Project#2

2018044893 이수정

<함수설명>

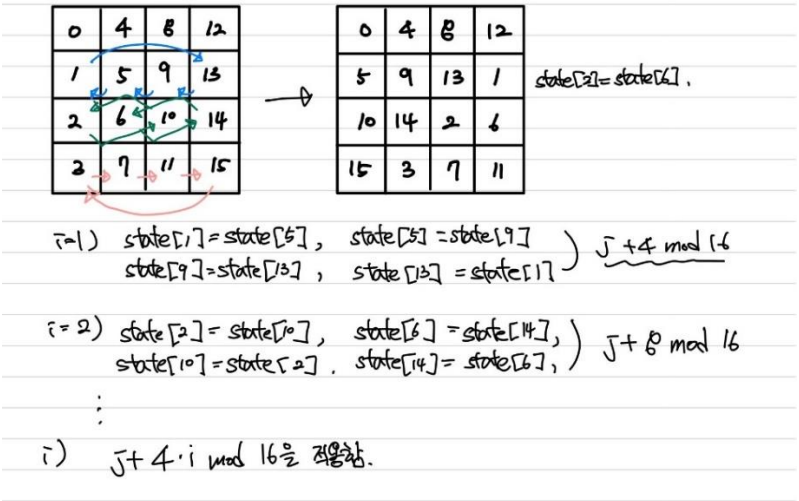
- static void SubBytes(uint8_t *state, int mode)

```
56 static void SubBytes(uint8_t *state, int mode)
57 {
58         if(mode == 1){//sbox를 이용해 값들을 치환해준다
59                 for(int i = 0; i < Nb * Nk; i++)        state[i] = sbox[state[i]];}
60         else{//isbox를 이용해 값들은 sbox의 역으로 치환해준다.
61                 for(int i = 0; i < Nb * Nk; i++)        state[i] = isbox[state[i]];}
62 }
63
```

표를 이용하여 치환하는 과정으로 state[i]에 해당하는 값을 s-box 와 inverse s-box array를 이용하여 해당하는 값을 구한다.

- static void ShiftRows(uint8_t *state, int mode)

```
64 static void ShiftRows(uint8_t *state, int mode)
65 {
66         uint8_t tmp_k[Nk*Nb];//가져온 state값을 보존하기 위해 tmp_k에 임시로 저장
67         for(int k = 0;k < Nk*Nb; k++)
68         {
69                 tmp_k[k] = state[k];
70         }
71         if(mode == 1){//encrypt
72                 for(int i = 1; i < Nb; i++)//0번째 row는 이동하지 않으므로 1번째 부터
73                 {
74                         for(int j = 0;j < Nk; j++)      state[Nb*j+i] = tmp_k[((Nb*j+i)+i*Nk)%16];//i씩 왼쪽
   으로 이동하고, 값이 0~15사이를 벗어나는 경우 고려
75                 }
76         }
77         else{//decrypt
78                 for(int i=1;i<Nb;i++)
79                 {
80                         for(int j=0;j<Nk;j++)   state[Nb*j+i] = tmp_k[((Nb*j+i)-i*Nk+16)%16];//i씩 오른으로
   이동하고, 값이 0~15사이를 벗어나는 경우 고려
81                 }
82         }
83 }
```

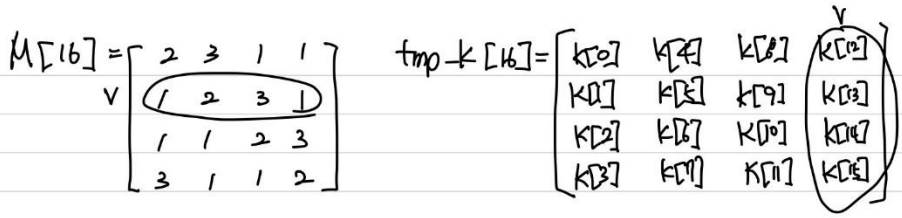


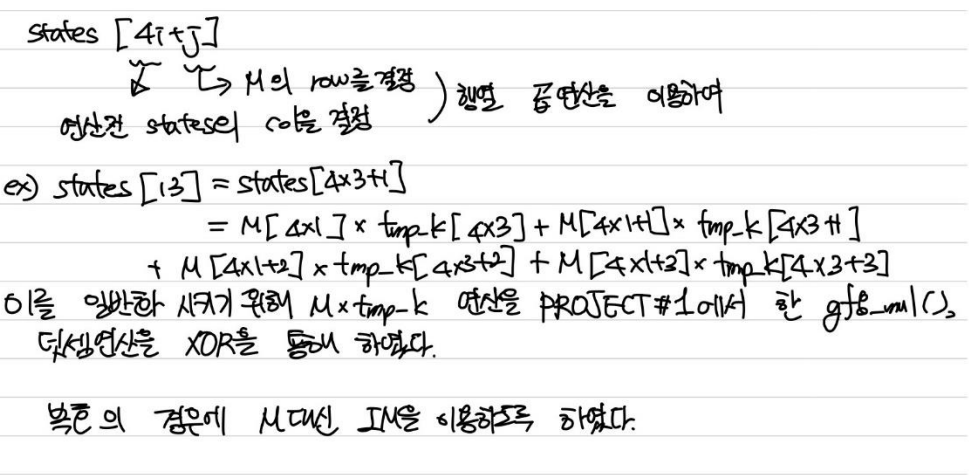
int i를 1부터 시작한 이유는 0첫번째 row에서는 shift가 발생하지 않기 때문에 불필요한 연산을 없앴다.

