AES

Advanced Encryption Standard, Rijndael

20192211 김민지

목차

1. AES

Advanced Encryption Standard, Rijndael

Input	Output	Data block	key
128-bit	128-bit	128-bit	128-bit, 192-bit, 256-bit

AES Parameters

*1 word = 32-bit

알고리듬	블록 크기 (N_b-word)	키 길이 (N_k-word)	라운드 수 (N_r)	라운드키 길이 (word)	라운드키 개수 ($N_r + 1$)	라운드키 전체크기 ($16(N_r + 1)$-word)
AES-128	4	4	10	4	11	44
AES-192	4	6	12	4	13	52
AES-256	4	8	14	4	15	60

1. AES

표기

$\oplus$: XOR 연산

$B = \{0,1\}^8$: 8비트열 집합

$\{\ \}_{16}$: 16진수

$\{\ \}_{10}$: 10진수

$\{\ \}_2$: 2진수

Ex. $\{0b\}$: 16진수

$\{a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0\}$: 바이트

*1 word = 32-bit
*1 byte = 8-bit

1. AES - 구조


```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
       byte state[4,Nb]

       state = in

       AddRoundKey(state, w[0, Nb-1])

       for round = 1 step 1 to Nr-1
              SubBytes(state)
              ShiftRows(state)
              MixColumns(state)
              AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
       end for

       SubBytes(state)
       ShiftRows(state)
       AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

       out = state
end
```

1. AES - 구조 - SubBytes

`def SubBytes() :` $s(a) \coloneqq T(inv(a))$

- **Multiplicative Inverse** in the finite field $GF(2^8)$

$$inv(a) \coloneqq \begin{cases} 0, & if\ a = 0, \\ a^{-1}, & otherwise. \end{cases}$$

- **Affine Transformation** (over $GF(2^8)$)

$$b_i' = b_i \oplus b_{(i+4)\ mod\ 8} \oplus b_{(i+5)\ mod\ 8} \oplus b_{(i+6)\ mod\ 8} \oplus b_{(i+7)\ mod\ 8} \oplus c_i.$$

for $0 \le i < 8$, where b_i is the i^{th} bit of the byte, and c_i is the i^{th} bit of a byte c with the value $\{63\}$ or $\{01100011\}_2$.

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

1. AES - 구조 - ShiftRows


```
def ShiftRows()
```

$S_{r,0}$ $S_{r,1}$ $S_{r,2}$ $S_{r,3}$ $\xrightarrow{\texttt{ShiftRows()}}$ $S'_{r,0}$ $S'_{r,1}$ $S'_{r,2}$ $S'_{r,3}$

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

S

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

S'

1. AES - 구조 - MixColumns, AddRoundKey


```
def MixColumns()
```

$\otimes$: Multiplication of two polynomials modulo $x^4 + 1$.

Columns : polynomials over $GF(2^8)$

multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

$$s'(x) = a(x) \otimes s(x).$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2.c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad , \text{ for } 0 \le c < Nb.$$

```
def AddRoundKey()
```

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{l,c}, w_{l+1,c}, w_{l+2,c}, w_{l+3,c}] \text{ for } 0 \le c < Nb,$$

where $\mathrm{l} = \mathrm{round} * Nb$, $[w_i]$: key schedule word, $0 \le round \le Nr$.

	Nb (word)	Nr
AES-128	4	10

1. AES - 구조 - Key Expansion


```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin

        word temp

        i = 0

        while (i < Nk)
                w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
                i = i+1
        end while

        i = Nk

        while (i < Nb * (Nr+1)]
                temp = w[i-1]
                if (i mod Nk = 0)
                        temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
                else if (Nk > 6 and i mod Nk = 4)
                        temp = SubWord(temp)
                end if
                w[i] = w[i-Nk] xor temp
                i = i + 1
        end while
end
```

2. 유한체 연산

Finite Field

A finite field is a field F which contains a finite number of elements.
$order(F) = p^k$ $(p \text{ is prime}, k \in \mathbb{Z}^+)$.

