

CHAPTER 5

Introduction to Modern Symmetric-Key Ciphers

(Solution to Practice Set)

Review Questions

1. The traditional symmetric-key ciphers are character-oriented ciphers. The modern symmetric-key ciphers are bit-oriented ciphers.
2. To be resistant to exhaustive-search attack, a modern block cipher needs to be designed as a substitution cipher because in a transposition cipher we have the same number of 1s in the plaintext and ciphertext, which makes exhaustive-search attack simpler.
3. A transposition is definitely a permutation of bits. A substitution of bits can be thought of a permutation if we add decoding and encoding to the operation.
4. We mentioned several components of a modern block ciphers in this chapter: P-boxes, S-boxes, the exclusive-or operation, the swap operation, the circular shift operation, the split operation, and the combine operation.
5. A P-box (permutation box) transposes bits. We have three types of P-boxes in modern block ciphers: straight P-boxes, expansion P-boxes, and compression P-boxes. A straight P-box is invertible; the other two are not.
6. An S-box is an $m \times n$ substitution unit, where m and n are not necessarily the same. The necessary condition for invertibility is that m should be equal to n .
7. A product cipher is a complex cipher combining substitution, permutation, and other components discussed in this chapter. We discussed two classes of product ciphers: Feistel and non-Feistel ciphers.
8. Diffusion hides the relationship between the ciphertext and the plaintext. Confusion hides the relationship between the ciphertext and the key.
9. A Feistel block cipher uses both invertible and noninvertible components. A non-Feistel block cipher uses only invertible components.
10. A differential cryptanalysis is based on a nonuniform differential distribution table of the S-boxes in a block cipher; it is a chosen-plaintext attack. A linear cryptanal-

ysis is based on a linear approximation of an S-box; it is a known-plaintext attack.

11. In a synchronous stream cipher the key stream is independent of the plaintext or ciphertext. In a nonsynchronous stream cipher the key stream is somehow dependent on the plaintext or ciphertext.
12. A feedback shift register is made of a shift register and a feedback function. We discussed two variations: linear feedback shift register (LFSR) and nonlinear feedback shift register (NLFSR).

Exercises

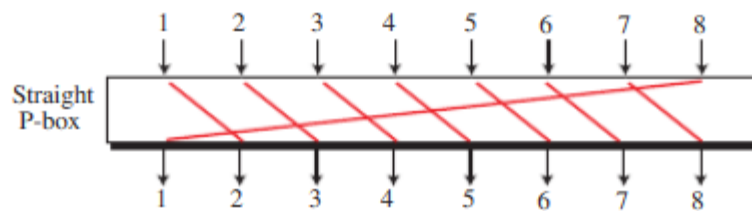
13. The order of the group is $10! = 3,626,800$. The key size is $\lceil \log_2(10!) \rceil = 22$ bits. Note that a key of 22 bits can select $2^{22} = 4,194,304$ different permutations, but only 3,626,800 of them are used here.
14. The order of the group is $(2^{10})! = 1024!$. The key size is $\lceil \log_2(1024!) \rceil = 8770$ bits. Note that a key of 8770 bits can select 2^{8770} different permutations, but only $(1024!)$ of them are used here.
15.
 - a. $\text{CircularLeftShift}_3(\mathbf{10011011}) \rightarrow \mathbf{11011100}$
 - b. $\text{CircularRightShift}_3(\mathbf{11011100}) \rightarrow \mathbf{10011011}$
 - c. The original word in Part a and the result of Part b are the same, which shows that circular left shift and circular right shift operations are inverses of each other.
16.
 - a. $\text{Swap}(\mathbf{10011011}) \rightarrow \mathbf{10111001}$
 - b. $\text{Swap}(\mathbf{10111001}) \rightarrow \mathbf{10011011}$
 - c. The original word in Part a and the result of Part b are the same, which shows that swapping is a self-invertible operation.
17. We show the complement of A with $\bar{A}$.
 - a. $(01001101) \oplus (01001101) = (\mathbf{00000000}) \rightarrow (A \oplus A = \text{All 0s})$
 - b. $(01001101) \oplus (10110010) = (\mathbf{11111111}) \rightarrow (A \oplus \bar{A} = \text{All 1s})$
 - c. $(01001101) \oplus (00000000) = (\mathbf{01001101}) \rightarrow (A \oplus \text{All 0s} = A)$
 - d. $(01001101) \oplus (11111111) = (\mathbf{10110010}) \rightarrow (A \oplus \text{All 1s} = \bar{A})$
18.
 - a. The value of $010_2 = 2$, which can be decoded as the 8-bit word $(0000\mathbf{100})$.
 - b. The 8-bit word $(00\mathbf{100000})$ has a 1 in position 5, which can be encoded as $(101)_2$.

