Assignment 2

2013312689 박찬열

Key1과 key2 값을 찾기 위하여 plaintext를 DES로 encrypt한 값과 AES로 decrypt 한 값을 비교하여 같은 값을 찾는 알고리즘을 통하여 key 값을 찾았다. 우선 내 프로그램을 돌리기 위해서 plaintext와 ciphertext가 있는 PlaintextCiphertext.txt 파일과, key값과 password값이 저장되어 있는 passwords.txt 파일이 필요하다. plaintextciphertext에서 plaintext와 base64로 encoding 되어 있는 ciphertext를 get_text로 읽어와 저장을 한 다음, change_base64 함수를 이용하여 outbase에 decoding 하였다. 그 뒤 plaintext의 앞 16바이트를 passwords.txt에 저장되어 있는 모든 key 값에 대하여 DES encrypt한 결과값과 그곳에 사용된 key값을 middes라는 구조체에 저장하였다. 그 뒤 encrypt한 결과값을 오름차순으로 정렬을 하였고, outbase 또한 des와 마찬가지로 모든 passwords.txt에 저장되어 있는 모든 key 값에 대하여 AES cbc decrypt를 진행한 뒤 앞 16바이트와 key값을 midaes에 저장한 뒤 오름차순 정렬을 해주었다. 그런 다음 모든 middes와 midaes를 비교하면서 같은 결과값을 찾을 경우 keys.txt 파일에 middes에 저장된 key를 출력, midaes에 저장된 key를 출력하였다. middes와 midaes 모두 정렬된 상태이기 때문에 비교하여 작은 쪽의 index값을 1늘려주는 방식을 사용하여 2*key값의 종류 시간에 모두 탐색 할 수 있었다.

```c
#define _CRT_SECURE_NO_WARNINGS
#define aes_enc 1		//AES_encrypt모드의 경우 1
#define aes_dec 0		//AES_decrypt 모드의 경우 0
#define base64_enc 1	//base64 encoding일 경우 1
#define base64_dec 0	//base64 decoding일 경우 0


#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<openssl/aes.h>
#include<openssl/des.h>

#define ALL_PW 184390

unsigned char base64[64];

//passwords.txt에서 key이름(name)과 password값(key)을 담기 위한 구조체
struct PWKEY
{
	unsigned char key[16];
	unsigned char name[20];
};

//ciphertext(midstr)와 어떤 key(key_name)을 썼는지 기록하기 위한 구조체
struct middlestring
{
	unsigned char midstr[17];
	unsigned char key_name[20];
};

//strcmp를 쓸 경우 '₩0'부분까지만 비교하므로 강제로 16자리를 비교하기 위한 함수
//앞 문자열이 더 크면 1, 더 작으면 -1, 같으면 0을 반환한다.
