DES 알고리즘

유튜브 주소 : https://youtu.be/ESmzSc2NlEg

DES 개요

파이스텔 구조, 암호화 과정

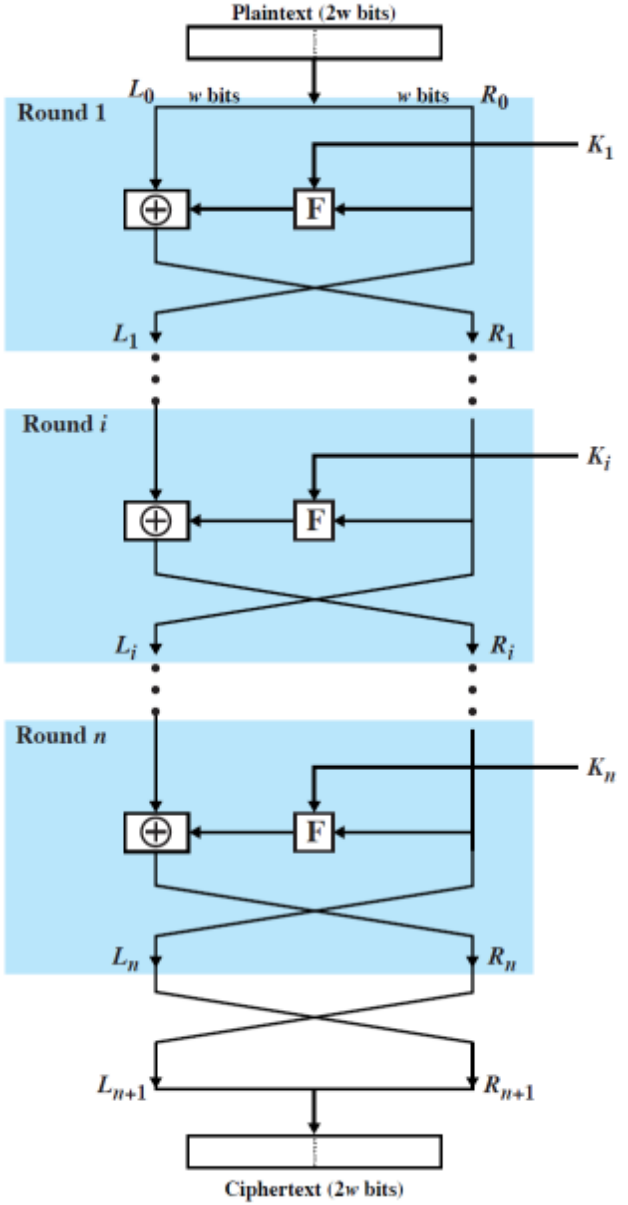
구현 코드 분석

DES

- 블록 암호의 일종
- 대칭키 암호
- 평문 길이 64비트, 키 길이 64비트
  - 실제 사용 되는 키는 56비트
- 파이스텔 사이퍼 구조
  - 데이터를 두 부분으로 나누어
    교대로 비선형 변환을 적용시키는 구조

