암호학

Project #2

소프트웨어학부
2017012251
윤영훈

1. source code 설명

source code 설명에 앞서 리눅스 환경의 영향으로 기존의 test.c / aes.h file과 makefile에서는 arc4random 함수가 정상적으로 동작하지 않아 error가 발생했다.

따라서

```
5   #ifndef AES_H
6   #define AES_H
7
8   #include <stdint.h>
9
10  #include <bsd/stdlib.h>
```

```
1 CC=gcc
2 CFLAGS=-Wall
3
4 all: test.o aes.o
5         $(CC) $(CFLAGS) -o test test.o aes.o -lbsd
6
7 test.o: test.c aes.h
8         $(CC) $(CFLAGS) -c test.c
9
10 aes.o: aes.c aes.h
11        $(CC) $(CFLAGS) -c aes.c
12
13 clean:
14        rm -rf *.o
15        rm -rf test
```

aes.h file에서 line10. #include <bsd/stdlib.h> 헤더를 추가하고 makefile에서 line5의 끝에 -lbsd 옵션을 추가하였다.

① AddRoundKey

```
148 // 지역 함수 1 AddRoundKey : 라운드 키를 XOR 연산을 사용하여 state에 더함
149 // round에 따라 roundKey를 구분해서 적용하기 위해 int round 인자를 추가
150 static void AddRoundKey(uint8_t *state, const uint32_t *roundKey, int round)
151 {
152   uint8_t *sub;
153   uint32_t a[4];
154
155   // uint8_t 자료형인 state를 uint32_t 자료형인 roundKey와 xor 하기 위해선 4번의 연산이 필요함
156   // xor 연산 수를 1번으로 줄이기 위해 state 값을 합쳐 uint32_t 자료형으로 변형
157   for (int i=0; i<Nb; i++){
158     a[i] = (uint32_t) state[Nb*i] + (uint32_t) state[Nb*i + 1] * 0x00000100 + (uint32_t) state[Nb*i + 2] * 0x00010000 + (uint32_t) state[Nb*i + 3] * 0x01000000;
159   }
160
161   // 변형된 uint32_t state를 roundkey와 xor 연산
162   for (int i=0; i<Nb; i++){
163     a[i] ^= roundKey[Nb*round + i];
164   }
165
166   // 변형된 uint32_t state를 다시 uint8_t state로 복구
167   for (int i=0; i<Nb; ++i) {
168     sub = (uint8_t *)(a+i);
169     state[Nb*i] = sub[0]; state[Nb*i + 1] = sub[1]; state[Nb*i + 2] = sub[2]; state[Nb*i + 3] = sub[3];
170   }
171 }
```

AddRoundKey의 경우 uint8_t 자료형인 state와 uint32_t 자료형인 roundKey의 연산이므로 효율적인 함수 구현을 위해 더 큰 범위의 uint32_t 자료형으로 통일하여 4번의 연산을 1번으로 줄였다. 이때 uint8_t인 state를 uint32_t로 변환하기 위해 state 4개를 합쳐 하나로 만들었다.

state[Nb*i]	state[Nb*i + 1]	state[Nb*i + 2]	state[Nb*i +3]
aa	bb	cc	dd
a[i]			
aabbccdd			

이후 roundKey와 xor 연산을 처리한 뒤 변형된 uint32_t state를 다시 uint8_t state로 복구했다.

② SubBytes

```
173  // 지역 함수 2 SubBytes : mode에 따라 순방향 혹은 역방향으로 바이트를 치환한다.
174  static void SubBytes(uint8_t *state, int mode)
175  {
176
177    // 암호화
178    if (mode == ENCRYPT) {
179      for (uint8_t i=0; i<BLOCKLEN; i++){
180        state[i] = sbox[state[i]];
181      }
182    }
183
184    // 복호화
185    else if (mode == DECRYPT) {
186      for (uint8_t i=0; i<BLOCKLEN; i++){
187        state[i] = isbox[state[i]];
188      }
189    }
190  }
```

암호화에 필요한 sbox, 복호화에 필요한 isbox가 이미 제공되었으므로 간단한 table lookup을 통해 SubBytes를 처리하였다.

