
ID1 = 104 337 378 (Jason Choquette)

ID2 = 103 385 550 (Yu Sheng Tian)

Group Code (J, Y) = (3,7)

Assigned Plaintext and Key:

00 00 00 00 00 00 00 00 00 00 00 00 00 00 ab d1 (plaintext)

1a 0c 24 f2 87 54 93 bc b7 08 0e 43 93 0f 56 81 (key)

The program is written in C for operating system Windows (Microsoft).

Key Schedule Results for Each Round with the modified AES:

Round: 0:

Key: 1a 0c 24 f2 87 54 93 bc b7 08 0e 43 93 0f 56 81

Round: 1:

Key: 6d bd 28 2e ea e9 bb 92 5d e1 b5 d1 ce ee e3 50

Round: 2:

Key: 47 ac 7b a5 ad 45 c0 37 f0 a4 75 e6 3e 4a 96 b6

Round: 3:

Key: 95 3c 35 17 38 79 f5 20 c8 dd 80 c6 f6 97 16 70

Round: 4:

Key: 15 7b 64 55 2d 02 91 75 e5 df 11 b3 13 48 07 c3

Round: 5:

Key: 57 be 4a 28 7a bc db 5d 9f 63 ca ee 8c 2b cd 2d

Round: 6:

Key: 86 03 92 4c fc bf 49 11 63 dc 83 ff ef f7 4e d2

Round: 7:

Key: ae 2c 27 93 52 93 6e 82 31 4f ed 7d de b8 a3 af

Round: 8:

Key: 42 26 5e 8e 10 b5 30 0c 21 fa dd 71 ff 42 7e de

Round: 9:

Key: 75 d5 43 98 65 60 73 94 44 9a ae e5 bb d8 d0 3b

Round: 10:

Key: 22 a5 a1 72 47 c5 d2 e6 03 5f 7c 03 b8 87 ac 38

Data Results for Each Round with the modified AES:

Round 0:

-----Start: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ab d1

----Output: 1a 0c 24 f2 87 54 93 bc b7 08 0e 43 93 0f fd 50

Round 1:

----Output: aa ea 12 fe 49 eb b4 c6 dd 9b a9 bb b2 b0 4c 39

Round 2:

----Output: 41 11 42 07 75 96 6b 2a 26 44 1d e3 a5 0e 02 2a

Round 3:

----Output: 62 96 41 6c 90 85 c2 7c 12 f6 b0 92 09 1e 5f a5

Round 4:

----Output: 19 d0 86 cc b4 fc c6 f8 92 87 91 cf d0 52 0d eb

Round 5:

----Output: 47 60 17 b7 de 36 29 87 04 a1 ce f9 c2 f6 3b 76

Round 6:

----Output: 1b 50 08 0e 18 0c c4 6c bd b6 f1 6e f2 04 5a e1

Round 7:

----Output: ab 64 3c cd d4 d0 0c d3 a8 ca f5 91 a1 2c fb 45

Round 8:

----Output: 9e fb 29 62 ae b9 ce ce 5b 8c e4 81 21 6e 70 2c

Round 9:

----Output: 81 b8 5f 58 b4 ea 1e 8c a2 f9 df 99 cc b1 3b bb

Round 10:

----Output: 2e 22 3f 98 ca 5c f3 8c 39 97 b3 67 f3 eb de d6

```
1  #define DEMO 1
2
3  void AesEncrypt(unsigned char *blk, unsigned char *key, int Nr);
4  static void AddRoundKey(unsigned char *col, unsigned char *key, int round);
5  static void SubBytes(unsigned char *col);
6  static void ShiftRows(unsigned char *col);
7  static void MixColumns(unsigned char *col);
8  static unsigned char xtime(unsigned char x);
9  unsigned char gmultiply(unsigned char a, unsigned char b);
10 unsigned char gmul_inverse(unsigned char in);
11 void rotate(unsigned char * in);
12 unsigned char rcon(unsigned char in);
13 unsigned char csbox(unsigned char in);
14 void schedule_core(unsigned char * in, unsigned char i);
15 void print_expand_key(unsigned char * in);
16
```

```c
#include <stdio.h>
#include "prototypes.h"


int main(void)
{

#if DEMO
    /* Data from project demo for testing*/
    unsigned char input[16] = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
      0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff };
    unsigned char key[256]  = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };
    char * sboxType = "original";
#else

    unsigned char input[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xab, 0xd1 };
    unsigned char key[256]  = { 0x1a, 0x0c, 0x24, 0xf2, 0x87, 0x54, 0x93, 0xbc,
      0xb7, 0x08, 0x0e, 0x43, 0x93, 0x0f, 0x56, 0x81 };
    char * sboxType = "modified";
#endif

    printf("\n\n-------------------------------------------------------------\n\n");
    printf("ID1 = 104 337 378  (Jason Choquette) \n");
    printf("ID2 = 103 385 550  (Yu Sheng Tian) \n");
    printf("Group Code (J, Y)= (3,7) \n\n\n");
    printf("Assigned Plaintext and Key:\n");
    printf("\t");
    for (int i = 0; i < 16; i++)
        printf("%02x ", input[i]);

    printf("\n\t");
    for (int i = 0; i < 16; i++)
        printf("%02x ", key[i]);

    printf("\n\n\n");
    printf("-------------------------------------------------------------\n");
    printf("Key Schedule Results for Each Round with the %s AES:\n", sboxType);
    printf("-------------------------------------------------------------\n");

    print_expand_key(key);

    printf("\n\n");
    printf("-------------------------------------------------------------\n");
    printf("Data Results for Each Round with the %s AES:\n", sboxType);
    printf("-------------------------------------------------------------\n");

    AesEncrypt(input, key, 10);

    getchar();
    return 0;
```

```
49  }
50
51
```

```
 1
 2
 3  /* Log table using 0xe5 (229) as the generator */
 4  unsigned char ltable[256] = {
 5      0x00, 0xff, 0xc8, 0x08, 0x91, 0x10, 0xd0, 0x36,
 6      0x5a, 0x3e, 0xd8, 0x43, 0x99, 0x77, 0xfe, 0x18,
 7      0x23, 0x20, 0x07, 0x70, 0xa1, 0x6c, 0x0c, 0x7f,
 8      0x62, 0x8b, 0x40, 0x46, 0xc7, 0x4b, 0xe0, 0x0e,
 9      0xeb, 0x16, 0xe8, 0xad, 0xcf, 0xcd, 0x39, 0x53,
10      0x6a, 0x27, 0x35, 0x93, 0xd4, 0x4e, 0x48, 0xc3,
11      0x2b, 0x79, 0x54, 0x28, 0x09, 0x78, 0x0f, 0x21,
12      0x90, 0x87, 0x14, 0x2a, 0xa9, 0x9c, 0xd6, 0x74,
13      0xb4, 0x7c, 0xde, 0xed, 0xb1, 0x86, 0x76, 0xa4,
14      0x98, 0xe2, 0x96, 0x8f, 0x02, 0x32, 0x1c, 0xc1,
