Regular Paper

Communications of the CCISA
Vol.30 No. 2 May. 2024

The fastest matrices multiplication using involutory matrix in AES MixColumns-InvMixcolumns transformation

San Yuan Wang[1], Fu Jung Kan[2], Yan Haw Chen[3*], Shui Hsiang Su[4],
Ling Ling Dai[5], Kes Shan Lin[6]
[1,3,5,6]Dept. of Information Engineering, I-Shou University, Kaohsiung, Taiwan 84008.
[2,4]Dept. of Electronic Engineering, I-Shou University, Kaohsiung, Taiwan 84008.
[1]sywang@isu.edu.tw, [2]kanfujung@gmail.com, [3]yanchen@isu.edu.tw, [4]shsu@isu.edu.tw
[5]isu1113081a@cloud.isu.edu.tw, [6]isu11103093a@cloud.isu.edu.tw

Abstract

The traditional computer will be difference attacks by future quantum computer. Now, AES seems a resistant primitive in the post quantum world, with a bigger security margin against quantum computer attacks. In this paper, the key idea here is to propose a method with a variations $2^k\times 2^k=n\times n$ involutory matrix for enhancing diffusion data in AES MixColumns-InvMixColumns step that the Branch Number of the confusion capability is increased n+1 where $k\geq 1$ integer number, but the matrix multiplication is required a lot of the finite field multiplications. A 16×16 involutory matrix for matrix multiplication needs 256 multiplications and 240 additions for using encryption and decryption in AES MixColumns transformation. By utilizing both properties, the addition of the same elements over GF(2^m) results in zero properties, and dividing the involutory matrix into four sets of submatrices circulant matrix properties; the matrix multiplication can be simplified by Scheme 3 (16×16 matrix) that matrix multiplications can use 81 multiplications and 260 additions with good branch number 17. Using Scheme 3 and the proposed method of the multiplication running on Intel CPU, to compare traditional matrix multiplication, the computational cost of matrix multiplication can be reduced by ~67%. Finally, using Scheme 1, Scheme 2, and Scheme 3 into AES Cipher and InvCipher procedure that the methods can increase encryption and decryption speed for data transmission.

Keywords: **AES, circulant matrix, involutory matrix, quantum computer**

1. Introduction

The Advanced Encryption Standard (AES) algorithm [1] was chosen as the encryption standard in 1999, replacing the original Data Encryption Standard (DES) from among 15 candidates. It has since been widely adopted globally due to its quickly computational and robust security features, establishing itself as the new standard in symmetric cryptography. In September 2000, Rijndael was officially designated as FIPS PUB 197 [2], with a complete round including four steps: SubBytes, ShiftRows, MixColumns, and AddRoundKey. Using larger matrices for AES computation is proposed in reference [3] which the matrix utilizes a 16×16 involutoy matrix for data diffusion. However, increasing data diffusion also reduces the speed of matrix multiplication operations. When performing matrix operations, this matrix allows for rapid calculation, providing comparable performance for both encryption and decryption. This method replaces the original 4×4 cyclic matrices in MixColumns steps of the AES. Reference [4] enhances encryption strength using 4×4 involutory matrix and 8×8 involutory matrices. There are research directions suggested by searching methods for finding MDS matrices in [5-6]. There are many block ciphers use Maximum Distance Separable (MDS) codes as diffusion layers [7-9]. The well-known ciphers the Khazad [10] and ARIA [11] are using involutory matrix. The diversity circulant matrices are used in the modern cryptographic method in AES [12]. As descried in this paper, may be designed as a circuit in VLSI, see [13-16], which can be used to decrease logic gates. In [17] presents a quantum circuit to implement the S-box of AES. The matrix multiplication needs a finite filed multiplication for speeding operation [18-19]. The method also can provide the security of the data transfer to the health monitoring system on ARM-based microcontrollers [20]. We propose using involutory matrix for encoding and decoding for AES MixColumns steps that is not required the inverse polynomial $A(x)$, denoted as $A^{-1}(x)$ [21-22]. The method using the 16×16 involutory matrix would be more difficult for attackers to locate and thus less prone to attacks in general. The matrix product operation can be reduction like as circulant matrix method. 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 number of multiplications in different n×n involutory matrix products for the AES encryption-decryption which these methods are called Scheme 1, Scheme 2, and Scheme 3, respectively. Section 4 presents a performance analysis of AES Cipher-InvCipher on Intel CPU. Section 5 concludes the paper.

2. Enhanced security in AES MixColumns transformation

This paper mainly is not focused on fix matrix element in AES MixColumns transformation. We aim to enhance security of AES algorithm with different matrix size for MixColumns steps that can be for increasing security. Since, the determining the key would require an exhaustive search and must to know what kind of matrix element in Table A as shown in Figure 1 for encrypting and decrypting. In other words, the key cannot be known from the plaintext and the ciphertext because there are not using AES standard MixColumns (02x, 03x, 01x, 01x) and InvMixColumns (0Ex, 0Bx, 0Dx, 09x) transformation. Furthermore, it might be sent different the involutory matrix elements by elliptic curve cryptography of the ECDH algorithm or RSA algorithm to receiver. In Table A, the value of elements from first row involutory MDS matrix (Hadamard matrix) is Hexadecimal.