- **static void MixColumns(uint8_t *state, int mode)**

  uint8_t gf8_mul(uint8_t a, uint8_t b)

```
85 uint8_t gf8_mul(uint8_t a, uint8_t b)//project 1 에서 이용한 galios filed연산을 가져옴
86 {
87         uint8_t res = 0;
88         while(b>0)
89         {
90                 if(b&1) res = res^a;
91                 b = b>>1;
92                 a = ((a<<1)^((a>>7)&1 ?0X1B:0));
93         }
94         return res;
95 }
96
97 static void MixColumns(uint8_t *state, int mode)//M[16], IM[16]을 이용
98 {
99         uint8_t tmp_k[Nk*Nb];//가져온 state값을 보존하기 위해 tmp_k에 임시로 저장
100        for(int k = 0;k < Nk*Nb; k++)
101        {
102                tmp_k[k] = state[k];
103        }
104        for(int i=0;i<4;i++)
105        {
106                for(int j=0;j<4;j++)
107                {
108                        if(mode == 1){//그림설명
109                                state[4*i+j] = gf8_mul(M[4*j],tmp_k[4*i]) ^
110                                        gf8_mul(M[4*j+1],tmp_k[4*i+1]) ^
111                                        gf8_mul(M[4*j+2],tmp_k[4*i+2]) ^
112                                        gf8_mul(M[4*j+3],tmp_k[4*i+3]);
113                        }
114                        else{
115                                state[4*i+j] = gf8_mul(IM[4*j],tmp_k[4*i]) ^
116                                        gf8_mul(IM[4*j+1],tmp_k[4*i+1]) ^
117                                        gf8_mul(IM[4*j+2],tmp_k[4*i+2]) ^
118                                        gf8_mul(IM[4*j+3],tmp_k[4*i+3]);
119
120                        }
121                }
122        }
123
124 }
```



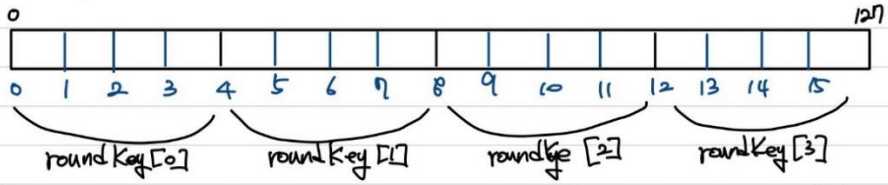


tmp_k와 M array의 진행 순서를 유의해야한다.

- void KeyExpansion(const uint8_t *key, uint32_t *roundKey)

```
127 void KeyExpansion(const uint8_t *key, uint32_t *roundKey)
128 {
129         int i=0;
130         uint32_t tmp_k,Rot,Sub;
131
132         while(i<Nk)//input된 key값을 (key[0~3],key[4~7],...)로 쪼개 subkey(roundKey0~3) 생성
133         {
134                 roundKey[i] = ((uint32_t)key[4*i+3]<<24^(uint32_t)key[4*i+2]<<16^
135                                 (uint32_t)key[4*i+1]<<8^(uint32_t)key[4*i]);
136                 i++;
137         }
138         while(i<(Nb*(Nr+1)))
139         {
140                 tmp_k = roundKey[i-1];
141
142                 if(i%Nk == 0)//128비트마다 강의 자료에 나온g값을 포함하여 계산
143                 {
144                         //rotword
145                         Rot = (tmp_k<<24 ^ tmp_k>>8);
146                         //subword
147                         Sub = 0;//초기화
148                         for(int j=0;j<32;j+=8){
149                                 Sub = Sub ^ (uint32_t)(sbox[(uint8_t)(Rot>>j)])<<j;
150                         }
151                         //
152                         tmp_k = Sub ^ Rcon[i/Nk];
153                 }
154                 roundKey[i] = roundKey[i-Nk] ^ tmp_k;
155                 i++;
156         }
157
158 }
159
```

① *key ⇒ 8bit x16 즉 128bit

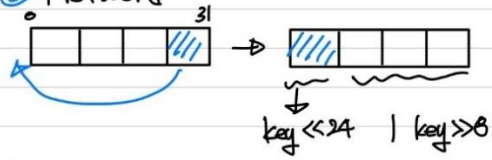


key[4·i], key[4i+1], key[4i+2], key[4i+3]을 비트단위 이동시키고 합치는 연산을 통해 roundKey 0~3을 생성함.