15      0x33, 0xee, 0xef, 0x81, 0xfd, 0x30, 0x5c, 0x13,
16      0x9d, 0x29, 0x17, 0xc4, 0x11, 0x44, 0x8c, 0x80,
17      0xf3, 0x73, 0x42, 0x1e, 0x1d, 0xb5, 0xf0, 0x12,
18      0xd1, 0x5b, 0x41, 0xa2, 0xd7, 0x2c, 0xe9, 0xd5,
19      0x59, 0xcb, 0x50, 0xa8, 0xdc, 0xfc, 0xf2, 0x56,
20      0x72, 0xa6, 0x65, 0x2f, 0x9f, 0x9b, 0x3d, 0xba,
21      0x7d, 0xc2, 0x45, 0x82, 0xa7, 0x57, 0xb6, 0xa3,
22      0x7a, 0x75, 0x4f, 0xae, 0x3f, 0x37, 0x6d, 0x47,
23      0x61, 0xbe, 0xab, 0xd3, 0x5f, 0xb0, 0x58, 0xaf,
24      0xca, 0x5e, 0xfa, 0x85, 0xe4, 0x4d, 0x8a, 0x05,
25      0xfb, 0x60, 0xb7, 0x7b, 0xb8, 0x26, 0x4a, 0x67,
26      0xc6, 0x1a, 0xf8, 0x69, 0x25, 0xb3, 0xdb, 0xbd,
27      0x66, 0xdd, 0xf1, 0xd2, 0xdf, 0x03, 0x8d, 0x34,
28      0xd9, 0x92, 0x0d, 0x63, 0x55, 0xaa, 0x49, 0xec,
29      0xbc, 0x95, 0x3c, 0x84, 0x0b, 0xf5, 0xe6, 0xe7,
30      0xe5, 0xac, 0x7e, 0x6e, 0xb9, 0xf9, 0xda, 0x8e,
31      0x9a, 0xc9, 0x24, 0xe1, 0x0a, 0x15, 0x6b, 0x3a,
32      0xa0, 0x51, 0xf4, 0xea, 0xb2, 0x97, 0x9e, 0x5d,
33      0x22, 0x88, 0x94, 0xce, 0x19, 0x01, 0x71, 0x4c,
34      0xa5, 0xe3, 0xc5, 0x31, 0xbb, 0xcc, 0x1f, 0x2d,
35      0x3b, 0x52, 0x6f, 0xf6, 0x2e, 0x89, 0xf7, 0xc0,
36      0x68, 0x1b, 0x64, 0x04, 0x06, 0xbf, 0x83, 0x38 };
37
38
39  /* Anti-log table: */
40  unsigned char atable[256] = {
41      0x01, 0xe5, 0x4c, 0xb5, 0xfb, 0x9f, 0xfc, 0x12,
42      0x03, 0x34, 0xd4, 0xc4, 0x16, 0xba, 0x1f, 0x36,
43      0x05, 0x5c, 0x67, 0x57, 0x3a, 0xd5, 0x21, 0x5a,
44      0x0f, 0xe4, 0xa9, 0xf9, 0x4e, 0x64, 0x63, 0xee,
45      0x11, 0x37, 0xe0, 0x10, 0xd2, 0xac, 0xa5, 0x29,
46      0x33, 0x59, 0x3b, 0x30, 0x6d, 0xef, 0xf4, 0x7b,
47      0x55, 0xeb, 0x4d, 0x50, 0xb7, 0x2a, 0x07, 0x8d,
48      0xff, 0x26, 0xd7, 0xf0, 0xc2, 0x7e, 0x09, 0x8c,
49      0x1a, 0x6a, 0x62, 0x0b, 0x5d, 0x82, 0x1b, 0x8f,
50      0x2e, 0xbe, 0xa6, 0x1d, 0xe7, 0x9d, 0x2d, 0x8a,
51      0x72, 0xd9, 0xf1, 0x27, 0x32, 0xbc, 0x77, 0x85,
52      0x96, 0x70, 0x08, 0x69, 0x56, 0xdf, 0x99, 0x94,
```

```
53      0xa1, 0x90, 0x18, 0xbb, 0xfa, 0x7a, 0xb0, 0xa7,
54      0xf8, 0xab, 0x28, 0xd6, 0x15, 0x8e, 0xcb, 0xf2,
55      0x13, 0xe6, 0x78, 0x61, 0x3f, 0x89, 0x46, 0x0d,
56      0x35, 0x31, 0x88, 0xa3, 0x41, 0x80, 0xca, 0x17,
57      0x5f, 0x53, 0x83, 0xfe, 0xc3, 0x9b, 0x45, 0x39,
58      0xe1, 0xf5, 0x9e, 0x19, 0x5e, 0xb6, 0xcf, 0x4b,
59      0x38, 0x04, 0xb9, 0x2b, 0xe2, 0xc1, 0x4a, 0xdd,
60      0x48, 0x0c, 0xd0, 0x7d, 0x3d, 0x58, 0xde, 0x7c,
61      0xd8, 0x14, 0x6b, 0x87, 0x47, 0xe8, 0x79, 0x84,
62      0x73, 0x3c, 0xbd, 0x92, 0xc9, 0x23, 0x8b, 0x97,
63      0x95, 0x44, 0xdc, 0xad, 0x40, 0x65, 0x86, 0xa2,
64      0xa4, 0xcc, 0x7f, 0xec, 0xc0, 0xaf, 0x91, 0xfd,
65      0xf7, 0x4f, 0x81, 0x2f, 0x5b, 0xea, 0xa8, 0x1c,
66      0x02, 0xd1, 0x98, 0x71, 0xed, 0x25, 0xe3, 0x24,
67      0x06, 0x68, 0xb3, 0x93, 0x2c, 0x6f, 0x3e, 0x6c,
68      0x0a, 0xb8, 0xce, 0xae, 0x74, 0xb1, 0x42, 0xb4,
69      0x1e, 0xd3, 0x49, 0xe9, 0x9c, 0xc8, 0xc6, 0xc7,
70      0x22, 0x6e, 0xdb, 0x20, 0xbf, 0x43, 0x51, 0x52,
71      0x66, 0xb2, 0x76, 0x60, 0xda, 0xc5, 0xf3, 0xf6,
72      0xaa, 0xcd, 0x9a, 0xa0, 0x75, 0x54, 0x0e, 0x01 };
73
74
75  unsigned char gmultiply(unsigned char a, unsigned char b)
76  {
77      unsigned char product = 0;