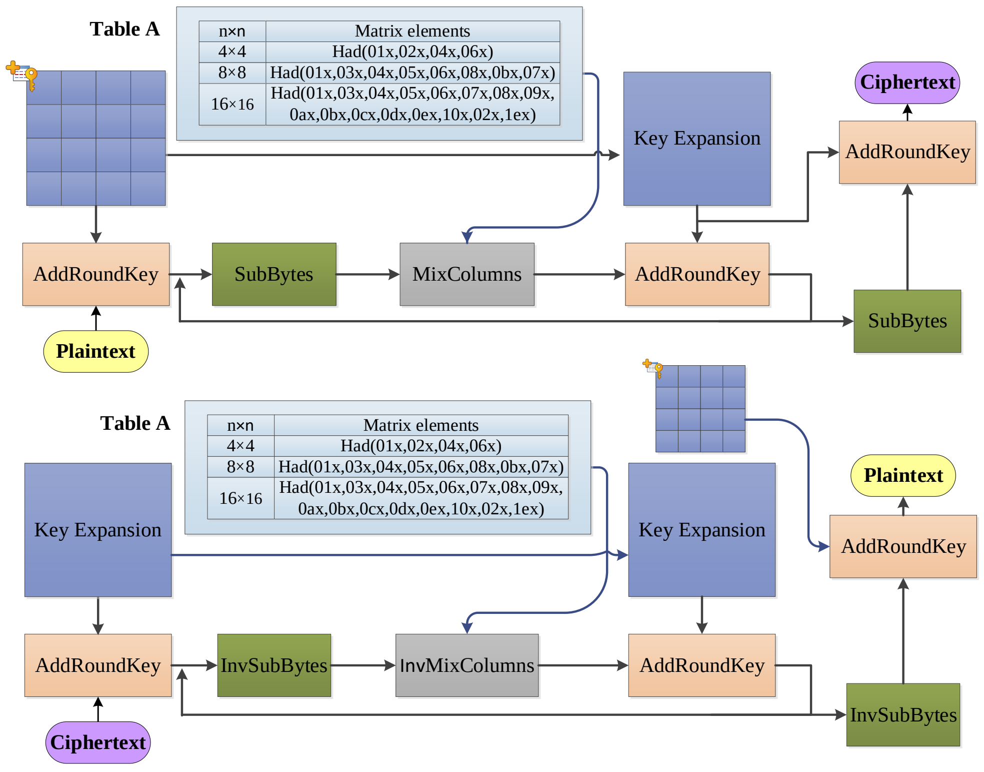

Table A

n×n	Matrix elements
4×4	Had(01x,02x,04x,06x)
8×8	Had(01x,03x,04x,05x,06x,08x,0bx,07x)
16×16	Had(01x,03x,04x,05x,06x,07x,08x,09x, 0ax,0bx,0cx,0dx,0ex,10x,02x,1ex)

Table A

n×n	Matrix elements
4×4	Had(01x,02x,04x,06x)
8×8	Had(01x,03x,04x,05x,06x,08x,0bx,07x)
16×16	Had(01x,03x,04x,05x,06x,07x,08x,09x, 0ax,0bx,0cx,0dx,0ex,10x,02x,1ex)

Figure 1: The *n*×*n* involutory matrix elements for AES-like cipher-invcipher

3. Fast matrix multiplication in AES MixColumns transformation

The method for computing of involutory 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, uses two point cyclic matrix the properties for reducing multiplication of the matrix product, Scheme 1, Scheme 2 and Scheme 3 are 4×4, 8×8, and 16×16 matrices for reducing multiplications, respectively.

3.1 Multiplication over *GF*(2^m)

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 the polynomials over *GF*(2^m), where a_i, b_i ∈ {0, 1}. The finite field addition is defined as:

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

The symbol of "+" is XOR operation. The finite field multiplication is defined as:

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

where the modulo *f(x)* is irreducible polynomial $f(x)=x^8+x^4+x^3+x+1$ in the AES algorithm. In (2), the *a*(*x*) polynomial can be form as follows:

$$a(x)=\left((a_7x^2+a_6x+a_5)x^3+(a_4x^2+a_3x+a_2)\right)x^2+(a_1x+a_0) \tag{3}$$

The polynomial *b*(*x*) is represented as *B*. In (3) submitted (2) as following formula,

$$\begin{aligned} c(x)=&\left((a_7Bx^2+a_6Bx+a_5B)x^3 \bmod f(x)+(a_4Bx^2+a_3Bx+a_2B)\right)x^2 \bmod f(x) \\ &+(a_1Bx+a_0B) \end{aligned} \tag{4}$$

Table 1: Lookup table LTB(a_i, a_j, a_k) for multiplication over *GF*(2^8)

LTB $[a_i, a_j, a_k]$	$(a_iBx^2+a_jBx+a_kB) \bmod f(x)$	Using logical operation by python
LTB[0,0,0]	0	0
LTB[0,0,1]	*B*	*B*
LTB[0,1,0]	LTB[0,0,1] *x* mod *f*(*x*)	(LTB[0,0,1] <<1)&0XFF ^ FT[*B*>>7]&0X01
LTB[1,0,0]	LTB[0,1,0] *x* mod *f*(*x*)	(LTB[0,1,0] <<1)&0XFF ^ FT[LTB[0,1,0]>>7]&0X01
LTB[0,1,1]	LTB[0,1,0] + LTB[0,0,1]	LTB([,1,0] ^ LTB[0,0,1]
LTB[1,0,1]	LTB[1,0,0] + LTB[0,0,1]	LTB[1,0,0] ^ LTB[0,0,1]
LTB[1,1,0]	LTB[1,0,0] + LTB[0,1,0]	LTB[1,0,0] ^ LTB[0,1,0]
LTB[1,1,1]	LTB[1,1,0] + LTB[0,0,1]	LTB[1,1,0] ^ LTB[0,0,1]

In Table 1, LTB(0,0,1)=B, LTB(0,0,1)x mod $f(x)$ is expression $Bx \bmod f(x)$ that can be represented as $(B$<<1)&$0x$FF^FT$((B >> 7)$&$0x01)$. (*e.g.*, FT(0)=0, FT(1)=$x^4 + x^3 + x + 1$, binary 11011, Hex 0x1b). The modulo operation to make a table is as shown in Table 2. The method is rewritten in python programming as below:

Table 2: Lookup table FT(a_i, a_j, a_k) for modulo operation

	FT$[a_i, a_j, a_k]$	
FT[0,0,0]= FT[0]=0	FT[0,1,1]= FT[3]=0x2d	FT[1,1,0]= FT[6]=0x5a
FT[0,0,1]= FT[1]=0x1b	FT[1,0,0]= FT[4]=0x6c	FT[1,1,1]= FT[7]=0x41
FT[0,1,0]= FT[2]=0x36	FT[1,0,1]= FT[5]=0x77	

The multiplication over $GF(2^m)$ by python program

```
Def GFM(a,b):
    LTB=[0]*8
    LTB[0]=0; LTB[1]=b; LTB[2]=((b<<1)&0xFF)^FT[b>>7]; LTB[3]=LTB[2]^LTB[1]
    LTB[4]=(LTB[2]<<1)&0XFF^FT[LTB[2]>>7]; LTB[5]=LTB[4]^LTB[1];
    LTB[6]=LTB[4]^LTB[2]
    LTB[7]=LTB[6]^LTB[1]
    C=LTB[(a>>5)]
    C=((C<<3)&0XFF)^FT[C>>5]^LTB[(a>>2)&0X7]
    C=((C<<2)&0XFF)^FT[C>>6]^LTB[a&0X3]
    return C
```

3.2 Reducing the multiplications in 4×4 involutory matrix for matrix multiplication

In AES MixColumns transformation that need four times matrix multiplication by 4×4 matrix. The product of the involutory matrix A and matrix B, is presented form as:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{5}$$

Where $D = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}$, $A = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix}$, and $B = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$.

