CHAPTER 7

AES

(Solution to Odd-Numbered Problems)

Review Questions

1. The criteria defined by NIST for selecting AES fall into three areas: *security*, *cost*, and *implementation*.

3. The number of round keys and the number of transformation depend on the number of rounds. The following table list the information. Note that there is one transformation for pre-round. Also note that the last round uses only three transformations.

Version	*Number of rounds*	*Number of round keys*	*Number of Transformations*
AES-128	10	11	$1 + 9 \times 4 + 3 = 40$
AES-192	12	13	$1 + 11 \times 4 + 3 = 48$
AES-256	14	15	$1 + 13 \times 4 + 3 = 56$

5. Before and after each stage, the data block is referred to as a state. States, like blocks, are made of 16 bytes, but they are normally treated as matrices of 4×4 bytes. The following table shows the number of states used in each version. We have used two extra states: one before the pre-round and one after the last round. If you do not consider this, the number of states is reduced by two.

Version	*Number of rounds*	*Number of States*
AES-128	10	$\mathbf{(1)} + 1 + 9 \times 4 + 3 + \mathbf{(1)} = 42$
AES-192	12	$\mathbf{(1)} + 1 + 11 \times 4 + 3 + \mathbf{(1)} = 50$
AES-256	14	$\mathbf{(1)} + 1 + 13 \times 4 + 3 + \mathbf{(1)} = 58$

7. Substitution in DES is done by S-boxes. Each box substitutes a 6-bit value with a 4-bit value. We need eight S-boxes to create a 32-bit half block. Substitution in

AES is done through SubBytes transformation that transforms a whole state to another state. However, we can say that SubBytes actually substitutes 16 bytes with new 16 bytes.

9. In DES, the size of the block is 64 bits, but the size of the round key is 48 bits. In AES the size of the block and the round key are both 128 bits (for all versions).

Exercises

11. The disadvantage of using keyed S-boxes is that it makes the design of the cipher more difficult. In particular, it is more difficult to create S-boxes that are inverse of each other in the encryption and decryption cipher. The advantage of using keyed S-boxes is that a keyed S-box is normally non-linear, which protect the cipher against linear cryptanalysis.

13. Having different number of rounds has the advantage that new versions of cipher with more number of rounds can be used, without changing the structure of cipher, if the cipher is attacked by differential and linear cryptanalysis (or other attacks that depend on the number of rounds). For example, we can use AES with 10 rounds as long as it is not secured. If it is attacked, we can move to the version with 12 or 14 rounds without changing the structure of the cipher.

15. A cipher in which the size of the round is the same as the size of the round key is easier to design because we do not have to use expansion (or compression) permutation to match the size of the block to the size of the round key. This enable us to make the non-Feistel ciphers.

17. We use two plaintexts that differ only in the first bit:

$$\mathbf{P_1}: \quad (0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$$
$$\mathbf{P_2}: \quad (8000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$$

a. Applying the SubBytes transformation to P_1 and P_2, we get:

$$\mathbf{T_1}: \quad (6363\ 6363\ 6363\ 6363\ 6363\ 6363\ 6363\ 6363)_{16}$$
$$\mathbf{T_2}: \quad (CD63\ 6363\ 6363\ 6363\ 6363\ 6363\ 6363\ 6363)_{16}$$

T_2 and T_1 differ only in 5 bits.

b. Applying the ShiftRows transformation to P_1 and P_2, we get:

$$\mathbf{T_1}: \quad (0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$$
$$\mathbf{T_2}: \quad (8000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$$

T_2 and T_1 differ only in 1 bit.

Applying the MixColumn transformation to P_1 and P_2, we get:

T_1:	$(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$
T_2:	$(1B80\ 809B\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

T_2 and T_1 differ in 9 bits.

c. Applying the AddRoundKey of 128 0-bit transformation to P_1 and P_2, we get:

T_1:	$(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$
T_2:	$(8000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

T_2 and T_1 differ only in 1 bit. Note that we have used a round key with 128 0's, but the result is the same if we use any key.

19.

a. The SubBytes transformation is repeated in each round. So we have N_r of this transformation.

b. The ShiftRows transformation is repeated in each round. So we have N_r of this transformation.

c. The MixColumns transformation is repeated in each round except the last round. So we have $(N_r - 1)$ of this transformation.

d. The AddRoundKey transformation is repeated in each round. In addition, we have one of this transformation in the pre-round section. So we have $(N_r + 1)$ of this transformation.

e. The total number of transformation is

$$\text{Total number of transformation} = N_r + N_r + (N_r - 1) + (N_r + 1) = 4 \times N_r$$

21.

a. We can use (x^{11-1} mod prime) and (x^{12-1} mod prime), in which the prime is the the irreducible polynomial ($x^8 + x^4 + x^3 + x + 1$), to find the first terms of RCon[11] and RCon[12]. The following shows the constants for AES-192.

