정보보호개론 Assignment 2

2016310936 우승민

이번 과제는 AES-128(DES(plain))되어 암호화 된 cipher를 plain과 함께 비교하여, 각각에 사용된 key를 찾는 것입니다. 우선 header file과 전역변수를 보면 아래와 같습니다.

des와 aes의 암호화 알고리즘을 사용하기위해 openssl의 header파일들을 추가해주었고, plaintext, ciphertext, MD5 value, MD5 key를 저장하기 위한 공간들을 전역변수로 선언해주었습니다.

main문에서 전역변수에 선언하였던 변수들에 malloc을 사용하여 공간을 할당하는 것입니다.

plain의 경우는 처음 64byte만을 암호화한 후 비교할 것이기 때문에 64의 크기로 할당하였고, cipher는 plain의 64byte를 암호화한 결과를 여유롭게 받기위해 512의 크기로 할당하였습니다.

pass와 key의 경우는 passwords.txt의 총 key의 개수가 184389이기 때문에 각각 이에 맞게 할당하였고, 글자수 크기는 최대인 32에 맞추어서 할당해주었습니다.

다음으로 plain과 cipher에 각 크기에 맞게 입력 받는 과정입니다.

```
66 ^|for(i=0; fscanf(fpIn, "%c", &ch) != EOF && i<64; ++i){  //plaintext의 64byte -> plain에 저장$
67 ^|^|if(ch == '₩n') break;$
68 ^|^|plain[i]=ch;$
69 ^|}$
70 ^|if(i<64) plain[i]='₩0';$
71 ^|if(i==64){^|^|//plaintext의 크기가 64byte보다 크면 ciphertext가 올 때까지 넘김$
72 ^|^|for(i=0; fscanf(fpIn, "%c", &ch) != EOF; ++i){$
73 ^|^|^|if(ch == '₩n') break;$
74 ^|^|}$
75 ^|}$
76 $
77 ^|$
78 ^|int cnt = 0;$
79 ^|for(i=0; fscanf(fpIn, "%c", &ch) != EOF && i<512; ++i){  //ciphertext의 512byte -> cipher에 저장$
80 ^|^|cipher[cnt]=ch;$
81 ^|^|cnt++;$
82 ^|}$
83 ^|cipher[cnt]='₩0';$
84 ^|fclose(fpIn);$
```

만약 입력 받는 plaintext가 64byte보다 작으면, for문이 종료되고 마지막 공간에 '₩0'을 넣어주었습니다. 반대로 64byte보다 크면 64 byte만큼만 저장하고 이후는 '₩n' 즉 cipher 문이 나올 때까지 넘겨주었습니다.

cipher도 마찬가지로 최대 512 byte만큼 저장하고 마지막에 '₩0'을 넣어주었습니다. 이제는 'PlaintextCiphertext.txt'의 용도가 끝났음으로 file을 종료해주었습니다.

```
87 ^|int j=0;$
88 ^|for(i=0; i<184389; i++){^|^|//passwords.txt의 value와 key를 각각 pass와 key에 저장$
89 ^|^|int cnt = 0;$
90 ^|^|for(; fscanf(fpPs, "%c", &ch) != EOF; ++j){$
91 ^|^|^|if(ch == ' ') break;$
92 ^|^|^|pass[i][cnt++] = ch;$
93 ^|^|}$
94 ^|^|cnt = 0;$
95 ^|^|for(; fscanf(fpPs, "%c", &ch) != EOF; ++j){$
96 ^|^|^|if(ch == '₩n') break;$
97 ^|^|^|key[i][cnt++] = ch;$
98 ^|^|}$
99 $
100 ^|}$
101 ^|fclose(fpPs);$
```

마찬가지로 MD5의 value와 key값도 for문을 통해서 각각 pass array와 key array에 순서대로 저장하였습니다. 이것 또한 저장이후 'passwords.txt'를 사용할 필요가 없음으로 종료해주었습니다.

다음은 des 암호화 이전에 input할 plain문을 des_in array에 옮겨주는 과정입니다.

```
unsigned char des_in[8][8];//des_ebc 암호화의 input 생성 64byte를 8 * 8byte로 쪼갬
for(int i=0; i<8; i++) memset(des_in[i],0,8);

for(int i=0; i<8; i++) {
    for(int j=0; j<8; j++) {
        if(plain[i*8+j]) des_in[i][j] = plain[i*8+j];
        else des_in[i][j]=0;
    }
}
```

openssl의 des 함수가 8byte씩 나누어서 이루어졌기 때문에 64byte의 plain을 각각 8*8byte로 나누어서 옮겨주었습니다.