19. Using eight bits for each character, $|M| = 8 \times 2000 = 16,000$ bits. Therefore, we have

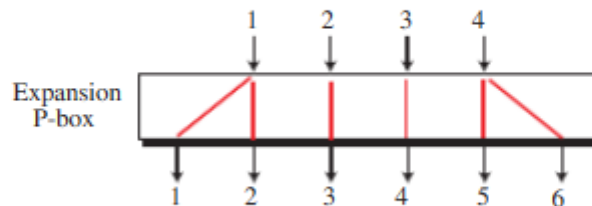
$$|M| + |Pad| = 0 \bmod 64 \rightarrow |Pad| = -|M| \bmod 64 \rightarrow |Pad| = -16,000 \bmod 64 = 0$$

This means no padding is needed. The message is divided into 250 blocks.

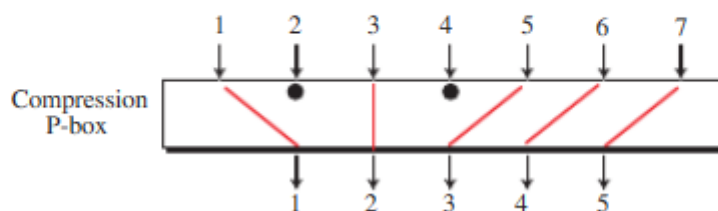
20. The permutation table is [2 5 4 1 3].
 21. The permutation table is [1 3 5].
 22. The permutation table is [1 3 3 1 2].
 23. See the figure below:



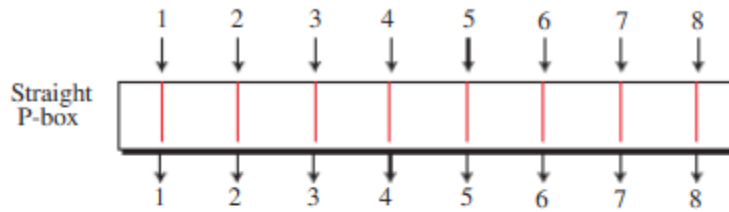
24. It is an expansion P-box with 4 inputs and 6 outputs as shown below:



25. It is a compression P-box with 7 inputs and 5 outputs as shown below:



26. It is a straight P-box with 6 inputs and 6 outputs as shown below:



27. We use the following procedure:

a. We first find the input/output relation based on the given S-box:

Input: 00 01 10 11
Output: 01 11 10 00

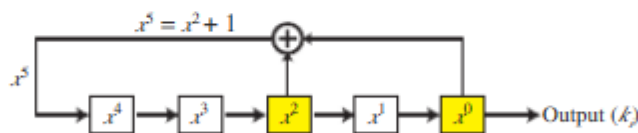
b. We then find the inverse input/output relation (sorted on input):

Input: 00 01 10 11
Output: 11 00 10 01

c. Now we create the table for the inverse S-box (the left row defines the first input bit, the first column defines the second input bit, and the entries define the output):

	0	1
0	11	00
1	10	01

28. The characteristic polynomial is $x^5 + x^2 + 1 = 0$. The feedback function can be found as $x^5 = x^2 + 1$. The following figure shows the LFSR.