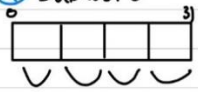
② W0 W1 W2 W3 → g → W4 W5 W6 W7

n이 Nk의 배수일 때
$W_n = W_{n-1} \oplus g \oplus W_{n-Nk}$를 통해 roundKey를 확장하고
n이 Nk의 배수가 아닐 때
$W_n = W_{n-1} \oplus W_{n-Nk}$
즉 $W_n = W_{n-1} \oplus \alpha$ (n % Nk == 0 → $g \oplus W_{n-Nk}$ ③+④ / n % Nk != 0 → W_{n-Nk})

③ RotWord



④ SubWord



uint8_t 단위로 쪼개 sbox array를 이용하여 치환해주고
RotWord, SubWord를 완료한 key를 Rcon array와 XOR연산
해준값이 ⑨가된다.

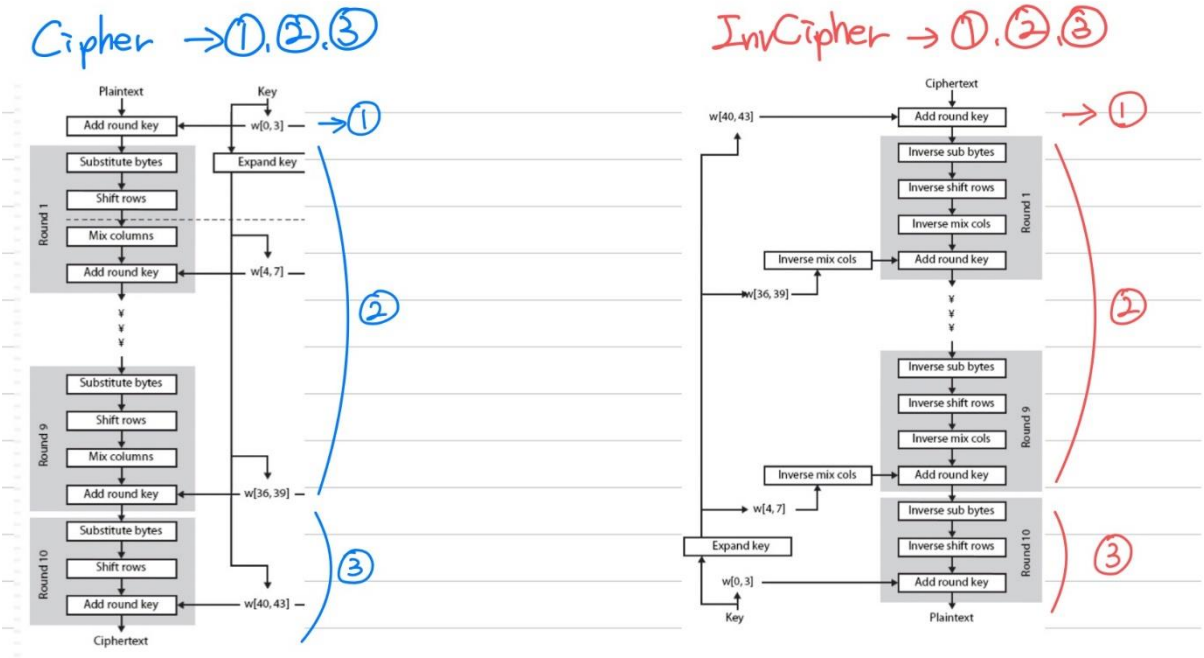
- static void AddRoundKey(uint8_t *state, const uint32_t *roundKey)

```
167 static void AddRoundKey(uint8_t *state, const uint32_t *roundKey)//roundKey와 Plaintext를 연산하는 과정
168 {
169         uint32_t *tmp_k = (uint32_t*)state;//roundkey와 XOR연산을 위해 자료형을 맞춤
170         for(int i=0;i<Nb;i++)   tmp_k[i] = tmp_k[i] ^ roundKey[i];
171 }
```

State를 roundeKey와 XOR연산을 하기 위해 uint32_t타입으로 치환한 후 XOR 연산을 해준다.

