대칭키 암호의 구현

YouTube: https://youtu.be/Z_bileFAnKc
Git: https://github.com/minpie/CryptoCraftLab-minpie_public

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

자기소개

계획한 향후 발표 주제

AES의 C언어 구현

DES의 C언어 구현

자기소개

- 차상민 입니다.
- 22학번 사이버보안, 정보시스템 트랙
- 현재 학부 3학년 입니다.
- 관심 분야:
  - 양자컴퓨팅, 병렬컴퓨팅, 블록체인, 하드웨어/임베디드 시스템, AI, etc.
- 취미:
  - 게임, 게임 모딩, 밀리터리 장비 관련 검색, 자전거타기, etc.

계획한 향후 발표 주제

- 1: 대칭키 암호 단일블록 암복호화의 C언어 구현 – AES, DES
- 2: OpenMPI와 AES 및 블록암호 운영모드별 병렬연산의 C언어 구현
- 3: 64비트 이상 키 길이의 공개키 암호의 C언어 구현 – RSA, Rabin, Elgamal, ECDSA

AES의 C언어 구현 - 개요

Federal Information

Processing Standards Publication 197

November 26, 2001

Announcing the

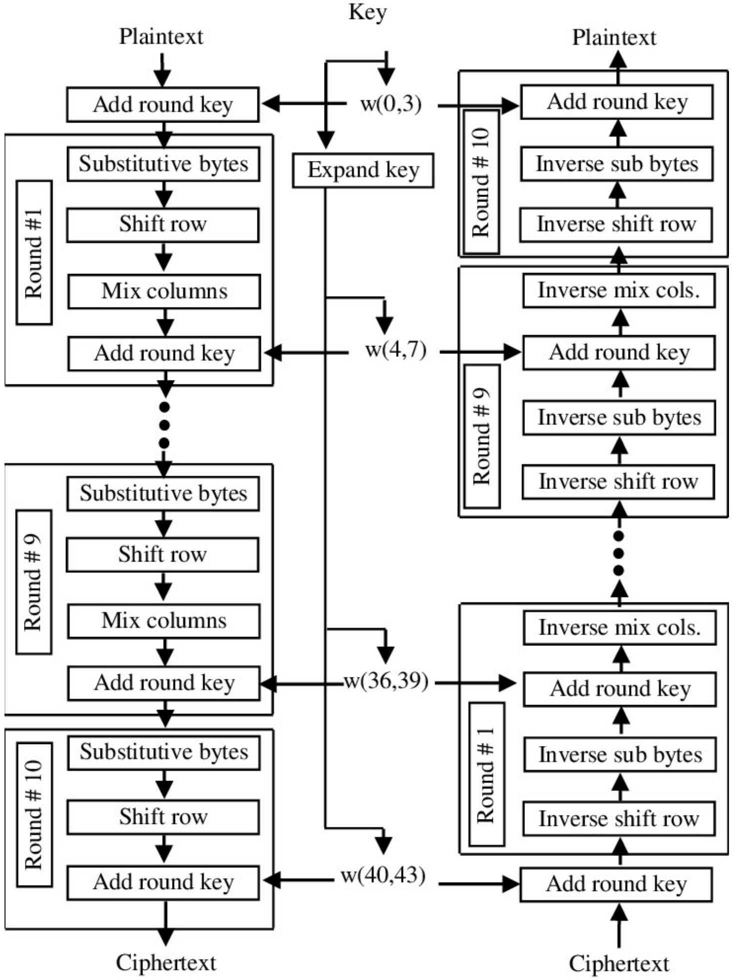
ADVANCED ENCRYPTION STANDARD (AES)

Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology (NIST) after approval by the Secretary of Commerce pursuant to Section 5131 of the Information Technology Management Reform Act of 1996 (Public Law 104-106) and the Computer Security Act of 1987 (Public Law 100-235).