유한체 F_{2^8}의 원소	벡터공간 F_{2^8}의 벡터	정수 0 ~ 255
$a_7x^7 + a_6x^6 + \cdots + a_0$	$(a_7, a_6, \ldots, a_0)$	$\Sigma_{j=0}^{7} a_j 2^j$
$x^6 + x^5 + x + 1$	$(0,1,1,0,0,0,1,1)$	$\{99\}_{10}$ $\{63\}$

$GF(2^8)$에서의 덧셈 – XOR ($\oplus$)

두 바이트 $\{a_7a_6a_5a_4a_3a_2a_1a_0\}$와 $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$의 덧셈 = $\{c_7c_6c_5c_4c_3c_2c_1c_0\}$

where $each\ c_i = a_i \oplus b_i,\ 0 \le i \le 7.$

2. 유한체 연산

$GF(2^8)$에서의 곱셈

$$a(x) \cdot b(x) \coloneqq a(x) \times b(x) \ \ mod\ m(x).$$

기약 다항식 $m(x) = x^8 + x^4 + x^3 + x + 1.$

→ 곱셈의 결과를 항상 8차 미만으로 유지시킨다.

- associative
- $\{01\}$: multiplicative identity

→ 8차 미만의 0이 아닌 이진 다항식 $b(x)$에 대해 multiplicative inverse가 존재한다. $(b^{-1}(x))$

Extended Euclidean Algorithm.

$$b(x)a(x) + m(x)c(x) = 1.$$

$$a(x) \cdot b(x)\ mod\ m(x) = 1.$$

$$b^{-1}(x) = a(x)\ mod\ m(x).$$

For any $a(x), b(x)$ and $c(x)$ in the field, it holds that

$$a(x) \cdot \big(b(x) + c(x)\big) = a(x) \cdot b(x) + a(x) \cdot c(x).$$

$* \ m(x) = x^8 + x^4 + x^3 + x + 1.$

2. 유한체 연산

$GF(2^8)$에서의 곱셈 - $xtime()$

$$a(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0.$$

$$x \cdot a(x) = a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x.$$

$$\text{If} \begin{cases} a_7 = 0, \\ a_7 = 1. \quad \rightarrow \oplus\, m(x) = x^8 + x^4 + x^3 + x + 1. \end{cases}$$

(byte) left shift ($\ll 1$)

→ $xtime()$을 이용하면 다항식의 차수를 **7차 이하로 유지**하면서 연산할 수 있다.

2. 유한체 연산

계수가 $GF(2^8)$인 다항식

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0. \qquad \rightarrow [a_0, a_1, a_2, a_3] \text{ (word)}$$

계수가 $GF(2^8)$인 다항식 – 덧셈

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0).$$

계수가 $GF(2^8)$인 다항식 – 곱셈

$$c(x) = a(x) \cdot b(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 ,$$

$$\begin{aligned} where\ c_0 &= (a_0 \cdot b_0), \\ c_1 &= (a_1 \cdot b_0) \oplus (a_0 \cdot b_1), \\ c_2 &= (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2), \end{aligned}$$

$$\begin{aligned} c_3 &= (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3), \\ c_4 &= (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3), \\ c_5 &= (a_3 \cdot b_2) \oplus (a_2 \cdot b_3), \\ c_6 &= (a_3 \cdot b_3). \end{aligned}$$

계수가 $GF(2^8)$인 다항식 – 곱셈 – AES

polynomial $x^4 + 1$, $\qquad x^i \bmod (x^4+1) = x^{i \bmod 4}$.

