암 호 학
과 제 #2

과 목 명 (subject)	암호학
담당교수 (professor)	오희국 교수님
대 학	소프트웨어융합대학
학 과	소프트웨어학부
이름	2017012188 엄태호

#1 구현 함수

1) gf8_mul

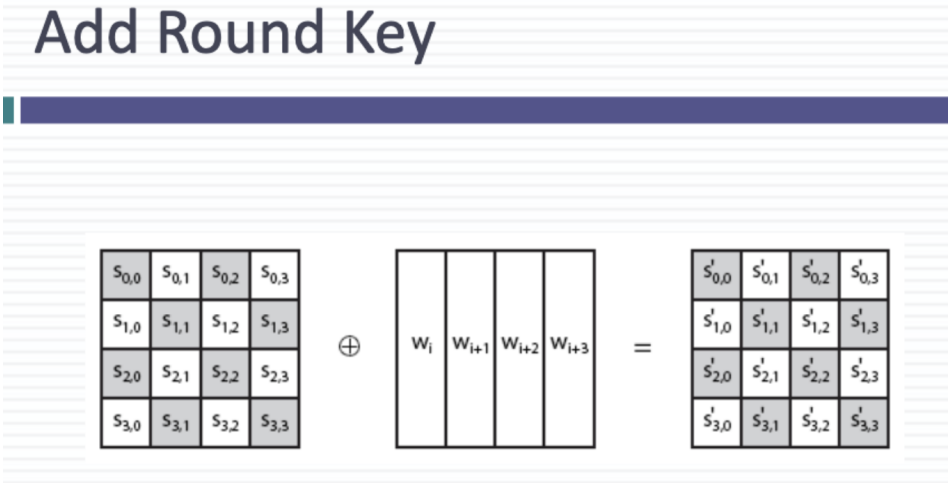
```
uint8_t gf8_mul(uint8_t a, uint8_t b)
{
    uint8_t r = 0;
    while(b > 0)
    {
      if(b & 1) r = r ^ a;
      b = b >> 1;
      a = XTIME(a);
    }
    return r;
}
```

MixColumns 함수에서 사용될 gf8_mul 함수입니다.
저번 프로그래밍 과제 #1에서 구현한 코드와 동일하며, “aes.h”파일에 정의되어 있는 XTIME을 추가로 사용하였습니다.

2) AddRoundKey

```
void AddRoundKey(uint8_t *state, const uint32_t *roundKey)
{
    uint32_t *tmp = (uint32_t *) state;
    for(int i = 0 ; i < Nb ; i++)
    {
        tmp[i] ^= roundKey[i];
    }
}
```

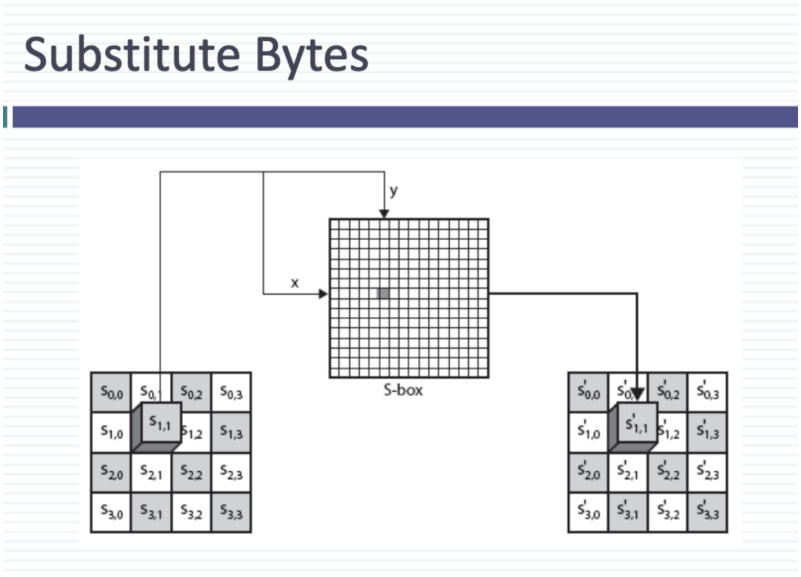
해당 라운드에 알맞은 RoundKey를 state와 XOR연산을 진행합니다.
state가 uint8_t 자료형을 가지고 있어, uint32_t형으로 형변환을 하였습니다.