78      unsigned char counter;
79      unsigned char hi_bit_set;
80
81      for (counter = 0; counter < 8; counter++)
82      {
83          if ((b & 1) == 1)
84              product ^= a;
85
86          hi_bit_set = (a & 0x80);
87          a <<= 1;
88
89          if (hi_bit_set == 0x80)
90              a ^= 0x1b;
91
92          b >>= 1;
93      }
94
95      return product;
96  }
97
98  unsigned char gmul_inverse(unsigned char in) {
99      /* 0 is self inverting */
100     if (in == 0)
101         return 0;
102     else
103         return atable[(255 - ltable[in])];
104 }
```

```
105
106
107
108 void rotate(unsigned char * in)
109 {
110     unsigned char a = in[0];
111     unsigned char c;
112
113     for (c = 0; c < 3; c++)
114         in[c] = in[c + 1];
115
116     in[3] = a;
117     return;
118 }
119
120
121 unsigned char rcon(unsigned char in)
122 {
123     unsigned char c = 1;
124     if (in == 0) return 0;
125
126     while (in != 1)
127     {
128         c = gmultiply(c, 2);
129         in--;
130     }
131
132     return c;
133 }
134
135 /* Calculate the s-box for a given number */
136 unsigned char csbox(unsigned char in)
137 {
138     unsigned char c, s, x;
139     s = x = gmul_inverse(in);
140
141     for (c = 0; c < 4; c++)
142     {
143         /* One bit circular rotate to the left */
144         s = (s << 1) | (s >> 7);
145         /* xor with x */
146         x ^= s;
147     }
148
149     x ^= 99; /* 0x63 */
150     return x;
151 }
152
153 /* This is the core key expansion, which, given a 4-byte value, does some
      scrambling. */
154 void schedule_core(unsigned char * in, unsigned char i)
155 {
```

```
156     char a;
157
158     /* Rotate the input 8 bits to the left */
159     rotate(in);
160
161     /* Apply Rijndael's s-box on all 4 bytes. */
162     for (a = 0; a < 4; a++)
163         in[a] = csbox(in[a]);
164
165     /* On just the first byte, add 2^i to the byte */
166     in[0] ^= rcon(i);
167 }
168
169
170 void print_expand_key(unsigned char * in)
171 {
172     unsigned char t[4];
173
174     /* let c = 16, since first sub-key is user-supplied key */
175     unsigned char c = 16;
176     unsigned char i = 1;
177     unsigned char a;
178     int round = 1;
179
180     /* Since we have 11 rounds (Round 0 up to round 10 for 16-byte key), we need ↵
          11 sets of
181     16 bytes each, for 128-bit mode (AES). */
182     while (c < 176)
183     {
184         /* Copy the temp variable over from the last 4-byte block */
185         for (a = 0; a < 4; a++)
186             t[a] = in[a + c - 4];
187
188         if (c > 16)
189             for (a = 0; a < 4; a++)
190                 printf("%02x ", t[a]);
191
192         /* Every four blocks (4-bytes) do a complex calculation */
193         if (c % 16 == 0)
194         {
195             schedule_core(t, i);
196             i++;
197             printf("\nRound: %d:\n     Key: ", round++);
198         }
199
200         for (a = 0; a < 4; a++)
201         {
202             in[c] = in[c - 16] ^ t[a];
203             c++;
204             if (c > 172) printf("%02x ", in[c - 1]);
205         }
206     }
```

```
207  }
208
```

```c