The matrix *A* can be represented as $Had(a_0, a_1, a_2, a_3)$. In (5), the matrix *D* is the product of the matrix *A* and the matrix *B* that is needed 16 multiplications and 12 additions (16**M**, 12**A**) listed as follows:

(16M, 12A)
$d_0 = a_0b_0 + a_1b_1 + a_2b_2 + a_3b_3$
$d_1 = a_1b_0 + a_0b_1 + a_3b_2 + a_2b_3$
$d_2 = a_2b_0 + a_3b_1 + a_0b_2 + a_1b_3$
$d_3 = a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3$

Using the 2×2 cyclic matrix property for matrices multiplication is given by:

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

Therefore, in (6) only requires 3 multiplications and 4 additions, namely, (**3M, 4A**) as shown in Table 1.

Table 3: The 2×2 cyclic matrix with (**3M, 4A**).

$t_0 = (b_0 + b_1)a_0$	$t_1 = a_0 + a_1$
$d_0 = t_0 + t_1b_1$	$d_1 = t_0 + t_1b_0$

In Table 3, 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 2×2 cyclic matrix method only uses 3 multiplications and 3 additions. The python program is as shown below.

The 2×2 involutory matrix (3M, 4A)

```
def FGH2(a0,a1,b0,b1):
    t0=GFM((b0^b1),a0)
    t1=a0^a1
    d0=t0^GFM(t1,b1)
```

```
d1=t0^GFM(t1,b0)
return [d0,d1]
```

Theorem 1 Let A be any $n\times n$ involutory matrix, where $n=2^k$, then the matrix A can be partitioned into four cyclic matrices, in which entries are $(\frac{n}{2}\times\frac{n}{2})$ submatrix where k is greater than 1.

Using (4), by Theorem 1, the 4×4 involutory matrix can be partitioned into four as,

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

where $D_0=\begin{bmatrix} d_0 \\ d_1 \end{bmatrix}, D_1=\begin{bmatrix} d_2 \\ d_3 \end{bmatrix}, A_0=\begin{bmatrix} a_0 & a_1 \\ a_1 & a_0 \end{bmatrix}, A_1=\begin{bmatrix} a_2 & a_3 \\ a_3 & a_2 \end{bmatrix},\ B_0=\begin{bmatrix} b_0 \\ b_1 \end{bmatrix},$ and $B_1=\begin{bmatrix} b_2 \\ b_3 \end{bmatrix}.$

In (7), it can be used to reduce the multiplications in term of Equation (6) form as follows:

$$\begin{bmatrix} D_0 \\ D_1 \end{bmatrix} = \begin{bmatrix} A_0(B_0\oplus B_1)\oplus(A_0\oplus A_1)B_1 \\ A_0(B_0\oplus B_1)\oplus(A_0\oplus A_1)B_0 \end{bmatrix} = \begin{bmatrix} F2\oplus G2 \\ F2\oplus H2 \end{bmatrix}, \tag{8}$$

where the symbol of $\oplus$ is represented the matrix addition,

$F2=A_0(B_0\oplus B_1)=\begin{bmatrix} a_0 & a_1 \\ a_1 & a_0 \end{bmatrix}\begin{bmatrix} b_0+b_2 \\ b_1+b_3 \end{bmatrix}$, $G2=(A_0\oplus A_1)B_1=\begin{bmatrix} a_0+a_2 & a_1+a_3 \\ a_1+a_3 & a_0+a_2 \end{bmatrix}\begin{bmatrix} b_2 \\ b_3 \end{bmatrix}$, and

$H2=(A_0\oplus A_1)B_0=\begin{bmatrix} a_0+a_2 & a_1+a_3 \\ a_1+a_3 & a_0+a_2 \end{bmatrix}\begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$. So that, the matrices $F2$, $G2$, and $H2$ can call function *FGH*2() yields:

$$\begin{aligned} F2 &= FGH2(a_0,a_1,b_0+b_2,b_1+b_3) \\ G2 &= FGH2(a_0+a_2,a_1+a_3,b_2,b_3) \\ H2 &= FGH2(a_0+a_2,a_1+a_3,b_0,b_1) \end{aligned} \tag{9}$$

Obviously, the matrix *F*2 is calling function *FGH2*() with parameters that does not need combination of the sets with element b_i. The matrix *G*2 and *H*2 are calling function *FGH*2()

with parameters that are combination of the sets with element *aᵢ*. In (9), rewrite the terms in $t_0 = a_0 + a_2$ and $t_1 = a_1 + a_3$ as follows:

$$\begin{aligned} F2 &= FGH2(a_0, a_1, b_1 + b_3, b_0 + b_2) \\ G2 &= FGH2(t_0, t_1, b_2, b_3) \\ H2 &= FGH2(t_0, t_1, b_0, b_1) \end{aligned} \tag{10}$$

Next, the 4×4 involutory matrix for matrix multiplication is given as

$$\begin{bmatrix} D_0 \\ D_1 \end{bmatrix} = \begin{bmatrix} F2 \oplus G2 \\ F2 \oplus H2 \end{bmatrix},$$

where the symbol of $\oplus$ is represented the matrix addition. In the simplified case, the matrix multiplication can be performed by 9 multiplications and 20 additions **(9M, 20A)**. Two items $t_0 = a_0 + a_2$ and $t_1 = a_3 + a_1$ are known because the value a_i of the elements of the matrix *A*, can be precomputed in the program. So that the method only uses 9 multiplications and 18 additions, that is remarked as **(9M, 18A)**. The 4×4 involutory matrix for the matrix multiplication is written by python program that is called Scheme 1. So that, Scheme 1 is need 4 times in AES MixColumns transformation that there are 36 multiplications and 72 additions **(36M, 72A)**.

Scheme 1: 4×4 involutory matrix (9M, 20A)

```
def FGH4(a0,a1,a2,a3,b0,b1,b2,b3):
    t0=a0^a2; t1=a1^a3
    F2=FGH2(a0,a1,b0^b2,b1^b3)
    G2=FGH2(t0,t1,b2,b3)
    H2=FGH2(t0,t1,b0,b1)
    d0=F2[0]^G2[0]
    d1=F2[1]^G2[1]
    d2=F2[0]^H2[0]
    d3=F2[1]^H2[1]
    return [d0,d1,d2,d3]
```

