AES 규격

김동현(wlswudpdlf31@kookmin.ac.kr)

March 12, 2025

Contents

1 참고 자료

2 정의

2.1 state

AES 블록 암호는 상태(state) 라고 하는 4×4 크기의 2차원 바이트 배열을 가지고 수행한다. 이 문서에서 상태 배열은 s로 표현하며, 각 개별 바이트는 두 개의 인덱스를 가진다. 행 인덱스 r의 범위는 $0 \leq r < 4$이고, 열 인덱스 c의 범위는 $0 \leq c < 4$이다. *state* 배열의 개별 바이트는 $s_{r,c}$로 나타낸다. state 배열 S를 word로 표현할 때는 다음과 같이 표현한다.

$$S[c] = \begin{pmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{pmatrix}$$

$S[i]$는 인덱스 i에 해당하는 워드를 나타내며, 4개의 연속된 워드 시퀀스는 $S[i : i + 4]$로 표현한다.

이후 절에서 설명할 AES 블록 암호 규격에서는 입력 바이트 배열 $I_0, I_1, \cdots, I_{15}$를 state 배열 S에 복사하며, 이는 다음과 같이 수행한다.

$$S_{r,c} = I_{r+4c} \quad \text{for } 0 \leq r < 4 \text{ and } 0 \leq c < 4.$$

AES 블록 암호 규격에서 state 배열을 변환한 후에는 출력 바이트 배열 $O_0, O_1, \cdots, O_{15}$에 복사하며, 다음과 같이 수행한다.

$$O_{r+4c} = S_{r,c} \quad \text{for } 0 \leq r < 4 \text{ and } 0 \leq c < 4.$$

2.2 $\mathbb{F}_{2^8}$에서의 덧셈

state 배열의 각 바이트는 유한 체의 256개의 원소 중 하나로 해석할 수 있다. 이 유한체는 $\mathbb{F}_{2^8}$로 표현한다. $\mathbb{F}_{2^8}$에서의 덧셈이나 곱셈을 정의하기 위해, 바이트 $b = \{b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0\}$를 다항식 $b(x)$로 표기하면 다음과 같다.

$$b(x) = b_7 x^7 + b_6 x^6 + \cdots + b_1 x + b_0.$$

유한체 $\mathbb{F}2^8$에서 두 원소를 더하기 위해, 원소를 나타내는 다항식의 계수들은 2를 법으로 하는 덧셈, 즉 배타적 논리합 연산을 수행하여 더해진다. 두 바이트는 각각의 대응하는 비트에 배타적 논리합 연산을 적용하여 더할 수 있다. 즉, 아래 덧셈은 모두 동등하다.

- $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$.
- `0b01010111` $\oplus$ `0b10000011` = `0b11010100`.
- `0x57` $\oplus$ `0x83` = `0xd4`.

다항식 계수가 2를 법으로 하여 줄어드므로, 계수 -1은 계수 1과 동등하다. 따라서 뺄셈은 덧셈과 동일하다.

2.3 $\mathbb{F}_{2^8}$에서의 곱셈

두 바이트의 곱셈은 다음과 같이 정의한다.

- 바이트를 나타내는 두 다항식을 곱한다.
- 곱해서 나온 다항식을 다음 다항식으로 나눈 나머지를 취한다.

$$x^8 + x^4 + x^3 + x + 1.$$

두 단계 모두, 다항식의 개별 계수는 2로 나눈 나머지로 계산한다. $b(x)$와 $c(x)$를 각각 두 바이트 b, c로 표현하면, 두 바이트 간 곱셈은 다음과 같이 다항식 간 모듈로 곱셈으로 표현할 수 있다.

$$b(x)c(x) \mod m(x).$$

다항식 $m(x)$에 대한 모듈로 감소는 $c = \texttt{0x02}$인 경우를 고려하면 유용하게 계산할 수 있다. b의 각 비트를 $b_0, \cdots, b_7$로 표현하여 $b = [b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0]$로 표현할 때, b와 0x02와의 곱셈을 다음과 같이 xTimes(b)로 나타낼 수 있다.