The characteristic polynomial $x^5 + x^2 + 1$ or $(25)_{16}$ is both a primitive polynomial (see Appendix G). However, the corresponding number of cells is 5, which is odd. This means that period is less than $31 (2^m - 1)$.

29. The characteristic polynomial is $x^4 + x^3 + x^2 + 1$ or $(11101)_2$ or $(1D)_{16}$. The polynomial is not primitive (see Appendix G). The maximum period is then less than $2^4 - 1$ or less than 15.
30. The following table shows the initial state (seed) and the next twenty states.

States	b_4	b_3	b_2	b_1	b_0	k_i
Initial	1	1	1	1	0	
01	0	1	1	1	1	0
02	0	0	1	1	1	1
03	0	0	0	1	1	1
04	1	0	0	0	1	1
05	0	1	0	0	0	1
06	0	0	1	0	0	0
07	1	0	0	1	0	0
08	1	1	0	0	1	0
09	0	1	1	0	0	1
10	1	0	1	1	0	0
11	0	1	0	1	1	0
12	1	0	1	0	1	1
13	1	1	0	1	0	1
14	1	1	1	0	1	0
15	1	1	1	1	0	1
16	0	1	1	1	1	0
17	0	0	1	1	1	1
18	0	0	0	1	1	1
19	1	0	0	0	1	1
20	0	1	0	0	0	1

The initial state in the above table is the same as the state 11 in Table 5.6. The next 9 states are also the same; state 09 in the above table is the same as the state 20 in Table 5.6. The rest of the states are different in the two tables.

31. To have a maximum period of 32, the characteristic polynomial should be of degree 6 because $2^5 - 1 = 31 < 32$. However, if the characteristic polynomial is primitive, the maximum period is $2^6 - 1 = 63$. But the problem says that the maximum period is 32, therefore, the characteristic polynomial is not primitive. In other words, we have an LFSR of degree 6, with 6 cells (6-bit register) whose characteristic polynomial is not primitive.
32. We assume that bits are numbered b_0 to b_5 from right to left. Other assumptions yield different results.

Input: **110010** → Left bit = $1 \oplus 0 \oplus 1 = 0$ Right bit = $1 \oplus 0 \oplus 0 = 1$ → Output: 01
 Input: **101101** → Left bit = $1 \oplus 1 \oplus 0 = 0$ Right bit = $0 \oplus 1 \oplus 1 = 0$ → Output: 00

33.

Input: **1011** → Left rotate → Output: 110
 Input: **0110** → Right rotate → Output: 011

34.

```
split(word [0 ...  $n$ ],  $n$ )
{
     $i \leftarrow 1$ 
    while ( $i \leq n/2$ )
    {
        rightWord[ $i$ ]  $\leftarrow$  word[ $i$ ]
        leftWord[ $i$ ]  $\leftarrow$  word[ $i + n/2$ ]
         $i \leftarrow i + 1$ 
    }
    return(rightWord, leftWord)
}
```

35.

```
combine(rightWord [1 ...  $m$ ], leftWord [1 ...  $m$ ],  $m$ )
{
     $i \leftarrow 1$ 
    while ( $i \leq m$ )
    {
        word[ $i$ ]  $\leftarrow$  rightWord[ $i$ ]
        word[ $i + m$ ]  $\leftarrow$  leftWord[ $i$ ]
         $i \leftarrow i + 1$ 
    }
    return(word[1 ...  $n$ ])           //  $n = 2m$ 
}
```

36.

```
swap(word [1 ...  $n$ ],  $n$ )
{
     $i \leftarrow 1$ 
    while ( $i \leq n/2$ )
    {
        temp[ $i + n/2$ ]  $\leftarrow$  word[ $i$ ]
        temp[ $i$ ]  $\leftarrow$  word[ $i + n/2$ ]
         $i \leftarrow i + 1$ 
    }
    return(temp[0 ...  $n$ ])
}
```