③ ShiftRows

```
192  // 지역 함수 3 ShiftRows : mode에 따라 순방향 혹은 역방향으로 바이트의 위치를 변경한다.
193  static void ShiftRows(uint8_t *state, int mode)
194  {
195    uint8_t temp[16];
196
197    // 암호화
198    // 변경되는 index가 modulo 16 안에서 5씩 증가하는 규칙을 발견하여 하나의 for loop로 구현
199    if (mode == ENCRYPT) {
200      for (int i=0; i<BLOCKLEN; i++){
201        temp[i] = state[5*i % BLOCKLEN];
202      }
203
204      for (int i = 0; i < BLOCKLEN; i++) {
205        state[i] = temp[i];
206      }
207    }
208
209    // 복호화
210    // 변경되는 index가 modulo 16 안에서 3씩 감소하는 규칙을 발견하여 하나의 for loop로 구현
211    else if (mode == DECRYPT) {
212      for (int i=BLOCKLEN-1; i>=0; i--){
213        temp[i] = state[(BLOCKLEN-i)*3 % BLOCKLEN];
214      }
215
216      for (int i=0; i<BLOCKLEN; i++) {
217        state[i] = temp[i];
218      }
219    }
220  }
```

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

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

Shift 후의 결과를 확인하면 index에 따라 순차적으로 0 -> 5 -> 10 -> 15 -> 4 -> 9 -> 14 ... 의 값이 들어감을 확인할 수 있다. 이는 0 -> 5 -> 10 -> 15 -> 20 (16+4) -> 25 (16+9) -> 30 (16+14)를 modulo 16으로 표현했을때와 같은 결과이다. 따라서 5*i % BLOCKLEN 으로 표현할 수 있고 복호화에서도 마찬가지로 (BLOCK-i)*3 % BLOCKLEN 으로 표현하여 결과적으로 각각 하나씩의 for loop만으로 구현할 수 있다.

④ gf8_mul & MixColumns

```
193  uint8_t gf8_mul(uint8_t a, uint8_t b){
194    uint8_t r = 0;
195
196    while (b>0){
197      if (b & 1) r = r ^ a;
198      b = b >> 1;
199      a = XTIME(a);
200    }
201
202    return r;
203  }
```

```
235  // 지역 함수 4 MixColumns : 기약 다항식 x^8 + x^4 + x^3 + x + 1 을 사용한 GF(2^8)에서 행렬곱셈을 수행한다. mode가 DECRYPT이면 역행렬을 곱한다.
236  static void MixColumns(uint8_t *state, int mode)
237  {
238
239    // 암호화
240    if (mode == ENCRYPT){
241      uint8_t array[BLOCKLEN];
242
243      memcpy(array, state, sizeof(uint8_t) * BLOCKLEN);
244
245      // 제공된 M matrix와 state 간 modulo x^8 + x^4 + x^3 + x + 1 안에서의 행렬곱셈을 계산 후 xor 처리
246      for (uint8_t i=0; i<4; i++){
247        for (int j=0; j<4; j++){
248          state[i*4 + j] = gf8_mul(M[j*4], array[i*4]) ^ gf8_mul(M[j*4 + 1], array[i*4 + 1]) ^ gf8_mul(M[j*4 + 2], array[i*4 + 2]) ^ gf8_mul(M[j*4 + 3], array[i*4 + 3]);
249        }
250      }
251    }
252
253    // 복호화
254    else if (mode == DECRYPT){
255      uint8_t array[BLOCKLEN];
256
257      memcpy(array, state, sizeof(uint8_t) * BLOCKLEN);
258
259      // 제공된 IM matrix와 state 간 modulo x^8 + x^4 + x^3 + x + 1 안에서의 행렬곱셈을 계산 후 xor 처리
260      for (uint8_t i=0; i<4; i++){
261        for (int j=0; j<4; j++){
262          state[i*4 + j] = gf8_mul(IM[j*4], array[i*4]) ^ gf8_mul(IM[j*4 + 1], array[i*4 + 1]) ^ gf8_mul(IM[j*4 + 2], array[i*4 + 2]) ^ gf8_mul(IM[j*4 + 3], array[i*4 + 3]);
263        }
264      }
265    }
266  }
```

$x^8 + x^4 + x^3 + x + 1$을 사용한 GF(2^8) 행렬곱셈을 수행하기 위해 project#1에서 구현했던 gf8_mul 함수를 사용한다. 이때 XTIME은 aes.h에 미리 정의된 함수를 사용한다. MixColumns 역시 미리 제공된 M matrix, IM matrix가 있으므로 단순히 행렬곱셈 후 xor 처리하여 state에 넣어준다.