$$\text{xTimes}(b) = \begin{cases} [b_6 b_5 b_4 b_3 b_2 b_1 b_0 0], & \text{if } b_7 = 0 \\ [b_6 b_5 b_4 b_3 b_2 b_1 b_0 0] \oplus \texttt{0b00011011}, & \text{if } b_7 = 1. \end{cases}$$

이를 이용하면 거듭 제곱, 즉 b와 0x04, 0x08과 같은 곱셈은 xTimes를 반복 적용하여 구현할 수 있다.

이제 b와 임의의 바이트간의 곱셈을 xTimes를 사용하여 유용하게 계산할 수 있다. 예를 들어, 0x57과 0x13간의 곱셈은 다음과 같이 계산할 수 있다.

$$\begin{aligned} \texttt{0x57} \cdot \texttt{0x13} &= \texttt{0x57} \cdot (\texttt{0x01} \oplus \texttt{0x02} \oplus \texttt{0x10}) \\ &= \texttt{0x57} \oplus (\texttt{0x57} \cdot \texttt{0x02}) \oplus (\texttt{0x57} \cdot \texttt{0x10}). \end{aligned}$$

2.4 $\mathbb{F}_{2^8}$에서의 역원

0x00이 아닌 바이트 b에 대해, 곱셈 역원 b^{-1}은 유일하게 정의되며, 다음을 만족한다.

$$b \cdot b^{-1} = \texttt{0x01}.$$

그리고 곱셈 역원은 확장 유클리드 알고리듬을 사용하여 계산하거나, 다음과 같이 계산할 수 있다.

$$b^{-1} = b^{254}.$$

AES 블록 암호 규격에서 SubBytes변환은 $\mathbb{F}2^8$에서의 곱셈 역원을 포함하는데, 이를 테이블로 대체하여 역원 계산 없이 구현할 수 있다.

3 AES 규격

AES-128, AES-192 또는 AES-256을 실행하는 일반적인 함수는 Cipher로 나타내며, 그 역함수는 InvCipher로 표시한다.

Cipher및 InvCipher알고리즘의 state에 대한 일정한 변환 과정인 라운드(round)의 연속적인 수행이다. 각 라운드는 라운드 키(round key)라고 하는 추가 입력을 필요로 하며, 라운드 키는 일반적으로 네 개의 워드(word)로 구성된 블록, 즉 16바이트로 표현된다.

KeyExpansion이라고 하는 확장 함수는 블록 암호화 키를 입력으로 받아 라운드 키를 생성합니다. 구체적으로, KeyExpansion의 입력은 워드 배열 key로 표현되며, 출력은 확장된 워드 배열 w 로 나타납니다. 이 확장된 키 배열을 키 스케줄(key schedule)이라고 한다.

AES-128, AES-192 및 AES-256 블록 암호는 다음 세 가지 측면에서 차이가 있다.

- 키 길이
- 라운드 수 (이는 필요한 키 스케줄의 크기를 결정함)
- KeyExpansion내에서의 재귀(recursion) 규격

각 알고리즘에서 라운드 수는 n_r, 키 길이의 워드 수는 n_k로 표시되고, 블록의 워드 수는 n_b로 나타낸다. n_b, n_k, n_r 값은 표 3에 제시되어 있다.

	블록 길이 n_b	키 길이 n_k	라운드 수 n_r
AES-128	4 (128 bits)	4 (128 bits)	10
AES-192	4 (128 bits)	6 (192 bits)	12
AES-256	4 (128 bits)	8 (256 bits)	14

Cipher함수 규격은 1 절을 참고한다. InvCipher함수 규격은 2 절을 참고한다. KeyExpansion 함수 규격은 3 절을 참고한다.