3.3 Reducing the multiplications in 8×8 involutory matrix for matrix multiplication

Matrix multiplication is needed **M**=64, **A**=56, where, **M** is multiplications, **A** is additions. The matrix is 8×8 involutory matrix in AES MixColumns transformation that is required twice times matrix multiplication:

$$\left[\begin{array}{c} d_0 \\ d_1 \\ d_2 \\ d_3 \\ \hline d_4 \\ d_5 \\ d_6 \\ d_7 \end{array}\right] = \left[\begin{array}{cccc|cccc} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\ a_1 & a_0 & a_3 & a_2 & a_5 & a_4 & a_7 & a_6 \\ a_2 & a_3 & a_0 & a_1 & a_6 & a_7 & a_4 & a_5 \\ a_3 & a_2 & a_1 & a_0 & a_7 & a_6 & a_5 & a_4 \\ \hline a_4 & a_5 & a_6 & a_7 & a_0 & a_1 & a_2 & a_3 \\ a_5 & a_4 & a_7 & a_6 & a_1 & a_0 & a_3 & a_2 \\ a_6 & a_7 & a_4 & a_5 & a_2 & a_3 & a_0 & a_1 \\ a_7 & a_6 & a_5 & a_4 & a_3 & a_4 & a_1 & a_0 \end{array}\right] \left[\begin{array}{c} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \hline b_4 \\ b_5 \\ b_6 \\ b_7 \end{array}\right], \tag{11}$$

$$\begin{matrix} D_0 & A_0 & A_1 & B_0 \\ D_1 & A_1 & A_0 & B_1 \end{matrix}$$

where $B_0 = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$, $B_1 = \begin{bmatrix} b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$, $A_0 = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix}$, $A_1 = \begin{bmatrix} a_4 & a_5 & a_6 & a_7 \\ a_5 & a_4 & a_7 & a_6 \\ a_6 & a_7 & a_4 & a_5 \\ a_7 & a_6 & a_5 & a_4 \end{bmatrix}$.

By (11), it can be used to reduce the multiplications in term of Equation (6) form as follows:

$$\begin{bmatrix} D_0 \\ D_1 \end{bmatrix} \begin{bmatrix} A_0(B_0 \oplus B_1) \oplus (A_0 \oplus A_1)B_1 \\ A_0(B_0 \oplus B_1) \oplus (A_0 \oplus A_1)B_0 \end{bmatrix} = \begin{bmatrix} F4 \oplus G4 \\ F4 \oplus H4 \end{bmatrix},.$$

where the symbol of $\oplus$ is represented the matrix addition, $F4 = A_0(B_0 \oplus B_1)$, $G4 = (A_0 \oplus A_1)B_1$, and $H4 = (A_0 \oplus A_1)B_0$.

$$F4 = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 + b_4 \\ b_1 + b_5 \\ b_2 + b_6 \\ b_3 + b_7 \end{bmatrix} \tag{12}$$

In (12), the matrix $F4$ calling function $FGH4()$ is as below.

$F4 = FGH_4(a_0, a_1, a_2, a_3, b_0 + b_4, b_1 + b_5, b_2 + b_6, b_3 + b_7)$,

$$G4 = \begin{bmatrix} a_0 + a_4 & a_1 + a_5 & a_2 + a_6 & a_3 + a_7 \\ a_1 + a_5 & a_0 + a_4 & a_3 + a_7 & a_2 + a_6 \\ a_2 + a_6 & a_3 + a_7 & a_0 + a_4 & a_1 + a_5 \\ a_3 + a_7 & a_2 + a_6 & a_1 + a_5 & a_0 + a_4 \end{bmatrix} \begin{bmatrix} b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}. \tag{13}$$

The matrix *G*4 calling function *FGH*4() is rewritten as below:

$$G4 = FGH_4(a_0 + a_4, a_1 + a_5, a_2 + a_6, a_3 + a_7, b_4, b_5, b_6, b_7)$$

$$H4 = \begin{bmatrix} a_0 + a_4 & a_1 + a_5 & a_2 + a_6 & a_3 + a_7 \\ a_1 + a_5 & a_0 + a_4 & a_3 + a_7 & a_2 + a_6 \\ a_2 + a_6 & a_3 + a_7 & a_0 + a_4 & a_1 + a_5 \\ a_3 + a_7 & a_2 + a_6 & a_1 + a_5 & a_0 + a_4 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}. \quad (14)$$

In (14), matrix *H*4 calling function *FGH*4() is rewritten as below.

$$H4 = FGH_4(a_0 + a_4, a_1 + a_5, a_2 + a_6, a_3 + a_7, b_0, b_1, b_2, b_3).$$

It needs 27 multiplications and 76 additions, namely, (**27M, 76A**), the two items can be replaced as $t_0 = a_0 + a_4$, $t_1 = a_1 + a_5$, $t_2 = a_2 + a_6$ and $t_3 = a_3 + a_7$ that precomputing for matrix multiplication is only 72 additions, namely, (**27M, 72A**). Consequently, the matrix multiplication is simplified in MixColumns step in AES that needs two times 8x8 matrix multiplication, the operation is needs 54 multiplications and 148 additions, namely, (**54M, 144A**). The 8×8 involutory matrix for matrix multiplication is called Scheme 2 that can further be rewritten by python program as follows:

Scheme 2: 8×8 involutory matrix (27M, 76A)

```
def FGH8(a0,a1,a2,a3,a4,a5,a6,a7,b0,b1,b2,b3,b4,b5,b6,b7):
    t0=a0^a4;t1=a1^a5;t2=a2^a6;t3=a3^a7
    F=FGH4(a0,a1,a2,a3,b0^b4,b1^b5,b2^b6,b3^b7)
    G=FGH4(t0,t1,t2,t3,b4,b5,b6,b7)
    H=FGH4(t0,t1,t2,t3,b0,b1,b2,b3)
    D=[0]*8
    for i in range (0,len(F)):
        D[i]=F[i]^G[i]
    for i in range (0,len(F)):
        D[i+4]=F[i]^H[i]
    return D
```

3.4 Reducing the multiplications in 16×16 involutory matrix for matrix multiplication

Using the 16×16 matrix is only a matrix multiplication in AES MixColumns, which is 256 multiplications and 240 additions. A 16×16 involutory matrix for matrix multiplication is as shown below.

