REPORT

암호학 프로젝트2 AES구현(FIPS-197)

과목명	암호학
담당교수	오희국 교수님
학생이름	지현도
학과	컴퓨터학부
학번	2021004866
제출일	2023/10/04

HANYANG UNIVERSITY

*본인이작성한함수에대한설명:

1. void LRotWord_4

$$g(w_3) = \text{S-Box}(\text{LRotWord}(w_3)) \oplus \text{RCon}_i$$

강의노트에 나와있듯 Gfunction에 사용되는 LRotWord를 다음과 같이 구현해주었다.

```
66 void LRotWord_4(uint8_t *n0, uint8_t *n1, uint8_t *n2, uint8_t *n3, int n){
67     // n칸씩 왼쪽으로 이동
68   if(n == 1){
69     uint8_t tmp = *n0;    // swap 매크로처럼 tmp에 첫값 n0을 저장한 후 변수들을 바꿔준다.
70     *n0 = *n1;
71     *n1 = *n2;
72     *n2 = *n3;
73     *n3 = tmp;
74   }else if(n == 2){
75     uint8_t tmp = *n0;
76     *n0 = *n2;
77     *n2 = tmp;
78     tmp = *n1;
79     *n1 = *n3;
80     *n3 = tmp;
81   }else if(n == 3){
82     uint8_t tmp = *n3;
83     *n3 = *n2;
84     *n2 = *n1;
85     *n1 = *n0;
86     *n0 = tmp;
87   }
88 }
```

함수 이름에서 알 수 있듯 기본적으론 4개의 데이터를 움직이는 것을 기본으로 구현하였다.

2. void uint8_to_uint32

Uint8_t 의 데이터를 쉬프트연산을 통해 32비트의 uint32_t 형태로 만들어주는 함수를 구현하였다.

```
90  void uint8_to_uint32(const uint8_t *n, uint32_t *r){
91    *r = ((uint32_t)n[3] << 24) | ((uint32_t)n[2] << 16) | ((uint32_t)n[1] << 8) |
          (uint32_t)n[0];
92  }
```

이는 아래에서 사용될 KeyExpansion에서 첫라운드 계산에서 사용되었습니다.

3. uint32_t Gfunc

```
94  uint32_t Gfunc(const uint32_t *round_key, int roundNum){
95    uint8_t tmp[4];
96    memcpy(tmp, round_key, 4);
97    LRotWord_4(tmp,tmp+1,tmp+2,tmp+3,1);
98    tmp[0] = sbox[tmp[0]];
99    tmp[1] = sbox[tmp[1]];
100   tmp[2] = sbox[tmp[2]];
101   tmp[3] = sbox[tmp[3]];
102   uint32_t r;
103   uint8_to_uint32(tmp, &r); //uint32로 형변환
104
105   return r ^ Rcon[roundNum];
106 }
```

KeyExpansion에서 가장 중요한 Gfunction을 구현하였다.

위에서 구현한 LRotWord와 구현되있는 sbox를 사용하여 계산한다. 이때 round_key값이 바뀌면 안되므로 memcpy를 활용하여 tmp로 복사해놓는다. 그다음 Rcon과 XOR연산을 실행한다.

4. void KeyExpansion

```
108 void KeyExpansion(const uint8_t *key, uint32_t *round_Key)
109 {
110   // key는 1바이트, round key는 4바이트
111
112   // 첫 라운드는 사용자 키를 사용
113   uint8_to_uint32(key, &round_Key[0]);
114   uint8_to_uint32(key+4, &round_Key[1]);
115   uint8_to_uint32(key+8, &round_Key[2]);
116   uint8_to_uint32(key+12, &round_Key[3]);
117
118   for(int i=Nk; i<RNDKEYLEN; i++){ // expansion 과정
119     if(i % Nk == 0) round_Key[i] = Gfunc(round_Key+i-1, i/Nk) ^ round_Key[i-Nk];
120     else round_Key[i] = round_Key[i-1] ^ round_Key[i-Nk];
121   }
122 }
```

키길에 따라 반복문으로 Gfunc을 활용하여 KeyExpansion을 구현하였다.