int new_strcpr(unsigned char* s1, unsigned char* s2)
{
	int i;
	for (i = 0; i < 16; i++)
	{
		if (s1[i] > s2[i])
			return 1;
		else if(s1[i] < s2[i])
			return -1;
	}
	return 0;
}

//stdlib에 있는 qsort함수를 사용하기 위한 compare함수
//더 작은 문자열이 앞에 오도록 오름차순 정렬
int compare_func(const void* v1, const void* v2)
{
	return (new_strcpr(((struct middlestring*)v1)->midstr, ((struct middlestring*)v2)->midstr));
}

//base[64]에 치환되는 문자를 넣기 위한 함수
```

```
void set_base64()
{
	int i;

	for (i = 0; i < 26; i++)
		base64[i] = 'A' + i;
	for (i = 0; i < 26; i++)
		base64[26 + i] = 'a' + i;
	for (i = 0; i < 10; i++)
		base64[52 + i] = '0' + i;
	base64[62] = '+';
	base64[63] = '/';
}

//base64를 이용하여 encoding (enc_dec == 1) 또는 decoding (enc_dec == 0)을 위한 함수
void change_base64(unsigned char* input, unsigned char* output, unsigned int enc_len, unsigned
int enc_dec)
{
	int i;
	int carry;
	int idx;
	unsigned char tmp;
	unsigned char j;


	idx = 0;
	carry = 0;
	if (enc_dec)	//encoding의 경우 6비트씩 끊어서 base64배열을 이용하여 치환
	{
		for (i = 0; i < enc_len; i++)
		{
			if (i % 3 == 0)
			{
				output[idx] = input[i] / 4;
				output[idx] = base64[output[idx]];
				idx++;
				output[idx] = ((input[i] % 4) << 4);
			}
			if (i % 3 == 1)
			{
				output[idx] += input[i] / 16;
				output[idx] = base64[output[idx]];
				idx++;
				output[idx] += (input[i] % 16) * 4;
			}
			if (i % 3 == 2)
			{
				output[idx] += input[i] / 64;
				output[idx] = base64[output[idx]];
				idx++;
				output[idx] += (input[i] % 64);
				output[idx] = base64[output[idx]];
				idx++;
			}
		}
```

```
        }
        else    //decoding의 경우 6비트씩 끊어진걸 8비트씩 모아 base[64]를 이용하여 치환
        {
                for (i = 0; i < strlen(input); i++)
                {
                        for (j = 0; j < 64; j++)
                                if (base64[j] == input[i])
                                        tmp = j;

                        if (i % 4 == 0)
                        {
                                output[idx] = tmp * 4;
                        }
                        if (i % 4 == 1)
                        {
                                output[idx] += tmp / 16;
                                idx++;
                                output[idx] += (tmp % 16) * 16;
                        }
                        if (i % 4 == 2)
                        {
                                output[idx] += (tmp / 4);
                                idx++;
                                output[idx] += (tmp % 4) * 64;
                        }
                        if (i % 4 == 3)
                        {
                                output[idx] += tmp;
                                idx++;
                        }
                }
        }
}

//FILE* table에서 password(key)와 key(name)을 읽어오는 함수
void get_password(unsigned char* key, unsigned char* name, FILE* table)
{
        int i;
        for (i = 0; i < 16; i++)
                fscanf(table,"%2x", (unsigned int*)&key[i]);
        fscanf(table, "%s", name);
        return;
}

//FILE* txt에서 plaintext(plain)와 ciphertext(cipher)를 읽어오는 함수
void get_text(unsigned char* plain, unsigned char* cipher, FILE* txt)
{
        int len;
        fgets(plain, 140000, txt);
        len = strlen(plain);
        plain[len - 1] = '₩0';
        fgets(cipher, 140000, txt);
        return;
}
```

```
//DES를 실행하는 함수
int do_des(unsigned char* plaintxt, unsigned char* ciphertxt, unsigned int msg_len, unsigned
char* key)
{
	int i;

	int check;

	unsigned int enc_len;

	DES_key_schedule des;

	//msg_len('₩0'포함)의 길이에 따라 enc_len을 설정
	//다만 msg_len으로 16밖에 넘겨주지 않기 때문에