- void Cipher(uint8_t *state, const uint32_t *roundKey, int mode)

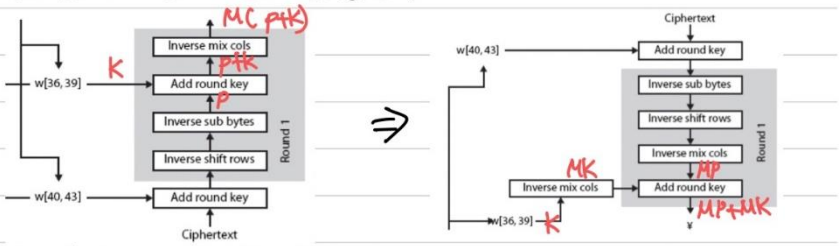
```
177 void Cipher(uint8_t *state, const uint32_t *roundKey, int mode)
178 {
179         uint32_t round_k[4];//임시로 roundKey를 가져오기 위함
180
181         if(mode == 1)   AddRoundKey(state,roundKey); //첫번째 roundKey
182         else{
183                 for(int k=0;k<Nk;k++)   round_k[k] = roundKey[4*Nr+k];//마지막 roundkey이용(역순으로 사용)
184                 AddRoundKey(state,round_k);
185         }
186
187         for(int i=1;i<Nr;i++){
188                 SubBytes(state, mode);
189                 ShiftRows(state, mode);
190                 MixColumns(state, mode);
191                 if(mode == 1){
192                         for(int j=0;j<Nk;j++)   round_k[j]=roundKey[4*i+j];
193                 }
194                 else{
195                         for(int j=0;j<Nk;j++)   round_k[j] = roundKey[4*(Nr-i)+j];//roundkey10부터 역순으로
    적용
196                         MixColumns((uint8_t*)round_k,mode);//roundKey에 대해 mixcol을 해줘야함.
197                 }
198                 AddRoundKey(state,round_k);
199         }
200         SubBytes(state, mode);
201         ShiftRows(state, mode);
202         if(mode == 1){
203                 for(int k=0;k<Nk;k++)   round_k[k] = roundKey[4*Nr+k];//마지막 roundKey이용
204                 AddRoundKey(state,round_k);
205         }
206         else    AddRoundKey(state,roundKey);//첫번째 roundKey이용
207
208
209 }
```



① 에서 암호화는 첫번째 roundKey를 Add해주는 반면 복호화에서는 마지막 roundKey를 이용함.

② -1 암호화는 위와같이 순차적으로 진행한다.

-2 복호화



그림과 같이 roundKey를 더한 후 inverse Mixcol한 값과 roundKey와 암호를 Mixcol한 값을 더한(Add) 결과는 같다.
이를 이용하여 inverseCipher은 Cipher 방법을 그대로 이용한다.

③ ①과 반대로 암호화는 마지막 roundKey를 복호화는 첫번째 roundKey를 Add 해준다.

<컴파일 과정>

환경 : ubuntu-20.04.3-desktop-amd64(리눅스)

리눅스 환경을 고려하여 arc4random()을 사용하기 위해 2가지를 변경하였다.

1. test.c 함수의 #include<stdlib.h>를 #inlcude<bsd/stdlib.j>로 변경하였다.

```
#include <stdio.h>
#include <string.h>
#include <bsd/stdlib.h>
#include "aes.h"
```

2. gcc 링크 시 -lbsd옵션을 사용하기 위해서 Makefile에 -lbsd를 추가하였다.

```
CC=gcc
CFLAGS=-Wall

all: test.o aes.o
	$(CC) $(CFLAGS) -o test test.o aes.o -lbsd

test.o: test.c aes.h
	$(CC) $(CFLAGS) -c test.c

aes.o: aes.c aes.h
	$(CC) $(CFLAGS) -c aes.c

clean:
	rm -rf *.o
	rm -rf test
```

위 내용들은 PROJECT#1을 참고하였다.

컴파일 결과

```
soo@soo-VirtualBox:~/Documents/crypto$ make clean
rm -rf *.o
rm -rf test
soo@soo-VirtualBox:~/Documents/crypto$ make
gcc -Wall -c test.c
gcc -Wall -c aes.c
gcc -Wall -o test test.o aes.o -lbsd
soo@soo-VirtualBox:~/Documents/crypto$
```

<실행 결과물>

최종 실행결과

```
soo@soo-VirtualBox:~/Documents/crypto$ ./test
<키>
0f 15 71 c9 47 d9 e8 59 0c b7 ad d6 af 7f 67 98
<라운드 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
---
<평문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<암호문>
ff 0b 84 4a 08 53 bf 7c 69 34 ab 43 64 14 8f b9
<복호문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<역암호문>
1f e0 22 1f 19 67 12 c4 be cd 5c 1c 60 71 ba a6
<복호문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Random testing.................No error found
soo@soo-VirtualBox:~/Documents/crypto$
```

라운드키가 제대로 생성되었는지 예상출력-1.txt를 통해 확인하였고

제시된 <평문>에 대한 암호문과 <역암호문>이 제대로 작동하여 <복호문>이 <평문>과 같음을 확인하였다. 또한 랜덤테스팅에 대한 에러는 발생하지 않았다.

