2) Implement DES

ref: "The DES Algorithm Illustrated" :

<http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm>

(얘는 노트필기 안하고 10/15 화욜에 사진 찍음)

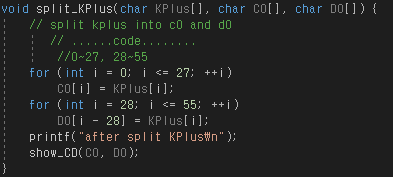
DES 알고리즘을 작성하기 위해 미리 올려진 템플릿을 다운받고 채워야될 함수를 채워줬다.

**step 2.2)** Split into C0(left half) and D0(right half)

C0=1111000 0110011 0010101 0101111

D0=0101010 1011001 1001111 0001111

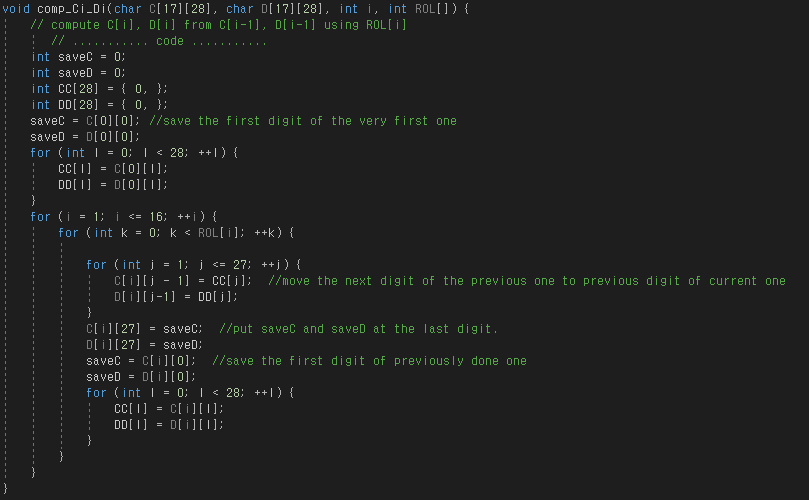






step 2.1에서 64-bit K가 56-bit KPlus로 바뀌어졌다. split\_KPlus 함수에서는 앞서 만들어진 KPlus를 반으로 쪼개서 각각 C0와 D0에 저장시켜야한다. KPlus의 길이는 총 56이기 때문에 배열로써는 0번째부터 55번째까지 있다. 이것을 고려해서 C0에는 KPlus 배열의 0번째부터 27번째까지 대입해주고 D0에는 28번째부터 55번째까지 대입해줬다

**step 2.3)** Compute Cn, Dn (n=1,2,...,16) by doing "rotate left" for C(n-1), D(n-1) once or twice depending on following schedule

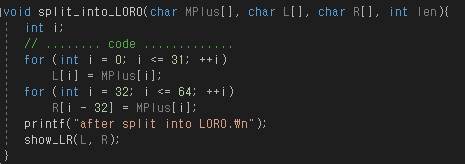




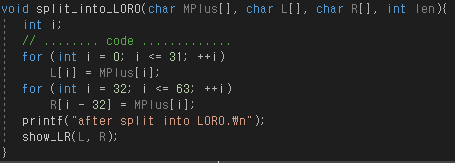
그 뒤로는 쪼개진 KPlus를 rotation table에 따라서 한번 또는 두번 rotate하게 된다. 여기서 rotate한다는건 맨 앞의 숫자를 맨 뒤로 옮긴다는 의미가 된다. 그러기 위해서 바로 이전의 C와 D의 상태를 기록해주는 배열 CC와 DD를 먼저 따로 만들어주고 CC에는 C0를, DD에는 D0를 넣어줬다. 또 첫번째 비트를 마지막으로 옮겨야하기 때문에 첫번째 비트들을 기억할 saveC와 saveD도 만들어주었다. 그러고 나서 CC와 DD의 두번째 비트부터 차례대로 C1, D1에 넣어주고, 미리 저장해둔 saveC와 saveD를 C1과 D1의 마지막에 넣어주었다. 이제 saveC와 saveD에는 C1과 D1의 첫번째 비트를 넣어주고, CC와 DD는 C1과 D1이 된다. 이렇게해서 C16과 D16이 구해질 때까지 for 루프를 돌게 되고, for(int k=0; k<ROL[i]; ++k)를 사용해 rotation을 두번 해야할 경우 그렇게 해주도록 만들었다.

