정보보호론 lect8_final

12141163 이욱진

Homework
2) Implement DES
ref: "The DES Algorithm Illustrated" :
http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm

처음에 K배열이 table에존재한다는 것을 모르고 K변수를 직접 할당하여 사용하려고하였으나 뒤늦게 table.cpp에 존재한다는것을 알고 다시 진행하였습니다.

step 2.1)

```
~/De/정 /lect8/desCode-template ./a.out ✔ 16:27:59
after permute_8_7. dest (size:56) is
1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0
0 1 1 0 0 1 1 1 1 0 0 0 1 1 1 1
```

처음 step 2.1)을 통하여 PC-1 배열을 활용하여 K+배열을 만들어주었고 이 K+배열은 다음과같이 출력문에서 확인할 수 있었습니다.

step 2.2)
keysched.cpp 파일에서 step2.2)단계에서 splitKPlus(KPlus, C[0], D[0])함수를 찾았고 이를 구현하였습니다.

```
// step 2.2. split KPlus into 28-bit C0 and 28-bit D0
char C[17][28], D[17][28];
split_KPlus(KPlus, C[0], D[0]);
```

```
void split_KPlus(char KPlus[], char C0[], char D0[]){
// split kplus into c0 and d0
   // ......code........

    int i = 0;
   for(i=0;i<28;i++)
       C0[i] = KPlus[i];
   for(int j=0;j<28;j++)
       D0[j] = KPlus[i++];
   printf("after split KPlust\n");
   show_CD(C0, D0);
}
```

step 2.3)

```
// step 2.3. compute Cn, Dn for n=1,2,...16
comp_Cn_Dn(C, D, ROL);
```

역시나 keysched.cpp 파일에 construct_key_schedule 함수 내부에서 발견하였으며 다음과같이

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Iteration Number i번째마다 Left Shifts 횟수가 있었고 이를 토대로 코드를 구현하였습니다.

```
49 void comp_Ci_Di(char C[17][28], char D[17][28], int i, int ROL[]){
50 // compute C[i], D[i] from C[i-1], D[i-1] using ROL[i]
51    // ........... code ...........
52    // start index 1
53     if (ROL[i] == 1) // shift 1
54     {
55         int Ctemp = C[i-1][0];
56         int Dtemp = D[i-1][0];
57         for(int j=1;j<28;j++)
58         {
59             C[i][j-1] = C[i-1][j];
60             D[i][j-1] = D[i-1][j];
61         }
62         C[i][27] = Ctemp;
63         D[i][27] = Dtemp;
64     }
65     else //shift 2
66     {
67         int Ctemp1 = C[i-1][0];
68         int Ctemp2 = C[i-1][1];
69         int Dtemp1 = D[i-1][0];
70         int Dtemp2 = D[i-1][1];
71         for(int j=2;j<28;j++)
72         {
73             C[i][j-2] = C[i-1][j];
74             D[i][j-2] = D[i-1][j];
75         }
76         C[i][26] = Ctemp1;
77         C[i][27] = Ctemp2;
78         D[i][26] = Dtemp1;
79         D[i][27] = Dtemp2;
80     }
81 }
```

1,2,9,16 번째에는 쉬프트가 한번 일어나고 그 외에는 전부 두번 일어났기 때문에 첫번째 쉬프트에서는 Ctemp, Dtemp 라는변수에 값을 저장해두고 옮겨 닮은 후에 다시 저장해두었던값을 할당해주었습니다. 같은맥락으로 두번 쉬프트가 일어날 경우도 동일한 틀로 구현하였습니다.

```
1th CD
11100001100110010101010111
10101010110011001111000111102th CD
11000011001100101010101111
01010101100110011111000111013th CD
00001100110010101010111111
01010110011001111000111010
4th CD
00110011001010101011111111
01011001100111100011110101
5th CD
11001100101010101111111100
01100110011110001111010101
6th CD
00110010101010111111110000
10011001111000111101010101
7th CD
11001010101011111111000011
01100111100011110101010110
8th CD
00101010101111111100001100
10011110001111010101011001
9th CD
01010101011111111000011001
00111100011110101010110011
10th CD
01010101111111100001100110
11110001111010101010011001
11th CD
01010111111110000110011001
11000111101010101100110011
12th CD
01011111111000011001100101
00011110101010101100110011
13th CD
01111111100001100110010101
01111010101010110011001111
14th CD
11111110000110011001010101
11101010101011001100111100
15th CD
11111000011001100101010111
10101010101100110011110001
16th CD
11110000110011001010101011
01010101011001100111100011
```