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 .$$

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0,$$

$$\begin{aligned}
d_0 &= (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3),\\
d_1 &= (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a_2 \cdot b_3),\\
d_2 &= (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3),\\
d_3 &= (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3).
\end{aligned}$$

$a(\mathrm{x})$: fixed polynomial

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

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

3. 유한체 연산을 적용한 AES 구현 - SubBytes

$$a(x) \cdot b(x) \coloneqq a(x) \times b(x) \ \ mod\, m(x).$$

$$* \, m(x) = x^8 + x^4 + x^3 + x + 1.$$

Multiplication Polynomial

```python
def mul_polynomial(a, b):
    if a == 0 or b == 0:
        return 0
    c = 0

    while a != 0:
        if a & 1 == 1:
            c ^= b

        b <<= 1
        a >>= 1

    return c
```

Ex. $a = 11101, b = 1001.$

```
    1001
x  11101
--------
    1001
   0000
  1001
 1001
1001
--------
11100101
```

3. 유한체 연산을 적용한 AES 구현 - SubBytes

$$a(x) \cdot b(x) \coloneqq a(x) \times b(x) \ \ mod\ m(x).$$

$$* \ m(x) = x^8 + x^4 + x^3 + x + 1.$$

```
# Modulo an irreducible polynomial
```

```python
def mod_polynomial(a, m):
    bit_m = m.bit_length()
        while True:
        bit_a = a.bit_length()
        if bit_a < bit_m:
            break
        mshift = m << (bit_a - bit_m)
        a ^= mshift
    return a
```

Ex. $a = 11100101, \quad m = 1001001.$

$$\begin{array}{r|l} & \ \ \ \ \ \ \ \ \ \ \ \ \ 11 \\ \hline 1001001 & 11100101 \\ & 1001001\ \ \\ \hline & 01110111 \\ & 1001001 \\ \hline & 0111110 \end{array}$$

3. 유한체 연산을 적용한 AES 구현 - SubBytes

$$a(x) \cdot b(x) \coloneqq a(x) \times b(x) \ \ mod\, m(x).$$

$$* \, m(x) = x^8 + x^4 + x^3 + x + 1.$$

```
# Multiplication in Gf(2^8)
```

```
def aes_gmult(a, b):
    # m(x) = x^8 + x^4 + x^3 + x + 1
    # m = (100011011)_2
    return mod_polynomial(mul_polynomial(a, b), 0b100011011)
```

3. 유한체 연산을 적용한 AES 구현 - SubBytes

xtime(f(x)) = x * f(x)

```
def xtime(a):
    b = (a >> 7) & 0x01
    if b == 1:
        res = (a << 1) ^ 0x1b
    else:
        res = a << 1
    return res & 0xff
```

Multiplication in Gf(2^8)- xtime()

```
def aes_gmult(a, b):
    d = 0x00

    for i in range(7, -1, -1):
        coef = (a >> i) & 1
        d = xtime(d)
        if coef == 1:
            d = gadd(d, b)

    return d
```

$$a(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0.$$

$$b(x)a(x) = b(x)a_0 + x(b(x)a_1 + x(b(x)a_2 + x \ldots .$$

```
def gadd(a, b):
    return a^b
```

3. 유한체 연산을 적용한 AES 구현 - SubBytes

Multiplicative Inverse - Extended Euclidean Algorithm

Define u_j *and* v_j *recursively as follows.*

$$u_0 = 1, \qquad v_0 = 0,$$
$$u_1 = 0, \qquad v_1 = 1$$

and for $j = 2, 3, \ldots, n,$

$$u_j = u_{j-2} - q_{j-1}u_{j-1}, \qquad v_j = v_{j-2} - q_{j-1}v_{j-1}$$

where q_j *is the quotient in Euclidean algorithm. Then, for* $j = 0, 1, \ldots, n,$