3) SubBytes

```
void SubBytes(uint8_t *state, int mode)
{
    for(int i = 0 ; i < BLOCKLEN ; i++)
    {
        state[i] = mode ? sbox[state[i]] : isbox[state[i]];
    }
}
```

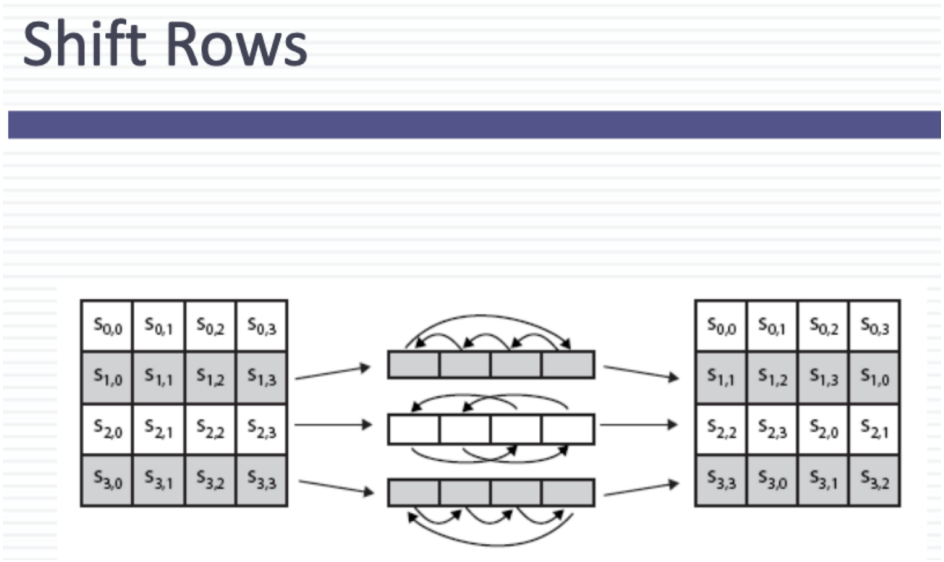
Substituion 연산을 진행하는 함수입니다.
모드가 ENCRYPT일 경우, Sbox를 통해, DECRYPT일 경우 ISBox를 통해 변환합니다.



4) ShiftRows

```
void ShiftRows(uint8_t *state, int mode)
{
    uint8_t tmp[16] = {};
    if(mode)
    {
        for(int i = 0 ; i < 16 ; i++)
        {
            if(i%4 == 1)
            {
                if(i - 4 > 0) tmp[i - 4] = state[i];
                else tmp[i - 4 + 16] = state[i];
            }
            else if(i%4 == 2)
            {
                if(i - 8 > 0) tmp[i - 8] = state[i];
                else tmp[i - 8 + 16] = state[i];
            }
            else if(i%4 == 3)
            {
                if(i - 12 > 0) tmp[i - 12] = state[i];
                else tmp[i - 12 + 16] = state[i];
            }
            else
            {
                tmp[i] = state[i];
            }
        }
    }

    else
    {
        for(int i = 0 ; i < 16 ; i++)
        {
            if(i%4 == 1)
            {
                if(i + 4 <= 15) tmp[i + 4] = state[i];
                else tmp[i + 4 - 16] = state[i];
            }
            else if(i%4 == 2)
            {
                if(i + 8 <= 15) tmp[i + 8] = state[i];
                else tmp[i + 8 - 16] = state[i];
            }
            else if(i%4 == 3)
            {
                if(i + 12 <= 15) tmp[i + 12] = state[i];
                else tmp[i + 12 - 16] = state[i];
            }
            else
            {
                tmp[i] = state[i];
            }
        }
    }
    for(int i = 0; i < 16 ; i++) state[i] = tmp[i];
}
```