37.

```
circularShift(shift, word [1 ...  $n$ ],  $n$ ,  $k$ )
{
     $i \leftarrow 1$ 
    while ( $i \leq k$ )
    {
        if (shift = left)
            word  $\leftarrow$  circularShiftLeft(word,  $n$ )
        else
            word  $\leftarrow$  circularShiftRight(word,  $n$ )
         $i \leftarrow i + 1$ 
    }
    return(word[1 ...  $n$ ])
}
```

```
circularShiftLeft(word [1 ...  $n$ ],  $n$ )
```

```
{
    temp  $\leftarrow$  word[ $n$ ]
     $j \leftarrow n - 1$ 
    while ( $j \geq 0$ )
    {
        word[ $j+1$ ]  $\leftarrow$  word[ $j$ ]
         $j \leftarrow j - 1$ 
    }
    word[1]  $\leftarrow$  temp
    return(word[0 ...  $n$ ])
}
```

```
circularShiftRight(word [0 ...  $n$ ],  $n$ )
```

```
{
    temp  $\leftarrow$  word[1]
     $j \leftarrow 1$ 
    while ( $j \leq n$ )
    {
        word [ $j-1$ ]  $\leftarrow$  word[ $j$ ]
         $j \leftarrow j + 1$ 
    }
    word[ $n$ ]  $\leftarrow$  temp
    return(word[0 ...  $n$ ])
}
```

38.

```
P-box (inputBits [1 ...  $n$ ], Table [1 ...  $m$ ],  $n$ ,  $m$ )
{
     $i \leftarrow 1$ 
    while ( $i \leq m$ )
    {
        outputBits[ $i$ ]  $\leftarrow$  inputBits[Table[ $i$ ]]
         $i \leftarrow i + 1$ 
    }
    return (outputBits [1 ...  $n$ ])
}
```

39. The table can be designed in many different ways. We assume, we have a linear table of n cells, in which each cell contains a value of m bits. The input defines the cell number, the contents of the cell defines the output. With this configuration, the routine looks like the one shown below:

```
S-box (inputBits [1 ...  $n$ ], Table [1 ...  $n$ ],  $n$ ,  $m$ )
{
    index  $\leftarrow$  binaryToDecimal(inputBits)
    value  $\leftarrow$  Table [index]
    outputBits  $\leftarrow$  decimalToBinary(value)
    return (outputBits [0 ...  $m$ ])
}
```

40. We assume that the key mixer apply the XOR operation on the input and the round key. The following shows the general idea.

```
Non_FeistelRound(inputBits [1 ...  $n$ ], roundKey[1 ...  $n$ ],  $n$ , PermuteTable)
{
    temp  $\leftarrow$  inputBits  $\oplus$  roundKey
    temp  $\leftarrow$  substitute(temp, SubstituteTables)
    outputBits  $\leftarrow$  permute(temp, PermuteTable)
    return(outputBits [1 ...  $n$ ])
}
```

41.

```
FeistelRound(inputBits [1 ...  $n$ ], roundKey[1 ...  $n$ ],  $n$ )
{
    (tempLeft, tempRight)  $\leftarrow$  split(word,  $n$ )
    tempLeft  $\leftarrow$  tempLeft  $\oplus$  function(tempRight, roundKey)
    (tempLeft, tempRight)  $\leftarrow$  swap(tempLeft, tempRight)
    outputBits  $\leftarrow$  combine(tempLeft, tempRight)
    return(outputBits [1 ...  $n$ ])
}
```

42.

```
LFSR(initialState [0 ...  $n-1$ ],  $n$ )
{
    b[ $n$ ]  $\leftarrow$  f(initialState)
     $i \leftarrow n$ 
    while ( $i > 0$ )
    {
        b[ $i-1$ ]  $\leftarrow$  b[ $i$ ]
         $i \leftarrow i-1$ 
    }
    return(b[0])
}
```