#include "prototypes.h"



/* Encrypt a single block with 10 rounds */
void AesEncrypt(unsigned char *blk, unsigned char *key, int Nr)
{
    printf("Round 0:\n");
    printf("-----Start:  ");
    for (int i = 0; i < 16; i++)
        printf("%02x ", blk[i]);

    printf("\n");
    printf("----Output:  ");
    AddRoundKey(blk, key, 0);

    for (int i = 0; i < 16; i++)
        printf("%02x ", blk[i]);

    for (int x = 1; x <= (Nr - 1); x++)
    {
        SubBytes(blk);
        ShiftRows(blk);
        MixColumns(blk);
        AddRoundKey(blk, key, x);
        printf("\nRound %d:\n", x);
        printf("----Output:  ");
        for (int i = 0; i < 16; i++)
            printf("%02x ", blk[i]);

    }

    printf("\nRound 10:\n");
    SubBytes(blk);
    ShiftRows(blk);
    AddRoundKey(blk, key, Nr);
    printf("----Output:  ");
    for (int i = 0; i < 16; i++)
        printf("%02x ", blk[i]);
}




/* The AES Substitution Table */
static const unsigned char 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,
```

```
51      0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, ↵
            0xD8, 0x31, 0x15,
52      0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, ↵
            0x27, 0xB2, 0x75,
53      0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, ↵
            0xE3, 0x2F, 0x84,
54      0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, ↵
            0x4C, 0x58, 0xCF,
55      0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, ↵
            0x3C, 0x9F, 0xA8,
56      0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, ↵
            0xFF, 0xF3, 0xD2,
57      0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, ↵
            0x5D, 0x19, 0x73,
58      0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, ↵
            0x5E, 0x0B, 0xDB,
59      0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, ↵
            0x95, 0xE4, 0x79,
60      0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, ↵
            0x7A, 0xAE, 0x08,
61      0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, ↵
            0xBD, 0x8B, 0x8A,
62      0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, ↵
            0xC1, 0x1D, 0x9E,
63      0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, ↵
            0x55, 0x28, 0xDF,
64      0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, ↵
            0x54, 0xbb, 0x16 };
65
66
67  static const unsigned char modifiedsbox[256] = {
68      0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, ↵
            0xd7, 0xab, 0x76,    //  0
69      0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, ↵
            0xA4, 0x72, 0xC0,    //  1
70      0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, ↵
            0xD8, 0x31, 0x15,    //  2
71      0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, ↵
            0xFF, 0xF3, 0xD2,    //  3 ----------------- swap with 7
72      0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, ↵
            0xE3, 0x2F, 0x84,    //  4
73      0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, ↵
            0x4C, 0x58, 0xCF,    //  5
74      0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, ↵
            0x3C, 0x9F, 0xA8,    //  6