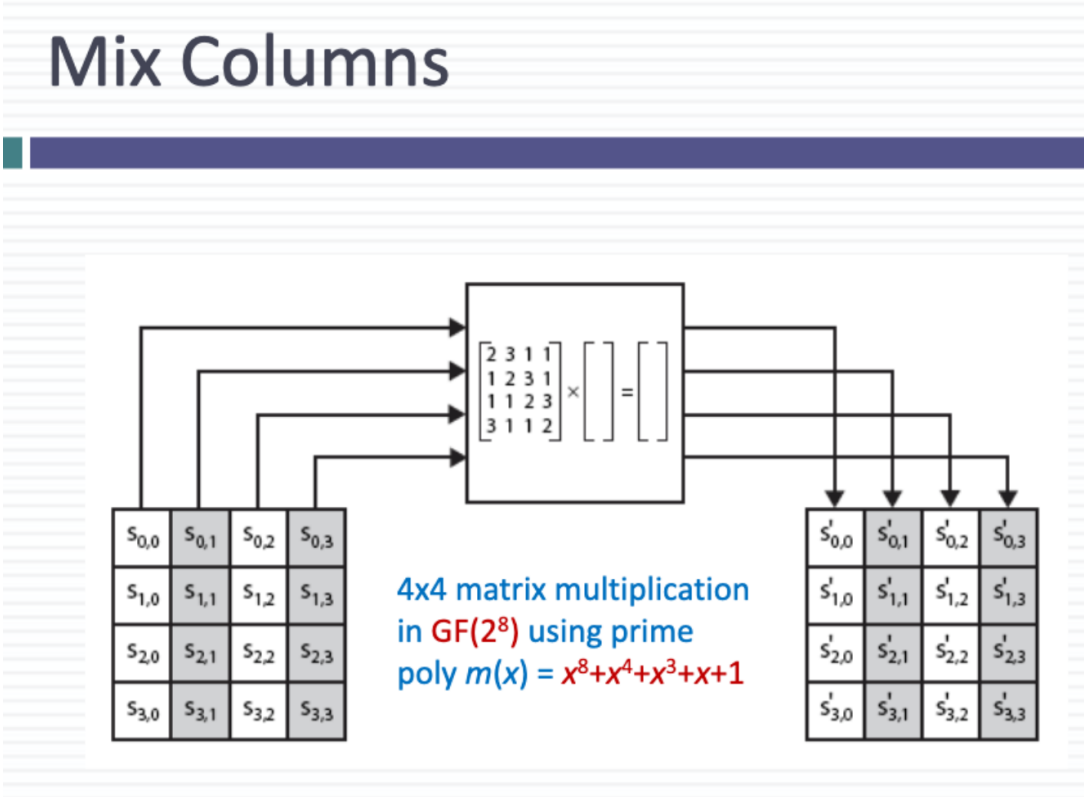
mode가 ENCRYPT라면 1열을 제외하고 2, 3, 4열을 왼쪽으로 1, 2, 3번 각각 쉬프트.
mode가 DEDCRYPT라면 반대 방향인 오른쪽으로 1, 2, 3번 각각 쉬프트합니다.

AES 128을 구현하기 때문에, 4로 나눈 나머지가 동일한 칸들의 경우 이동하는 개수가 똑같다는 것을 발견하여 구현하였습니다.

5) MixColumns

```
void MixColumns(uint8_t *state, int mode)
{
    uint8_t tmp[Nb*Nb] = {0,};

    for(int i = 0 ; i < Nb ; i++)
    {
        for(int j = 0 ; j < Nb ; j++)
        {
            for(int k = 0 ; k < Nb ; k++)
            {
                if(mode) tmp[j * Nb + i] ^= gf8_mul(M[i * Nb + k], state[j * Nb + k]);
                else tmp[j * Nb + i] ^= gf8_mul(IM[i * Nb + k], state[j * Nb + k]);
            }
        }
    }
    for(int i = 0 ; i < Nb * Nb ; i++) state[i] = tmp[i];
}
```