DES

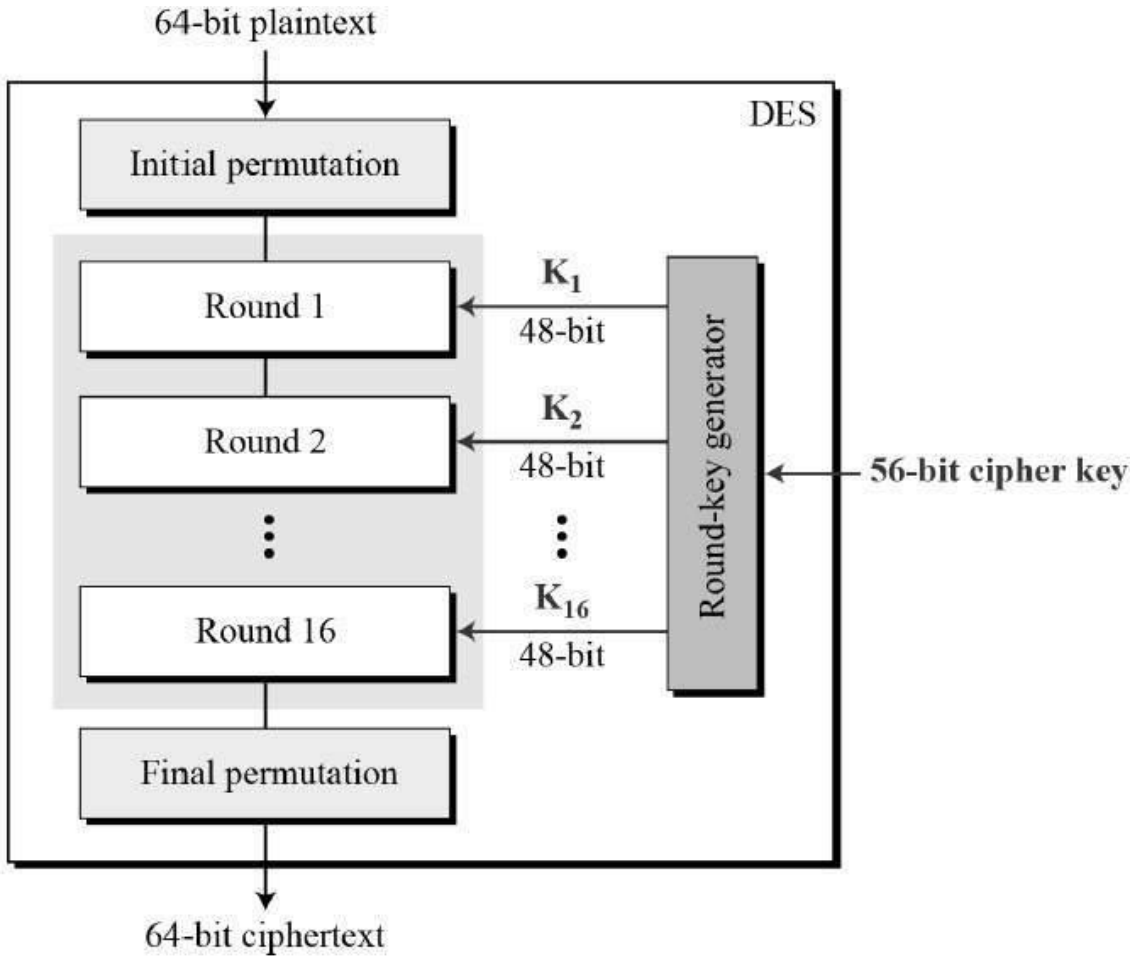
- 파이스텔 사이퍼 구조
- 암복호화 과정에서 라운드 함수(F 함수) 사용
- 라운드 수는 원하는 만큼 늘릴 수 있음
  - 라운드 수가 늘어날수록 보안성 강화
  - 일반적으로 16라운드 사용
- 암복호화 과정이 동일



DES 암호화 과정

- 크게 3가지로 나뉘어짐
  - Initial Permutation & Final Permutation
  - 16 Round Function
  - Round key Generator

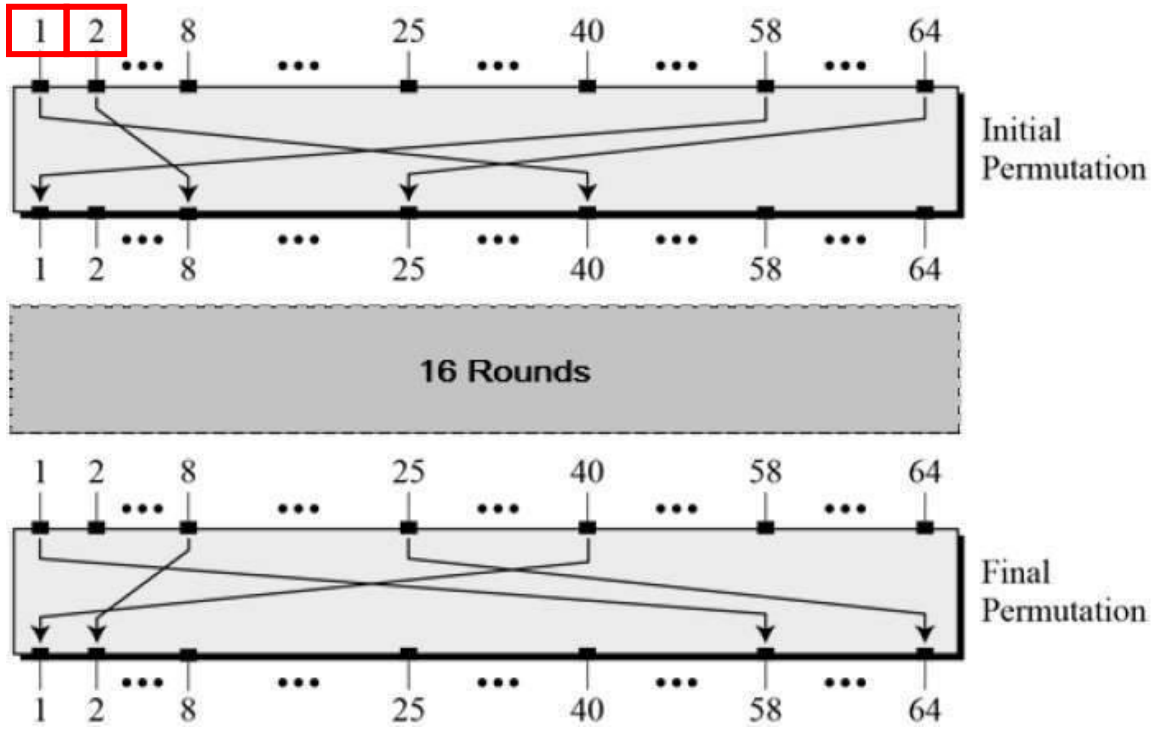
- 과정 요약
  - 64비트 입력 initial permutation
  - 32비트씩 쪼개기
  - 오른쪽 32비트는 다음 라운드의 왼쪽 32비트가 됨
  - 오른쪽 32비트를 F함수를 거친 키와 xor
  - 라운드 반복 but 16라운드에는 교차X
  - 최종 permutation