$$\begin{bmatrix} d_0\\ d_1\\ d_2\\ d_3\\ d_4\\ d_5\\ d_6\\ d_7\\ \hline d_8\\ d_9\\ d_{10}\\ d_{11}\\ d_{12}\\ d_{13}\\ d_{14}\\ d_{15} \end{bmatrix} = \left[\begin{array}{cccccccc|cccccccc}
a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15}\\
a_1 & a_0 & a_3 & a_2 & a_5 & a_4 & a_7 & a_6 & a_9 & a_8 & a_{11} & a_{10} & a_{13} & a_{12} & a_{15} & a_{14}\\
a_2 & a_3 & a_0 & a_1 & a_6 & a_7 & a_4 & a_5 & a_{10} & a_{11} & a_8 & a_9 & a_{14} & a_{15} & a_{12} & a_{13}\\
a_3 & a_2 & a_1 & a_0 & a_7 & a_6 & a_5 & a_4 & a_{11} & a_{10} & a_9 & a_8 & a_{15} & a_{14} & a_{13} & a_{12}\\
a_4 & a_5 & a_6 & a_7 & a_0 & a_1 & a_2 & a_3 & a_{12} & a_{13} & a_{14} & a_{15} & a_8 & a_9 & a_{10} & a_{11}\\
a_5 & a_4 & a_7 & a_6 & a_1 & a_0 & a_3 & a_2 & a_{13} & a_{12} & a_{15} & a_{14} & a_9 & a_8 & a_{11} & a_{10}\\
a_6 & a_7 & a_4 & a_5 & a_2 & a_1 & a_0 & a_1 & a_{14} & a_{15} & a_{12} & a_{13} & a_{10} & a_{11} & a_8 & a_9\\
a_7 & a_6 & a_5 & a_4 & a_1 & a_2 & a_1 & a_0 & a_{15} & a_{14} & a_{13} & a_{12} & a_{11} & a_{10} & a_9 & a_8\\ \hline
a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7\\
a_9 & a_8 & a_{11} & a_{10} & a_{13} & a_{12} & a_{15} & a_{14} & a_1 & a_0 & a_3 & a_2 & a_5 & a_4 & a_7 & a_6\\
a_{10} & a_{11} & a_8 & a_9 & a_{14} & a_{15} & a_{12} & a_{13} & a_2 & a_3 & a_0 & a_1 & a_6 & a_7 & a_4 & a_5\\
a_{11} & a_{10} & a_9 & a_8 & a_{15} & a_{14} & a_{13} & a_{12} & a_3 & a_2 & a_1 & a_0 & a_7 & a_6 & a_5 & a_4\\
a_{12} & a_{13} & a_{14} & a_{15} & a_8 & a_9 & a_{10} & a_{11} & a_4 & a_5 & a_6 & a_7 & a_0 & a_1 & a_2 & a_3\\
a_{13} & a_{12} & a_{15} & a_{14} & a_9 & a_8 & a_{11} & a_{10} & a_5 & a_4 & a_7 & a_6 & a_1 & a_0 & a_3 & a_2\\
a_{14} & a_{15} & a_{12} & a_{13} & a_{10} & a_{11} & a_8 & a_9 & a_6 & a_7 & a_4 & a_5 & a_2 & a_3 & a_0 & a_1\\
a_{15} & a_{14} & a_{13} & a_{12} & a_{11} & a_{10} & a_9 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0
\end{array}\right] \begin{bmatrix} b_0\\ b_1\\ b_2\\ b_3\\ b_4\\ b_5\\ b_6\\ b_7\\ \hline b_8\\ b_9\\ b_{10}\\ b_{11}\\ b_{12}\\ b_{13}\\ b_{14}\\ b_{15} \end{bmatrix}$$

$$\begin{matrix} D_0 & A_0 & A_1 & B_0\\ D_1 & A_1 & A_0 & B_1 \end{matrix}$$

The matrix can be used to reduce the multiplications in term of Equation (6) form as follows:

$$\begin{bmatrix} D_0\\ D_1 \end{bmatrix}\begin{bmatrix} A_0(B_0\oplus B_1)\oplus(A_0\oplus A_1)B_1\\ A_0(B_0\oplus B_1)\oplus(A_0\oplus A_1)B_0 \end{bmatrix}=\begin{bmatrix} F8\oplus G8\\ F8\oplus H8 \end{bmatrix},$$

where the symbol of $\oplus$ is the matrix addition, the matrices $F8=A_0(B_0\oplus B_1)$, $G8=(A_0\oplus A_1)B_1$, and $H8=(A_0\oplus A_1)B_0$ can be representation form as below:

Communications of the CCISA
Vol.30 No. 2 May. 2024

$$F8=\begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \\ a_1 & a_0 & a_3 & a_2 & a_5 & a_4 & a_7 & a_6 \\ a_2 & a_3 & a_0 & a_1 & a_6 & a_7 & a_4 & a_5 \\ a_3 & a_2 & a_1 & a_0 & a_7 & a_6 & a_5 & a_4 \\ a_4 & a_5 & a_6 & a_7 & a_0 & a_1 & a_2 & a_3 \\ a_5 & a_4 & a_7 & a_6 & a_1 & a_0 & a_3 & a_2 \\ a_6 & a_7 & a_4 & a_5 & a_2 & a_3 & a_0 & a_1 \\ a_7 & a_6 & a_5 & a_4 & a_3 & a_4 & a_1 & a_0 \end{bmatrix}\begin{bmatrix} b_0+b_8 \\ b_1+b_9 \\ b_2+b_{10} \\ b_3+b_{11} \\ b_4+b_{12} \\ b_5+b_{13} \\ b_6+b_{14} \\ b_7+b_{15} \end{bmatrix}.$$

The matrix *F*8 calling function *FGH*8() is written as below.

$$F8=FGH_8(a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7,b_0+b_8,b_1+b_9,b_2+b_{10},b_3+b_{11},b_4+b_{12},b_5+b_{13},b_6+b_{14},\\ b_7+b_{15})$$

$$G8=\begin{bmatrix} a_0+a_8 & a_1+a_9 & a_2+a_{10} & a_3+a_{11} & a_4+a_{12} & a_5+a_{13} & a_6+a_{14} & a_7+a_{15} \\ a_1+a_9 & a_0+a_8 & a_3+a_{11} & a_2+a_{10} & a_5+a_{13} & a_4+a_{12} & a_7+a_{15} & a_6+a_{14} \\ a_2+a_{10} & a_3+a_{11} & a_0+a_8 & a_1+a_9 & a_6+a_{14} & a_7+a_{15} & a_4+a_{12} & a_5+a_{13} \\ a_3+a_{11} & a_2+a_{10} & a_1+a_9 & a_0+a_8 & a_7+a_{15} & a_6+a_{14} & a_5+a_{13} & a_4+a_{12} \\ a_4+a_{12} & a_5+a_{13} & a_6+a_{14} & a_7+a_{15} & a_0+a_8 & a_1+a_{15} & a_2+a_{10} & a_3+a_{11} \\ a_5+a_{13} & a_4+a_{12} & a_7+a_{15} & a_6+a_{14} & a_1+a_9 & a_0+a_8 & a_3+a_{11} & a_2+a_{10} \\ a_6+a_{14} & a_7+a_{15} & a_4+a_{12} & a_5+a_{13} & a_2+a_{10} & a_3+a_9 & a_0+a_8 & a_1+a_9 \\ a_7+a_{15} & a_6+a_{14} & a_5+a_{13} & a_4+a_{12} & a_3+a_{11} & a_4+a_{10} & a_1+a_9 & a_0+a_8 \end{bmatrix}\begin{bmatrix} b_8 \\ b_9 \\ b_{10} \\ b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \end{bmatrix}.$$