75      0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, ↵
            0x27, 0xB2, 0x75,    //  7 ----------------- swap with 3
76      0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, ↵
            0x5D, 0x19, 0x73,    //  8
77      0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, ↵
            0x5E, 0x0B, 0xDB,    //  9
78      0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, ↵
```

```
          0x95, 0xE4, 0x79,    // 10
 79     0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, ↵
          0x7A, 0xAE, 0x08,    // 11
 80     0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, ↵
          0xBD, 0x8B, 0x8A,    // 12
 81     0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, ↵
          0xC1, 0x1D, 0x9E,    // 13
 82     0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, ↵
          0x55, 0x28, 0xDF,    // 14
 83     0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, ↵
          0x54, 0xbb, 0x16 };  // 15
 84
 85
 86 /* The key schedule rcon table */
 87 static const unsigned char Rcon[10] =
 88 { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36 };
 89
 90 /* The *x function */
 91 static unsigned char xtime(unsigned char x)
 92 {
 93     if (x & 0x80) { return ((x << 1) ^ 0x1B) & 0xFF; }
 94     return x << 1;
 95 }
 96
 97
 98
 99 /* MixColumns: Processes the entire block */
100 static void MixColumns(unsigned char *col)
101 {
102     unsigned char tmp[4], xt[4];
103
104     for (int x = 0; x < 4; x++, col += 4)
105     {
106         xt[0]  = xtime(col[0]);
107         xt[1]  = xtime(col[1]);
108         xt[2]  = xtime(col[2]);
109         xt[3]  = xtime(col[3]);
110         tmp[0] = xt[0]  ^ xt[1]  ^ col[1] ^ col[2] ^ col[3];
111         tmp[1] = col[0] ^ xt[1]  ^ xt[2]  ^ col[2] ^ col[3];
112         tmp[2] = col[0] ^ col[1] ^ xt[2]  ^ xt[3]  ^ col[3];
113         tmp[3] = xt[0]  ^ col[0] ^ col[1] ^ col[2] ^ xt[3];
114         col[0] = tmp[0];
115         col[1] = tmp[1];
116         col[2] = tmp[2];
117         col[3] = tmp[3];
118     }
119 }
120
121
122
123
124 /* ShiftRows: Shifts the entire block */
```

```
125 static void ShiftRows(unsigned char *col)
126 {
127     unsigned char t;
128 
129     /* 2nd row */
130     t       = col[1];
131     col[1]  = col[5];
132     col[5]  = col[9];
133     col[9]  = col[13];
134     col[13] = t;
135 
136     /* 3rd row */
137     t       = col[2];
138     col[2]  = col[10];
139     col[10] = t;
140     t       = col[6];
141     col[6]  = col[14];
142     col[14] = t;
143 
144     /* 4th row */
145     t        = col[15];
146     col[15]  = col[11];
147     col[11]  = col[7];
148     col[7]   = col[3];
149     col[3]   = t;
150 }
151 
152 
153 
154 /* SubBytes */
155 static void SubBytes(unsigned char *col)
156 {
157 #if DEMO
158     for (int x = 0; x < 16; x++)
159         col[x] = sbox[col[x]];
160 #else
161     for (int x = 0; x < 16; x++)
162         col[x] = modifiedsbox[col[x]];
163 #endif // DEMO
164 }
165 
166 
167 
168 /* AddRoundKey */
169 static void AddRoundKey(unsigned char *col, unsigned char *key, int round)
170 {
171     for (int x = 0; x < 16; x++)
172         col[x] ^= key[(round << 4) + x];
173 }
```