4.2 DES 구조

4.2.1 DES 전체 구조

DES는 IBM에서 Lucifer 시스템을 개선하여 개발한 암호 시스템으로, 1977년 미국 상무성의 국립 표준국(NBS, National Bureau of Standard, 현재 NIST)에서 미국 표준 암호 알고리즘으로 채택한 64비트 블록의 입·출력을 갖는 64비트 블록 암호이다. DES는 암호화 및 복호화를 위해서 동일한 키를 사용하는 대표적인 비밀 키 블록 암호로서 [그림 4.3]과 같이 Feistel 구조를 갖는다. 64비트의 키 블록 중 56비트가 암호화 및 복호화에 사용되고, 나머지 8비트는 키 블록의 패리티 검사용으로 사용된다. DES는 기본적으로 16라운드로 구성되며, 각 라운드는 서로 동일한 과정을 반복적으로 수행하다. 복호화는 암호화 과정과 동일하나 사용되는 키만 역순으로 적용하면 된다.

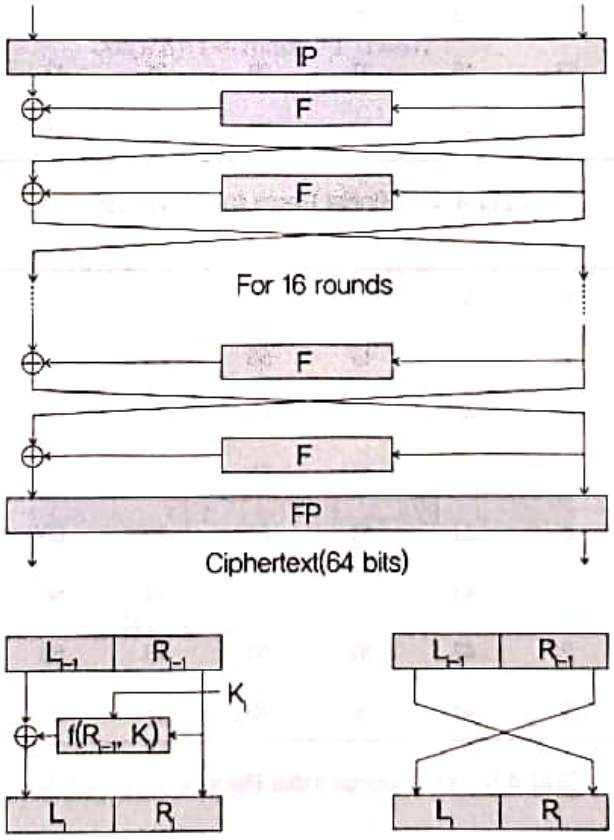

그림 4.3 DES의 기본 구조

4.2.2 초기 순열 및 역초기 순열

64비트 평문 입력은 아래 그림에 의해 초기 순열(Initial Permutation, IP) 과정을 거친다. 이 표는 64비트 입력 중 58번째 비트가 첫 번째 위치로, 50번째 비트가 두 번째 위치로 오는 등의 순열을 정해주는 표이다. 16라운드의 암호화 과정을 수행한 결과는 초기 순열의 역과정인 역초기 순열(inverse IP) IP^{-1}을 거쳐 최종적으로 출력되어 암호문이 된다. 그러나 초기 순열과 역초기 순열은 암호학적 강도가 없는 것으로 알려져 있다.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

그림 4.4 IP(Initial Permutation) 테이블

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

그림 4.5 IP^{-1}(Inverse Initial Permutation) 테이블

초기 순열 프로그램

```
1.  void IP(BYTE *in, BYTE *out)
2.  {
3.      int i;
4.      BYTE index, bit, mask=0x80;
5.
6.      for(i=0;i<64;i++)
7.      {
8.          index = (ip[i]-1) / 8;
9.          bit = (ip[i]-1) % 8;
10.
11.         if(in[index] & (mask >> bit))
12.             out[i/8] |= mask >> (i%8);
13.     }
14. }
```

- 1 ~ 14 : 입력된 64비트 블록을 초기 순열 테이블을 이용하여 변환한다.
- 6 ~ 13 : 초기 순열 테이블에 있는 비트의 위치를 계산하여 & 연산과 시프트 연산을 통해 해당 위치의 비트 값을 추출하여 결과 값을 저장할 배열에 상위 비트부터 저장한다.

역초기 순열 프로그램

```
1.  void In_IP(BYTE *in, BYTE *out)
2.  {
3.      int i;
4.      BYTE index, bit, mask=0x80;
5.
```

```
6.        for(i=0;i<64;i++)
7.        {
8.                index = (ip_1[i]-1) / 8;
9.                bit = (ip_1[i]-1) % 8;
10.
11.               if(in[index] & (mask >> bit))
12.                       out[i/8] |= mask >> (i%8);
13.       }
14. }
```

- 1 ~ 14 : 입력된 64비트 블록을 역초기 순열 테이블을 이용하여 변환한다.
- 6 ~ 13 : 역초기 순열 테이블에 있는 비트의 위치를 계산하여 & 연산과 시프트 연산을 통해 해당 위치의 비트 값을 추출하여 결과 값을 저장할 배열에 상위 비트부터 저장한다.