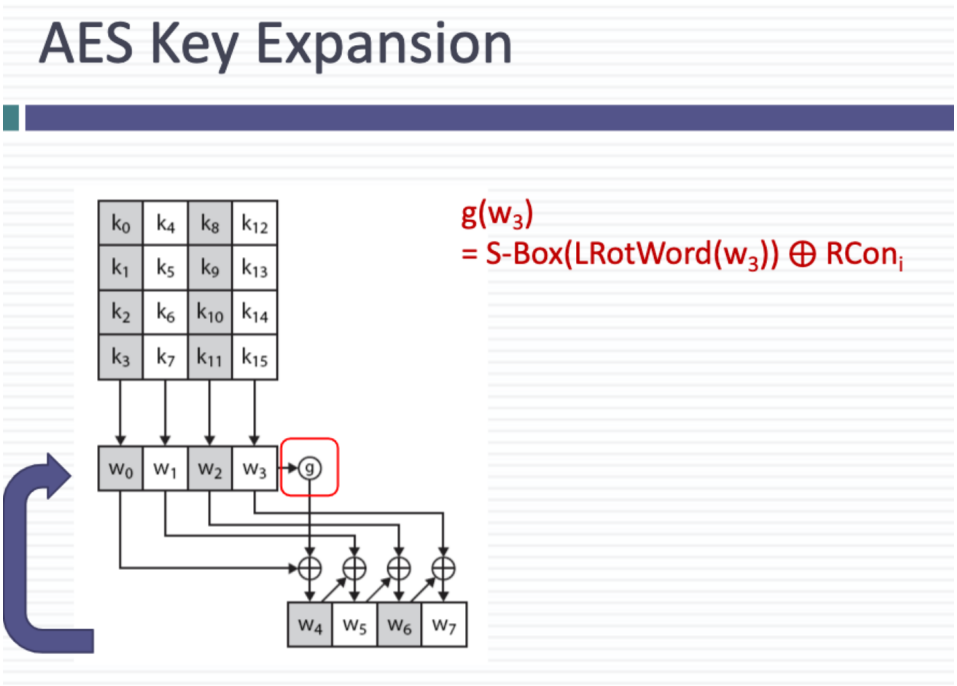
state의 각 열과 ENCRYPT의 경우 M 행렬과, DECRYPT IM 행렬과 곱하는 연산을 진행합니다. tmp 배열을 하나 생성하여 연산을 진행한 값을 저장한 뒤,
마지막으로 state 배열로 옮기는 방식으로 구현하였습니다.

6) KeyExpansion

```
void KeyExpansion(const uint8_t *key, uint32_t *roundKey)
{
    uint8_t p[Nb* RNDKEYSIZE];
    for(int i = 0 ; i < KEYLEN ; i++)
    {
        p[i] = key[i];
    }

    for(int i = 1 ; i <= Nr ; i++)
    {
        uint8_t w3[KEYLEN/4] = {p[i * KEYLEN - 4], p[i * KEYLEN - 3], p[i * KEYLEN - 2], p[i * KEYLEN - 1]};
        LRotWord(w3, uint8_t);
        for(int j = 0 ; j < KEYLEN/4 ; j++)
        {
            w3[j] = sbox[w3[j]];
            if(!j) w3[j] ^= Rcon[i]; // g(w3) 생성
        }

        for(int j = KEYLEN * i ; j < KEYLEN * (i + 1) ; j++) // 현재 라운드부터 다음 라운드 전 까지
        {
            if(j - KEYLEN * i < KEYLEN / 4) p[j] = p[j - KEYLEN] ^ w3[j - KEYLEN * i]; // (w4의 경우 g(w3)과 XOR연산)
            else p[j] = p[j - KEYLEN] ^ p[j - 4]; // (w5, w6, w7의 경우 바로 이전 것과 이전 라운드와 XOR
        }
    }
    memcpy((uint8_t *)roundKey, p, sizeof(uint8_t) * Nb * RNDKEYSIZE);
}
```



각 라운드마다 사용할 라운드 키를 확장해나가는 함수입니다.
공식문서의 수도 코드의 경우 이해가 완벽하게 되지 않아 pdf에 나와있는 연산식을 참고하여 구현하였습니다.
라운드 키의 자료형이 달라 3칸씩 출력의 차이가 있어 같은 자료형을 갖는 배열을 생성하여 맞춰주었으며,
LRotWord의 경우 매크로 함수를 정의하여 사용하였습니다.
마지막으로 memcpy를 이용해 생성했던 배열의 메모리를 roundKey로 복사합니다.

7) Cipher

```
void Cipher(uint8_t *state, const uint32_t *roundKey, int mode)
{
    if(mode)
    {
        AddRoundKey(state, roundKey);
        roundKey += Nk;
        for(int i = 1; i <= Nr - 1 ; i++){
            SubBytes(state, mode);
            ShiftRows(state, mode);
            MixColumns(state, mode);
            AddRoundKey(state, roundKey);
            roundKey += Nk;

        }
        SubBytes(state, mode);
        ShiftRows(state, mode);
        AddRoundKey(state, roundKey);
    }
    else
    {
        roundKey += Nk * Nr;
        AddRoundKey(state, roundKey);
        roundKey -= Nk;
        for(int i = Nr - 1 ; i >= 1 ; i--)
        {
            ShiftRows(state, mode);
            SubBytes(state, mode);
            AddRoundKey(state, roundKey);
            MixColumns(state, mode);
            roundKey -= Nk;
        }
        ShiftRows(state, mode);
        SubBytes(state, mode);
        AddRoundKey(state, roundKey);
    }
}
```

```
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])                  // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                              // See Sec. 5.1.1
      ShiftRows(state)                             // See Sec. 5.1.2
      MixColumns(state)                            // See Sec. 5.1.3
      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
```