Round	*(RCon)*	*Round*	*(RCon)*	*Round*	*(RCon)*
1	(**01** 00 00 00)$_{16}$	5	(**10** 00 00 00)$_{16}$	9	(**1B** 00 00 00)$_{16}$
2	(**02** 00 00 00)$_{16}$	6	(**20** 00 00 00)$_{16}$	10	(**36** 00 00 00)$_{16}$
3	(**04** 00 00 00)$_{16}$	7	(**40** 00 00 00)$_{16}$	11	(**6C** 00 00 00)$_{16}$
4	(**08** 00 00 00)$_{16}$	8	(**80** 00 00 00)$_{16}$	12	(**D8** 00 00 00)$_{16}$

b. We can use (x^{13-1} mod prime) and (x^{14-1} mod prime), in which the prime is the the irreducible polynomial ($x^8 + x^4 + x^3 + x + 1$), to find the first terms of RCon[13] and RCon[14]. The following shows the constants for AES-256.

Round	*(RCon)*	*Round*	*(RCon)*	*Round*	*(RCon)*
1	(**01** 00 00 00)$_{16}$	6	(**20** 00 00 00)$_{16}$	11	(**6C** 00 00 00)$_{16}$
2	(**02** 00 00 00)$_{16}$	7	(**40** 00 00 00)$_{16}$	12	(**D8** 00 00 00)$_{16}$
3	(**04** 00 00 00)$_{16}$	8	(**80** 00 00 00)$_{16}$	13	(**AB** 00 00 00)$_{16}$
4	(**08** 00 00 00)$_{16}$	9	(**1B** 00 00 00)$_{16}$	14	(**4D** 00 00 00)$_{16}$
5	(**10** 00 00 00)$_{16}$	10	(**36** 00 00 00)$_{16}$		

23. The result is an identity matrix as shown below. Note that the addition and multiplication of elements are in GF(2).

$$\underset{\mathrm{X}}{\begin{bmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&0 \end{bmatrix}} \times \underset{\mathrm{X}^{-1}}{\begin{bmatrix} 0&0&1&0&0&1&0&1\\ 1&0&0&1&0&0&1&0\\ 0&1&0&0&1&0&0&1\\ 1&0&1&0&0&1&0&0\\ 0&1&0&1&0&0&1&0\\ 0&0&1&0&1&0&0&1\\ 1&0&0&1&0&1&0&0\\ 0&1&0&0&1&0&1&0 \end{bmatrix}} = \underset{\mathrm{X} \times \mathrm{X}^{-1}}{\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}}$$

25. Most of the code matches, line by line, with the steps in the process. The only section that needs some explanation is the loop. The iterations of the loop gives us

$$\begin{array}{lcl}
\mathbf{c}_0 = \mathbf{b}_0 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 & \rightarrow & \mathbf{c}_0 = \mathbf{b}_0 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \\
\mathbf{c}_1 = \mathbf{b}_1 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \oplus \mathbf{b}_0 & \rightarrow & \mathbf{c}_1 = \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \\
\mathbf{c}_2 = \mathbf{b}_2 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \oplus \mathbf{b}_0 \oplus \mathbf{b}_1 & \rightarrow & \mathbf{c}_2 = \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \\
\mathbf{c}_3 = \mathbf{b}_3 \oplus \mathbf{b}_7 \oplus \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 & \rightarrow & \mathbf{c}_3 = \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_7 \\
\mathbf{c}_4 = \mathbf{b}_4 \oplus \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 & \rightarrow & \mathbf{c}_4 = \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \\
\mathbf{c}_5 = \mathbf{b}_5 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 & \rightarrow & \mathbf{c}_5 = \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \\
\mathbf{c}_6 = \mathbf{b}_6 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 & \rightarrow & \mathbf{c}_6 = \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \\
\mathbf{c}_7 = \mathbf{b}_7 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 & \rightarrow & \mathbf{c}_7 = \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7
\end{array}$$

The rearranged code is the result of matrix multiplication $\mathbf{c} = \mathbf{X} \times \mathbf{b}$ if we ignore the zero terms. After each line is created $\mathbf{d} = \mathbf{c} + \mathbf{y}$ is made.

27.

```
InvSubBytes (S[0 … 3][0 … 3])
{
    for ($r$ = 0 to 3)
        for (c = 0 to 3)
            $S[r][c] \leftarrow$ invsubbyte ($S[r][c]$)
    return (S)
}
```

```
invsubbyte (byte)
{
    $\mathbf{d} \leftarrow$ ByteToMatrix (byte)
    $\mathbf{c} \leftarrow \mathbf{d} \oplus$ ByteToMatrix (0x63)
    $\mathbf{b} \leftarrow \mathbf{X}^{-1} \times \mathbf{c}$
    $a \leftarrow$ MatrixToByte (b)
    return ($a^{-1}$)
}
```

29.

```
CopyRow (row[0 … 3], t[0 … 3])
{
    for ($c$ = 0 to 3)
        $t[c] \leftarrow row[c]$
}
```