The matrix *G*8 calling the function *FGH*8() is as follows:

$$G8=FGH_8(a_0+a_8,a_1+a_9,a_2+a_{10},a_3+a_{11},a_4+a_{12},a_5+a_{13},a_6+a_{14},a_7+a_{15},b_8,b_9,b_{10},b_{11},b_{12},\\ b_{13},b_{14},b_{15})$$

$$H8=\begin{bmatrix} a_0+a_8 & a_1+a_9 & a_2+a_{10} & a_3+a_{11} & a_4+a_{12} & a_5+a_{13} & a_6+a_{14} & a_7+a_{15} \\ a_1+a_9 & a_0+a_8 & a_3+a_{11} & a_2+a_{10} & a_5+a_{13} & a_4+a_{12} & a_7+a_{15} & a_6+a_{14} \\ a_2+a_{10} & a_3+a_{11} & a_0+a_8 & a_1+a_9 & a_6+a_{14} & a_7+a_{15} & a_4+a_{12} & a_5+a_{13} \\ a_3+a_{11} & a_2+a_{10} & a_1+a_9 & a_0+a_8 & a_7+a_{15} & a_6+a_{14} & a_5+a_{13} & a_4+a_{12} \\ a_4+a_{12} & a_5+a_{13} & a_6+a_{14} & a_7+a_{15} & a_0+a_8 & a_1+a_{15} & a_2+a_{10} & a_3+a_{11} \\ a_5+a_{13} & a_4+a_{12} & a_7+a_{15} & a_6+a_{14} & a_1+a_9 & a_0+a_8 & a_3+a_{11} & a_2+a_{10} \\ a_6+a_{14} & a_7+a_{15} & a_4+a_{12} & a_5+a_{13} & a_2+a_{10} & a_3+a_9 & a_0+a_8 & a_1+a_9 \\ a_7+a_{15} & a_6+a_{14} & a_5+a_{13} & a_4+a_{12} & a_3+a_{11} & a_4+a_{10} & a_1+a_9 & a_0+a_8 \end{bmatrix}\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}.\mathrm{I}$$

The matrix *H*8 calling function *FGH*8() is written as below.

$$H8 = FGH_8(a_0 + a_8, a_1 + a_9, a_2 + a_{10}, a_3 + a_{11}, a_4 + a_{12}, a_5 + a_{13}, a_6 + a_{14}, a_7 + a_{15}, b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$$

Matrix multiplication, using 4×4 matrix needs 16 multiplications and 12 additions; using 8×8 matrix needs 64 multiplications and 56 additions; using 16×16 matrix needs 256 multiplications and 240 additions. Using Scheme 1 needs 36 multiplications and 80 additions; using Scheme 2 needs 27 multiplications and 76 additions; Scheme 3 only uses 81 multiplications and 260 additions. In summary, the number of the multiplications and memory sizes for different Schemes are as shown in Table 4. In Table 4, the symbol "**M**" represents the multiplications and the symbol "**A**" represents the additions.

Scheme 3: 16×16 involutory matrix (81M, 260A)

```
def FGH16(a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,b0,b1,b2,b3,b4,b5,b6,b7,
          b8,b9,b10,b11,b12,b13,b14,b15):
    t0=a0^a8;t1=a1^a9;t2=a2^a10;t3=a3^a11;t4=a4^a12;t5=a5^a13;t6=a6^a14;t7=a7^a15
    F=FGH8(a0,a1,a2,a3,a4,a5,a6,a7,
       b0^b8,b1^b9,b2^b10,b3^b11,b4^b12,b5^b13,b6^b14,b7^b15)
    G=FGH8(t0,t1,t2,t3,t4,t5,t6,t7,b8,b9,b10,b11,b12,b13,b14,b15)
    H=FGH8(t0,t1,t2,t3,t4,t5,t6,t7,b0,b1,b2,b3,b4,b5,b6,b7)
    D=[0]*16
    for i in range (0,len(F)):
        D[i]=F[i]^G[i]
    for i in range (0,len(F)):
        D[i+8]=F[i]^H[i]
    return D
```

Note that the traditional circulant matrix multiplication in AES, which has to find inverse matrix for decryption processes. However, Scheme 1, Scheme 2 and Scheme 3 can be used both encryption and decryption in AES MixColumns and InvMixColumns transformation.

Table 4: Different schemes need operations and memory sizes.

Matrix multiplication	(M, A)	Memory sizes	Schemes	(M, A)	Memory sizes
4x4 matrix	**(64M, 48A)**	4 bytes	Scheme 1	(**36M, 80A**)	12 bytes

8x8 matrix	**(128M, 84A)**	8 bytes	Scheme 2	(**54M, 152A**)	48 bytes
16x16 matrix	**(256M, 240A)**	16 bytes	Scheme 3	(**81M, 260A**)	168 bytes

4. Simulation results

The different methods of the multiplication execute time running 100,000 times in Intel Core i9-12900 @ 2.4GHz by python program and the results are given in Table 5.

Table 5: Different schemes of multiplication executes time.

Multiplication	Execution time	Memory size
Russian Peasant algorithm	0.17 s	0 bytes
Horner's rule [21]	0.10 s	8 bytes
The proposed method	0.09 s	16 bytes

The proposed method is faster than Horner’s rule. As a result, the finite field multiplication is utilized for performing matrix multiplication in the 4×4 involutory matrix, designated as $A_{4\times4}$; the 8×8 involutory matrix, designated as $A_{8\times8}$, and the 16×16 involutory matrix, designated as $A_{16\times16}$, as detailed below:

$$A_{4\times4} = \begin{bmatrix} 01 & 02 & 04 & 06 \\ 02 & 01 & 06 & 04 \\ 04 & 06 & 01 & 02 \\ 06 & 04 & 02 & 01 \end{bmatrix}, \quad A_{8\times8} = \begin{bmatrix} 01 & 03 & 04 & 05 & 06 & 08 & 0B & 07 \\ 03 & 01 & 05 & 04 & 08 & 06 & 07 & 0B \\ 04 & 05 & 01 & 03 & 0B & 07 & 06 & 08 \\ 05 & 04 & 03 & 01 & 07 & 0B & 08 & 06 \\ 06 & 08 & 0B & 07 & 01 & 03 & 04 & 05 \\ 08 & 06 & 07 & 0B & 03 & 01 & 05 & 04 \\ 0B & 07 & 06 & 08 & 04 & 05 & 01 & 03 \\ 07 & 0B & 08 & 06 & 05 & 04 & 03 & 01 \end{bmatrix}, \text{ and}$$