5. void AddRoundKey, SubBytes, ShiftRows

```
124 static void AddRoundKey(uint8_t *state, const uint32_t *round_Key){
125   uint8_t *tmp = (uint8_t *)round_Key;
126   for(int i=0; i<BLOCKLEN; i++) state[i] ^= tmp[i];
127 }
128
129 static void SubBytes(uint8_t *state, int mode){
130   if(mode == ENCRYPT){
131     for(int i=0; i<BLOCKLEN; i++) state[i] = sbox[state[i]];
132   }
133   else{
134     for(int i=0; i<BLOCKLEN; i++) state[i] = isbox[state[i]];
135   }
136 }
137
138 static void ShiftRows(uint8_t *state, int mode){
139   if(mode == ENCRYPT){
140     LRotWord_4(state+1, state+5, state+9, state+13, 1); // 왼쪽으로 n번 이동
141     LRotWord_4(state+2, state+6, state+10, state+14, 2);
142     LRotWord_4(state+3, state+7, state+11, state+15, 3);
143   }
144   else{
145     LRotWord_4(state+1, state+5, state+9, state+13, 3); // 오른쪽으로 n번 이동 == 왼쪽으로
          Nb-n번 이동
146     LRotWord_4(state+2, state+6, state+10, state+14, 2);
147     LRotWord_4(state+3, state+7, state+11, state+15, 1);
148   }
149 }
```

AddRoundKey, SubBytes, ShiftRows는 위와같이 정해진 규격에 따라 구현이 가능하였다. ShiftRows는 미리 구현해둔 LRotWord_4 를 활용하였는데 4개의 데이터를 하나로 사용하므로 입력변수가 4씩 차이나는것을 확인할 수 있다.

6. void MixColumns

```
151 uint8_t gf8_mul(uint8_t a, uint8_t b){ //g(2^8)
152     uint8_t r = 0;
153     while(b>0){
154         if(b&1) r=r^a;
155         b = b >> 1;
156         a = XTIME(a);
157     }
158     return r;
159 }
160
161
162 static void MixColumns(uint8_t *state, int mode){
163   // memcpy 활용한 g(2^8)행렬곱 계산, 모드가 DECRPYT이면 역행렬을 곱한다
164   // if (mode==ENCRYPT)state = M*state, if (mode==DECRYPT)state = IM*state
165   uint8_t tmp[BLOCKLEN], tmp2[BLOCKLEN];
166
167   memcpy(tmp2,state,BLOCKLEN);
168   if(mode == ENCRYPT) memcpy(tmp,M,sizeof(M));
169   else memcpy(tmp,IM,sizeof(IM));
170
171   for(int i=0; i<4; i++){
172     for(int j=0; j<Nb; j++){
173       int idx = 4*j+i;
174       state[idx] = 0;
175
176       for(int k=0; k<4; k++){ //tmp[i][k]*state[k][j]
177         state[idx] ^= gf8_mul(tmp[4*i+k],tmp2[4*j+k]);
178       }
179     }
180   }
181 }
```

우선 MixColumns를 구현하기 위해 8차기약다항식의 행렬곱을 구현하기위한 gf8_mul를 구현해주었다. 이는 이미 헤더파일에 존재하는 XTIME를 활용하여 강의노트에 있는 함수 그

대로 사용해주었다. 그 후 memcpy를 이용하여 만약 모드가 암호화(ENCRYPT)라면 행렬M을 state와 곱해주고, 모드가 복호화(DECRYPT)라면 IM를 state와 곱해준다. State값은 변하면 안되므로 tmp2에 저장한 후 이를 계산에 활용한다.