4.2.3 라운드 함수

가. 확장 순열(Expansion Permutation)

초기 순열을 거친 64비트는 32비트씩 두 부분으로 분할되어 총 16라운드를 수행하는데 한 라운드는 기본 변환 A와 B를 거치게 된다. 기본 변환 A에서 왼쪽 32비트 L_{i-1}은 $f(R_{i-1}, K_i)$와의 XOR에 의해 L_i가 되고, 오른쪽 32비트 R_{i-1}은 그대로 R_i가 된다. 여기서 함수 f는 [그림 4.6]과 같은 구조를 가지고 있으며, 확장 순열(Expansion Permutation, EP), S-Box, P-Box 순열을 거친다. 확장은 32비트를 48비트로, S-Box는 이를 다시 32비트로 변환한다. 기본 변환 B는 단순히 왼쪽 절반과 오른쪽 절반을 교환하는 것이다.

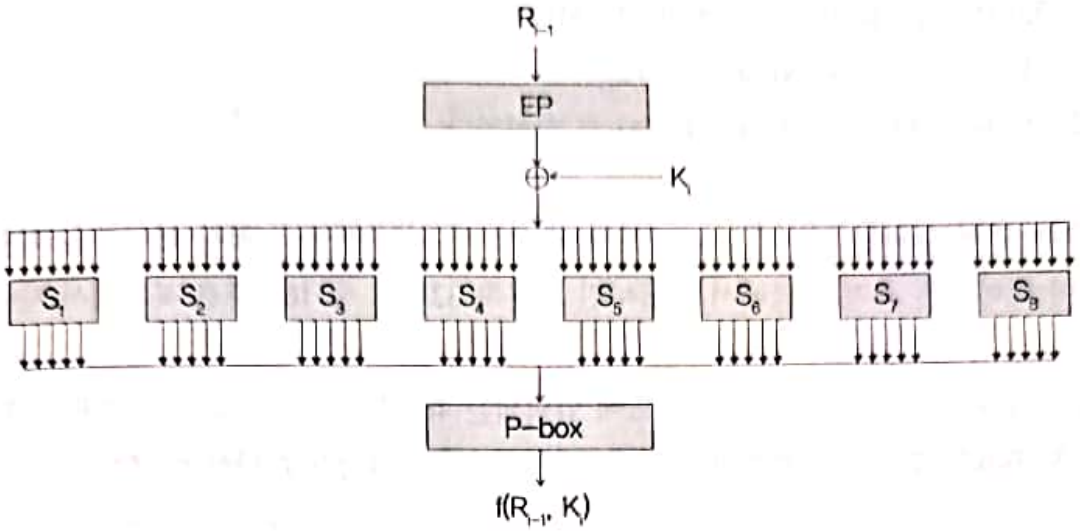

그림 4.6 f함수

f 함수 프로그램

```
1.  UINT f(UINT r, BYTE* rkey)
2.  {
3.      int i;
4.      BYTE data[6] = {0,};
5.      UINT out;
6.
7.      EP(r, data);     // 확장 치환
8.
9.      // 라운드 키와 XOR
10.     for(i=0;i<6;i++)
11.             data[i] = data[i] ^ rkey[i];
12.     out = Permutation(S_box_Transfer(data)); // S-box 변환 결과를 치환
13.
14.     return out;
15. }
```

• 7 : 32비트 입력을 48비트로 확장 치환한다.
• 9 ~ 11 : 라운드 키와 XOR 연산한다.
• 12 : S-box 변환 결과를 다시 P-Box로 치환한다.

f함수는 먼저 입력된 32비트를 48비트로 확장하게 되며 다음의 확장 테이블을 이용하게 된다. 이 테이블은 초기 순열에서 만들어진 오른쪽 32비트 중 16비트를 복제해서 사용하고 있다.

확장 과정을 거친 데이터는 각 라운드에 입력되는 48비트의 서브 키 K_i와 XOR한 후 다음 단계인 S-Box의 입력으로 들어가기 위해 6비트씩 8개의 그룹으로 나누어진다.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

그림 4.7 EP : Expansion Permutation 테이블

확장 순열 프로그램

```
1.  void EP(UINT r, BYTE* out)
2.  {
3.      int i;
4.      UINT mask = 0x80000000;
5.
6.      for(i=0;i<48;i++)
```

```
7.      {
8.          if(r & (mask >> (E[i]-1)))
9.          {
10.             out[i/8] |= (BYTE)(0x80 >> (i%8));
11.         }
12.     }
13. }
```

- 1 ~ 13 : 32비트의 입력 값을 48비트로 확장한다.
- 6 ~ 12 : 입력 값에서 확장 순열 테이블이 나타내는 위치의 비트 값을 & 연산과 시프트 연산을 이용하여 추출한 뒤 결과 값을 저장할 배열의 상위 비트부터 추출한 값을 저장한다.

