Cryptography Project #1

소프트웨어학부

2018044720 석예림

1. aes.c 소스코드

```c
/*
 * Copyright 2020. Heekuck Oh, all rights reserved
 * 이 프로그램은 한양대학교 ERICA 소프트웨어학부 재학생을 위한 교육용으로 제작되었습니다.
 */
#include <stdio.h>
#include <string.h>
#include "aes.h"

static const uint8_t sbox[256] = {
  0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
  0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
  0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
  0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
  0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
  0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
  0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
  0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
  0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
  0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
  0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
  0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
  0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
  0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
  0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
  0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };

static const uint8_t isbox[256] = {
  0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
  0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
  0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
  0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
  0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
  0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
  0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
  0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
  0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
  0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
  0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
  0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
  0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
  0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
  0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
  0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };

static const uint8_t Rcon[11] = {0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36};

static const uint8_t M[16] = {2, 3, 1, 1, 1, 2, 3, 1, 1, 1, 2, 3, 3, 1, 1, 2};

static const uint8_t IM[16] = {0x0e, 0x0b, 0x0d, 0x09, 0x09, 0x0e, 0x0b, 0x0d, 0x0d, 0x09, 0x0e, 0x0b, 0x0b, 0x0d, 0x09, 0x0e};

static uint32_t RotWord(uint32_t word){
  uint32_t rw = word & 0xFF;
  word = (word >> 8) ^ (rw << 24);
  return word;
}
```

```
57  static uint32_t SubWord(uint32_t word){
58    uint32_t sw = 0;
59    uint8_t sb;
60    for(uint8_t i = 0; i < 4; i ++){
61      sb = sbox[(uint8_t)(word >> i*8) & 0xFF];
62      sw ^= (uint32_t)sb << i*8;
63    }
64    return sw;
65  }
66
67
68  /*
69   * Generate an AES key schedule
70   */
71  void KeyExpansion(const uint8_t *key, uint32_t *roundKey)
72  {
73    uint32_t temp;
74    uint8_t i;
75    for(i = 0; i < Nk; i++){
76      roundKey[i] = *((uint32_t*)&key[i*4]);
77    }
78
79    for(i = Nk; i < RNDKEYSIZE; i++){
80      temp = roundKey[i-1];
81      if(i % Nk == 0){
82        temp = SubWord(RotWord(temp)) ^ Rcon[i/Nk];
83      }
84      else if((Nk > 6) && (i % Nk == 4)) {
85        temp = SubWord(temp);
86      }
87      roundKey[i] = roundKey[i-Nk] ^ temp;
88    }
89  }
90
91  // 라운드 키를 XOR 연산을 사용하여 state에 더한다.
92  static void AddRoundKey(uint8_t *state, const uint32_t *roundKey){
93    uint32_t *p;
94    p = (uint32_t *)state;
95    for(uint8_t i = 0; i < Nb; i++){
96      p[i] ^= roundKey[i];
97    }
98  }
99
100 // mode에 따라 순방향 또는 역방향으로 바이트를 치환한다.
101 static void SubBytes(uint8_t *state, int mode){
102   for(uint8_t i = 0; i < BLOCKLEN; i++){
103     if(mode != 0){
104       *(state+i) = sbox[*(state+i)];
105     }
106     else{
107       *(state+i) = isbox[*(state+i)];
108     }
109   }
110 }
111
```

```
112 // mode에 따라 순방향 또는 역방향으로 바이트의 위치를 변경한다.
113 static void ShiftRows(uint8_t *state, int mode){
114   uint8_t temp;
115     if(mode != 0){
116     // row1
117     temp        = *(state+1);
118     *(state+1)  = *(state+5);
119     *(state+5)  = *(state+9);
120     *(state+9)  = *(state+13);
121     *(state+13) = temp;
122     // row2
123     temp        = *(state+2);
124     *(state+2)  = *(state+10);
125     *(state+10) = temp;
126     temp        = *(state+6);
127     *(state+6)  = *(state+14);
128     *(state+14) = temp;
129     // row3
130     temp        = *(state+15);
131     *(state+15) = *(state+11);
132     *(state+11) = *(state+7);
133     *(state+7)  = *(state+3);
134     *(state+3)  = temp;
135     }
136     else{
137     // row1
138     temp        = *(state+13);
139     *(state+13) = *(state+9);
140     *(state+9)  = *(state+5);
141     *(state+5)  = *(state+1);
142     *(state+1)  = temp;
143     // row2
144     temp        = *(state+14);
145     *(state+14) = *(state+6);
146     *(state+6)  = temp;
147     temp        = *(state+10);
148     *(state+10) = *(state+2);
149     *(state+2)  = temp;
150     // row3
151     temp        = *(state+3);
152     *(state+3)  = *(state+7);
153     *(state+7)  = *(state+11);
154     *(state+11) = *(state+15);
155     *(state+15) = temp;
156     }
157 }
```

```
159 static void MixColumns(uint8_t *state, int mode)
160 {
161   uint8_t m, s, r;
162   uint8_t temp[16];
163   for(uint8_t i = 0; i < BLOCKLEN; i ++){
164     temp[i] = *(state+i);
165   }
166   for(uint8_t i = 0; i < Nb; i ++){
167     for(uint8_t j = 0; j < Nb; j++){
168       r = 0;
169       for(uint8_t k = 0; k < Nb; k++){
170         s = temp[4*i+k];
171         if(mode != 0) m = M[4 * j + k];
172         else m = IM[4 * j + k];
173         while(m > 0){
174           if(m & 1) r ^= s;
175           m = m >> 1;
176           s = XTIME(s);
177         }
178       }
179       *(state+(4*i+j)) = r;
180     }
181   }
182 }
```

```
/*
 * AES cipher function
 * If mode is nonzero, then do encryption, otherwise do decryption.
 */
void Cipher(uint8_t *state, const uint32_t *roundKey, int mode)
{
  uint32_t rk[4];

  // first round
  if(mode != 0) AddRoundKey(state, roundKey);
  else{
    for(uint8_t j = 0; j < Nk; j++){
      rk[j] = roundKey[4*Nr+j];
    }
    AddRoundKey(state,rk);
  }

  // 9 rounds
  for (uint8_t i = 1; i < Nr; ++i) {
    SubBytes(state, mode);
    ShiftRows(state, mode);
    if(mode != 0){
      for(uint8_t j = 0; j < Nk; j++){
        rk[j] = roundKey[4*i+j];
      }
    }
    else{
      for(uint8_t j = 0; j < Nk; j++){
        rk[j] = roundKey[4*(Nr-i)+j];
      }
      MixColumns((uint8_t*)rk,mode);
    }
    MixColumns(state, mode);
    AddRoundKey(state,rk);
  }

  // last round
  SubBytes(state, mode);
  ShiftRows(state, mode);
  if(mode != 0){
    for(uint8_t j = 0; j < Nk; j++){
      rk[j] = roundKey[4*Nr+j];
    }
    AddRoundKey(state,rk);
  }
  else{
    AddRoundKey(state,roundKey);
  }
}
```