DES 암호화 과정

Initial Permutation & Final Permutation

- Permutaion Table에 따라서 bit 교환
- 암호화를 하진 않음



Permutaion 구조

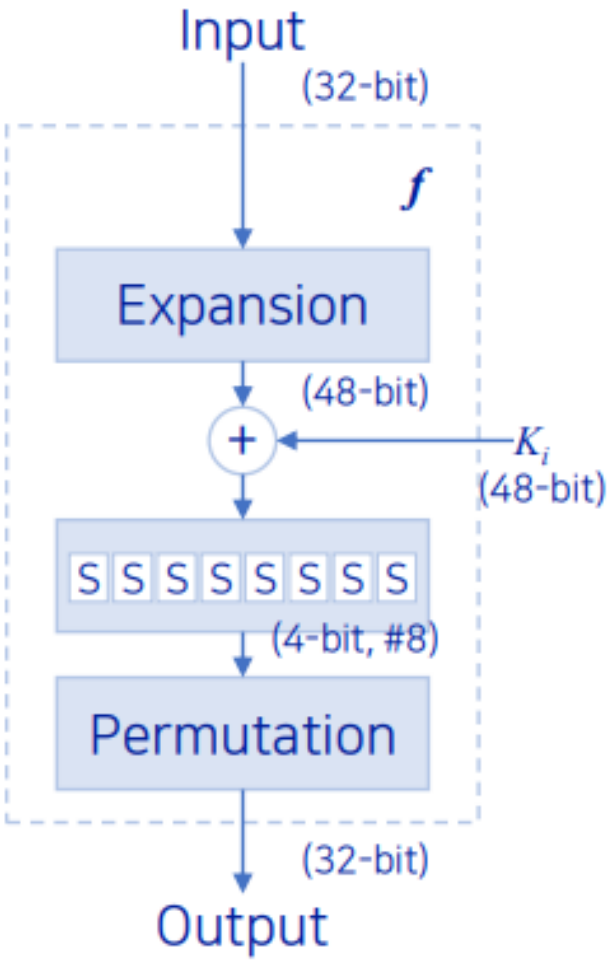
Initial Permutation								*Final Permutation*							
58	50	42	34	26	18	10	02	40	08	48	16	56	24	64	32
60	52	44	36	28	20	12	04	39	07	47	15	55	23	63	31
62	54	46	38	30	22	14	06	38	06	46	14	54	22	62	30
64	56	48	40	32	24	16	08	37	05	45	13	53	21	61	29
57	49	41	33	25	17	09	01	36	04	44	12	52	20	60	28
59	51	43	35	27	19	11	03	35	03	43	11	51	19	59	27
61	53	45	37	29	21	13	05	34	02	42	10	50	18	58	26
63	55	47	39	31	23	15	07	33	01	41	09	49	17	57	25