7. void Cipher

```
188 void Cipher(uint8_t *state, const uint32_t *round_Key, int mode)
189 {
190     // 암호화일때와 복호화일때는 각각 SubBytes와 ShiftRows의 순서와 Mixcolumns와 AddRoundKey의
            순서가 반대로 변한다.
191   if(mode == ENCRYPT){
192     AddRoundKey(state, round_Key);
193     for(int i=1; i<Nr; i++){
194       SubBytes(state,mode);
195       ShiftRows(state,mode);
196       MixColumns(state,mode);
197       AddRoundKey(state,round_Key+Nb*i);
198     }
199     SubBytes(state,mode);
200     ShiftRows(state,mode);
201     AddRoundKey(state,round_Key+Nb*Nr);
202   }
203   else{
204     AddRoundKey(state, round_Key+Nr*Nb);
205     for(int i=Nr-1; i>0; i--){
206       ShiftRows(state,mode);
207       SubBytes(state,mode);
208       AddRoundKey(state,round_Key+Nb*i);
209       MixColumns(state,mode);
210     }
211     ShiftRows(state,mode);
212     SubBytes(state,mode);
213     AddRoundKey(state,round_Key);
214   }
215 }
```

앞서 구현한 4가지 수식함수들을 이용하여 암호화일때와 복호화일때를 구분하여 state를 round_key를 사용하여 암호화 혹은 복호화해준다.

*컴파일 과정

```
jihyeondo@192 proj#2 % make
gcc -Wall -O3 -c aes.c
gcc -o test test.o aes.o
jihyeondo@192 proj#2 %
```

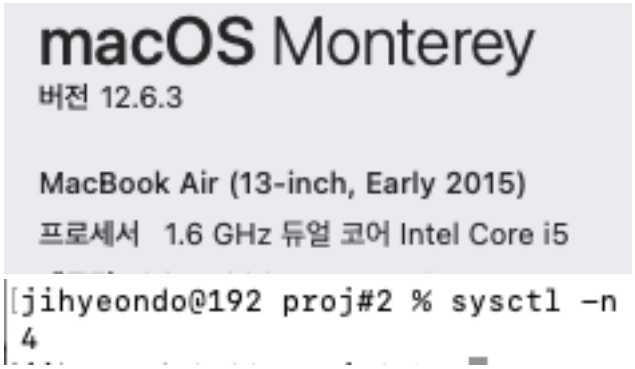
오류없이 정상적으로 컴파일되는 모습이다.

*실행 결과물

```
jihyeondo@192 proj#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 .....PASSED
---
AES 성 능 시 험 .....PASSED
CPU 사 용 시 간  = 35.9358초
jihyeondo@192 proj#2 %
```

실행 결과과 예상 출력과 모두 일치하고 오류없이 모두 통과되는 모습을 볼 수 있다. 아마 모든 함수가 정상적으로 작동

하는 것 같다. 그런데 예상결과와 속도차이가 꽤 많이 나는 것으로 보였는데 이는 제가 구동한 노트북의 cpu사양이 좋지 않기 때문이거나 혹은 제 코드의 구성이 속도에 큰 영향을 끼칠 정도로 교수님의 답안 코드에 비해 깔끔하지 못하기 때문인 것 같습니다.



개인적으론 전자인 노트북의 성능문제라고 믿고 싶습니다.

*프로젝트 소감 및 부족한 점

우선 교수님의 pdf파일의 점검 사항에 나와있듯 Uint8_t 정수와 uint32_t 정수가 섞여있을때 큰 타입으로 변환하여 계산하는 것이 유리하다 하셔서 그런 과정을 추가하였고, 반복문을 줄이기위해 LRotWord_4같은 함수를 구현할때는 다소 하드코딩의 느낌이 나게끔 구현한 것 같습니다. 이부분이 제 코드의 속도 향상에 도움을 준 것인지는 고민이 더 필요할 것 같습니다. 특히 C로 구현하는 코드이다보니 메모리 관리에 어려움이 더 많은 것 같습니다.

또한 aes128비트 뿐만아니라 aes192, aes256비트에도 사용가

능한 함수를 구현해볼 수 있다면 좋을 것 같습니다. 때문에 key 길이에 따른 유동적인 코드작성을 더 연습해봐야할 필요성을 느끼게 되었습니다.