999.

[14] J. Lacan and J. Fimes, "Systematic MDS erasure codes based on vandermonde matrices," *IEEE Trans. Commun. Lett.*, vol. 8, no. 9, pp. 570-572, 2004

[15] M. H. Jing and Z. H. Chen, "System for high-speed and diversified AES using FPGA," *Microprocessors and Microsystems*, vol. 31, no. 12, pp. 94–102, 2006.

[16] M. H. Jing, J. H. Chen, and Z. H. Chen, "Diversified Mixcolumn transformation of AES," *Proc. Int. Conf. ICICS 2007*, Singapore, December, pp. 10-13, 2007.

[17] P. Junod, S. Vaudenay, Perfect diffusion primitives for block ciphers. building efficient MDS Matrices. Federalede Lausanne, Switzerland, 2004.

[18] S. Rizwana, B. Jagrutee, and B. Pragna, "Securing E-healthcare records on Cloud Using Relevant data classification and Encryption," *International Journal Of Engineering And Computer Science*, vol. 6, no. 2, pp. 20215-20220, 2017

[19] W. S. Pienaar and M. Reza, "Survey on A Smart Health Monitoring System Based on Context Awareness Sensing," *Communications_of_the_CCISA*, vol. 25, no. 1, pp. 1-13, 2019.

[20] Y. Wang, L. Ni, C. H. Chang, and H. Yu, "DW-AES: A Domain-Wall Nanowire-Based AES for high throughput and energy-efficient data encryption in Non-Volatile memory," *IEEE T INF FOREN SEC*, vol. 11, no. 11, pp. 2426-2440, 2016.