$$r_j = u_j a + v_j b,$$

and especially, $\gcd(a, b) = r_n = u_n a + v_n b.$

$$m(x) = x^8 + x^4 + x^3 + x + 1. \rightarrow \{01\}\{1b\}$$

$$\gcd(a, m) = 1 = u_n a + v_n m. \longrightarrow u_n a \equiv 1 \mod m.$$

a의 역원 $= u_n.$

3. 유한체 연산을 적용한 AES 구현 - SubBytes

Multiplicative Inverse - Extended Euclidean Algorithm

$$m(x) = x^8 + x^4 + x^3 + x + 1. \rightarrow \{283\}_{10}$$

```
def mul_inverse(a):
    m = 283
    if a == 0:
        return 0
    u0 = 1
    u1 = 0
    t0 = a
    t1 = m

    while t1!=0 and t1!=1:
        t2 = t0
        t0 = t1

        q_r = qr(t2, t1)
        q = q_r[0]
        t1 = q_r[1]

        u2 = u0
        u0 = u1
        u1 = mod_polynomial((u2 ^ mul_polynomial(q, u1)), m)

    return u1
```

3. 유한체 연산을 적용한 AES 구현 - SubBytes

Affine Transformation

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

```python
def affine_trans(a):
    affine = [0b11111000, 0b01111100, 0b00111110, 0b00011111,
0b10001111, 0b11000111, 0b11100011, 0b11110001]
    b = ''

    for i in range(8):
        b += str(bin(a & affine[i]).count('1') % 2)

    b = int(b, 2) ^ int('01100011', 2)

    return b
```

def SubBytes() : $s(a) := T(inv(a))$

```python
def sub_bytes(state):
    for i in range(4):
        for j in range(Nb):
            state[Nb*i+j] = affine_trans(mul_inverse(state[Nb*i+j]))
    return state
```

3. 유한체 연산을 적용한 AES 구현 - InvSubBytes

Inverse of the Affine Transformation

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

```python
def inv_affine_trans(a):
    inv_affine = [0b01010010, 0b00101001, 0b10010100,
0b01001010, 0b00100101, 0b10010010, 0b01001001, 0b10100100]
    b = ''

    for i in range(8):
        b += str(bin(a & inv_affine[i]).count('1') % 2)

    b = int(b, 2) ^ int('00000101', 2)

    return b
```

def Inv_SubBytes() : $s^{-1}(x) \coloneqq inv\left(T^{-1}(a)\right)$

```python
def inv_sub_bytes(state):
    for i in range(4):
        for j in range(Nb):
            state[Nb*i+j] = mul_inverse(inv_affine_trans(state[Nb*i+j]))
    return state
```

3. 유한체 연산을 적용한 AES 구현 - KeyExpansion

Addition of 4 byte words

```
def coef_add(a, b):
    d = list(range(4))
    d[0] = a[0]^b[0]
    d[1] = a[1]^b[1]
    d[2] = a[2]^b[2]
    d[3] = a[3]^b[3]
    return d
```

def key_expansion()

```
def aes_key_expansion(key, w):
    tmp = list(range(4))
    k_len = Nb*(Nr+1)

    for i in range(Nk):
        for k in range(4):
            w[4*i+k] = key[4*i+k]

    for i in range(Nk, k_len):
        for k in range(4):
            tmp[k] = w[4*(i-1)+k]

        if (i%Nk == 0):
            tmp = rot_word(tmp)
            tmp = sub_word(tmp)
            tmp = coef_add(tmp, Rcon(i/Nk))

        for k in range(4):
            w[4*i+k] = w[4*(i-Nk)+k]^tmp[k]

    return w
```

3. 유한체 연산을 적용한 AES 구현 - MixColumns

Multiplication of 4 byte words

```
def coef_mult(a, b):
    d = list(range(4))
    d[0] = aes_gmult(a[0],b[0])^aes_gmult(a[3],b[1])^aes_gmult(a[2],b[2])^aes_gmult(a[1],b[3])
    d[1] = aes_gmult(a[1],b[0])^aes_gmult(a[0],b[1])^aes_gmult(a[3],b[2])^aes_gmult(a[2],b[3])
    d[2] = aes_gmult(a[2],b[0])^aes_gmult(a[1],b[1])^aes_gmult(a[0],b[2])^aes_gmult(a[3],b[3])
    d[3] = aes_gmult(a[3],b[0])^aes_gmult(a[2],b[1])^aes_gmult(a[1],b[2])^aes_gmult(a[0],b[3])
    return d
```

def mix_columns()