나. S-Box(Substitution)

f함수의 두 번째 과정은 S-Box로서 총 8개의 S-Box가 있으며, S-Box 테이블에 의해 각각 6비트를 입력으로 받아들여서 4비트로 출력한다. 이 변환은 아래 [그림 4.8]과 같이 나타내며 강한 비선형성을 가지고 있다.

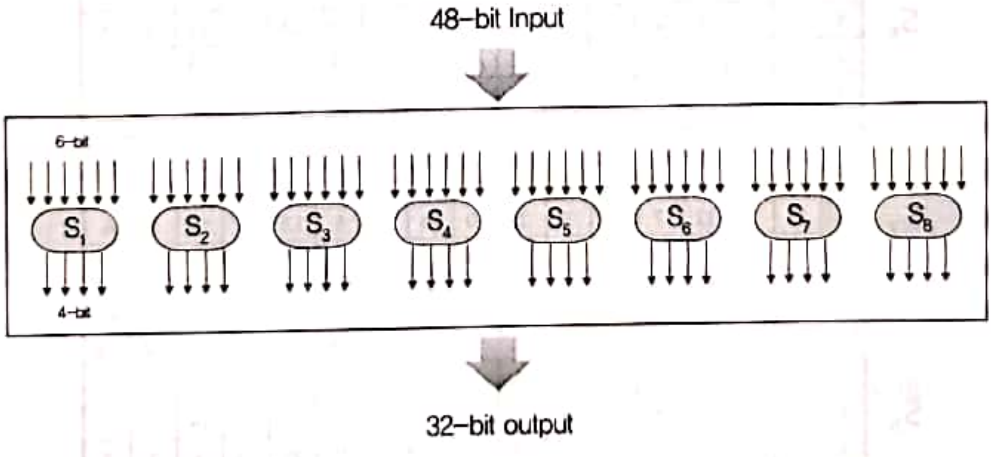

그림 4.8 S-Box의 입출력 관계

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

그림 4.9 S-Box 테이블

S-Box의 동작에서 입력되는 6비트를 a1a2a3a4a5a6라 할 때, a1과 a6을 병렬로 결합하여 생긴 이진수를 십진수로 변화시킨 값이 S-Box에서 행을 결정한다. 비슷한 방법으로 a2,a3,a4,a5를 병렬로 결합하여 생긴 이진수를 십진수로 변화시킨 값이 S-Box에서 열을 결정한다. 열과 행이 만나는 숫자를 이진수로 바꾼 것이 출력 4비트가 된다.

예를 들어 S_1-Box의 입력이 100110이라고 하자. 이때, 최상위 1개 비트와 최하위 1개 비트는 각각 1과 0이므로 [00] [01] [10] 순서로 3번째 행을 참조하고, 가운데 4개 비트 값은 0011이므로 [0000] [0001] ... [0011] 순서인 4번째 행을 참조한다. 최종 결과 값은 S_1-Box에서 3번째 행과 4번째 열에 위치한 십진수 8이 되고 이진수로 표현하면 1000으로 출력되게 된다.

S-Box 프로그램

```
1.  UINT S_box_Transfer(BYTE* in)
2.  {
3.      int i, row, column, shift = 28;
4.      UINT temp = 0, result = 0, mask = 0x00000080;
5.
6.      for(i=0;i<48;i++)
7.      {
8.          if(in[i/8] & (BYTE)(mask >> (i%8)))
9.              temp |= 0x20 >> (i%6);
10.
11.         if((i+1) % 6 == 0)
12.         {
13.             row = ((temp & 0x20) >> 4) + (temp & 0x01);
14.             column = (temp & 0x1E) >> 1;
15.             result += ((UINT)s_box[i/6][row][column] << shift);
16.             shift -= 4;
17.             temp = 0;
```

```
18.                 }
19.         }
20.
21.         return result;
22.  }
```

- 1 ~ 22 : 48비트의 입력을 6비트씩 나누고, 그 값을 S-box 테이블에 적용하여 최종적으로 32비트의 결과 값을 만든다.
- 8 ~ 9 : 입력 값의 상위 비트부터 1비트씩 차례로 추출하여 temp에 저장한다.
- 11 ~ 18 : 추출한 값이 6비트가 되면, 첫 번째 비트와 6번째 비트를 추출(13)하여 S-box의 행의 값을 계산하고, 2~5번째 비트로 열의 값을 계산(14)하여 S-box 테이블을 적용한 후 얻은 4비트의 결과 값을 result에 상위 비트부터 4비트 씩 저장한다. 그 후에 시프트 횟수를 줄이고, 다음의 6비트 값을 계산하기 위해 temp를 초기화 한다.
- 21 : 모든 과정을 수행하고 나면 결과 값을 반환 한다.

다. 순열 함수(Permutation Function)

S-Box를 거친 32비트 데이터는 다음 순열 (P-Box) 테이블에 의해 재배치된다.