(subbytes, shiftrows, mixcolumns)함수에 대한 진행 과정중 실행 결과

Subbytes

```
<평문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<암호문>
7c 26 6e 85 a7 62 bd df bb 86 f4 46 38 20 23 ca
```

Shiftrows

```
<평문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<암호문>
7c 62 f4 ca a7 86 23 85 bb 20 6e df 38 26 bd 46
```

Mixcolums

```
<평문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<암호문>
60 75 a8 9d 62 50 f3 46 bc 96 3d 3d e1 ee b5 5f
```

<소감>

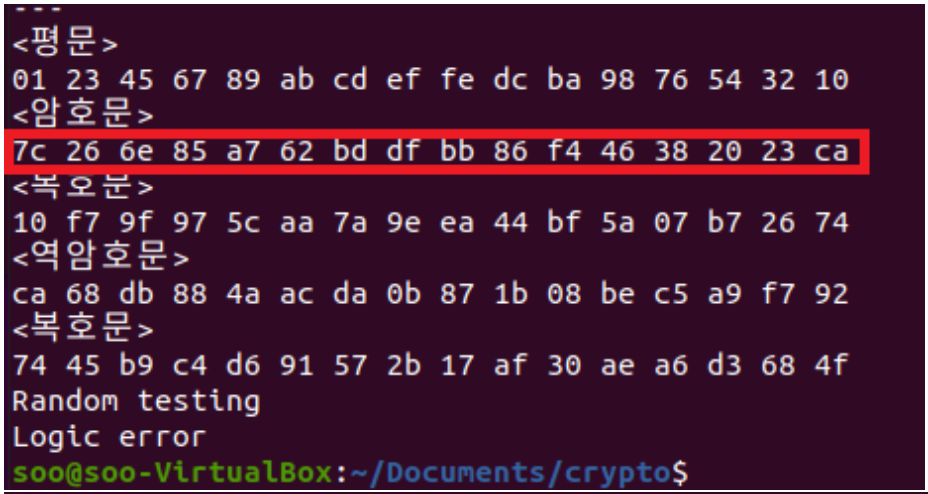
어려웠던점1

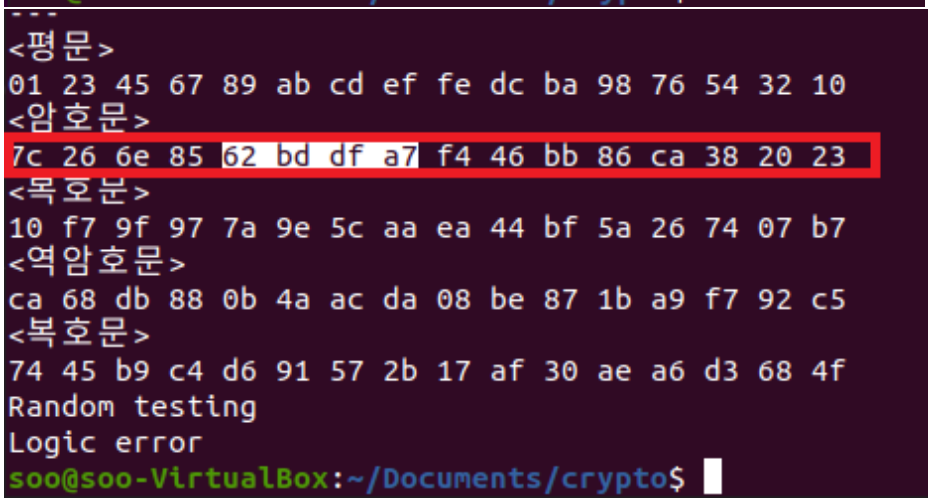
static void ShiftRows(uint8_t *state, int mode) 에서 state 순서를 잘못 인지하여 Mixcolumns()를 진'행하면서 문제를 확인할 수 있었다.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

