C Programming, Autumn 2013, Exercises for the Fourth Week

1. Consider a two-dimensional 4×4 array whose elements are of the type `uint8_t`. We can use the following definition

   ```
   typedef uint8_t state [4][4];
   ```

   Write a function

   ```
    state* read_block(FILE* fptr)
   ```

   which reads a plain text block of 128 bits or 16 bytes from a text file and stores it into an array whose type is *state*. The first byte should go into state[0][0], the next into state[0][1], etc. The function returns a pointer to that array. Hint: use fread.

2. Write two functions

   ```
   void shift_row(state* arr);
   void inv_shift_row(state* arr);
   ```

   The first function takes a pointer to an array of *state* type and transforms the array in the following way. The first row of the array is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following example shows what happens:

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

   $\longrightarrow$

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

   The second function performs the circular shifts in the opposite direction for each of the last three rows.

3. Let us define

   ```
    typedef struct {
       uint8_t i;
       uint8_t j;
       } intPair;
   ```

   Write a function

   ```
    intPair construct_indices(uint8_t byte);
   ```

that returns two integers as follows. The four leftmost bits of *byte* form the integer i and the four rightmost bits of *byte* form the integer j. The function returns i and j in the structure of type `intPair`.

4. S-Box is an 16×16 array whose elements are `uint8_t` integers. S-Box is given as an attachment where elements have been written in hexadecimal notation. Make a function

   ```
   Sbox* read_sbox(FILE* fptr);
   ```

   that reads S-Box from a file and stores the values into a Sbox array. The function returns a pointer to that array. The *Sbox* is defined as

   ```
   typedef uint8_t sbox [16][16];
   ```

5. Write a function

   ```
   void sub_bytes(state* st, Sbox* box);
   ```

   that transforms the state array *st* as follows. For every element (byte) of *st* (for example `(*st)[m][n]`), use first the function *constructIndices* to construct two integers i and j from the element. These i and j serve as indices into the S-box to select a unique 8-bit value which replaces the original value of the element. (i.e. `(*st)[m][n] = (*box)[i][j]`);

6. Consider a byte b. It can be regarded as a polynomial with modulo 2 integer coefficients (i.e. 0 or 1). Assume that the least significant bit in the byte is on the right. (This depends on the machine architecture, but is the most common.) Thus for example 01100111 describes the polynomial $x^6 + x^5 + x^2 + x + 1$. The sum of two this kind of polynomials is simply got by making the xor operation between the bytes. The multiplication of a byte with x modulo $f(x) = x^8 + x^4 + x^3 + x + 1$ is done as follows. Make first one step shift for the byte to the left. If the leftmost bit of the byte was 1, it drops out from the left. Then it is necessary to make a xor operation between the byte and 00011011. Otherwise the mere shift is enough.

   Example. Consider the byte 01100111. It represents the polynomial $x^6 + x^5 + x^2 + x + 1$. When this is multiplied with x^2, we must divide the task into two parts. Multiply first with x. Thus make a shift to the left and the result is 11001110. Then this must be multiplied still with x. First the shift and the result is 10011100. Now one 1 has dropped out and we must make the xor operation with 00011011. The result is 10000111 which represents the polynomial $x^7 + x^2 + x + 1$. This polynomial is the result of the multiplication $x^2 \times (x^6 + x^5 + x^2 + x + 1)$ modulo $f(x)$.

   (The modulo operation is needed for two reasons. First, the result must consist at most of 8 bits. Secondly, now the bytes form a finite field with 2^8 elements. See some good text book on algebra.)

Make a function `uint8_t mul1(uint8_t byte, int n)` which multiplies (modulo $f(x)$) the polynomial represented by *byte* with x^n, $0 < n < 8$.

7. Make a function `uint8_t mul2(uint8_t byte1, uint8_t byte2)` which multiplies the polynomial *byte1* with polynomial *byte2* modulo $f(x)$.

8. Consider the Mix_columns operation in Rijndael or AES. A state

$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

is multiplied with a constant matrix

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Remember the matrix multiplication:

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}.$$

Thus the element $s'_{0,0}$ in the resulting matrix is calculated by the formula

$$s'_{0,0} = (2 \bullet s_{0,0}) \oplus (3 \bullet s_{1,0}) \oplus s_{2,0} \oplus s_{3,0},$$

where the bytes are intepreted as polynomials and $\oplus$ is the normal addition and $\bullet$ is the multiplication modulo the previous $f(x)$.

Make a function `state* mix\_columns(state* st)` which multiplies the state matrix with the above constant matrix. The result state matrix is returned.

9. Write the function

```
void inv_mix_columns(state* st);
```

that does the same as mix_columns, but this time using the constant matrix (in hexadecimal notation)

$$\begin{matrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{matrix}$$

10. Make a function `state* add_round_key(state* text, state* key)` which makes the xor operation between *text* and *key*. That is, every byte in *text* is xored with the corresponding byte in *key*.

11. Now you can perform one round AES encryption. To get points for this exercise it is good enough that encryption and decryption of the encrypted produce the original result. You do not have to implement anything new for this. However, you can use the following steps to perform AES yourself. Let the plaintext be already in a state array st. The encryption consists of the following steps:

```
sub_bytes(st, sbox);
shift_rows(st);
st = mix_columns(st);
st = add_round_key(st, key);
```

The decryption proceeds as follows:

```
st = add_round_key(st, key);
st = inv_mix_columns(st);
inv_shift_rows(st);
sub_bytes(st, invBox);
```

The sub_bytes in the decryption is the same as in the encryption, but this time it uses the inverse Sbox which is given below.

Make the encryption and decryption and check if you will get the original plaintext.

Inverse Sbox:

```
52 09 6a d5 30 36 a5 38 bf 40 a3 9e 81 f3 d7 fb
7c e3 39 82 9b 2f ff 87 34 8e 43 44 c4 de e9 cb
54 7b 94 32 a6 c2 23 3d ee 4c 95 0b 42 fa c3 4e
08 2e a1 66 28 d9 24 b2 76 5b a2 49 6d 8b d1 25
72 f8 f6 64 86 68 98 16 d4 a4 5c cc 5d 65 b6 92
6c 70 48 50 fd ed b9 da 5e 15 46 57 a7 8d 9d 84
90 d8 ab 00 8c bc d3 0a f7 e4 58 05 b8 b3 45 06
d0 2c 1e 8f ca 3f 0f 02 c1 af bd 03 01 13 8a 6b
3a 91 11 41 4f 67 dc ea 97 f2 cf ce f0 b4 e6 73
96 ac 74 22 e7 ad 35 85 e2 f9 37 e8 1c 75 df 6e
47 f1 1a 71 1d 29 c5 89 6f b7 62 0e aa 18 be 1b
fc 56 3e 4b c6 d2 79 20 9a db c0 fe 78 cd 5a f4
1f dd a8 33 88 07 c7 31 b1 12 10 59 27 80 ec 5f
60 51 7f a9 19 b5 4a 0d 2d e5 7a 9f 93 c9 9c ef
a0 e0 3b 4d ae 2a f5 b0 c8 eb bb 3c 83 53 99 61
17 2b 04 7e ba 77 d6 26 e1 69 14 63 55 21 0c 7d
```