- AES-128을 구현.
- 키 길이: 128비트
- 블록 길이: 128비트
- 구조: SPN 구조
- 라운드 수: 10라운드
- GF(2^8) 에서 연산을 정의

https://github.com/minpie/CryptoCraftLab-minpie_public/blob/main/%EC%95%94%ED%98%B8%EA%B5%AC%ED%98%84/AdvancedEncryptionStandard/main.c

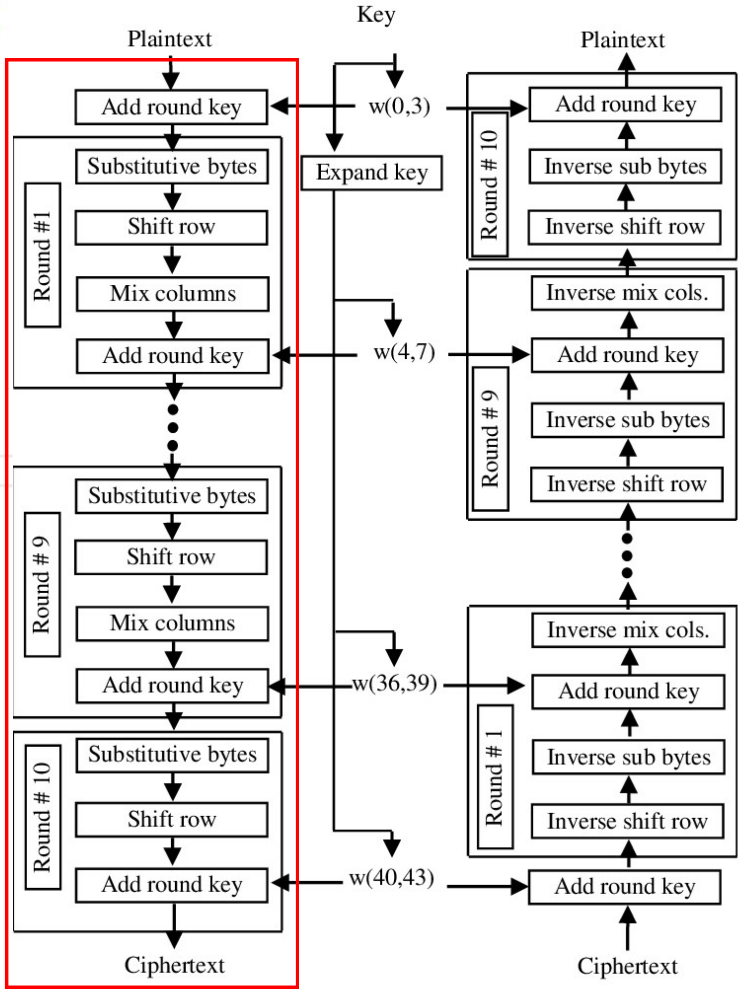
AES의 C언어 구현 - 개요



- AES-128을 구현.
- 키 길이: 128비트
- 블록 길이: 128비트
- 구조: SPN 구조
- 라운드 수: 10라운드
- GF(2^8) 에서 연산을 정의

AES의 C언어 구현 – 단일 블록 암호화

```c
void Encrypt(UINT8 * cipher, const UINT8 * plain, const UINT8 * initialKey){
    // ## 단일 블록 암호화 함수
    // # 0. 변수 초기화
    UINT8 state[16]; // 내부 state
    UINT8 roundKey[176]; // roundkey
    // # 1. Key Expansion
    KeyExpansion(roundKey, initialKey); // key expansion
    // # 2. Input To State
    ConvInputToState(state, plain); // plain to state
    // # 3. Rounds
    AddRoundKey(state, &(roundKey[0]));
    for(UINT8 rnd=1; rnd<10; rnd++){
        // SubBytes
        SubBytes(state);
        // ShiftRows
        ShiftRows(state);
        // MixColumns
        MixColumns(state);
        // AddRoundKey
        AddRoundKey(state, &(roundKey[(rnd*16)]));
    }
    // SubBytes
    SubBytes(state);
    // ShiftRows
    ShiftRows(state);
    // AddRoundKey
    AddRoundKey(state, &(roundKey[160]));
    // # 4. State To Output
    ConvStateToOutput(cipher, state); // state to cipher
}
```



AES의 C언어 구현 – KeyExpansion()

```
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
```

Note that *Nk*=4, 6, and 8 do not all have to be implemented; they are all included in the conditional statement above for conciseness. Specific implementation requirements for the Cipher Key are presented in Sec. 6.1.