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

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.$$

```
def mix_columns(state):
    a = [0x02, 0x01, 0x01, 0x03]
    col = list(range(4))

    for j in range(Nb):
        for i in range(4):
            col[i] = state[Nb*i+j]

        res = coef_mult(a, col)

        for i in range(4):
            state[Nb*i+j] = res[i]
    return state
```

3. 유한체 연산을 적용한 AES 구현 - InvMixColumns

Multiplication of 4 byte words

```
def coef_mult(a, b):
    d = list(range(4))
    d[0] = aes_gmult(a[0],b[0])^aes_gmult(a[3],b[1])^aes_gmult(a[2],b[2])^aes_gmult(a[1],b[3])
    d[1] = aes_gmult(a[1],b[0])^aes_gmult(a[0],b[1])^aes_gmult(a[3],b[2])^aes_gmult(a[2],b[3])
    d[2] = aes_gmult(a[2],b[0])^aes_gmult(a[1],b[1])^aes_gmult(a[0],b[2])^aes_gmult(a[3],b[3])
    d[3] = aes_gmult(a[3],b[0])^aes_gmult(a[2],b[1])^aes_gmult(a[1],b[2])^aes_gmult(a[0],b[3])
    return d
```

def inv_mix_columns()

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

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

```
def inv_mix_columns(state):
    a = [0x0e, 0x09, 0x0d, 0x0b]
    col = list(range(4))

    for j in range(Nb):
        for i in range(4):
            col[i] = state[Nb*i+j]

        res = coef_mult(a, col)

        for i in range(4):
            state[Nb*i+j] = res[i]
    return state
```

3. 유한체 연산을 적용한 AES 구현 - MixColumns

def xtime_mul()

```
def xtime_mul(a, num):
    if num == 0x01:
        return a
    elif num == 0x02:
        return xtime(a)
    elif num == 0x03:
        return xtime_mul(a, 0x02) ^ xtime_mul(a, 0x01)
    elif num == 0x04:
        return xtime_mul(xtime_mul(a, 0x02), 0x02)
    elif num == 0x08:
        return xtime_mul(xtime_mul(a, 0x02), 0x04)
    elif num == 0x09:
        return xtime_mul(a, 0x08) ^ xtime_mul(a, 0x01)
    elif num == 0x0b:
        return xtime_mul(a, 0x08) ^ xtime_mul(a, 0x03)
    elif num == 0x0d:
        return xtime_mul(a, 0x08) ^ xtime_mul(a, 0x04)
                    ^ xtime_mul(a, 0x01)
    elif num == 0x0e:
        return xtime_mul(a, 0x08) ^ xtime_mul(a, 0x04)
                    ^ xtime_mul(a, 0x02)
```

$$A \cdot \mathbf{0x01} = A,$$

$$A \cdot \mathbf{0x02} = (A \ll 1) \oplus (A \gg 7) \cdot \mathbf{0x1b},$$

$$A \cdot \mathbf{0x03} = A \cdot (\mathbf{0x02{+}0x01}) = (A \cdot \mathbf{0x02}) \oplus A,$$

$$A \cdot \mathbf{0x04} = \mathbf{0x02} \cdot (A \cdot \mathbf{0x02}),$$

$$A \cdot \mathbf{0x08} = \mathbf{0x02} \cdot (A \cdot \mathbf{0x04}),$$

$$A \cdot \mathbf{0x09} = (A \cdot \mathbf{0x08}) \oplus (A \cdot \mathbf{0x01}),$$

$$A \cdot \mathbf{0x0b} = (A \cdot \mathbf{0x08}) \oplus (A \cdot \mathbf{0x03}),$$

$$A \cdot \mathbf{0x0d} = (A \cdot \mathbf{0x08}) \oplus (A \cdot \mathbf{0x04}) \oplus (A \cdot \mathbf{0x01}),$$

$$A \cdot \mathbf{0x0e} = (A \cdot \mathbf{0x08}) \oplus (A \cdot \mathbf{0x04}) \oplus (A \cdot \mathbf{0x02}).$$