이제는 본격적으로 암호화하는 과정입니다. 제가 구현한 code는 des와 aes 모두 사용된 key의 쌍을 찾기 위해서 for문을 2번 사용하여 바깥쪽 for문은 aes 암호화를 안쪽 for문은 des 암호화를 진행합니다. 각 for문은 모든 MD5의 key를 확인하기위해 184389번씩 반복합니다.

$$AES\_CBC128_{k_i}\left(DES_{ECB_{k_j}}(des_{in})\right)$$

```
for(int i=0; i<184389; i++){//밖의 for문은 aes_cbc 암호화 반복문
    unsigned char aes_key[16];
    char ch3[3];
    int num2;
    for(int k=0; k<16; k++){//aes 암호화의 key값 저장 -> aes_key
        ch3[0] = pass[i][k*2+0];
        ch3[1] = pass[i][k*2+1];
        ch3[2] = '₩0';
        num2 = strtol(ch3, NULL, 16);
        aes_key[k] = num2;
    }

    for(int j=0; j<184389; j++){//안의 for문은 des_ebc 암호화 반복문
        char ch2[3];
        int num1;
        DES_cblock key1;
        for(int k=0; k<8; k++){//des 암호화의 key값 저장 -> key1
            ch2[0] = pass[j][k*2+0];
            ch2[1] = pass[j][k*2+1];
            ch2[2] = '₩0';
            num1 = strtol(ch2, NULL, 16);
            key1[k] = num1;
        }
```

각 for문의 시작할 때는 aes_key에는 aes_cbc에서 사용할 key를 key1에는 des_ebc에서 사용할 key를 저장합니다. pass array에는 key가 char형으로 저장되어 있으므로 strtol함수를 사용하였습니다.

이후 des_ebc 의 암호화를 진행합니다. des_out 도 des_in 과 마찬가지로 8*8byte 형태로 만들어주었습니다.

```
unsigned char des_out[8][8];//des 암호화 output 생성
for(int i=0; i<8; i++) memset(des_out[i],0,8);

DES_key_schedule keysched;

DES_set_key(&key1, &keysched);
for(int k=0; k<8; k++) {//des 암호화과정 8byte나눈 것을 8번 반복 des_in -> des_out
DES_ecb_encrypt((DES_cblock*)des_in[k],(DES_cblock*)des_out[k], &keysched, DES_ENCRYPT);
}

unsigned char aes_input[64];//aes 암호화 input 생성
memset(aes_input,0,64);
for(int k=0; k<8; k++){
for(int l=0; l<8; l++){//des의 output을 aes input으로 옮김
aes_input[k*8+l] = des_out[k][l];

}
}
```

DES_ecb_encrypt 함수를 8번 반복하여 des_out이 나오면 aes_cbc 암호화를 진행하기위해 다시 aes_input에 des_out의 값을 옮겨주었습니다.

바로 이어서 aes_cbc의 암호화를 진행합니다. initialization vector를 NULL값으로 생성해준 후 암호화를 진행하였습니다. 이어서 바로 암호화된 enc_out을 base64 형식으로 변환해주었습니다.

```
unsigned char iv[AES_BLOCK_SIZE];//initalization vector 생성 NULL으로 사용
memset(iv, 0x00, AES_BLOCK_SIZE);

unsigned char enc_out[sizeof(aes_input)];//aes 암호화 output 생성
memset(enc_out, 0, sizeof(enc_out));
AES_KEY key2;
AES_set_encrypt_key(aes_key, sizeof(aes_key)*8, &key2); // aes_key -> key2로 변환

// aes_cbc 암호화 aes_input -> enc_out
AES_cbc_encrypt(aes_input, enc_out, sizeof(aes_input), &key2, iv, AES_ENCRYPT);
int k=0;

int encode_size = 4 * ((sizeof(enc_out)+ 2) / 3);//base64 변환 공간 할당

unsigned char base_out[encode_size];

base64_encode(enc_out, base_out, sizeof(enc_out));//base64 변환 enc_out -> base_out
```

base64로 변환된 크기는 8bit를 6bit 형식으로 변환하는 것이기 때문에 $\frac{4}{3}$ 배 해주었습니다. base64_encode 함수는 openssl의 header 파일에서 base64 encoding을 찾지 못해 직접 구현하였습니다.

```
int base64[64] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '/'};
```

우선 base64 array에 base64에서 사용되는 문자를 모두 입력해주었습니다.