31. The **MixColumns** algorithm calls the **mixcolum** routine four times, once for each column of the old state to create the correspond column of the new state. The **mixcolumn** routine actually performs the following matrix multiplication $\mathbf{col} = \mathbf{C} \times \mathbf{t}$, in which **col** is the new column matrix, **t** is the old column matrix, and the **C** is the square constant matrix. We can write the code in the **mixcolumn** routine as

$$\begin{array}{lllllllll}
\mathbf{col_0} & \leftarrow & C_{00} \bullet \mathbf{t}_0 & \oplus & C_{01} \bullet \mathbf{t}_1 & \oplus & C_{02} \bullet \mathbf{t}_2 & \oplus & C_{03} \bullet \mathbf{t}_3 \\
\mathbf{col_1} & \leftarrow & C_{10} \bullet \mathbf{t}_0 & \oplus & C_{11} \bullet \mathbf{t}_1 & \oplus & C_{12} \bullet \mathbf{t}_2 & \oplus & C_{13} \bullet \mathbf{t}_3 \\
\mathbf{col_2} & \leftarrow & C_{20} \bullet \mathbf{t}_0 & \oplus & C_{21} \bullet \mathbf{t}_1 & \oplus & C_{22} \bullet \mathbf{t}_2 & \oplus & C_{23} \bullet \mathbf{t}_3 \\
\mathbf{col_3} & \leftarrow & C_{30} \bullet \mathbf{t}_0 & \oplus & C_{31} \bullet \mathbf{t}_1 & \oplus & C_{32} \bullet \mathbf{t}_2 & \oplus & C_{33} \bullet \mathbf{t}_3
\end{array}$$

33.

```
MixColumns (S[0 … 3][0 … 3]
{
        for ($c = 0$ to 3)
                mixcolumn (S[c])
}
```

```
mixcolumn (col [0 … 3])
{
        CopyColumn (col, t)
        col[0] ← MultField (0x02, t[0]) ⊕ MultField (0x03, t[1]) ⊕ t[2] ⊕ t[3]
        col[1] ← t[0] ⊕ MultField (0x02, t[1]) ⊕ MultField (0x03, t[2]) ⊕ t[3]
        col[2] ← t[0] ⊕ t[1] ⊕ MultField (0x02, t[2]) ⊕ MultField (0x03, t[3])
        col[3] ← MultField (0x03, t[0]) ⊕ t[1] ⊕ t[2] ⊕ MultField (0x02, t[3])
}
```

35. The algorithm uses a loop that iterates four times, one for each column of the current state. In each iteration, a column of the old state is exclusive-ored with a key word to create a new column.

$$\mathbf{S}[c] \leftarrow \mathbf{S}[c] \oplus \mathbf{W}[4 \times \text{round} + c]$$

For example, in round 5, we have

S[0]	←	**S**[0] ⊕ **W**[20]
S[1]	←	**S**[1] ⊕ **W**[21]
S[2]	←	**S**[2] ⊕ **W**[22]
S[3]	←	**S**[3] ⊕ **W**[23]

37.

a.

```
KeyExpansion (Key[0 … 23], W[0 … 51]
{
        for ($i$ = 0 to 5)
                W[$i$] ← Key[$4i$] | Key[$4i$ +1] | Key[$4i$ +2] | Key[$4i$ +3]
        for ($i$ = 6 to 51)
                if ($i$ mod 6 = 0)
                {
                        t ← subword(rotWord (W[$i$ – 1]) ⊕ Rcon[$i$ /6]
                        W[$i$] ← t ⊕ W[$i$ – 6]
                }
                else
                        $W[i] \leftarrow W[i-1] \oplus W[i-6]$
}
```

b.

```
KeyExpansion (Key[0 … 31], W[0 … 59])
{
        for ($i$ = 0 to 7)
                $W[i] \leftarrow Key[4i] \mid Key[4i+1] \mid Key[4i+2] \mid Key[4i+3]$
        for ($i$ = 8 to 59)
                if ($i$ mod 8 = 0)
                {
                        $t \leftarrow subword(rotWord (W[i-1]) \oplus Rcon[i/8]$
                        $W[i] \leftarrow t \oplus W[i-8]$
                }
                if ($i$ mod 8 = 4)
                {
                        $t \leftarrow subword(W[i-1])$
                        $W[i] \leftarrow t \oplus W[i-8]$
                }
                else
                        $W[i] \leftarrow W[i-1] \oplus W[i-8]$
}
```

39.

```
InvCipher (InBlock[0 … 16], outBlock[0 … 16], W[0 … 43]
{
        BlockToState (Inblock, S)
        $S \leftarrow$ AddRoundKey (S, W[40 … 43])
        for ($r$ = 1 to 10)                        // r defines the round
        {
                $S \leftarrow$ InvShiftRows (S)
                $S \leftarrow$ InvSubBytes (S)
                $S \leftarrow$ AddRoundKey (S, W[$(10-r) \times 4$ … $(10-r) \times 4 + 3$)
                if ($r \neq 10$)
                        $S \leftarrow$ InvMixColumns (S)
        }
        StateToBlock (S, outBlock)
}
```