3.1 Cipher

Algorithm 1 Cipher

Require: 입력 배열 I, 라운드 수 n_r, 라운드 키 R
Ensure: 출력 배열 O

procedure Cipher(I, n_r, R)
  $S \leftarrow I$
  $S \leftarrow$ AddRoundKey($S, R_{[0:4]}$)
  for $i = 1$ to $n_r - 1$ **do**
    $S \leftarrow$ SubBytes(S)
    $S \leftarrow$ ShiftRows(S)
    $S \leftarrow$ MixColumns(S)
    $S \leftarrow$ AddRoundKey($S, R_{[4i:4(i+1)]}$)
  end for
  $S \leftarrow$ SubBytes(S)
  $S \leftarrow$ ShiftRows(S)
  $S \leftarrow$ AddRoundKey($S, R_{[4n_r:4(n_r+1)]}$)
  $O \leftarrow S$
  return O
end procedure

Cipher의 입력 데이터, 라운드 수, 라운드 키 세 개이다. 데이터는 16 바이트 선형 배열로 표현되는 블록이며, 라운드 수는 해당 AES 인스턴스에 대한 라운드 수이다. 예를 들어, AES-128의 Cipher 함수는 다음과 같이 표현된다.

$$\text{Cipher}(in, 10, \text{KeyExpansion}(key)).$$

Cipher에서 라운드는 SubBytes, ShiftRows, MixColumns, AddRoundKey 네 가지 바이트 단위 변환을 포함한다. 이 네 가지 변환 규격은 하위 절에서 설명한다.

첫 번째 단계(line 2)는 입력을 *state* 에 복사한다. 초기 라운드 키 추가(line 3) 후, *state*는 n_r 번의 라운드 함수 변환을 거친다(line 4 to 9). 마지막 라운드(line 10 to 12)는 MixColumns변환이 생략된다는 점에서 이전 라운드들과 다릅니다. 최종 state는 출력으로 반환된다(line 13).

3.1.1 AddRoundKey

AddRoundKey은 *state*에 라운드 키를 적용하여 비트 단위 XOR 연산을 수행하는 변환이다. 과정은 다음과 같이 표현된다.

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{0,4r+c}, w_{1,4r+c}, w_{2,4r+c}, w_{3,4r+c}], \quad \text{for } 0 \le c < 4.$$

AddRoundKey는 Cipher에서 총 $n_r + 1$번 호출된다. 첫번째 라운드 함수 적용 전에 한 번, 그리고 n_r 번의 라운드에서 한 번씩 호출된다.

3.1.2 SubBytes

SubBytes는 *state*에 대해 가역적이고 비선형적인 변환을 수행하는 함수로, 각 바이트에 대해 SBox라고 불리는 치환 테이블을 독립적으로 적용한다. 입력 바이트 b를 SBox에 입력한다고 하자. 출력 바이트 b'은 다음 두 가지 변환을 조합하여 구성된다.

- 중간값 $\tilde{b}$을 다음과 같이 계산한다.

$$\tilde{b} = \begin{cases} \texttt{0x00}, & \text{if } b = \texttt{0x00} \\ b^{-1}, & \text{if } b \neq \texttt{0x00}. \end{cases}$$

- $\tilde{b}$의 비트에 다음 아핀 변환을 적용해 b'의 비트를 생성한다. (이 때, c는 상수 바이트로, $c =$ `0b01100011`이다.)

$$b'_i = \tilde{b}_i \oplus \tilde{b}_{(i+4) \bmod 8} \oplus \tilde{b}_{(i+5) \bmod 8} \oplus \tilde{b}_{(i+6) \bmod 8} \oplus \tilde{b}_{(i+7) \bmod 8} \oplus c_i.$$

3.1.3 Shiftrows

ShiftRows은 *state*에서 마지막 세 개의 행에 속한 바이트들이 순환적으로 이동하는 변환이다. 각 행에서 바이트가 이동하는 횟수는 인덱스 r에 따라 달라지며, 다음과 같이 정의된다.

$$s'_{r,c} = s_{r,(c+r) \bmod 4} \quad \text{for } 0 \le r < 4 \text{ and } 0 \le c < 4.$$

이 변환은 각 바이트를 해당 행에서 r만큼 왼쪽으로 이동시키고, 왼쪽에서 밀려난 바이트를 행의 오른쪽 끝으로 순환 시키는 것이다. 단, 첫번째 행은 변환되지 않고 그대로 유지된다.