2. 코드 내 함수 설명

- RotWord(uint32_t word) : KeyExpansion 에 필요한 함수로 왼쪽으로 한 byte 씩 로테이션 한다. 변수 rw 는 word 의 마지막 바이트를 저장하여 word 를 한 바이트씩 옮기고 rw 를 제일 앞에 넣어준다.

- SubWord(uint32_t word) : KeyExpansion 에 필요한 함수로 word 바이트 값을 다른 바이트로 바꿔준다. 한 바이트 씩 이동시키면서 Word 바이트 값을 sbox 에 넣어 새로운 값(sw)을 얻어낸다.

- KeyExpansion(const uint8_t *key, uint32_t *roundKey) : 처음 key 값은 그대로 사용해 준다. 첫번째 라운드부터 keyexpansion 에 의해 만들어는 라운드키를 사용하기 때문에 16byte 마다 스케줄에 의해 새로운 키 값을 만들어 낸다.
- AddRoundKey (uint8_t *state, const uint32_t *roundKey): 라운드 키를 XOR 연산을 사용하여 state 에 더하는 함수로 state 와 roundkey 를 입력받아 각 바이트씩 xor 연산 해 준다.
- SubBytes(uint8_t *state, int mode) : SubWord 함수와 마찬가지로 바이트를 치환하는데 모드에 따라 순방향, 역방향으로 치환한다. 모드가 달라도 치환하는 table 만 달라지지 동작원리는 바뀌지 않는다.
- ShiftRows(uint8_t *state, int mode) : 바이트의 위치를 변경하는 함수이다. 이것도 마찬가지로 mode 를 입력받는데 역방향은 순방향의 반대로 돌리면 된다.
    - 1 번째 row 는 왼쪽으로 1 바이트씩 위치를 변경
    - 2 번째 row 는 왼쪽으로 2 바이트씩 위치를 변경
    - 3 번째 row 는 왼쪽으로 3 바이트씩 위치를 변경
- MixColumns(uint8_t *state, int mode) : 기약다항식 x^8 + x^4 + x^3 + x + 1 을 사용한 GF(2^8)에서 행렬 곱셈을 수행하는 함수이다. Mode 에 따라 encrypt 일때는 matrix in GF(2^8)를 사용하고 decrypt 일때는

inverse matrix in GF(2^8)을 사용한다. SubByte 함수와 마찬가지로 복호화할 때 행렬만 바뀔뿐 동작원리는 달라지지 않기때문에 Temp 에 state 값을 임시 저장해 두고 mode 에 따라 행렬을 선택한다. 그 후 a*b mod m(x)를 해주고 그 값들을 xor 하여 행렬 곱셈을 완성한다.

- Cipher(uint8_t *state, const uint32_t *roundKey, int mode): 복호화 할 때 equivalent inverse cipher 를 사용하여 복호화 순서를 바꾸어 암호화 과정과 같게 구현하였다. AES 암호화 과정은 먼저 plaintext 와 key 를 AddRoundKey 해 주고 SubBytes – ShiftRows – MixColumns - AddRoundKey 순으로 9 라운드를 진행하고 마지막 라운드에서는 SubBytes – ShiftRows – AddRoundKey 순으로 진행하게 된다. 매 라운드에서는 KeyExpansion 에서 생성된 roundkey 를 사용한다. equivalent inverse cipher 는 암호화 과정과 같지만 roundkey 를 거꾸로 사용하며 매 라운드마다 roundkey 를 MixColumns 해 state 를 MixColumns 한 것과 AddRoundKey 하게된다.

3. 실행 결과

```
yerim ~/Downloads/2020 암호학/프로젝트 #1
❯ make
gcc -Wall -o test test.o aes_skeleton.o
yerim ~/Downloads/2020 암호학/프로젝트 #1
❯ make
gcc -Wall -c test.c
gcc -Wall -c aes.c
gcc -Wall -o test test.o aes.o
yerim ~/Downloads/2020 암호학/프로젝트 #1
❯ ./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
yerim ~/Downloads/2020 암호학/프로젝트 #1
❯
```