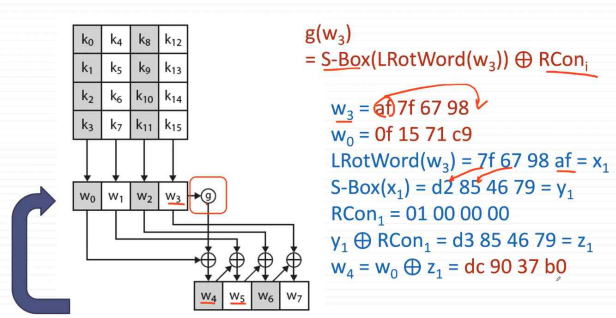
⑤ KeyExpansion

```
59  void KeyExpansion(const uint8_t *key, uint32_t *roundKey)
60  {
61    uint32_t temp;
62    uint8_t sub[4];
63    uint8_t *p;
64
65    // uint8_t 자료형의 key를 uint32_t 자료형인 roundKey에 넣기 위해 자릿수를 맞춰줌
66    // example : aa + bb + cc + dd > aa000000 + 00bb0000 + 0000cc00 + 000000dd > aabbccdd
67    for (uint8_t i=0; i<RNDKEYSIZE; i++){
68
69      // w0, w1, w2, w3은 변경사항이 없으므로 그대로 가져옴
70      if (i < Nk){
71        roundKey[i] = (uint32_t) key[4*i] + (uint32_t) key[4*i + 1] * 0x00000100 + (uint32_t) key[4*i + 2] * 0x00010000 + (uint32_t) key[4*i + 3] * 0x01000000;
72      }
73
74      // w4부터는 g(w(i)) : S-Box(LRotWord(w(i))) xor Rcon 처리와 더불어 w(i-4) xor w(i-1) 처리도 병행하여 업데이트
75      else{
76        temp = roundKey[i-1];
77
78        // i가 4의 배수인 경우 g(w(i)) 처리
79        if (i % Nk == 0){
80          p = (uint8_t *) (roundKey + i-1);
81
82          sub[0] = sbox[p[1]] ^ Rcon[i/Nk];
83          sub[1] = sbox[p[2]];
84          sub[2] = sbox[p[3]];
85          sub[3] = sbox[p[0]];
86
87          temp = (uint32_t) sub[0] + (uint32_t) sub[1] * 0x00000100 + (uint32_t) sub[2] * 0x00010000+ (uint32_t) sub[3] * 0x01000000 ;
88        }
89
90        // 최종적으로 w(i-4) xor w(i-1) 처리
91        roundKey[i] = roundKey[i-Nk] ^ temp;
92      }
93    }
94  }
```

uint8_t 자료형의 key를 사용하여 uint32_t 자료형의 roundKey를 만들기 위해 앞서 AddRoundKey에서 사용했던 방법처럼 4개의 key를 사용해 자릿수를 맞춰 uint32_t 자료형으로 변환해준다.

w0, w1, w2, w3은 변경사항이 없으므로 key값 그대로 가져오며, w4부터는 g(w(i)) 처리 후 w(i-4) xor w(i-1) 처리도 병행하여 업데이트 해준다.

g(w(i))
S-BOX(LRotWord(w(i)) xor RCon



w4, w8, w12 등 I가 4의 배수인 경우에만 g(w(i)) 처리해주며, 그 이외에는 g(w(i)) 처리 없이 w(i-4) 와 w(i-1)을 xor 처리하여 최종적으로 roundKey를 완성한다.