$$A_{16\times 16}=\begin{bmatrix}
01 & 03 & 04 & 05 & 06 & 07 & 08 & 09 & 0A & 0B & 0C & 0D & 0E & 10 & 02 & 1E \\
03 & 01 & 05 & 04 & 07 & 06 & 09 & 08 & 0B & 0A & 0D & 0C & 10 & 0E & 1E & 02 \\
04 & 05 & 01 & 03 & 08 & 09 & 06 & 07 & 0C & 0D & 0A & 0B & 02 & 1E & 0E & 10 \\
05 & 04 & 03 & 01 & 09 & 08 & 07 & 06 & 0D & 0C & 0B & 0A & 1E & 02 & 10 & 0E \\
06 & 07 & 08 & 09 & 01 & 03 & 04 & 05 & 0E & 10 & 02 & 1E & 0A & 0B & 0C & 0D \\
07 & 06 & 09 & 08 & 03 & 01 & 05 & 04 & 10 & 0E & 1E & 02 & 0B & 0A & 0D & 0C \\
08 & 09 & 06 & 07 & 04 & 05 & 01 & 03 & 02 & 1E & 0E & 10 & 0C & 0D & 0A & 0B \\
09 & 08 & 07 & 06 & 05 & 04 & 03 & 01 & 1E & 02 & 10 & 0E & 0D & 0C & 0B & 0A \\
0A & 0B & 0C & 0D & 0E & 10 & 02 & 1E & 01 & 03 & 04 & 05 & 06 & 07 & 08 & 09 \\
0B & 0A & 0D & 0C & 10 & 0E & 1E & 02 & 03 & 01 & 05 & 04 & 07 & 06 & 09 & 08 \\
0C & 0D & 0A & 0B & 02 & 1E & 0E & 10 & 04 & 05 & 01 & 03 & 08 & 07 & 06 & 07 \\
0D & 0C & 0B & 0A & 1E & 02 & 10 & 0E & 05 & 04 & 03 & 01 & 09 & 08 & 07 & 06 \\
0E & 10 & 02 & 1E & 0A & 0B & 0C & 0D & 06 & 07 & 08 & 09 & 01 & 03 & 04 & 05 \\
10 & 0E & 1E & 02 & 0B & 0A & 0D & 0C & 07 & 06 & 09 & 08 & 03 & 01 & 05 & 04 \\
02 & 1E & 0E & 10 & 0C & 0D & 0A & 0B & 08 & 09 & 06 & 07 & 04 & 05 & 01 & 03 \\
1E & 02 & 10 & 0E & 0D & 0C & 0B & 0A & 09 & 08 & 07 & 06 & 05 & 04 & 03 & 01
\end{bmatrix}.$$

Note that the value of the elements in the involutory matrix is Hexadecimal.

Using the traditional matrix multiplication and Schemes are for evaluating procedure running 10,000 times. There are many methods for computing matrix multiplication, in which using the proposed multiplication for Schemes is very fast when compared other traditional methods of the matrix multiplication as shown in Table 6.

Table 6: The computing time of the matrix multiplication.

Multiplication algorithms	(4×4) matrix	(8×8) matrix	(16×16) matrix	SCM 1 (9M,20A)	SCM 2 (27M,80A)	SCM 3 (54M,160A)	Reducing Percentage (16×16)- SCM 3 /(16×16)×100%
Russian Peasant	0.29s	1.14s	4.60s	0.16s	0.48s	1.48s	68%
Horner's rule	0.19s	0.73s	2.88s	0.10s	0.31s	0.95s	67%
The proposed method	0.16s	0.64s	2.55s	0.09s	0.27s	0.83s	67%

These matrices of diffusion data are used in the AES MixColumns transformation calculation as shown in Figure 2. Scheme 1 needs ShiftRows function, because the diffusion matrix of the size is smaller. Scheme 2 and Scheme 3 in matrix size have larger diffusion so that does not need ShiftRows functions.

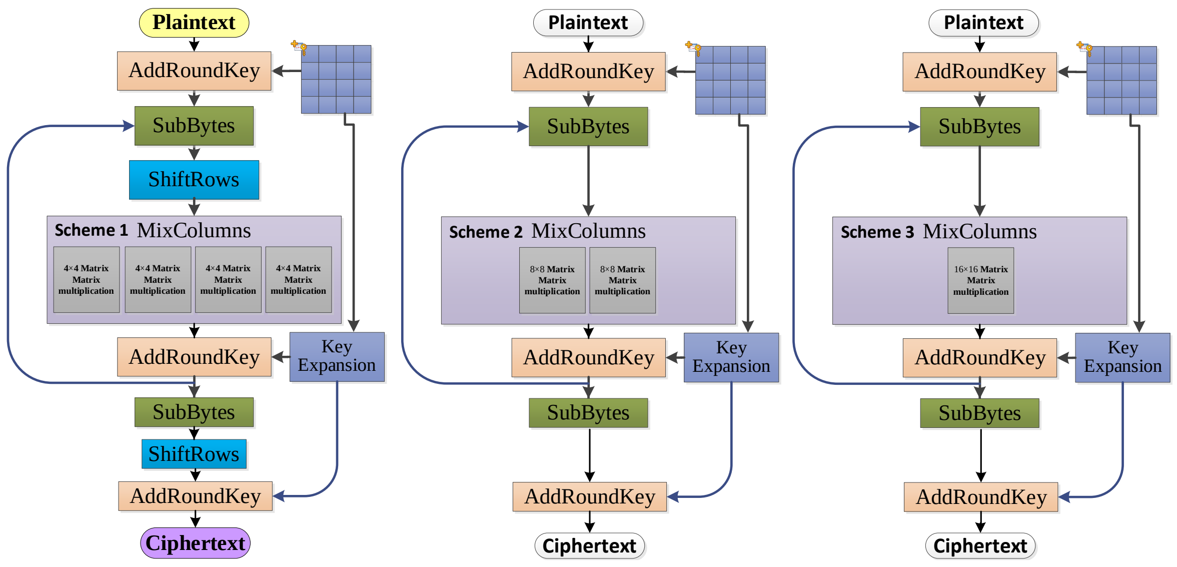

Figure 2: the matric of the different sizes using in the AES

Using the traditional methods and Schemes by python language are for evaluating encryption procedure with different AES key lengths. The keys sizes are 128, 192, and 256 bits for running cipher 10,000 execution time as shown in Table 7, which it can be into graphical from in Figure 3. The trade-off between matrix size and speed performance in AES Cipher would suggest that Scheme 2 (8×8 matrix) is better suitable for embedded systems that is faster than traditional matrix multiplication (4×4 matrix) running in AES MixColumns transformation; The keys of the lengths 128, 192, 256 bits can be reduced execution time ~12%, ~12%, and ~12%, respectively.