Ci, Di 출력결과.

step 2.4)

```
// step 2.4. compute Kn by applying PC_2 to CnDn
comp_keys(C, D, PC_2, keys);
```

```
void comp_keys(char C[17][28], char D[17][28], int PC_2[8][6], char
keys[17][48]){
   char CD[56];
   for(int i=1;i<=16;i++){
      combine_arr(C[i], D[i], CD, 28);
      permute_8_6(PC_2, CD, keys[i]);
   }
   printf("displaying keys\n");
   for(int i=0;i<17;i++){
      printf("K%d=",i);
      show_arr(keys[i], 48);
   }
}
```

step 2.4)에서 PC-2 배열을 이용하여 어레이를 합치고 계산하는과정을 확인할 수 있었습니다.

```
displaying keys
K0=1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 16 0 0 0 0 0 0 0 0 80 -61 9 1 0 0 0 0 0 0 0 0 0 0 0 88 64 -13 22 1 0 0 0
K1=0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 1 0
K2=0 1 1 1 1 0 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 0 1
K3=0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1
K4=0 1 1 1 0 0 1 0 1 0 1 0 1 1 0 1 1 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 1
K5=0 1 1 1 1 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 0 0 1 1 1 0 1 0 1 0 0 0
K6=0 1 1 0 0 0 1 1 1 0 1 0 0 1 0 1 0 0 1 1 1 1 1 1 0 0 1 0 1 0 0 0 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 1
K7=1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 1 0 0
K8=1 1 1 1 0 1 1 1 1 1 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1
K9=1 1 1 0 0 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 1
K10=1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 1 1
K11=0 0 1 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1 0 0 1 1 1 0 0 0 0 1 1 0
K12=0 1 1 1 0 1 0 1 0 1 1 1 0 0 0 1 1 1 1 1 0 1 0 1 1 0 0 1 0 1 0 0 0 1 1 0 0 1 1 1 1 1 1 0 1 0 0 1
K13=1 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 0 0 1 0 0 0 0 0 1
K14=0 1 0 1 1 1 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 1 0
K15=1 0 1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 0 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0
K16=1 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1
```

맨 위부터 각 K가 계산된 출력문을 확인할 수 있었습니다.
step 3.1) Initial permutation with IP
enc.cpp 파일내부에서 des_encrypt함수에 step 3.1)을 찾을 수 있었습니다.

```
// step 3.1. permute M into MPlus using IP matrix
char MPlus[64];
permute_8_8(IP, M, MPlus);
```

그리고 permute_8_8 함수의 결과로 다음의 출력결과를 얻을 수 있었습니다.

```
after permute_8_8. dest (size:64) is
1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 1 1 1 0
0 0 0 1 0 1 0 1 0 1 0
```

M+=1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010
1111 0000 1010 1010

강의노트의 M+ 값과 비교해보았고 정상적으로 맞게 출력된 것을 확인할 수 있었습니다.

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

L_0 = 1100 1100 0000 0000 1100 1100 1111 1111
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

```
after split into L0R0.
L: 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1
R: 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0
```

L0,R0 는 32비트씩 M+ 배열을 나눠서가졌기 때문에 코드는 다음과같이 구성하였습니다.

```
28 void split_into_L0R0(char MPlus[], char L[], char R[], int len){
29    // ........ code ............
30    // 32, 32 bit divide
31    int i = 0;
32    for (i = 0;i < 32; i++)
33        L[i] = MPlus[i];
34    for (int j = 0;j < 32; j++)
35        R[j] = MPlus[i++];
36    printf("after split into L0R0.\n");
37    show_LR(L, R);
38 }
```

step 3.3) Compute Ln and Rn from L(n-1) and R(n-1) as follows

Ln=R(n-1)
Rn=L(n-1) xor f(R(n-1), Kn)