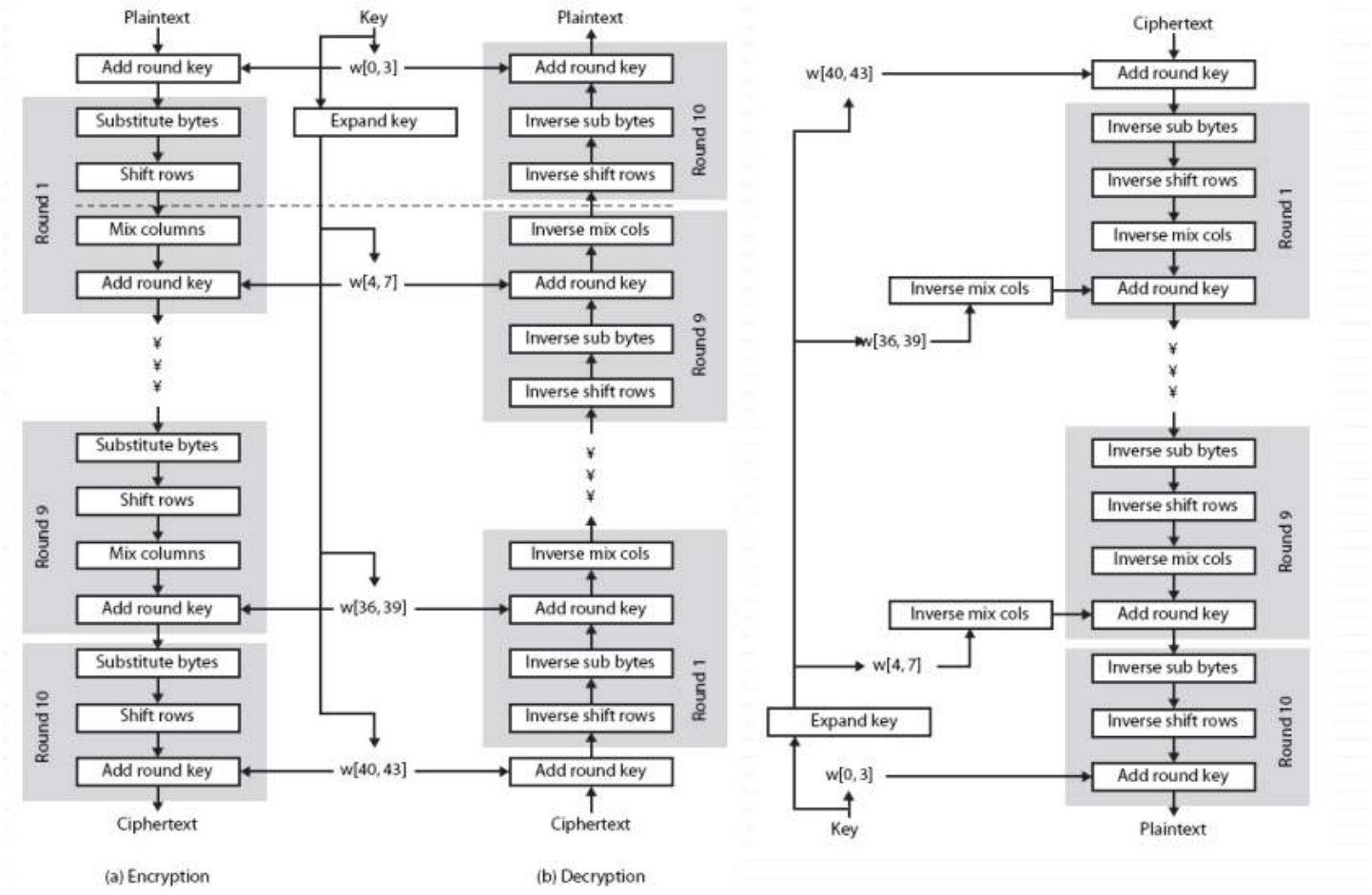
⑥ Cipher

```
100  void Cipher(uint8_t *state, const uint32_t *roundKey, int mode)
101  {
102
103    // 암호화
104    // round 0부터 10까지 반복적으로 AddRoundKey, SubBytes, ShiftRows, MixColumns 적용
105    // round 10에서는 MixColumns를 적용하지 않음
106    if (mode == ENCRYPT){
107      int round = 0;
108      AddRoundKey(state, roundKey, round);
109
110      round++;
111
112      for (int i=0; i<Nr-1; i++){
113        SubBytes(state, mode);
114        ShiftRows(state, mode);
115        MixColumns(state, mode);
116        AddRoundKey(state, roundKey, round);
117
118        round++;
119      }
120      SubBytes(state, mode);
121      ShiftRows(state, mode);
122      AddRoundKey(state, roundKey, round);
123    }
```

암호화

```
125    // 복호화
126    // round 10부터 0까지 반복적으로 AddRoundKey, SubBytes, ShiftRows, MixColumns 적용
127    // round 0에서는 MixColumns를 적용하지 않음
128    else if (mode == DECRYPT){
129      int round = Nr;
130      AddRoundKey(state, roundKey, round);
131
132      round--;
133
134      for(int i=0; i<Nr-1; i++){
135        SubBytes(state, mode);
136        ShiftRows(state, mode);
137        AddRoundKey(state, roundKey, round);
138        MixColumns(state, mode);
139
140        round--;
141      }
142      SubBytes(state, mode);
143      ShiftRows(state, mode);
144      AddRoundKey(state, roundKey, round);
145    }
146  }
```