를 ShiftRows하여

0	1	2	3
5	6	7	4
10	11	8	9
15	12	13	11



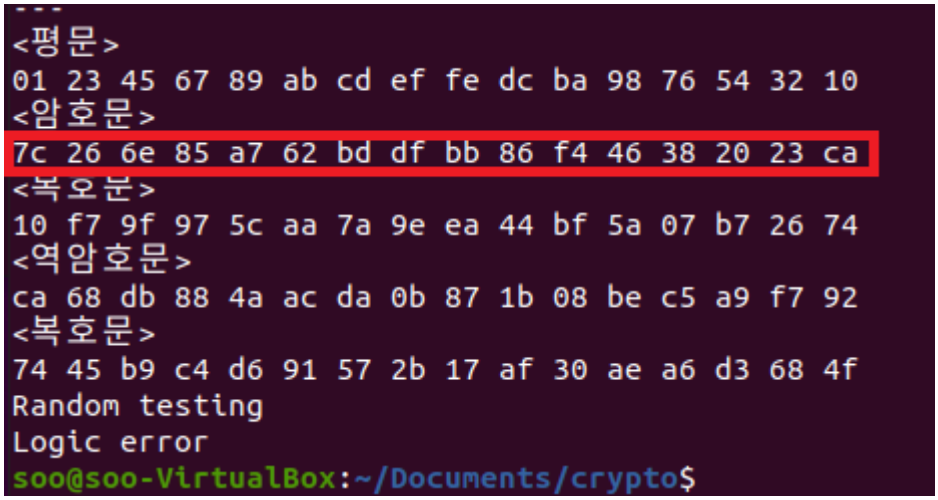


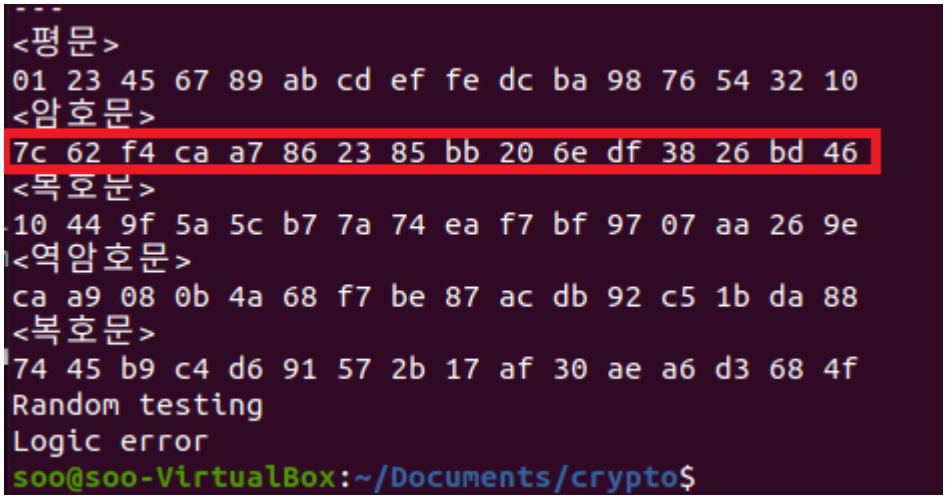
이렇게 적용하였지만 실제 position은 아래와 같기 때문에 수정하였다.

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

를 ShiftRows하여

0	4	8	12
5	9	13	1
10	14	2	6
15	3	7	11





어려웠던점2

void KeyExpansion(const uint8_t *key, uint32_t *roundKey)

```
134         while(i<Nk)//input된 key값을 (key[0~3],key[4~7],...)로 쪼개 subkey(roundKey0~3) 생성
135         {
136                 roundKey[i] = ((uint32_t)key[4*i+3]<<24^(uint32_t)key[4*i+2]<<16^
137                                (uint32_t)key[4*i+1]<<8^(uint32_t)key[4*i]);
138                 i++;
139         }
```

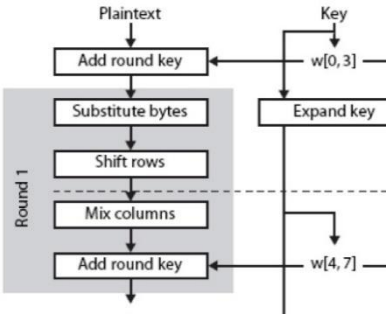
i%Nk == 0인 부분에 대한 정보는 수업시간에도 직접 연산과정을 통해 알아보았고, 제공받은 fips-197에서 충분히 다뤘지만, 첫 roundKey 4개를 생성하는 과정은 오히려 당연한 부분이라 처음에는 이해하기가 어려웠다.

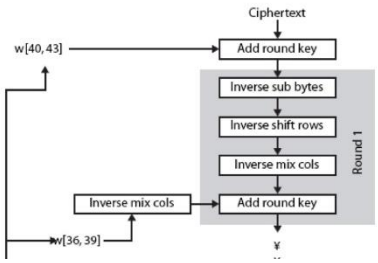
```
16 uint8_t key[KEYLEN] = {0x0f, 0x15, 0x71, 0xc9, 0x47, 0xd9, 0xe8, 0x59, 0x0c, 0xb7, 0xad, 0xd6, 0xa
   f, 0x7f, 0x67, 0x98};
```

test.c에 나와있는 key의 형태와 roundKey의 자료형을 통해 fips-197에 나와있는 pseudo코드를 이해하는 과정을 거쳐 roundKey를 생성하였다.

어려웠던점3

Chiper을 완성한 후 복호인 inverse chiper을 코딩할 때 같은 구조를 이용한다는 내용을 알지만 이해하지 못해 구조를 통해 다시 이해하는 과정을 거쳤다.