example: for n=1,
K1=000110 110000 001011 101111 111111 000111 000001 110010
L1=R0=1111 0000 1010 1010 1111 0000 1010 1010
R1=L0 xor f(R0, K1)

function f:
R0 ==> E(R0) ==> E(R0) xor K1 ==> B1 B2 B3 B4 B5 B6 B7 B8
==>SB = S1(B1) S2(B2) S3(B3) S4(B4) S5(B5) S6(B6) S7(B7) S8(B8)
==> P(SB)

```
55 void comp_Li_Ri(char L[17][32], char R[17][32], int i, int len, char *key){
56 // compute L[i], R[i] from L[i-1], R[i-1] with key. L, R has length len
57
58    // compute L[i]: copy R[i-1] into L[i]
59    int j;
60    for(j=0;j<len;j++) L[i][j]=R[i-1][j];
61
62    // compute R[i]: R[i]= L[i-1] xor f(R[i-1], key)
63    char f[32];
64    comp_f(R[i-1], key, f);
65    xoring(L[i-1], f, R[i], 32);
66 }
```

```
showing 1-th LR. L is
L: 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0
R: 1 1 0 0 1 1 0 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1
```

16번의 루프를 통해 각 L,R 을 출력구문으로 확인할 수 있었습니다.
더불어 위 코드의 64,65 번째 줄을 통하여 comp_f(R(n-1),Kn) 동작을 하고 xor 을 L(n-1)과 하는 작업을 확인할 수 있었습니다.

comp_f 함수 내부에서 split_B 함수를 호출하는것을 확인할 수 있었습니다.

```
 2 void comp_f(char *R, char *key, char *f){
 3 // compute f=f(R, key)
 4
 5    // step 3.3.1. expand 32-bit R into 48-bit ER using E matrix
 6    char ER[48];
 7    permute_8_6(E, R, ER);
 8
 9    // step 3.3.2. xor ER with key
10    char ERp[48]; // result of ER xor key
11    xoring(ER, key, ERp, 48);
12
13    // step 3.3.3. split ERp into B1, B2, ..., B8
14    char B[8][6];
15    split_B(ERp, B); // split 48-bit ERp into 6-bit B1, B2, ..., B8
16    printf("aftr split B. B is\n");
17    showB(B);
18
19    // step 3.3.4. Convert 6-bit B[i] into 4-bit SB[i]
20    char SB[8][4];
21    char totalSB[32];
22    do_sbox(B, SB, Sbox); // convert B1 to SB1, B2 to SB2, ...
23    printf("after do sbox. SB is\n");
24    show_SB(SB);
25
26    SBmerge(SB, totalSB); // merge SB1, SB2, .., SB8 into totalSB
27    printf("after SB merge. totalSB is\n");
28    show_arr(totalSB, 32);
29
30    // step 3.3.5. final permutation
31    permute_8_4(P, totalSB, f);
32
33 }
```

split_B 함수 내부코드에서 ERp 배열의 각 원소를 B배열의 할당하였습니다.

```
60 void split_B(char ERp[], char B[8][6]){
61    // ............ code
62    int k = 0;
63    for(int i=0;i<8;i++)
64        for(int j=0;j<6;j++)
65         B[i][j] = ERp[k++];
66 }
```

step 3.3.4) 부분의 do_sbox 내부를 살펴보았습니다.

```
// step 3.3.3. split ERp into B1, B2, ..., B8
char B[8][6];
split_B(ERp, B); // split 48-bit ERp into 6-bit B1, B2, ..., B8
printf("aftr split B. B is\n");
showB(B);

// step 3.3.4. Convert 6-bit B[i] into 4-bit SB[i]
char SB[8][4];
char totalSB[32];
do_sbox(B, SB, Sbox); // convert B1 to SB1, B2 to SB2, ...
printf("after do sbox. SB is\n");
show_SB(SB);
```

step 3.3.4)에서 6bit의 B[i]를 4bit의 SB[i]로 변환하는작업이 필요하였고 더불어 Sbox행렬을 활용하여 변환해야 했기 때문에 row와 col을 추출하는 함수 그리고 비트를 변환하는 함수를 이용하였습니다.