Permutaion table

DES 암호화 과정

F 함수

- 32비트 입력을 48비트로 확장
- 48비트 라운드 키와 xor연산
- 6비트씩 S박스에 통과
  - 박스당 6비트의 입력은 4비트의 아웃풋을 냄
  - 따라서 6*8=48비트는 S박스 통과 후 4*8=32비트가 됨
- 퍼뮤테이션 후 종료



	0	1	2	3	4	5	6	7
0	16	7	20	21	29	12	28	17
1	1	15	23	26	5	18	31	10
2	2	8	24	14	32	27	3	9
3	19	13	30	6	22	11	4	25

P-box

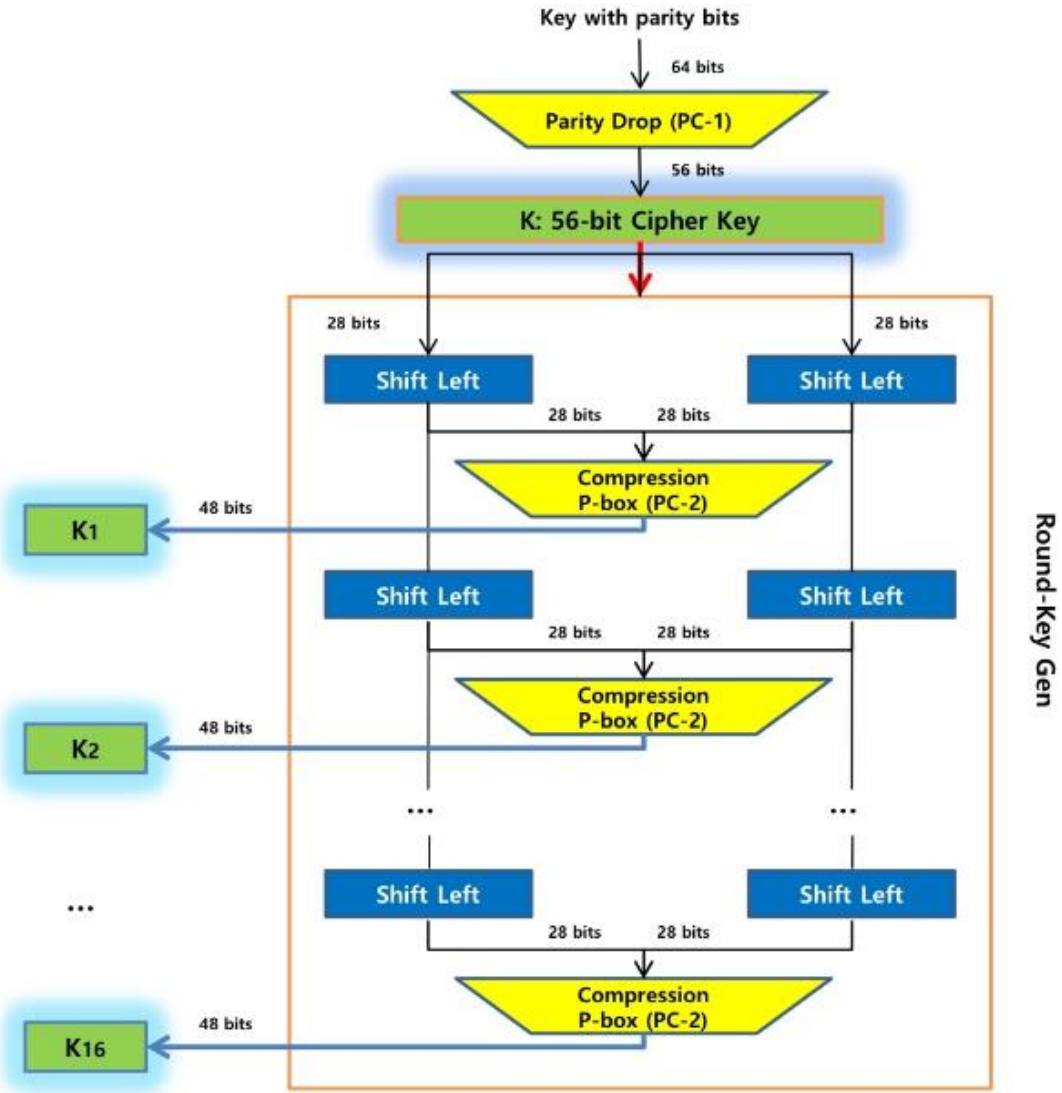
DES 암호화 과정

Key Generator

- 56비트의 기본 키를 이용
  - 64비트 키의 패리티 비트를 drop
- 16개의 48비트 라운드 키 생성
- 56비트를 왼쪽 오른쪽 나눠 shift left
  - 1,2,9,16 라운드에선 1bit shift left
  - 나머지 라운드에선 2bit shift left
- 56비트를 48비트로 축소
  - PC-2 테이블 이용