두가지를 비교해 본 결과 한가지 차이점을 확인할 수 있었는데 암호화의 경우 state인 Plaintext에 roundKey를 더한 이후 inverse mixcol하는 방식이지만 복호화의 경우엔 state와 roundKey각각을 inverse mixcol한 이후에 add해주는 방식이였다. 그 외의 과정은 같은 구조임을 확인할 수 있어 inverse chiper는 chiper을 이용할 수 있었다.

완벽하게 이해가 되지 않았을 때 roundKey에 대한 inverse mixcol을 놓쳐 복호화를 하지 못하기도 하였다.

```
<평문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<암호문>
60 ee 3d 46 75 b5 3d 62 a8 5f bc 50 9d e1 96 f3
<복호문>
b7 34 91 57 c3 cd cd a1 d8 4b 01 e9 c1 b0 ea 87
<역암호문>
fe 8a c9 f0 5b 23 a1 4e d5 11 43 9a 0f 7b 1e 65
<복호문>
50 64 f2 b8 94 bf a7 11 7e bd 97 3c 70 e4 12 d0
Random testing
Logic error
soo@soo-VirtualBox:~/Documents/crypto$
```

수정후

```
<평문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<암호문>
ff 0b 84 4a 08 53 bf 7c 69 34 ab 43 64 14 8f b9
<복호문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
<역암호문>
1f e0 22 1f 19 67 12 c4 be cd 5c 1c 60 71 ba a6
<복호문>
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Random testing.................No error found
soo@soo-VirtualBox:~/Documents/crypto$
```

속도 증진을 위한 노력

이해를 위해 중간중간 printf를 이용하였는데 상당한 양의 computing이 있다는 것을 알수있었다.

속도증진을 가져온state내용을 유지하기 위해 따로 저장해주는 과정을 거쳤는데 일반적인 array사용이 편리하여 for 문을 통해 저장해 주는 과정을 memcpy함수를 이용하여 메모리를 복사해 올수 있도록 하였다.

수정전

```
64 static void ShiftRows(uint8_t *state, int mode)
65 {
66         uint8_t tmp_k[Nk*Nb];//가져온 state값을 보존하기 위해 tmp_k에 임시로 저장
67         for(int k = 0;k < Nk*Nb; k++)
68         {
69                 tmp_k[k] = state[k];
70         }
71         if(mode == 1){//encrypt
72                 for(int i = 1; i < Nb; i++)//0번째 row는 이동하지 않으므로 1번째 부터
73                 {
74                         for(int j = 0;j < Nk; j++)      state[Nb*j+i] = tmp_k[((Nb*j+i)+i*Nk)%16];//i씩 왼쪽
   으로 이동하고, 값이 0~15사이를 벗어나는 경우 고려
75                 }
76         }
77         else{//decrypt
78                 for(int i=1;i<Nb;i++)
79                 {
80                         for(int j=0;j<Nk;j++)   state[Nb*j+i] = tmp_k[((Nb*j+i)-i*Nk+16)%16];//i씩 오른으로
   이동하고, 값이 0~15사이를 벗어나는 경우 고려
81                 }
82         }
83 }
```

```
 97 static void MixColumns(uint8_t *state, int mode)//M[16], IM[16]을 이용
 98 {
 99         uint8_t tmp_k[Nk*Nb];//가져온 state값을 보존하기 위해 tmp_k에 임시로 저장
100         for(int k = 0;k < Nk*Nb; k++)
101         {
102                 tmp_k[k] = state[k];
103         }
104         for(int i=0;i<4;i++)
105         {
106                 for(int j=0;j<4;j++)
107                 {
108                         if(mode == 1){//그림설명
109                                 state[4*i+j] = gf8_mul(M[4*j],tmp_k[4*i]) ^
110                                         gf8_mul(M[4*j+1],tmp_k[4*i+1]) ^
111                                         gf8_mul(M[4*j+2],tmp_k[4*i+2]) ^
112                                         gf8_mul(M[4*j+3],tmp_k[4*i+3]);
113                         }
114                         else{
115                                 state[4*i+j] = gf8_mul(IM[4*j],tmp_k[4*i]) ^
116                                         gf8_mul(IM[4*j+1],tmp_k[4*i+1]) ^
117                                         gf8_mul(IM[4*j+2],tmp_k[4*i+2]) ^
118                                         gf8_mul(IM[4*j+3],tmp_k[4*i+3]);
119
120                         }
121                 }
122         }
123
124 }
```