Figure 11. Pseudo Code for Key Expansion.[2]

- 총 11개의 라운드 키 생성
- 각 라운드키는 ‘word’ 4개로 구성
- Rcon은 미리 정해진 값
- SubWord()는 S-Box 연산
- RotWord()는 1바이트 circular-right-shift 연산

AES의 C언어 구현 – KeyExpansion()

```c
void KeyExpansion(UINT8 * roundKeys, const UINT8 * initialKey){
    // ## KeyExpansion 함수
    // # 0. 변수 초기화
    UINT8 temp1[4];
    UINT8 temp2[4];
    const UINT8 RoundConstant[10][4] = {
        {0x01, 0, 0, 0},
        {0x02, 0, 0, 0},
        {0x04, 0, 0, 0},
        {0x08, 0, 0, 0},
        {0x10, 0, 0, 0},
        {0x20, 0, 0, 0},
        {0x40, 0, 0, 0},
        {0x80, 0, 0, 0},
        {0x1b, 0, 0, 0},
        {0x36, 0, 0, 0}
    };

    // # 1. w0~w3:
    CopyBytes(roundKeys, initialKey, 16);

    // # 2. w4~w43:
    for(UINT8 i=4; i<44; i++){
        CopyBytes(temp1, &(roundKeys[(i-1)*4]), 4); // temp = w[i-1]
        if(!(i % 4)){
            // temp1 = SubWord(RotWord(temp1)):
            RotWord(temp1);
            SubWord(temp1);

            // temp1 = temp1 XOR RoundConstant:
            temp2[0] = temp1[0] ^ RoundConstant[(i/4)-1][0];
            temp2[1] = temp1[1] ^ RoundConstant[(i/4)-1][1];
            temp2[2] = temp1[2] ^ RoundConstant[(i/4)-1][2];
            temp2[3] = temp1[3] ^ RoundConstant[(i/4)-1][3];
            CopyBytes(temp1, temp2, 4); // temp1 = temp2
        }
        // # w[i] = w[i-4] ^ temp1:
        roundKeys[(i*4)] = roundKeys[((i-4)*4)] ^ temp1[0];
        roundKeys[(i*4)+1] = roundKeys[((i-4)*4)+1] ^ temp1[1];
        roundKeys[(i*4)+2] = roundKeys[((i-4)*4)+2] ^ temp1[2];
        roundKeys[(i*4)+3] = roundKeys[((i-4)*4)+3] ^ temp1[3];
    }
}
```

AES의 C언어 구현 – State 변환

```c
void ConvInputToState(UINT8 * state, const UINT8 * input){
    // ## Input to State 변환 함수
    state[0] = input[0];
    state[1] = input[4];
    state[2] = input[8];
    state[3] = input[12];

    state[4] = input[1];
    state[5] = input[5];
    state[6] = input[9];
    state[7] = input[13];
    state[8] = input[2];
    state[9] = input[6];
    state[10] = input[10];
    state[11] = input[14];

    state[12] = input[3];
    state[13] = input[7];
    state[14] = input[11];
    state[15] = input[15];
```

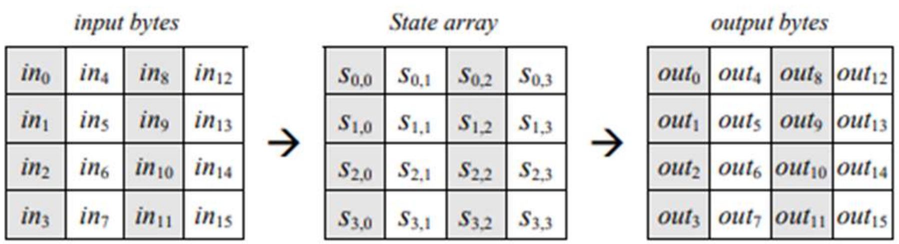


Figure 3. State array input and output.

- Byte 순서를 바꾸는 연산.

AES의 C언어 구현 – AddRoundKey()

```c
void AddRoundKey(UINT8 * state, const UINT8 * roundKey){
    // ## AddRoundKey 함수
    // # 0. 변수 초기화
    UINT8 temp[16]; // 임시 변수

    // # 1. 연산:
    CopyBytes(temp, state, 16);
    for(UINT8 i=0; i<4; i++){
        state[i] = temp[i] ^ roundKey[(i*4)];
        state[i+4] = temp[i+4] ^ roundKey[(i*4)+1];
        state[i+8] = temp[i+8] ^ roundKey[(i*4)+2];
        state[i+12] = temp[i+12] ^ roundKey[(i*4)+3];
    }
}
```

- KeyExpansion()을 통해 생성된 라운드키와 XOR 하는 연산

AES의 C언어 구현 – SubBytes()

```c
void SubBytes(UINT8 * state){
    // ## SubBytes 함수
    // # 0. 변수 초기화
    UINT8 temp[16]; // 임시 변수
    const UINT8 Sbox[16][16] = {
        {0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
        {0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
        {0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
        {0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
        {0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
        {0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
        {0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
        {0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
        {0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
        {0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
        {0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
        {0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
        {0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
        {0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
        {0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
        {0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
    };
    // # 1.연산:
    CopyBytes(temp, state, 16);
    for(UINT8 i=0; i<16; i++){
        state[i] = Sbox[(temp[i] >> 4)][(temp[i] & 0x0f)]; // Sbox 계산
    }
}
```

※ InvSubBytes() 도 동일한 방법으로 구현!