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

그림 4.10 순열(Permutation) P-Box 테이블

순열 함수 프로그램

```
1.  UINT Permutation(UINT in)
2.  {
3.      int i;
4.      UINT out = 0, mask = 0x80000000;
5.
6.      for(i=0;i<32;i++)
7.      {
8.          if(in & (mask >> (P[i]-1)))
9.              out |= (mask >> i);
10.     }
11.
12.     return out;
13. }
```

- 1 ~ 13 : 32비트의 입력 값을 순열 테이블을 이용하여 변환한다.
- 6 ~ 10 : 순열 테이블이 나타내는 위치의 비트를 추출한 결과 값을 상위 비트부터 저장한다.
- 12 : 순열 과정이 완료되면 결과 값을 반환한다.

4.2.4 키 생성

입력되는 64비트의 비밀 키를 이용하여 각 라운드에 입력되는 48비트의 키로 확장하는 과정은 [그림 4.11]과 같다. 먼저 64비트 키는 초기 순열(Permuted Choice 1, PC-1)을 거치고, 이 과정에서 8비트의 패리티 비트가 제거된 후, 28비트씩 두 부분으로 분할된다. 이 두 부분은 좌측 순환 이동(Left Cyclic Shift)이 되고, 다시 합쳐진 후, 압축 순열(Permuted Choice 2, PC-2)을 거쳐서 각 라운드에서 사용되는 48비트의 키가 된다.

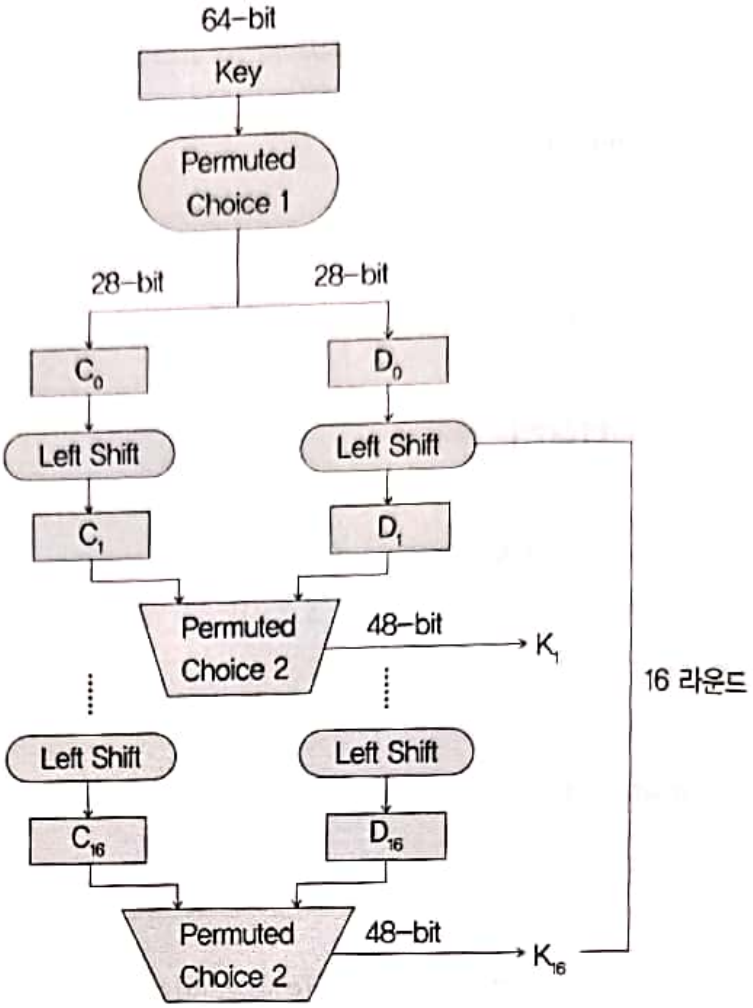

그림 4.11 라운드 키 생성 과정

키 생성 프로그램

```
1.  void key_expansion(BYTE *key, BYTE round_key[16][6])
2.  {
3.      int i;
4.      BYTE pc1_result[7] = {0,};
5.      UINT c = 0, d = 0;
6.
7.      PC1(key, pc1_result);
```

```
8.
9.        makeBit28(&c, &d, pc1_result);
10.
11.       for(i=0;i<16;i++)
12.       {
13.               // 28 bit 순환 시프트
14.               c = cir_shift(c, i);
15.               d = cir_shift(d, i);
16.
17.               PC2(c, d, round_key[i]);
18.       }
19. }
20.
21. void makeBit28(UINT* c, UINT *d, BYTE *data)
22. {
23.       int i;
24.       BYTE mask = 0x80;
25.
26.       for(i=0;i<56;i++)
27.       {
28.               if(i < 28)
29.               {
30.                       if(data[i/8] & (mask >> (i%8)))
31.                               *c |= 0x08000000 >> i;
32.               }
33.               else
34.               {
35.                       if(data[i/8] & (mask >> (i%8)))
```

```
36.                         *d |= 0x08000000 >> (i-28);
37.                 }
38.         }
39. }
```

- 1 ~ 19 : DES의 라운드 키를 생성한다.
- 7 : PC1( ) 함수를 호출하여 키를 순열 선택 1 테이블을 통해 재배치 한다.
- 9 : makeBit28( ) 함수를 호출하여 56비트의 데이터를 28비트로 나눈다.
- 11 ~ 18 : 라운드 키를 만드는 과정으로 순환 이동과 순열 선택 2를 사용하여 16라운드 키를 생성한다.
- 14 ~ 15 : 각각의 28비트 데이터를 좌측으로 순환 이동한다.
- 17 : 순환 이동된 총 56비트의 데이터를 PC2( ) 함수를 호출하여 순열 선택 2 테이블을 통해 재배치 된다. 여기서 재배치 된 48비트 데이터가 라운드 키가 된다.
- 21 ~ 39 : 56비트의 데이터를 28비트로 나누는 함수이다.