Table 7: The keys lengths of 128, 192, and 256 bits execution time with different matrix size.

Matrix size	AES 128	AES 192	AES 256
4x4 matrix	6.27	7.62	9.04
Scheme1	3.57	4.30	5.13
8x8 matrix	11.80	14.46	17.15
Scheme 2	5.34	6.56	7.75
16x16 matrix	23.01	28.04	33.21
Scheme 3	7.95	9.78	11.67

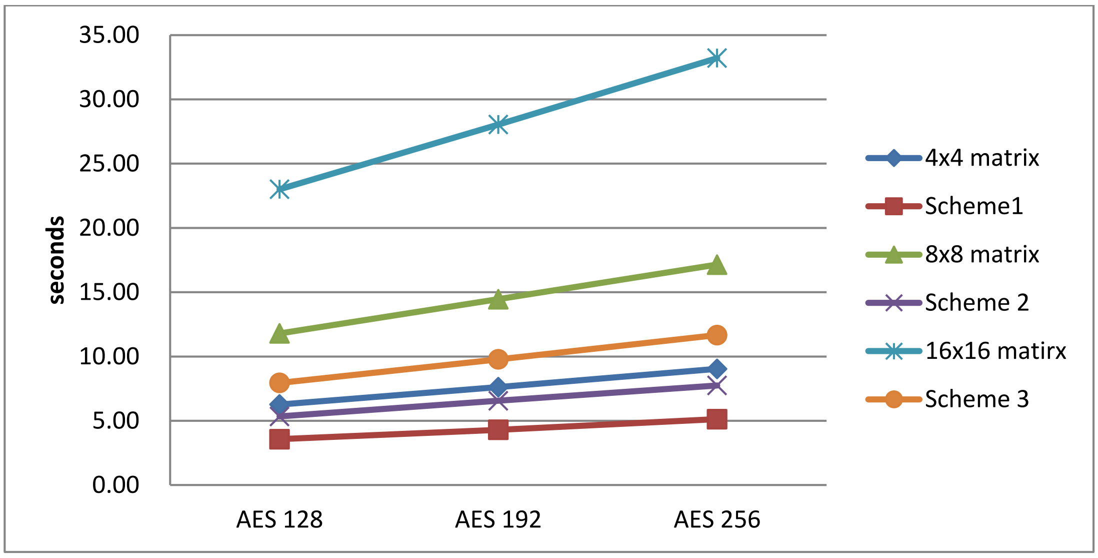

Figure 3: AES execution time with the different key lengths

5. Conclusion

To summarize, this study showed herein that the computational complexity matrix multiplication over GF(2^8) can be minimized by dividing the matrix into four submatrices and 2-point cyclic convolution property. In comparison for each of the matrix sizes, in AES a key size of 128 bits, 192 bits, and 256 bits, Scheme 2 can be run on MixColumns step of the AES faster than (16M, 12A) method. Scheme 1, Scheme 2, and Scheme 3 can also be used for different key sizes that you need cryptographic strength. Scheme 1 and Scheme 2 matrix multiplication are fast than Scheme 3 for encryption and decryption. However, Scheme 3 exhibits a superior branch number for enhancing data security. When using Scheme 2 and Scheme 3, in AES algorithm can remove ShiftRows step that is illustrated in Figure 1. In the future, Scheme 1, Scheme 2, and Scheme 3 could also be utilized in VLSI circuit design to reduce the number of logic gates required for the MixColumns and InvMixColumns transformations.

6. Acknowledgements

This study was supported in part by National Science and Technology Council NIST 111-2221-E-214-014 and NIST 112-2813-C-214-021.

References

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

[2] FIPS 197, Advanced Encryption Standard (AES) is updated by NIST FIPS 197-upd1, 2023 https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

[3] J. Nakahara, E. Abrahao, "A new invoutory MDS matrix for the AES," International Journal of newtwork security, vol. 9, no. 2, pp. 109-116, 2009.

[4] P. Junod, S. Vaudenay, "Perfect diffusion primitives for block ciphers. building efficient MDS Matrices," Lecture Notes in computer science, vol 3357, 2004. https://doi.org/10.1007/978-3-540-30564-4_6

[5] 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.

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

[7] D. Kwon, S. H. Sung, J. H. Song and S. Park, "Design of Block Ciphers and Coding Theory," Trends in Mathematics, vol. 8, no. 1, pp. 13-20, 2005.

[8] B. W. Koo, H. S. Jang, J. H. Song, "Constructing and Cryptanalysis of a 16x16 Binary Matrix as a Diffusion Layer," Proceedings of Information Security Applications: 4th International Workshop (WISA2003), Lecture Notes in Computer Science, Springer-Verlag, vol. 2908, pp. 489-503, 2003.

[9] B. W. Koo, H. S. Jang, J. H. Song, "On Constructing of a 32x32 binary matrix as a diffusion layer for a 256-bit block cipher," Proceedings of International Conference on Information Security and Cryptology, Lecture Notes in Computer Science, vol. 4296, pp. 51-64, Springer-Verlag, 2006.

[10] P. S. L. M. Barreto and V. Rijmen, "The Khazad legacy-level block cipher," First open NESSIE Workshop, Leuven, 2000.

[11] D. Kwon, J. Kim, S. Park, S.H. Sung, Y. Sohn, J.H. Song, Y. Yeom, E-J. Yoon, S, Lee, J. Lee, S. Chee, D. Han, and J. Hong, "New block cipher: ARIA," Proceedings of International Conference on Information Security and Cryptology, Lecture Notes in Computer Science, vol. 2971, pp. 432-445, 2004.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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, pp. 1-17, 2020.

[17] 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.

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

[19] 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.

[20] 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.

[21] Jeng-Jung Wang, Yan-Haw Chen, Guan-Hsiung Liaw, Jack Chang, Cheng-Chih Lee, "Efficient schemes with diverse of a pair of circulant matrices for AES MixColumns-InvMixcolumns transformation," Communications_of_the_CCISA, vol. 26, no. 2, pp. 1-20, 2020.

[22] J. J. Wang, Y. H. Chen, Y. W. Chen, and C. D. Lee, “Diversity AES in MixColomns step with 8x8 circulant matrix,” International Journal of Engineering Technology and Management Research, vol. 8 no. 9, pp. 19-35, 2021.