Figure 6 illustrates the effect of the **SubBytes()** transformation on the State.

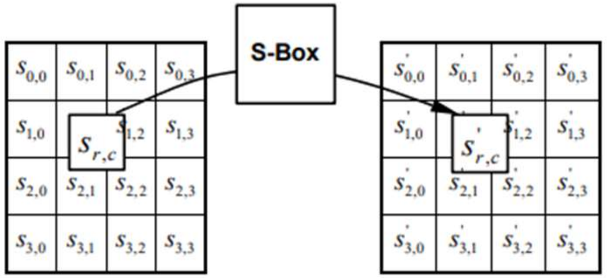


Figure 6. SubBytes() applies the S-box to each byte of the State.

The S-box used in the **SubBytes()** transformation is presented in hexadecimal form in Fig. 7. For example, if $s_{1,1} = \{53\}$, then the substitution value would be determined by the intersection of the row with index '5' and the column with index '3' in Fig. 7. This would result in $s'_{1,1}$ having a value of {ed}.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 7. S-box: substitution values for the byte xy (in hexadecimal format).

AES의 C언어 구현 – ShiftRows()

```c
void ShiftRows(UINT8 * state){
    // ## ShiftRows 함수
    // # 0. 변수 초기화
    UINT8 temp[16]; // 임시 변수
    // # 1.연산:
    CopyBytes(temp, state, 16);

    state[0] = temp[0];
    state[1] = temp[1];
    state[2] = temp[2];
    state[3] = temp[3];
```

```c
    state[4] = temp[5];
    state[5] = temp[6];
    state[6] = temp[7];
    state[7] = temp[4];

    state[8] = temp[10];
    state[9] = temp[11];
    state[10] = temp[8];
    state[11] = temp[9];

    state[12] = temp[15];
    state[13] = temp[12];
    state[14] = temp[13];
    state[15] = temp[14];
```

Figure 8 illustrates the **ShiftRows()** transformation.

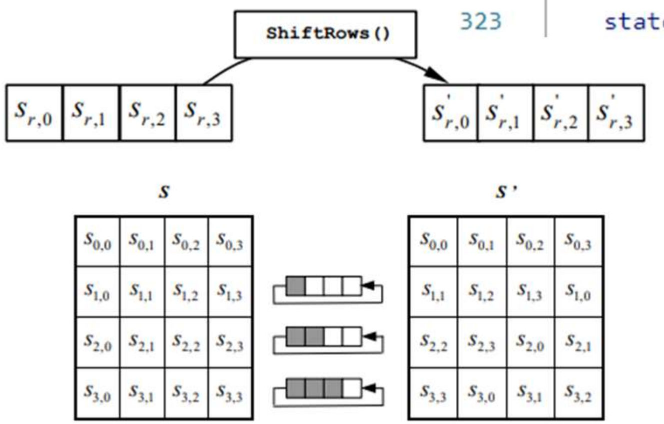


Figure 8. ShiftRows() cyclically shifts the last three rows in the State.

- 간단히 각 state 단위로 섞는 연산.
- InvShiftRows()도 동일한 방법으로 구현.

AES의 C언어 구현 – MixColumns()

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}. \tag{5.6}$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

As described in Sec. 4.3, this can be written as a matrix multiplication. Let $s'(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}. \tag{5.10}$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