가. 순열 선택 1(Permuted Choice 1)

입력된 64비트의 키는 순열 선택표(Permuted Choice 1, PC-1)에 의해 축소되어 재배치되는데 좌우로 28비트씩 2개로 나누어져 각각 C_0와 D_0가 된다.

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

그림 4.12 Permuted Choice 1(PC-1) 순열 테이블

순열 선택 -1 프로그램

```
1.  void PC1(BYTE* in, BYTE* out)
2.  {
3.      int i, index, bit;
4.      UINT mask = 0x00000080;
5.
6.      for(i=0;i<56;i++)
7.      {
8.          index = (PC_1[i]-1) / 8;
9.          bit = (PC_1[i]-1) % 8;
10.
11.         if(in[index] & (BYTE)(mask >> bit))
12.             out[i/8] |= (BYTE)(mask >> (i%8));
13.     }
14. }
```

- 1 ~ 14 : 64비트의 입력 값을 순열 선택-1 테이블을 이용하여 48비트로 변환한다.
- 6 ~ 13 : 순열 선택-1이 나타내는 위치를 계산하여 입력 값으로부터 해당 위치의 비트를 추출하여 결과 값을 저장할 배열에 상위 비트부터 저장한다.

나. 좌측 순환 이동(Left Cyclic Shift)

순열 선택 1을 거쳐 나온 출력 값 56비트의 키를 다음의 라운드별 좌측 쉬프트 테이블에 의해 왼쪽 방향으로 순환 쉬프트를 수행한다.

라운드	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
좌측 순환 쉬프트 수	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

그림 4.13 라운드당 shift 키 비트 수

좌측 순환 이동 프로그램

```
1.  UINT cir_shift(UINT n, int r)
2.  {
3.          int n_shift[16] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};
4.
5.          if(n_shift[r] == 1)
6.          {
7.                  n = ((n << 1) + (n >> 27)) & 0xFFFFFFFF;
8.          }
9.          else
10.         {
11.                 n = ((n << 2) + (n >> 26)) & 0xFFFFFFFF;
12.         }
13.
14.         return n;
15. }
```

- 1 ~ 15 : 각 라운드마다 정해진 시프트 횟수에 따라 28비트 데이터를 왼쪽으로 순환 이동한다.
- 5 ~ 12 : 시프트 횟수가 1인지 2인지에 따라 나누어 처리한다.
- 14 : 결과 값을 반환한다.

다. 순열 선택 2(Permuted Choice 2)

왼쪽 순환 쉬프트를 수행한 56비트의 키는 다음의 순열 선택 2 테이블에 따라 48비트로 축소되어 한 라운드 키를 출력한다. 이러한 과정은 16번 반복하여 각 라운드에 사용되는 16개의 라운드 키를 생성한다.

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

그림 4.14 Permuted Choice 2(PC-2) 순열 테이블

순열 선택 2 프로그램

```
1.  void PC2(UINT c, UINT d, BYTE* out)
2.  {
3.      int i;
4.      UINT mask = 0x08000000;
5.
6.      for(i=0;i<48;i++)
7.      {
8.          if((PC_2[i] -1)< 28)
9.          {
10.             if(c & (mask >> (PC_2[i]-1)))
11.                 out[i/8] |= 0x80 >> (i%8);
12.         }
13.         else
14.         {
15.             if(d & (mask >> (PC_2[i]-1-28)))
16.                 out[i/8] |= 0x80 >> (i%8);
17.         }
18.     }
19. }
```

• 1 ~ 19 : 56비트의 입력 값을 순열 선택-2 테이블을 이용하여 48비트로 변환한다.
• 6 ~ 13 : 순열 선택-2가 나타내는 위치의 비트를 입력 값으로부터 추출하여 결과 값을 저장할 배열에 상위 비트부터 저장한다.