row는 맨앞과 뒤에 2개의비트를 그리고 col은 처음과끝을 제외한 나머지비트를 계산해주었습니다.

```
 73 void do_sbox_i(char B[8][6], char SB[8][4], int Sbox[8][4][16], int i){
 74 // comp SB[i] from B[i] using Sbox[i]
 75    // ............. code
 76     int row, col;
 77     row = comp_row_from_B(B[i]);
 78     col = comp_col_from_B(B[i]);
 79     convert_val_to_4bit(Sbox[i][row][col], SB[i]);
 80 }
 81
 82 int comp_row_from_B(char B[]){
 83 // row is at B[0] and B[5]
 84    int row;
 85    row=B[0];
 86    row=row*2+B[5];
 87    return row;
 88 }
 89 int comp_col_from_B(char B[]){
 90 // col is at B[1] to B[4]
 91    int col=0,i;
 92    for(i=1;i<5;i++){
 93       col=col*2 + B[i];
 94    }
 95    return col;
 96 }
 97 void convert_val_to_4bit(int SBval, char SB[]){
 98    int temp=SBval;
 99    for(int i=3;i>=0;i--){
100       SB[i]=temp%2;
101       temp=temp/2;
102    }
103 }
```

```
aftr split B. B is
0 1 1 0 0 0
0 1 0 0 0 1
0 1 1 1 1 0
1 1 1 0 1 0
1 0 0 0 0 1
1 0 0 1 1 0
0 1 0 1 0 0
1 0 0 1 1 1
after do sbox. SB is
0 1 0 1
1 1 0 0
1 0 0 0
0 0 1 0
1 0 1 1
0 1 0 1
1 0 0 1
0 1 1 1
after SB merge. totalSB is
0 1 0 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 1 0 0 1 0 1 1 1
```

split B와 Sbox를 통해 계산된 totalSB를 확인할 수 있었습니다.

step 3.3.5)

```
// step 3.3.5. final permutation
permute_8_4(P, totalSB, f);
```

final permutation으로 permute_8_4()함수를 호출하고 P배열과 totalSB 를이용하여 최종적으로 f값에 계산하여값을 할당해주는 작업을 하였습니다.

step 3.4)

```
// step 3.4. reverse L16 and R16, and apply final permutation
reverse(L[16], R[16]);
char LR[64];
combineLR(L[16], R[16], LR);
permute_8_8(IP_1, LR, cipher);
```

step 3.4)에서는 이렇게 위 step3.3)과정으로 구한 f(R0,K1)값을 이용하여 reverse를 시키고 IP-1 배열을 활용하여 최종적으로 cipher text를 구하게됩니다.

```
67 void reverse(char L[], char R[]){
68    // ........... code ..............
69    // reverse 32bit L_array, R_array
70    char copy_temp[32];
71
72    copy_arr(copy_temp, L, 32);
73    copy_arr(L, R, 32);
74    copy_arr(R, copy_temp, 32);
75    printf("after reverse. L and R is\n");
76    show_LR(L, R);
77 }
78 void combineLR(char L[], char R[], char LR[]){
79 // combine L and R into LR
80    // ............code ..............
81    int index;
82    for(index = 0; index < 32; index++)
83        LR[index] = L[index];
84    for(int j = 0; j < 32; j++)
85        LR[index++] = R[j];
86    printf("after combineLR\n");
87    show_arr(LR, 64);
88 }
```

copy_temp배열을 이용하여 32비트의 배열을 copy_arr 함수를 이용하여 복사해주었고 combineLR에서는 기존의 붙여주었던 방식과 동일하게 LR배열에 순차적으로 값을 할당해주었습니다.

```
showing 16-th LR. L is
L: 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0
R: 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 0 1 0 1
after reverse. L and R is
L: 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 0 1 0 1
R: 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0
after combineLR
0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1
0 0 0 1 1 0 1 0 0
after permute_8_8. dest (size:64) is
1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0
0 0 0 0 0 0 1 0 1
cipher:
1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0
0 0 0 0 0 0 1 0 1
```

마지막 출력은 다음과같이 확인할 수 있었습니다.