base64_encode 함수 code입니다. 변환할 plain과 변환 후 저장될 encode, plain의 크기를 인자로 넘겨주었습니다.

```
15 void base64_encode(unsigned char *plain, unsigned char *encode, int plain_size){$
16 $
17 ^Iint i=0;$
18 ^Iint j=0;$
19 ^I$
20 ^Iwhile(i < plain_size-2){^I^I^I^I^I^I^I// 3개의 plain -> 4개의 encode$
21 ^I^Iencode[j++] = base64[plain[i] >> 2];$
22 ^I^Iencode[j++] = base64[((plain[i] << 4) & 0x30) | (plain[i+1] >> 4)];$
23 ^I^Iencode[j++] = base64[((plain[i+1] << 2) & 0x3c) | (plain[i+2] >> 6)];$
24 ^I^Iencode[j++] = base64[(plain[i+2] & 0x3f)];$
25 ^I^Ii+=3;$
26 ^I}$
27 ^I$
28 ^Iif(i == plain_size-2){^I^I^I^I^I^I^I^I^I^I// 2개의 plain + padding -> 4개의 encode$
29 ^I^Iencode[j++] = base64[plain[i] >> 2];$
30 ^I^Iencode[j++] = base64[((plain[i] << 4) & 0x30) | (plain[i+1] >> 4)];$
31 ^I^Iencode[j++] = base64[((plain[i+1] << 2) & 0x3c)];$
32 ^I^Iencode[j++] = '='; $
33 ^I$
34 ^I}$
35 ^I$
36 ^Ielse{^I^I^I^I^I^I^I^I^I^I^I^I^I^I^I^I^I^I// 1개의 plain + padding -> 4개의 encode$
37 ^I^Iencode[j++] = base64[plain[i] >> 2];$
38 ^I^Iencode[j++] = base64[((plain[i] << 4) & 0x30)];$
39 ^I^Iencode[j++] = '=';$
40 ^I^Iencode[j++] = '=';$
41 ^I}$
42 $
43 }$
```

기본적으로 base64 encoding은 8bit를 6bit로 변환하는 것임으로 3개 * 8bit -> 4개 * 6bit 형식으로 이루어집니다. 그렇기에 대부분의 변환은 while문을 통해 이루어집니다. 이외에 plain의 bit가 3으로 나누어 떨어지지 않을 경우를 대비해 if문과 else문을 통해 나머지 상황도 encoding 해주었습니다.

이후 다시 main문으로 넘어와서 암호화가 완료된 bit들을 cipher와 비교하는 과정만이 남았습니다. 우선 저는 40byte만큼 동일하게 암호화가 되었으면 올바른 key들을 찾았다고 생각하여 최대 비교를 40으로 설정하였고, 만약 base_out이나 cipher문의 크기가 더 작을 경우에는 더 작은 값으로 바꾸어 주었습니다.

```
177 ^I^I^Ibase64_encode(enc_out, base_out, sizeof(enc_out));^I//base64 변환 enc_out -> base_out$
178 ^I^I^I$
179 ^I^I^Ik = 0;$
180 ^I^I^Iint diff=40;^I^I^I//암호화 후 비교크기를 최대 40으로 설정$
181 ^I^I^I$
182 ^I^I^Iwhile(base_out[k++]);^I^I$
183 ^I^I^Iif(diff > k) diff = k;^I//만약 base_out이나 cipher 중에 size가 40보다 작은 것이 있으면 변경$
184 ^I^I^Ik = 0;$
185 ^I^I^I$
186 ^I^I^Iwhile(cipher[k++]);$
187 ^I^I^Iif(diff > k) diff = k;$
188 ^I^I^Ik = 0;$
```

diff값이 올바르게 암호화 되었다고 판단하는 최솟값인 것입니다.