AES의 C언어 구현 – MixColumns()

```c
void MixColumns(UINT8 * state){
    // ## MixColumns 함수
    UINT8 temp[16];
    CopyBytes(temp, state, 16);

    state[0] = (MulGF256(0x2, temp[0])) ^ (MulGF256(0x3, temp[4])) ^ (MulGF256(0x1, temp[8])) ^ (MulGF256(0x1, temp[12]));
    state[4] = (MulGF256(0x1, temp[0])) ^ (MulGF256(0x2, temp[4])) ^ (MulGF256(0x3, temp[8])) ^ (MulGF256(0x1, temp[12]));
    state[8] = (MulGF256(0x1, temp[0])) ^ (MulGF256(0x1, temp[4])) ^ (MulGF256(0x2, temp[8])) ^ (MulGF256(0x3, temp[12]));
    state[12] = (MulGF256(0x3, temp[0])) ^ (MulGF256(0x1, temp[4])) ^ (MulGF256(0x1, temp[8])) ^ (MulGF256(0x2, temp[12]));

    state[1] = (MulGF256(0x2, temp[1])) ^ (MulGF256(0x3, temp[5])) ^ (MulGF256(0x1, temp[9])) ^ (MulGF256(0x1, temp[13]));
    state[5] = (MulGF256(0x1, temp[1])) ^ (MulGF256(0x2, temp[5])) ^ (MulGF256(0x3, temp[9])) ^ (MulGF256(0x1, temp[13]));
    state[9] = (MulGF256(0x1, temp[1])) ^ (MulGF256(0x1, temp[5])) ^ (MulGF256(0x2, temp[9])) ^ (MulGF256(0x3, temp[13]));
    state[13] = (MulGF256(0x3, temp[1])) ^ (MulGF256(0x1, temp[5])) ^ (MulGF256(0x1, temp[9])) ^ (MulGF256(0x2, temp[13]));

    state[2] = (MulGF256(0x2, temp[2])) ^ (MulGF256(0x3, temp[6])) ^ (MulGF256(0x1, temp[10])) ^ (MulGF256(0x1, temp[14]));
    state[6] = (MulGF256(0x1, temp[2])) ^ (MulGF256(0x2, temp[6])) ^ (MulGF256(0x3, temp[10])) ^ (MulGF256(0x1, temp[14]));
    state[10] = (MulGF256(0x1, temp[2])) ^ (MulGF256(0x1, temp[6])) ^ (MulGF256(0x2, temp[10])) ^ (MulGF256(0x3, temp[14]));
    state[14] = (MulGF256(0x3, temp[2])) ^ (MulGF256(0x1, temp[6])) ^ (MulGF256(0x1, temp[10])) ^ (MulGF256(0x2, temp[14]));

    state[3] = (MulGF256(0x2, temp[3])) ^ (MulGF256(0x3, temp[7])) ^ (MulGF256(0x1, temp[11])) ^ (MulGF256(0x1, temp[15]));
    state[7] = (MulGF256(0x1, temp[3])) ^ (MulGF256(0x2, temp[7])) ^ (MulGF256(0x3, temp[11])) ^ (MulGF256(0x1, temp[15]));
    state[11] = (MulGF256(0x1, temp[3])) ^ (MulGF256(0x1, temp[7])) ^ (MulGF256(0x2, temp[11])) ^ (MulGF256(0x3, temp[15]));
    state[15] = (MulGF256(0x3, temp[3])) ^ (MulGF256(0x1, temp[7])) ^ (MulGF256(0x1, temp[11])) ^ (MulGF256(0x2, temp[15]));
}
```