4.3 DES의 S/W 구현

다음은 DES 암호 알고리즘을 전체적으로 구현한 것이다.

```
1.  #define _CRT_SECURE_NO_WARNINGS
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <string.h>
5.  /* 상수 정의 */
6.  #define BLOCK_SIZE      8       // DES 블록 사이즈
7.  #define DES_ROUND       16      // DES 라운드 수
8.
9.  /* 타입 정의 */
10. typedef unsigned char BYTE;
11. typedef unsigned int UINT;
12.
13. /* 함수 선언 */
14. void DES_Encryption(BYTE *p_text, BYTE *result, BYTE *key);
    // DES 암호화 함수
15. void DES_Decryption(BYTE *c_text, BYTE *result, BYTE *key);
    // DES 복호화 함수
16. void IP(BYTE *in, BYTE *out);           // 초기 순열 함수
17. void In_IP(BYTE *in, BYTE *out);        // 역 초기 순열 함수
```

```
18. void EP(UINT r, BYTE* out);                    // 확장 순열 함수
19. UINT Permutation(UINT in);                     // 순열 함수
20. void PC1(BYTE* in, BYTE* out);                 // 순열 선택 - 1 함수
21. void PC2(UINT c, UINT d, BYTE* out);           // 순열 선택 - 2 함수
22. UINT S_box_Transfer(BYTE* in);                 // S-box 변환 함수
23. UINT f(UINT in, BYTE* rkey);                   // f 함수
24. void key_expansion(BYTE *key, BYTE exp_key[16][6]);        // 키 확장 함수
25. void swap(UINT* x, UINT* y);                   // 자리바꿈 함수
26. void makeBit28(UINT* c, UINT *d, BYTE *data);  // 56 bit를 28 bit로 나누
    는 함수
27. UINT cir_shift(UINT n, int r);                 // 28 bit 순환 시프트 함수
28. void BtoW(BYTE *in, UINT *x, UINT *y);  // byte를 word로 바꾸는 함수
29. void WtoB(UINT l, UINT r, BYTE *out);   // word를 byte로 바꾸는 함수
30.
31. /* 전역 변수 */
32. // 초기 순열 테이블
33. BYTE ip[64] = { 58, 50, 42, 34, 26, 18, 10, 2,
34.                 60, 52, 44, 36 ,28, 20, 12, 4,
35.                 62, 54, 46, 38, 30, 22, 14, 6,
36.                 64, 56, 48, 40, 32, 24, 16, 8,
37.                 57, 49, 41, 33, 25, 17,  9, 1,
38.                 59, 51, 43, 35, 27, 19, 11, 3,
39.                 61, 53, 45, 37, 29, 21, 13, 5,
40.                 63, 55, 47, 39, 31, 23, 15, 7 };
41.
42. // 역 초기 순열 테이블
43. BYTE ip_1[64] = { 40, 8, 48, 16, 56, 24, 64, 32,
44.                   39, 7, 47, 15, 55, 23, 63, 31,
45.                   38, 6, 46, 14, 54, 22, 62, 30,
46.                   37, 5, 45, 13, 53, 21, 61, 29,
```

```
47.                                 36, 4, 44, 12, 52, 20, 60, 28,
48.                                 35, 3, 43, 11, 51, 19, 59, 27,
49.                                 34, 2, 42, 10, 50, 18, 58, 26,
50.                                 33, 1, 41,  9, 49, 17, 57, 25 };
51.
52.  // 확장 순열 테이블
53.  BYTE E[48] = { 32,  1,  2,  3,  4,  5,  4,  5,
54.                                   6,  7,  8,  9,  8,  9, 10, 11,
55.                            12, 13, 12, 13, 14, 15, 16, 17,
56.                            16, 17, 18, 19, 20, 21, 20, 21,
57.                            22, 23, 24, 25, 24, 25, 26, 27,
58.                            28, 29, 28, 29, 30, 31, 32,  1 };
59.
60.  // 순열 테이블
61.  BYTE P[32] = { 16,  7, 20, 21, 29, 12, 28, 17,
62.                  1, 15, 23, 26,  5, 18, 31, 10,
63.                  2,  8, 24, 14, 32, 27,  3,  9,
64.                 19, 13, 30,  6, 22, 11,  4, 25 };
65.
66.  // 순열 선택-1 테이블
67.  BYTE PC_1[56] = { 57, 49, 41, 33, 25, 17,  9,  1,
68.                                 58, 50, 42, 34, 26, 18, 10,  2,
69.                                 59, 51, 43, 35, 27, 19, 11,  3,
70.                                 60, 52, 44, 36, 63, 55, 47, 39,
71.                                 31, 23, 15,  7, 62, 54, 46, 38,
72.                                 30, 22, 14,  6, 61, 53, 45, 37,
73.                                 29, 21, 13,  5, 28, 20, 12,  4 };
74.
75.  // 순열 선택-2 테이블
76.  BYTE PC_2[48] = { 14, 17, 11, 24,  1,  5,  3, 28,
```

```
77.                         15,  6, 21, 10, 23, 19, 12,  4,
78.                         26,  8, 16,  7, 27, 20, 13,  2,
79.                         41, 52, 31, 37, 47, 55, 30, 40,
80.                         51, 45, 33, 48, 44, 49, 39, 56,
81.                         34, 53, 46, 42, 50, 36, 29, 32 };
82.
83. // S-BOX 테이블
84. BYTE s_box[8][4][16] =
85. {
86.         { 14,  4, 13, 1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9, 0,  7,
87.           0, 15,  7, 4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5, 3,  8,
88.           4,  1, 14, 8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10, 5,  0,
89.          15, 12,  8, 2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0, 6, 13 },
90.
91.         { 15,  1,  8, 14,  6, 11,  3,  4,  9, 7,  2, 13, 12, 0,  5, 10,
92.           3, 13,  4,  7, 15,  2,  8, 14, 12, 0,  1, 10,  6, 9, 11,  5,
93.           0, 14,  7, 11, 10,  4, 13,  1,  5, 8, 12,  6,  9, 3,  2, 15,
94.          13,  8, 10,  1,  3, 15,  4,  2, 11, 6,  7, 12,  0, 5, 14,  9 },
95.
96.         { 10,  0,  9, 14, 6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8,
97.          13,  7,  0,  9, 3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
98.          13,  6,  4,  9, 8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
99.           1, 10, 13,  0, 6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12 },
100.
101.        { 7, 13, 14, 3,  0,  6,  9, 10,  1, 2, 8,  5, 11, 12,  4, 15,
102.          13,  8, 11, 5,  6, 15,  0,  3,  4, 7, 2, 12,  1, 10, 14,  9,
103.          10,  6,  9, 0, 12, 11,  7, 13, 15, 1, 3, 14,  5,  2,  8,  4,
104.           3, 15,  0, 6, 10,  1, 13,  8,  9, 4, 5, 11, 12,  7,  2, 14 },
105.
106.        { 2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13, 0, 14,  9,
```

```