	0	1	2	3	4	5	6	7
0	14	17	11	24	1	5	3	28
1	15	6	21	10	23	19	12	4
2	26	8	16	7	27	20	13	2
3	41	52	31	37	47	55	30	40
4	51	45	33	48	44	49	39	56
5	34	53	46	42	50	36	29	32

PC-2 테이블



구현 코드 분석

DES 암호화 메인 코드

```
void DES_Encryption(BYTE *p_text, BYTE *result, BYTE *key) {
    int i;

    BYTE data[BLOCK_SIZE] = {0, };

    BYTE round_key[16][6] = { 0, };
    UINT left = 0, right = 0;

    key_expansion(key, round_key);    // 키 generation
    IP(p_text, data);                 // initial permutation

    BtoW(data, &left, &right);        // 32비트씩 쪼갬

    for (i = 0;i < DES_ROUND; i++) {
        left = left ^ f(right, round_key[i]);
        if (i != DES_ROUND - 1)
            swap(&left, &right);
    }

    WtoB(left, right, data);
    In_IP(data, result);              // final permutation
}
```

키 Generation, left shift 코드

```
void key_expansion(BYTE *key, BYTE round_key[16][6])
{
    int i;
    BYTE pc1_result[7] = { 0, };
    UINT c = 0, d = 0;

    PC1(key, pc1_result);             // 축소 전치(64 -> 56)

    makeBit28(&c, &d, pc1_result);  // 28비트씩 쪼갬

    for (i = 0;i<16;i++)
    {
        c = cir_shift(c, i);          // left shift
        d = cir_shift(d, i);

        PC2(c, d, round_key[i]);     // 축소 전치(56 -> 48)
    }
}
```

```
UINT cir_shift(UINT n, int r)
{
    int n_shift[16] = { 1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1 };

    n = (n << n_shift[r]) + (n >> (28 - n_shift[r]));

    return n & 0xFFFFFFF;       // 쓰레기값 제거
}
```

구현 코드 분석

Initial Permutation 코드

```
void IP(BYTE *in, BYTE *out)
{
    int i;
    BYTE index, bit, mask = 0x80;

    for (i = 0;i<64;i++)
    {

        index = (ip[i] - 1) / 8;
        bit = (ip[i] - 1) % 8;

        if (in[index] & (mask >> bit))
            out[i / 8] |= mask >> (i % 8);
    }
}
```

F함수 코드

```
UINT f(UINT r, BYTE* rkey)
{
    int i;
    BYTE data[6] = { 0, };
    UINT out;

    EP(r, data);

    for (i = 0;i<6;i++)
        data[i] = data[i] ^ rkey[i];

    out = Permutation(S_box_Transfer(data));

    return out;
}
```

F함수 내부 Expension코드

```
void EP(UINT r, BYTE* out)
{
    int i;
    UINT mask = 0x80000000;

    for (i = 0;i<48;i++)
    {

        if (r & (mask >> (E[i] - 1)))
        {
            out[i / 8] |= (BYTE)(0x80 >> (i % 8));
        }
    }
}
```

구현 코드 분석

- S-box 코드

```
UINT S_box_Transfer(BYTE* in)
{
    int i, row, column, shift = 28;
    UINT temp = 0, result = 0, mask = 0x80;

    for (i = 0;i<48;i++)
    {
        if (in[i / 8] & (BYTE)(mask >> (i % 8)))  // 마스크를 씌워 확인 후 temp에 해당 비트 1로 함
            temp |= 0x20 >> (i % 6);

        if ((i + 1) % 6 == 0)                          // 6비트마다
        {
            row = ((temp & 0x20) >> 4) + (temp & 0x01);           // 행 값
            column = (temp & 0x1E) >> 1;                              // 열 값

            result += ((UINT)s_box[i / 6][row][column] << shift);    // 값 더하고 쉬프트(4비트씩)

            shift -= 4;
            temp = 0;
        }
    }

    return result;
}
```

Q & A