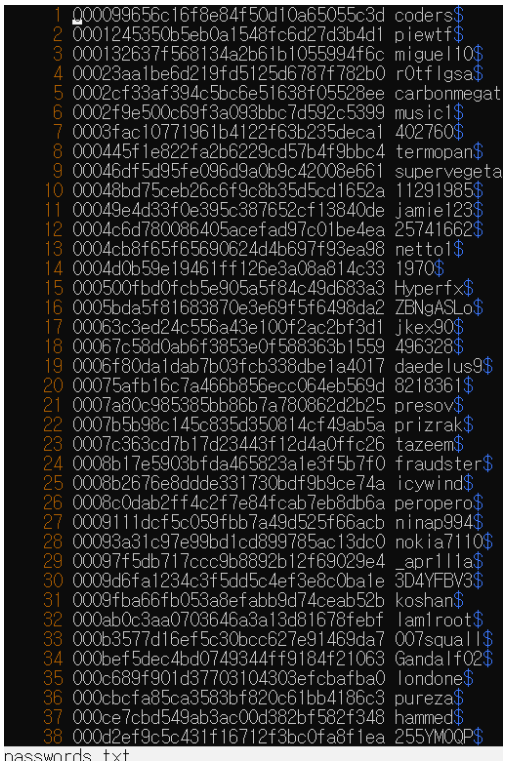
최종적으로 base_out과 cipher를 비교하는 과정입니다.

```
^I^I^Iwhile(base_out[k] && base_out[k] == cipher[k]){  //diff길이만큼 base_out과 cipher를 비교$
^I^I^I^Iif(k > diff){^I^I//만약 diff까지 왔으면 암호화가 제대로 이루어짐 -> key값 찾음$
^I^I^I^I^Ichar x[40];$
^I^I^I^I^Iint I=0;$
^I^I^I^I^Iwhile(key[j][I]){^I^I//x에 key1, key2 차례로 옮긴 후 fpOut에 출력$
^I^I^I^I^I^Ix[I] = key[j][I];$
^I^I^I^I^I^II++;$
^I^I^I^I^I}$
^I^I^I^I^Ix[I]='₩0';$
^I^I^I^I^Ifwrite(x,I, 1, fpOut);$
^I^I^I^I^Ich = '₩n';$
^I^I^I^I^Ifwrite(&ch, sizeof(ch), 1, fpOut);$
^I^I^I^I^II=0;$
^I^I^I^I^Iwhile(key[i][I]){$
^I^I^I^I^I^Ix[I] = key[i][I];$
^I^I^I^I^I^II++;$
^I^I^I^I^I}$
^I^I^I^I^Ix[I]='₩0';$
^I^I^I^I^Ifwrite(x,I, 1, fpOut);^I$
^I^I^I^I^Ifclose(fpOut);$
^I^I^I^I^I$
^I^I^I^I^Ifor(int I=0; I<184389; I++){^I// malloc으로 할당해준 공간 free$
^I^I^I^I^I^Ifree(pass[I]);$
^I^I^I^I^I^Ifree(key[I]);$
^I^I^I^I^I}$
^I^I^I^I^Ifree(pass);$
^I^I^I^I^Ifree(key);$
^I^I^I^I^Ifree(plain);$
^I^I^I^I^Ifree(cipher);$
$
^I^I^I^I^Ireturn 0;^I^I// 빠른 실행 종료를 위해 바로 return$
^I^I^I^I}$
^I^I^Ik++;$
^I^I}$
^I}$
$
}$
```

위에서 최소한으로 설정하였던 diff값의 기준을 통과하였다면, x array에 암호화에 사용하였던 key 값을 차례로 입력한 후 fwrite를 통해 'Keys.txt"에 입력하게 해주었습니다. 이후 모든 과정이 끝났음으로 for문을 계속 진행할 필요가 없기에 malloc으로 할당해주었던 공간들을 모두 free해준 후 return을 통해 main문이 바로 종료되게 해주었습니다.

```
^Ifclose(fpOut);^I^I//만약 key를 찾지 못했어도 할당한 공간 free 후 종료$
^Ifor(i=0; i<184389; i++){$
^I^Ifree(pass[i]);$
^I^Ifree(key[i]);$
^I}$
^Ifree(pass);$
^Ifree(key);$
^Ifree(plain);$
^Ifree(cipher);$
$
^Ireturn 0;$
}$
```

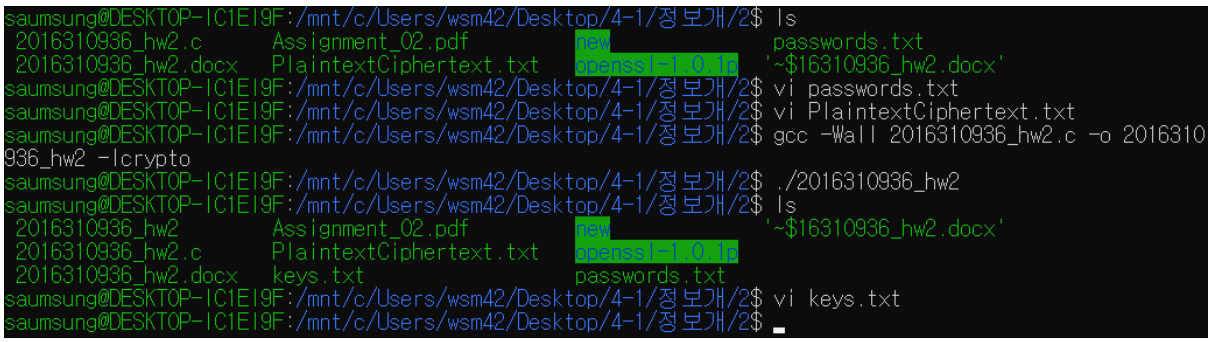
만약 for문을 모두 진행했음에도 찾지 못하면, 그대로 할당 공간들을 free한 후 종료하였습니다.



현재 'passwords.txt'에 왼쪽과 같이 저장 되어있고

'PlaintextCiphertext.txt'에는 아래와 같이 저장되어 있습니다.

다음으로 code를 compile한 후 실행하면 다음과 같이 keys.txt가 생성되는 것이 확인 가능합니다.



keys.txt를 보면 아래와 같이 key가 출력되는 것을 확인할 수 있습니다.