3.1.4 Mixcolumns

MixColumns는 *state*의 각 열을 고정된 행렬과 곱하는 변환이다. 과정은 다음과 같다.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} \texttt{0x02} & \texttt{0x03} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x02} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x01} & \texttt{0x02} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < 4.$$

3.2 InvCipher

InvCipher는 Cipher의 변환들이 역순으로 실행된다. *state*의 역변환은 AddRoundKey, InvSubBytes, InvShiftRows, InvMixColumns으로 구성되며, AddRoundKey을 제외한 각 변환은 하위 절에서 설명한다.

Algorithm 2 InvCipher

Require: I, n_r, R ▷ $R = \text{KeyExpansion}(key)$
Ensure: O

procedure InvCipher(I, n_r, R)
  $S \leftarrow I$
  $S \leftarrow \text{AddRoundKey}(S, R_{[4n_r:4(n_r+1)]})$
  for $i = 1$ to $n_r - 1$ **do**
    $S \leftarrow \text{InvShiftRows}(S)$
    $S \leftarrow \text{InvSubBytes}(S)$
    $S \leftarrow \text{AddRoundKey}(S, R_{[4i:4(i+1)]})$
    $S \leftarrow \text{InvMixColumns}(S)$
  end for
  $S \leftarrow \text{InvShiftRows}(S)$
  $S \leftarrow \text{InvSubBytes}(S)$
  $S \leftarrow \text{AddRoundKey}(S, R_{[0:4]})$
  $O \leftarrow S$
  return O
end procedure

3.2.1 InvSubBytes

InvSubBytes는 SubBytes의 역연산이다. *state*의 각 바이트에 SubBytes의 역함수인 InvSBox가 적용된다. 즉, InvSBox는 SBox의 입력과 출력의 역할이 바뀌어 생성된다.

3.2.2 InvShiftRows

InvShiftRows는 ShiftRows의 역연산이다. *state*의 마지막 세 개의 행에 있는 바이트들은 다음과 같이 순환 이동한다.

$$s'_{r,c} = s_{r,(c-r) \bmod 4} \quad \text{for } 0 \le r < 4 \text{ and } 0 \le c < 4.$$

이 변환은 각 바이트를 해당 행에서 r 칸 씩 오른쪽으로 이동시키고, 오른쪽 끝에서 밀려난 r 개의 바이트를 왼쪽 끝으로 순환 이동시킨다. 이 때, 첫 행은 변경되지 않는다.

3.2.3 InvMixColumns

InvMixColumns는 MixColumns의 역연산이다. InvMixColumns는 *state* 에 고정된 행렬을 곱한다. 과정은 다음과 같다.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} \texttt{0x0e} & \texttt{0x0b} & \texttt{0x0d} & \texttt{0x09} \\ \texttt{0x09} & \texttt{0x0e} & \texttt{0x0b} & \texttt{0x0d} \\ \texttt{0x0d} & \texttt{0x09} & \texttt{0x0e} & \texttt{0x0b} \\ \texttt{0x0b} & \texttt{0x0d} & \texttt{0x09} & \texttt{0x0e} \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < 4.$$

3.3 KeyExpansion

KeyExpansion은 키를 적용하여 $4(n_r + 1)$개의 워드를 생성하는 함수이다. 따라서, Cipher및 InvCipher에서 AddRoundKey를 $n_r + 1$ 번 적용하기 위해 4개의 워드 씩 생성된다. 이 함수의 출력은 $w_{[0:4(n_r+1)]}$로 표시되는 선형 배열들의 워드들로 구성된다.