3. 유한체 연산을 적용한 AES 구현 - MixColumns

def mix_columns() - xtime

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad , a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.$$

```python
def mix_columns(state):
    col = list(range(4))
    res = list(range(4))

    for j in range(Nb):
        for i in range(4):
            col[i] = state[Nb*i+j]

        res[0] = (xtime_mul(col[0], 0x02)) ^ (xtime_mul(col[1], 0x03)) ^ (xtime_mul(col[2], 0x01)) ^ (xtime_mul(col[3], 0x01))
        res[1] = (xtime_mul(col[0], 0x01)) ^ (xtime_mul(col[1], 0x02)) ^ (xtime_mul(col[2], 0x03)) ^ (xtime_mul(col[3], 0x01))
        res[2] = (xtime_mul(col[0], 0x01)) ^ (xtime_mul(col[1], 0x01)) ^ (xtime_mul(col[2], 0x02)) ^ (xtime_mul(col[3], 0x03))
        res[3] = (xtime_mul(col[0], 0x03)) ^ (xtime_mul(col[1], 0x01)) ^ (xtime_mul(col[2], 0x01)) ^ (xtime_mul(col[3], 0x02))

        for i in range(4):
            state[Nb*i+j] = res[i]

    return state
```

3. 유한체 연산을 적용한 AES 구현 - InvMixColumns

def inv_mix_columns() - xtime

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad , \; a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

```
def mix_columns(state):
    def inv_mix_columns(state):
    col = list(range(4))
    res = list(range(4))

    for j in range(Nb):
        for i in range(4):
            col[i] = state[Nb*i+j]

        res[0] = (xtime_mul(col[0], 0x0e)) ^ (xtime_mul(col[1], 0x0b)) ^ (xtime_mul(col[2], 0x0d)) ^ (xtime_mul(col[3], 0x09))
        res[1] = (xtime_mul(col[0], 0x09)) ^ (xtime_mul(col[1], 0x0e)) ^ (xtime_mul(col[2], 0x0b)) ^ (xtime_mul(col[3], 0x0d))
        res[2] = (xtime_mul(col[0], 0x0d)) ^ (xtime_mul(col[1], 0x09)) ^ (xtime_mul(col[2], 0x0e)) ^ (xtime_mul(col[3], 0x0b))
        res[3] = (xtime_mul(col[0], 0x0b)) ^ (xtime_mul(col[1], 0x0d)) ^ (xtime_mul(col[2], 0x09)) ^ (xtime_mul(col[3], 0x0e))

        for i in range(4):
            state[Nb*i+j] = res[i]

    return state
```

4. AES 8x32 Table 고속구현

8 bit 열 $\{x_j\}_{j=0}^{15} \rightarrow$ 32 bit 열 $\{X_j\}_{j=0}^{3}$

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
X_0				X_1				X_2				X_3			

$$X_j := (x_{4j} \ll 24) || (x_{4j+1} \ll 16) || (x_{4j+2} \ll 8) || (x_{4j+3}) \;, for\ j = 0, 1, 2, 3.$$

4. AES 8x32 Table 고속구현

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \xrightarrow{\text{SubBytes}} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} \xrightarrow{\text{ShiftRows}} \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} \xrightarrow{\text{MixColumns}} \begin{bmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{bmatrix}$$