107.          14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3, 9,  8,  6,
108.           4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6, 3,  0, 14,
109.          11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10, 4,  5,  3 },
110.
111.        { 12,  1, 10, 15, 9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
112.          10, 15,  4,  2, 7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
113.           9, 14, 15,  5, 2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
114.           4,  3,  2, 12, 9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13 },
115.
116.        { 4, 11,  2, 14, 15, 0,  8, 13,  3, 12, 9,  7,  5, 10, 6,  1,
117.          13,  0, 11,  7,  4, 9,  1, 10, 14,  3, 5, 12,  2, 15, 8,  6,
118.           1,  4, 11, 13, 12, 3,  7, 14, 10, 15, 6,  8,  0,  5, 9,  2,
119.           6, 11, 13,  8,  1, 4, 10,  7,  9,  5, 0, 15, 14,  2, 3, 12 },
120.
121.        { 13,  2,  8, 4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
122.          1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
123.          7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
124.          2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11 }
125. };
126.
127. void main( )
128. {
129.         int i;
130.         int msg_len = 0, block_count = 0;        // 메시지 길이와 블록 수
131.         BYTE p_text[128] = {0,};                 // 평문
132.         BYTE c_text[128] = {0,};                 // 암호문
133.         BYTE d_text[128] = {0,};                 // 복호문
134.         BYTE key[9] = {0,};                      // 비밀 키
135.
136.         // 평문 입력
```

```
137.    printf("* 평문 입력 : ");
138.    gets(p_text);
139.
140.    // 비밀키 입력
141.    printf("* 비밀키 입력 : ");
142.    scanf("%s", key);
143.
144.    // 메시지 길이와 블록 수를 계산
145.    msg_len = (int)strlen((char*)p_text);
146.    block_count = (msg_len % BLOCK_SIZE) ? (msg_len / BLOCK_SIZE + 1)
        : (msg_len / BLOCK_SIZE);
147.
148.    for(i=0;i<block_count;i++)
149.        DES_Encryption(&p_text[i*BLOCK_SIZE], &c_text[i*BLOCK_
            SIZE], key);          // DES 암호화
150.
151.    // 암호문 출력
152.    printf("₩n* 암호문 : ");
153.        for(i=0;i<block_count*BLOCK_SIZE;i++)
154.            printf("%x", c_text[i]);
155.    printf("₩n");
156.
157.    for(i=0;i<block_count;i++)
158.        DES_Decryption(&c_text[i*BLOCK_SIZE], &d_text[i*BLOCK_
            SIZE], key);          // DES 복호화
159.
160.    // 복호문 출력
161.    printf("₩n* 복호문 : ");
162.        for(i=0;i<msg_len;i++)
163.            printf("%c", d_text[i]);
```

```
164.        printf("Wn");
165. }
166.
167. // DES 암호화
168. void DES_Encryption(BYTE *p_text, BYTE *result, BYTE *key)
169. {
170.        int i;
171.        BYTE data[BLOCK_SIZE] = {0,};
172.        BYTE round_key[16][6] = {0,};
173.        UINT L = 0, R = 0;
174.
175.        key_expansion(key, round_key);                    // 라운드 키 생성
176.        IP(p_text, data);                                 // 초기 순열
177.
178.        // 64bit 블록을 32bit로 나눔
179.        BtoW(data, &L, &R);
180.
181.        // DES Round 1~16
182.        for(i=0;i<DES_ROUND;i++)
183.        {
184.              L = L ^ f(R, round_key[i]);
185.              // 마지막 라운드는 swap을 하지 않는다
186.              if(i != DES_ROUND-1)
187.                    swap(&L, &R);
188.        }
189.
190.        WtoB(L, R, data);       // 32bit로 나누어진 블록을 다시 64bit 블록으로 변환
191.        In_IP(data, result);                              // 역 초기 순열
192. }
193.
```

```
194. // DES 복호화
195. void DES_Decryption(BYTE *c_text, BYTE *result, BYTE *key)
196. {
197.         int i;
198.         BYTE data[BLOCK_SIZE] = {0,};
199.         BYTE round_key[16][6] = {0,};
200.         UINT L = 0, R = 0;
201.
202.         key_expansion(key, round_key);                    // 라운드 키 생성
203.         IP(c_text, data);                                 // 초기 순열
204.
205.         // // 64bit 블록을 32bit로 나눔
206.         BtoW(data, &L, &R);
207.
208.         // DES Round 1~16
209.         for(i=0;i<DES_ROUND;i++)
210.         {
211.                 L = L ^ f(R, round_key[DES_ROUND-i-1]);    // 복호할 때는
                        라운드 키를 역순으로 적용
212.
213.                 // 마지막 라운드는 swap을 하지 않는다
214.                 if(i != DES_ROUND-1)
215.                         swap(&L, &R);
216.         }
217.
218.         WtoB(L, R, data);       // 32bit로 나누어진 블록을 다시 64bit 블록으로 변환
219.         In_IP(data, result);    // 역 초기 순열
220. }
221.
222. // 자리바꿈
```

```
223. void swap(UINT* x, UINT* y)
224. {
225.         UINT temp;
226.
227.         temp = *x;
228.         *x = *y;
229.         *y = temp;
230. }
231.
232. // 8bit(byte) 단위의 데이터를 32bit(word) 단위의 데이터로 변환
233. void BtoW(BYTE *in, UINT *x, UINT *y)
234. {
235.         int i;
236.
237.         for(i=0;i<8;i++)
238.         {
239.                 if(i < 4)
240.                         *x |= (UINT)in[i] << (24-(i*8));
241.                 else
242.                         *y |= (UINT)in[i] << (56-(i*8));
243.         }
244. }
245.
246. // 32bit(word) 단위의 데이터를 8bit(byte) 단위의 데이터로 변환
247. void WtoB(UINT l, UINT r, BYTE *out)
248. {
249.         int i;
250.         UINT mask = 0xFF000000;
251.
252.         for(i=0;i<8;i++)
```

```
253.        {
254.                if(i < 4)
255.                        out[i] = (l & (mask >> i*8)) >> (24-(i*8));
256.                else
257.                        out[i] = (r & (mask >> (i-4)*8)) >> (56-(i*8));
258.        }
259. }
```