KeyExpansion은 $1 \le j \le 10$ 범위에서 Rcon[j]로 표시되는 10개의 고정된 워드를 호출한다. 이 10개의 워드를 라운드 상수라고 한다. Rcon[j] 값은 아래 표와 같다.

AES-128은 10개의 라운드 키를 생성할 때 마다 고유한 라운드 상수가 호출되고, AES-192 와 AES-256은 각 처음 8개와 7개의 라운드 상수를 호출한다.

Rcon[j]의 최상위 바이트 값은 다항식 형태로 x^{j-1}이다. $j > 0$인 경우, 이러한 바이트 값은 x^{j-1}로 표현된 바이트에 xTimes 연산을 반복적으로 생성할 수 있다.

KeyExpansion에서는 다음 두 가지 변환을 사용한다.

Rcon[1]:	[0x01, 0x00, 0x00, 0x00]	Rcon[6]:	[0x20, 0x00, 0x00, 0x00]
Rcon[2]:	[0x02, 0x00, 0x00, 0x00]	Rcon[7]:	[0x40, 0x00, 0x00, 0x00]
Rcon[3]:	[0x04, 0x00, 0x00, 0x00]	Rcon[8]:	[0x80, 0x00, 0x00, 0x00]
Rcon[4]:	[0x08, 0x00, 0x00, 0x00]	Rcon[9]:	[0x1b, 0x00, 0x00, 0x00]
Rcon[5]:	[0x10, 0x00, 0x00, 0x00]	Rcon[10]:	[0x36, 0x00, 0x00, 0x00]

- SubWord: 4 바이트 입력 워드를 $[a_0, a_1, a_2, a_3]$라고 할 때, 다음과 같다.

$$\text{SubWord}([a_0, a_1, a_2, a_3]) = [\text{SBox}(a_0), \text{SBox}(a_1), \text{SBox}(a_2), \text{SBox}(a_3)].$$

- RotWord: 위와 같은 입력 워드에 대해, 다음과 같다.

$$\text{RotWord}([a_0, a_1, a_2, a_3]) = [a_1, a_2, a_3, a_0].$$

Algorithm 3 KeyExpansion

Require: 키 K
Ensure: 라운드 키 R

procedure KeyExpansion(K)
  for $i = 0$ to $n_k - 1$ **do**
    $R_i \leftarrow K_i$
  end for
  for $i = n_k$ to $4n_r + 3$ **do**
    $t \leftarrow R_{i-1}$
    if $i \bmod n_k = 0$ **then**
      $t \leftarrow \text{SubWord}(\text{RotWord}(t)) \oplus \text{Rcon}[i/n_k]$
    else if $n_k > 6$ and $i \bmod n_k = 4$ **then** ▷ AES-256에서만 동작
      $t \leftarrow \text{SubWord}(t)$
    end if
    $R_i \leftarrow R_{i-n_k} \oplus t$
  end for
  return R
end procedure

키 확장 알고리즘은 위와 같다. 확장된 키의 첫 n_k 워드는 키 key이다. 이후 각 워드 $w[i]$는 이전 워드 $w[i-1]$ 및 n_k 앞 워드 $w[i-n_k]$를 사용하여 재귀적으로 생성한다. 이를 수식으로 나타내면 다음과 같다.

- 만약 i가 n_k의 배수면, $w[i]$를 다음과 같이 계산한다.

$$w[i] = w[i-n_k] \oplus \text{SubWord}(\text{RotWord}(w[i-1])) \oplus \text{Rcon}[i/n_k].$$

- 아니라면, 다음과 같이 계산한다.

$$w[i] = w[i-n_k] \oplus w[i-1].$$

  단, AES-256에서, $i+4$가 8의 배수일 때는 다음과 같이 계산한다.

$$w[i] = w[i-n_k] \oplus \text{SubWord}(w[i-1]).$$