**step 3.2)** Split into L0(left half) and R0(right half)

***L0*** = 1100 1100 0000 0000 1100 1100 1111 1111   
***R0*** = 1111 0000 1010 1010 1111 0000 1010 1010



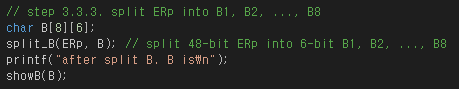


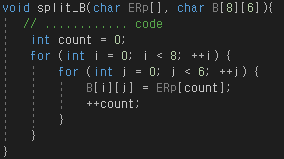


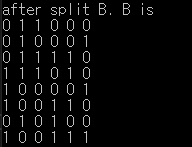
split\_into\_L0R0는 split\_KPlus와 비슷한 방식으로 구현했다. KPlus가 56비트였다면 MPlus는 64비트이기 때문에 MPlus 배열의 0번째부터 31번째까지는 L에, 32번째부터 63번째까지는 R에 넣어주었다.

**step 3.3.2), step 3.3.3)**   
***K1*** xor **E**(***R0***) = 011000 010001 011110 111010 100001 100110 010100 100111

B1 B2 B3 B4 B5 B6 B7 B8







step 3.3.1에서 K1 xor E(R0)를 한 값이 ERp에 저장되고, split\_B 함수에서는 두개의 for 루프를 사용해서 B[0]부터 B[7]까지 6비트씩 들어가게 만들어줬다.

**step 3.3.4)**

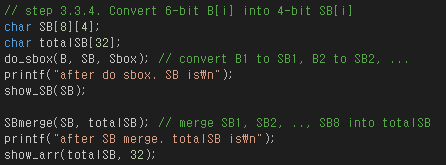
S1(B1)=S1(011000) =S1[00][1100]=S1[0][12]=5=0101

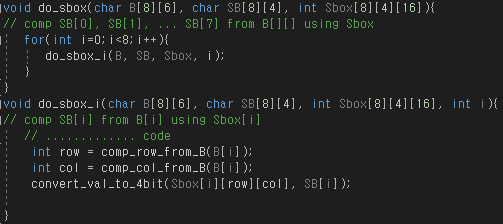
S2(B2)=S2(010001)=S2[1][8]=12=1100

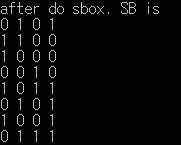
S3(B3)=S3(011110)=S3[0][15]=8=1000

........

SB=0101 1100 1000 0010 1011 0101 1001 0111







이 단계에서는 sbox를 통해서 6비트인 B를 4비트 단위로 끊어서 SB에 저장하게 된다. row는 6비트중 맨 앞과 맨 끝의 비트가 합쳐진 값이고, col은 그 두 비트를 제외한 가운데 4비트가 합쳐진 값이 된다. row와 col 전부 다 미리 정의된 함수들을 이용해서 구해줬다. 그리고나서 정의된 함수 convert\_val\_to\_4bit 를 이용해서 i번째 Sbox의 row번째 열 col번째 행의 값을 찾아낸 뒤 그 값을 이진수로 바꿔서 i번째 SB에 저장시켜줬다.

**step 3.4)** Reverse L16 and R16 and apply the final permutation

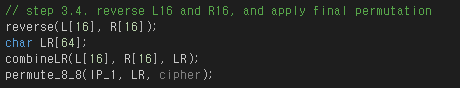
R16L16=

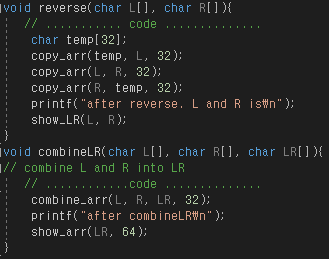
00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

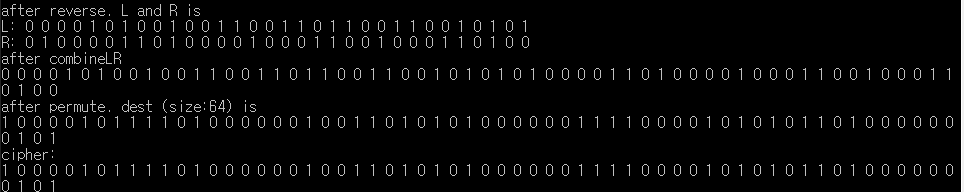
**IP-1**(R16L16) =

10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

= 85E813540F0AB405







reverse 함수에서는 임시 배열 temp와 미리 정의된 함수 copy\_arr을 이용해서 temp에 L을 복사한 후 L에 R을 복사하고, 원래의 L이 저장돼있는 temp를 다시 R에 복사해주게 된다. combineLR 함수에서도 미리 정의된 함수 combine\_arr에 L이랑 R, 합친 결과가 될 LR과 그 길이인 32를 인자로 넘겨주었다. 최종적으로 cipher에 내가 원하는 결과가 나오게 됐다.