- 1 ~ 3 : 헤더파일 선언
- 6 ~ 7 : 상수 정의
- 10 ~ 11 : 타입 정의
- 14 ~ 29 : 함수 선언
- 32 ~ 125 : 초기 순열, 역초기 순열, 순열, 확장 순열, 순열 선택-1, 순열 선택-2, S-box 테이블을 전역변수로 정의한다.
- 127 ~ 165 : 메인 함수로 DES의 암호화와 복호화를 수행한다.
- 137 ~ 138 : 평문을 입력 받는다.
- 141 ~ 142 : 비밀 키를 입력 받는다.
- 145 ~ 146 : 평문 메시지의 길이와 블록의 수를 계산한다.
- 148 ~ 149 : 블록 수만큼 암호화를 한다.
- 152 ~ 155 : 암호문을 출력한다.
- 157 ~ 158 : 블록 수만큼 복호화를 한다.
- 161 ~ 164 : 복호문을 출력한다.
- 168 ~ 192 : DES 암호화를 수행하는 함수
- 175 ~ 176 : 라운드 키를 생성하고 초기 순열을 수행한다.
- 179 : 초기 순열을 거친 64비트의 데이터를 32비트로 나눈다.
- 182 ~ 188 : DES 암호화의 16라운드를 수행한다. 여기서 마지막 라운드는 자리바꿈을 하지 않는다.
- 190 ~ 191 : 16라운드를 거친 32비트의 L과 R을 합쳐 64비트로 만든 후 역초기 순열을 수행한다.

- 195 ~ 220 : DES 복호화를 수행하는 함수
- 202 ~ 203 : 라운드 키를 생성하고 초기 순열을 수행한다.
- 206 : 초기 순열을 거친 64비트의 데이터를 32비트로 나눈다.
- 209 ~ 216 : DES 복호화의 16라운드를 수행한다. 여기서 라운드 키는 역순으로 적용되고, 암호화와 마찬가지로 마지막 라운드는 자리바꿈을 하지 않는다.
- 218 ~ 219 : 16라운드를 거친 32비트의 L과 R을 합쳐 64비트로 만든 후 역초기 순열을 수행한다.
- 223 ~ 230 : 자리바꿈 함수로 32비트의 L과 R 값을 서로 바꾼다.
- 233 ~ 244 : 8비트 바이트 단위의 데이터를 32비트 워드 단위로 변환하는 함수로 64비트 블록을 32비트 L과 R로 나눌 때 사용한다.
- 247 ~ 259 : 32비트 워드 단위의 데이터를 8비트 바이트 단위로 변환하는 함수로 32비트 데이터 L과 R을 64비트로 결합시킬 때 사용한다.

실행 화면