AES의 C언어 구현 – MixColumns()

```c
void InvMixColumns(UINT8 * state){
    // ## InvMixColumns 함수
    // # Inverse function of MixColumns().
    UINT8 temp[16];          // 전체 복사한 값
    CopyBytes(temp, state, 16); // 복사

    state[0] = (MulGF256(0xe, temp[0])) ^ (MulGF256(0xb, temp[4])) ^ (MulGF256(0xd, temp[8])) ^ (MulGF256(0x9, temp[12]));
    state[4] = (MulGF256(0x9, temp[0])) ^ (MulGF256(0xe, temp[4])) ^ (MulGF256(0xb, temp[8])) ^ (MulGF256(0xd, temp[12]));
    state[8] = (MulGF256(0xd, temp[0])) ^ (MulGF256(0x9, temp[4])) ^ (MulGF256(0xe, temp[8])) ^ (MulGF256(0xb, temp[12]));
    state[12] = (MulGF256(0xb, temp[0])) ^ (MulGF256(0xd, temp[4])) ^ (MulGF256(0x9, temp[8])) ^ (MulGF256(0xe, temp[12]));

    state[1] = (MulGF256(0xe, temp[1])) ^ (MulGF256(0xb, temp[5])) ^ (MulGF256(0xd, temp[9])) ^ (MulGF256(0x9, temp[13]));
    state[5] = (MulGF256(0x9, temp[1])) ^ (MulGF256(0xe, temp[5])) ^ (MulGF256(0xb, temp[9])) ^ (MulGF256(0xd, temp[13]));
    state[9] = (MulGF256(0xd, temp[1])) ^ (MulGF256(0x9, temp[5])) ^ (MulGF256(0xe, temp[9])) ^ (MulGF256(0xb, temp[13]));
    state[13] = (MulGF256(0xb, temp[1])) ^ (MulGF256(0xd, temp[5])) ^ (MulGF256(0x9, temp[9])) ^ (MulGF256(0xe, temp[13]));

    state[2] = (MulGF256(0xe, temp[2])) ^ (MulGF256(0xb, temp[6])) ^ (MulGF256(0xd, temp[10])) ^ (MulGF256(0x9, temp[14]));
    state[6] = (MulGF256(0x9, temp[2])) ^ (MulGF256(0xe, temp[6])) ^ (MulGF256(0xb, temp[10])) ^ (MulGF256(0xd, temp[14]));
    state[10] = (MulGF256(0xd, temp[2])) ^ (MulGF256(0x9, temp[6])) ^ (MulGF256(0xe, temp[10])) ^ (MulGF256(0xb, temp[14]));
    state[14] = (MulGF256(0xb, temp[2])) ^ (MulGF256(0xd, temp[6])) ^ (MulGF256(0x9, temp[10])) ^ (MulGF256(0xe, temp[14]));

    state[3] = (MulGF256(0xe, temp[3])) ^ (MulGF256(0xb, temp[7])) ^ (MulGF256(0xd, temp[11])) ^ (MulGF256(0x9, temp[15]));
    state[7] = (MulGF256(0x9, temp[3])) ^ (MulGF256(0xe, temp[7])) ^ (MulGF256(0xb, temp[11])) ^ (MulGF256(0xd, temp[15]));
    state[11] = (MulGF256(0xd, temp[3])) ^ (MulGF256(0x9, temp[7])) ^ (MulGF256(0xe, temp[11])) ^ (MulGF256(0xb, temp[15]));
    state[15] = (MulGF256(0xb, temp[3])) ^ (MulGF256(0xd, temp[7])) ^ (MulGF256(0x9, temp[11])) ^ (MulGF256(0xe, temp[15]));
}
```

AES의 C언어 구현 – GF(2^8) 연산 구현

```
UINT8 MulGF256(UINT8 op1, UINT8 op2){
    // ## Multiplacation in GF(2^8)
    UINT8 n = GetBitLength(op1);
    UINT8 result = 0;
    UINT8 a = op2;

    for(UINT8 i=0; i<n; i++){
        result = result ^ (((op1 >> i) & 0b1) * a);
        a = Mul2(a);
    }
    return result;
}
```

- ※ GetBitAmount(a) = $[\log_2 a] + 1$

AES의 C언어 구현 – GF(2^8) 연산 구현

```c
// ### AES 관련 함수:
UINT8 Mul2(UINT8 a)
{
    // ## Multiplacation by 2 in GF(2^8).
    UINT8 b = (a >> 7) & 0b1;
    UINT8 c = a << 1; // c(x) = x * b(x)
    if (b)
    {
        c = c ^ 0x1b; // c(x) = c(x) mod p(x)
    }
    return c;
}
```

4.2 Multiplication

In the polynomial representation, multiplication in GF(2^8) (denoted by •) corresponds with the multiplication of polynomials modulo an **irreducible polynomial** of degree 8. A polynomial is irreducible if its only divisors are one and itself. **For the AES algorithm, this irreducible polynomial is**

$$m(x) = x^8 + x^4 + x^3 + x + 1, \qquad (4.1)$$

- 0x1b = 0001 1011
- $p(x) = 0001\ 0001\ 1011$

subtracting (i.e., XORing) the polynomial $m(x)$. It follows that multiplication by x (i.e., {00000010} or {02}) can be implemented at the byte level as a left shift and a subsequent conditional bitwise XOR with {1b}. This operation on bytes is denoted by `xtime()`.

```c
// #### typedef
typedef unsigned char UINT8; // 부호없는 8비트 정수
typedef unsigned short UINT16; // 부호없는 16비트 정수
```

```c
UINT8 Mul2(UINT8 op1)
{
    // ## Multiplacation by 2 in GF(2^8).
    // b(x) = (x * a(x)) mod p(x)
    return (((!((op1 >> 7) & 0b1)) * (op1 << 1)) + (((op1 >> 7) & 0b1) * ((op1 << 1) ^ 0x1b)));
}
```

AES의 C언어 구현 – 어려웠던 부분

- MulGF2()
- MulGF256()
- ConvInputToState()
- ConvStateToOutput()

AES의 C언어 구현 - 참고문헌

- NIST FIPS 197-upd1

Q & A