수정후

```
64 static void ShiftRows(uint8_t *state, int mode)
65 {
66         uint8_t tmp_k[Nk*Nb];//가져온 state값을 보존하기 위해 tmp_k에 임시로 저장
67
68         memcpy(tmp_k,state,sizeof(uint8_t)*Nk*Nb);
69         if(mode == 1){//encrypt
70                 for(int i = 1; i < Nb; i++)//0번째 row는 이동하지 않으므로 1번째 부터
71                 {
72                         for(int j = 0;j < Nk; j++)      state[Nb*j+i] = tmp_k[((Nb*j+i)+i*Nk)%16];
   //i씩 왼쪽으로 이동하고, 값이 0~15사이를 벗어나는 경우 고려
73                 }
74         }
75         else{//decrypt
76                 for(int i=1;i<Nb;i++)
77                 {
78                         for(int j=0;j<Nk;j++)   state[Nb*j+i] = tmp_k[((Nb*j+i)-i*Nk+16)%16];//i씩
    오른으로 이동하고, 값이 0~15사이를 벗어나는 경우 고려
79                 }
80         }
81 }
```

```
95 static void MixColumns(uint8_t *state, int mode)//M[16], IM[16]을 이용
96 {
97         uint8_t tmp_k[Nk*Nb];//가져온 state값을 보존하기 위해 tmp_k에 임시로 저장
98
99         memcpy(tmp_k,state,sizeof(uint8_t)*Nk*Nb);
100        for(int i=0;i<4;i++)
101        {
102                for(int j=0;j<4;j++)
103                {
104                        if(mode == 1){//그림설명
105                                state[4*i+j] = gf8_mul(M[4*j],tmp_k[4*i]) ^
106                                        gf8_mul(M[4*j+1],tmp_k[4*i+1]) ^
107                                        gf8_mul(M[4*j+2],tmp_k[4*i+2]) ^
108                                        gf8_mul(M[4*j+3],tmp_k[4*i+3]);
109                        }
110                        else{
111                                state[4*i+j] = gf8_mul(IM[4*j],tmp_k[4*i]) ^
112                                        gf8_mul(IM[4*j+1],tmp_k[4*i+1]) ^
113                                        gf8_mul(IM[4*j+2],tmp_k[4*i+2]) ^
114                                        gf8_mul(IM[4*j+3],tmp_k[4*i+3]);
115
116                        }
117                }
118        }
119
120 }
```

비트연산이 컴퓨터에서 가장 빠른 연산이라는 정보를 통해 속도 증진을 위해 Mixcolumns에서는곱연산 대신 project#1에서 진행한 galios filed 비트의 곱연산을 이용하였고 Keyexpansion에서 또한 연산을 모두 비트연산으로 사용하였다.

```
95 static void MixColumns(uint8_t *state, int mode)//M[16], IM[16]을 이용
96 {
97         uint8_t tmp_k[Nk*Nb];//가져온 state값을 보존하기 위해 tmp_k에 임시로 저장
98
99         memcpy(tmp_k,state,sizeof(uint8_t)*Nk*Nb);
100        for(int i=0;i<4;i++)
101        {
102                for(int j=0;j<4;j++)
103                {
104                        if(mode == 1){//그림설명
105                                state[4*i+j] = gf8_mul(M[4*j],tmp_k[4*i]) ^
106                                        gf8_mul(M[4*j+1],tmp_k[4*i+1]) ^
107                                        gf8_mul(M[4*j+2],tmp_k[4*i+2]) ^
108                                        gf8_mul(M[4*j+3],tmp_k[4*i+3]);
109                        }
110                        else{
111                                state[4*i+j] = gf8_mul(IM[4*j],tmp_k[4*i]) ^
112                                        gf8_mul(IM[4*j+1],tmp_k[4*i+1]) ^
113                                        gf8_mul(IM[4*j+2],tmp_k[4*i+2]) ^
114                                        gf8_mul(IM[4*j+3],tmp_k[4*i+3]);
115
116                        }
117                }
118        }
119
120 }
```

```
127 void KeyExpansion(const uint8_t *key, uint32_t *roundKey)
128 {
129        int i=0;
130        uint32_t tmp_k,Rot,Sub;
131
132        while(i<Nk)//input된 key값을 (key[0~3],key[4~7],...)로 쪼개 subkey(roundKey0~3) 생성
133        {
134                roundKey[i] = ((uint32_t)key[4*i+3]<<24^(uint32_t)key[4*i+2]<<16^
135                                (uint32_t)key[4*i+1]<<8^(uint32_t)key[4*i]);
136                i++;
137        }
138        while(i<(Nb*(Nr+1)))
139        {
140                tmp_k = roundKey[i-1];
141
142                if(i%Nk == 0)//128비트마다 강의 자료에 나온g값을 포함하여 계산
143                {
144                        //rotword
145                        Rot = (tmp_k<<24 ^ tmp_k>>8);
146                        //subword
147                        Sub = 0;//초기화
148                        for(int j=0;j<32;j+=8){
149                                Sub = Sub ^ (uint32_t)(sbox[(uint8_t)(Rot>>j)])<<j;
150                        }
151                        //
152                        tmp_k = Sub ^ Rcon[i/Nk];
153                }
154                roundKey[i] = roundKey[i-Nk] ^ tmp_k;
155                i++;
156        }
157
158 }
159
```