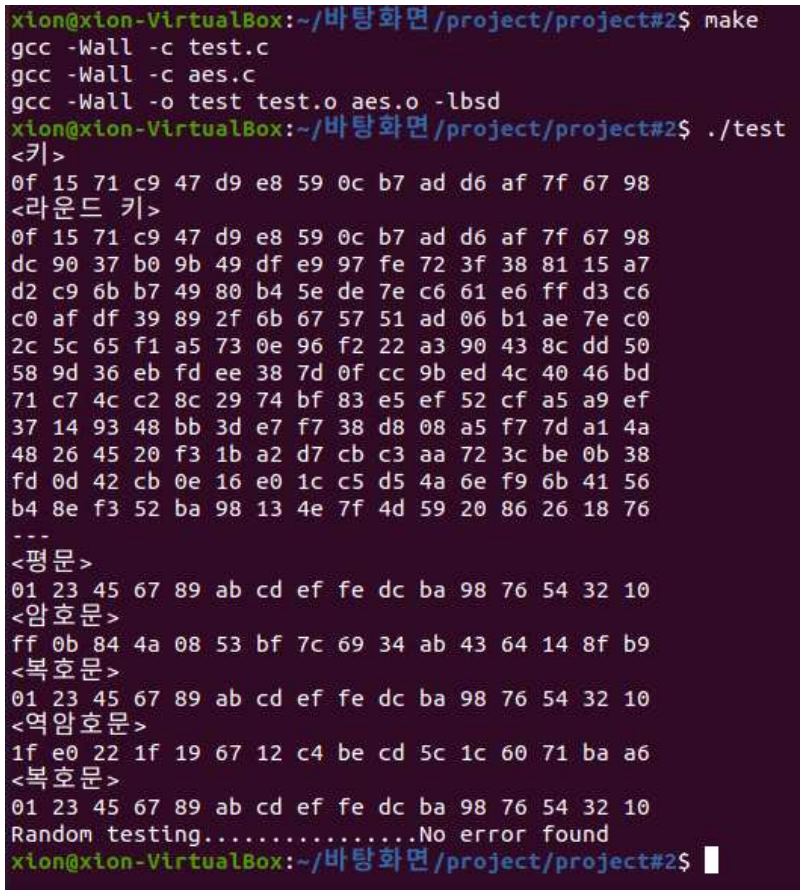
복호화



AES Structure에 맞춰 암호화에서는 round 0부터 round 10까지 반복적으로 AddRoundKey > SubBytes > ShiftRows > MixColumns를 적용하며 이때 마지막 round 10에서는 MixColumns를 적용하지 않는다.

복호화 시, 정석적으로는 암호화 순서의 반대로 AddRoundKey > ShiftRows > SubBytes > MixColumns 순서로 적용해야하지만 강의시간에 배웠던 대로 순서를 변형해도 무방하므로 최종적으로는 복호화에서도 암호화 순서와 같은 순서로 Cipher를 구현할 수 있다. 하지만 roundKey는 반대 순서로 사용해야 하므로 round는 Nr부터 시작하여 감소하며 round 0까지 반복하며, 이때 마지막 round 0에서는 MixColumns를 적용하지 않는다.

2. compile



warning, error 없이 정상적으로 compile 되었으며 test 실행도 정상적으로 완료되었음을 확인할 수 있다.

3. result

평문, 암호문, 복호문, 역암호문이 모두 정상적으로 출력되었으며 최종적으로 Random testing에서도 No error found가 출력됨을 확인할 수 있다. 터미널 명령어 '>'를 통해 aes.txt에 저장하여 별도로 제출한다.

4. feedback

AddRoundKey, KeyExpansion에서는 uint8_t와 uint32_t가 섞여있는 계산에서 연산 수를 줄이기 위해 uint8_t을 4개씩 합쳐 uint32_t로 변환하여 사용한 뒤 다시 uint8_t로 나눠주었다. 이론상 최대 4배의 이득을 얻을 수 있다고 하지만 uint8_t를 uint32_t로 합치는 과정, 다시 uint8_t로 나누는 과정 등 불필요한 요소들도 추가되었다. 따라서 정말 큰 타입을 작은 타입으로 변환하는 방법보다 결과가 뛰어난지 직접 확인하고 싶었으나 시간 감소 외에는 추가적으로 구별할 방법을 찾지 못해 자세히 비교해보지 못했다. 그래도 구현 중간의 프로토타입 버전과 최종 버전의 시간이 어느정도 차이나는 것으로 보아 해당 방법이 일정 수준의 영향을 끼친 것으로 사료된다.

또한 메모리 접근과 반복문의 횟수를 줄이기 위해 ShiftRows에서 마구잡이로 for loop를 사용해 구현하기보다 modulo 16이라는 규칙성을 찾아 하나의 for loop로 구현하였다. 프로토타입 버전에서는 이중 for loop를 사용해 구현하였고, 값을 SWAP하는데에 추가적인 SWAP 함수를 구현하여 사용하였었는데, 최종 버전에서는 하나의 for loop로 구현하면서 동시에 SWAP 함수도 사용하지 않게 되었으므로 이중 for loop의 부담과 함수 호출의 부담을 모두 줄일 수 있었다고 생각한다.