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,0}] \\ S[a_{1,1}] \\ S[a_{2,2}] \\ S[a_{3,3}] \end{bmatrix} = S[a_{0,0}] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S[a_{1,1}] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S[a_{2,2}] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S[a_{3,3}] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix}$$

$$= T0[a_{0,0}] \oplus T1[a_{1,1}] \oplus T2[a_{2,2}] \oplus T3[a_{3,3}].$$

4. AES 8x32 Table 고속구현

$$T0[x] = \begin{bmatrix} 02 * S[x] \\ S[x] \\ S[x] \\ 03 * S[x] \end{bmatrix}, T1[x] = \begin{bmatrix} 03 * S[x] \\ 02 * S[x] \\ S[x] \\ S[x] \end{bmatrix}, T2[x] = \begin{bmatrix} S[x] \\ 03 * S[x] \\ 02 * S[x] \\ S[x] \end{bmatrix}, T3[x] = \begin{bmatrix} S[x] \\ S[x] \\ 03 * S[x] \\ 02 * S[x] \end{bmatrix}.$$

```
def T_e0():
    res = []
    for i in range(256):
        res.append((((aes_gmult(0x02, s_box(i)) << 24) ^ ((s_box(i) << 16)) ^ ((s_box(i) << 8)) ^
         ((aes_gmult(0x03,s_box(i)))))))
    return res
```

```
# T_e1 = T_shift(T_e0)
```

```
def T_shift(table):
    for i in range(256):
        tmp = (table[i] & 0xff)
        table[i] = ((table[i] >> 8) & 0x00ffffff) ^ (tmp << 24)
    return table
```

4. AES 8x32 Table 고속구현


```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
        ...

        while (i < Nb * (Nr+1)]
                temp = w[i-1]
                if (i mod Nk = 0)
                        temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
                else if (Nk > 6 and i mod Nk = 4)
                        temp = SubWord(temp)
                end if
                w[i] = w[i-Nk] xor temp
                i = i + 1
        end while
end
```

$$T0[x] = \begin{bmatrix} 02 * S[x] \\ S[x] \\ S[x] \\ 03 * S[x] \end{bmatrix}.$$

```python
# s_box_table(Te0)로 고정
def s_box_table(table):
    for i in range(256):
        table[i] = (table[i] >> 8) & 0xff
    return table
```

5. 시간측정 및 비교

유한체 연산을 적용한 AES128 시간 측정

```
def time_test():
    global aes_in
    global key

    start_time = datetime.datetime.now()

    aes_out = list(range(16))

    w = list(range(Nb*(Nr+1)*4))
    w = aes_key_expansion(key, w)

    aes_out = aes_cipher(aes_in, aes_out, w)
    aes_in = aes_inv_cipher(aes_out, aes_in, w)

    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time

    return elapsed_time.microseconds
```

5. 시간측정 및 비교

8x32 Table look-up AES128 시간 측정

```
def time_test():
    global aes_in
    global key

    start_time = datetime.datetime.now()

    aes_out = list(range(16))
    w = list(range(Nb*(Nr+1)*4))
    w = KeySchedule(key, w)
    rk32 = [[0 for col in range(4)] for row in range(11)]
    rk32 = AES32_Enc_KeySchedule(w, rk32)

    aes_out = AES32_enc(aes_in, rk32, aes_out)

    decrk32 = AES32_Dec_KeySchedule(rk32)

    aes_in = AES32_dec(aes_out, decrk32, aes_in)

    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return elapsed_time.microseconds
```

5. 시간측정 및 비교

$*$ 동일한 plaintext와 key 사용 (AESAVS의 파라미터 사용)

$*$ 500번의 테스트 값 평균

# 유한체 연산을 적용한 AES128	# 8x32 Table look-up AES128
6502.396 us	207.514 us

$* \ 1 \text{ us} = 10^{-6} \ seconds$

*8x32 테이블의 메모리 사용량

4 bytes x 256개 원소 x 4개 Table x 2 (암복호화)
= 8.192KB ≈ 대략 8KB

6. 참고문헌

*NIST, FIPS 197, Advanced Encryption Standard (AES). November 26, 2001.

*Lawrence E. Bassham III, The Advanced Encryption Standard Algorithm Validation Suite (AESAVS). November 15, 2002.

*Secure Protocols, lecture note. Fall, 2020. 김동찬 교수님.

*보안 S/W 구현, lecture note. Fall, 2021. 염용진 교수님.

AES128 구현 코드 링크

https://github.com/enoma422/aes.py