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

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                        // See Sec. 5.3.1
      InvSubBytes(state)                         // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                       // See Sec. 5.3.3
   end for

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

   out = state
end
```

참고용 자료로 주신 수도코드를 참고하여 코드를 작성하였습니다.
라운드 키의 경우 ENCRYPT의 경우, 0에서 시작하여 키의 크기만큼 주소값을 이동하며 진행하였고, DECRYPT의 경우 증가된 상태에서 감소하는 방식으로 구현하였습니다.

#2 실행 화면

```
Random testing.................No error found
(base) umtaeho@eomtaehoui-MacBookPro project#2 % make
gcc -Wall -c aes.c
gcc -Wall -o test test.o aes.o
(base) umtaeho@eomtaehoui-MacBookPro project#2 % ./test
<키 >
0f 15 71 c9 47 d9 e8 59 0c b7 ad d6 af 7f 67 98
<라 운 드  키 >
0f 15 71 c9 47 d9 e8 59 0c b7 ad d6 af 7f 67 98
dc 90 37 b0 9b 49 df e9 97 fe 72 3f 38 81 15 a7
d2 c9 6b b7 49 80 b4 5e de 7e c6 61 e6 ff d3 c6
c0 af df 39 89 2f 6b 67 57 51 ad 06 b1 ae 7e c0
2c 5c 65 f1 a5 73 0e 96 f2 22 a3 90 43 8c dd 50
58 9d 36 eb fd ee 38 7d 0f cc 9b ed 4c 40 46 bd
71 c7 4c c2 8c 29 74 bf 83 e5 ef 52 cf a5 a9 ef
37 14 93 48 bb 3d e7 f7 38 d8 08 a5 f7 7d a1 4a
48 26 45 20 f3 1b a2 d7 cb c3 aa 72 3c be 0b 38
fd 0d 42 cb 0e 16 e0 1c c5 d5 4a 6e f9 6b 41 56
b4 8e f3 52 ba 98 13 4e 7f 4d 59 20 86 26 18 76
---
<평 문 >
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<암 호 문 >
ff 0b 84 4a 08 53 bf 7c 69 34 ab 43 64 14 8f b9
<복 호 문 >
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<역 암 호 문 >
1f e0 22 1f 19 67 12 c4 be cd 5c 1c 60 71 ba a6
<복 호 문 >
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Random testing.................No error found
```

#3 소감

짜는 과정에서 2차원 배열이였으면 조금 더 편했을 것 같은 과정들을 일차원 배열로 해결하려니 조금 헷갈리는 부분이 있었던 것 같습니다.

저번 과제 피드백에서 Swap처럼 단순한 함수를 따로 구현하는 것은 효율성 측면에서 좋지 않으니, 가독성을 위해서라면 매크로 함수를 정의해서 사용하라는 피드백을 받아 KeyExpansion 함수를 구현할 때, LRotWord를 매크로 함수를 정의하여 사용하였습니다.

```
#define LRotWord(word, type) type tmp = word[0]; word[0] = word[1]; word[1] = word[2];  word[2] = word[3]; word[3] = tmp;
```

추가로 속도 개선을 위해, ShiftRow 함수를 구현할 때, 처음에는 3중 반복문을 이용하여 하드하게 각 배열에서 위치를 바꾸는 방법으로 구현하였다가, 바뀌는 위치의 규칙성을 찾아 하나의 반복문으로 연산이 진행되도록 변경하였습니다.

수업을 들을 때 AES가 진행되는 과정을 볼 때에는 그냥 그렇구나라는 생각이 들었었는데, 막상 구현을 해보니 생각보다 쉽지 않다는 생각이 들었습니다. 그래도 직접 구현을해보니 AES를 이해하는 데에 큰 도움이 된 구현 과제였다고 생각합니다.