	//아주 만약의 상황을 위해 남겨둔 코드
	if (msg_len % 16)
		enc_len = (msg_len / 16 + 1) * 16;
	else
		enc_len = msg_len;

	check = DES_set_key((C_Block*)key, &des);

	if (check < 0)
	{
		printf("Error Occured during DES₩n");
		return -1;
	}

	memset(ciphertxt, 0, 17);

	//ciphertext에 key값으로 des encrypt
	for (i = 0; i < enc_len; i += 8)
		DES_ecb_encrypt((C_Block*)(plaintxt + i), (C_Block*)(ciphertxt + i), &des,
DES_ENCRYPT);

	return enc_len;
}

//AES를 실행하는 함수
void do_aes(unsigned char* plaintxt, unsigned char* ciphertxt, unsigned char* key, unsigned int
enc_len, unsigned int enc_dec)
{
	int i;

	int check;

	unsigned char iv[16];

	AES_KEY aes;

	if (enc_dec == 1)
		check = AES_set_encrypt_key(key, 128, &aes);
	else
		check = AES_set_decrypt_key(key, 128, &aes);
```

```
        if (check < 0)
        {
                printf("Error Occured during AESWn");
                return;
        }

        memset(iv, 0, 16);
        memset(ciphertxt, 0, enc_len);

        //plaintxt를 key값으로 encrypt 또는 decrypt
        AES_cbc_encrypt(plaintxt, ciphertxt, enc_len, &aes, iv, enc_dec);
}

int main()
{
        FILE* hashtable;
        FILE* plcitxt;
        FILE* result;

        struct PWKEY enc_key;
        struct middlestring* middes;
        struct middlestring* midaes;

        unsigned int enc_len;
        unsigned char* plntxt;
        unsigned char* outbase;
        unsigned char* cphtxt;

        int i, j, k;
        int index1;
        int index2;
        int check;

        AES_KEY aes;

        middes = (struct middlestring*)malloc(sizeof(struct middlestring)*ALL_PW);
        //des결과값을 저장
        midaes = (struct middlestring*)malloc(sizeof(struct middlestring)*ALL_PW);
        //aes결과값을 저장

        plntxt = (char*)malloc(sizeof(char) * 100100);     //plaintext 저장
        outbase = (char*)malloc(sizeof(char) * 100100);    //base64로 decoding된 값을저장
        cphtxt = (char*)malloc(sizeof(char) * 140000);     //ciphertext 저장

        hashtable = fopen("passwords.txt", "r");
        plcitxt = fopen("PlaintextCiphertext.txt", "r");
        result = fopen("keys.txt", "wb");

        //plaintext,ciphertext를 읽어옴
        get_text(plntxt, cphtxt, plcitxt);

        //ciphertext를 base64로 decoding
        memset(outbase, 0, 100100);
        set_base64();
        change_base64(cphtxt, outbase, 100100, base64_dec);
```

```
	//모든 key값으로 plaintext의 앞 16바이트에 대하여 DES encrypt를 실행
	for (i = 0; i < ALL_PW; i++)
	{
		memset(middes[i].midstr, 0, 17);
		memset(enc_key.name, 0, 20);
		get_password(enc_key.key, enc_key.name, hashtable);
		strcpy(middes[i].key_name, enc_key.name);
		do_des(plntxt, middes[i].midstr, 16, enc_key.key);
	}
	//정렬
	qsort(middes, ALL_PW, sizeof(struct middlestring), compare_func);

	fseek(hashtable, 0L, SEEK_SET);

	//enc_len값 설정
	enc_len = strlen(plntxt) + 1;
	if (enc_len % 16)
		enc_len = (enc_len / 16 + 1) * 16;

	//모든 key값으로 outbase에 대하여 AES Decrypt를 실행한 후 앞 16바이트를 저장
	for (i = 0; i < ALL_PW; i++)
	{
		memset(midaes[i].midstr, 0, 17);
		get_password(enc_key.key, enc_key.name, hashtable);
		strcpy(midaes[i].key_name, enc_key.name);
		do_aes(outbase, cphtxt, enc_key.key, enc_len, aes_dec);
		for (j = 0; j < 16; j++)
			midaes[i].midstr[j] = cphtxt[j];
	}
	//정렬
	qsort(midaes, ALL_PW, sizeof(struct middlestring), compare_func);


	index1 = 0;
	index2 = 0;
	check = 0;

	//DES encrypt 결과 값과 AES decrypt 결과 값 중 같은걸 찾는다
	while (index1 < ALL_PW && index2 < ALL_PW)
	{
		check = new_strcpr(middes[index1].midstr, midaes[index2].midstr);
		if (check == 0)
		{
			//찾으면 keys.txt에 key1과 key2를 기록
			fprintf(result, "%s₩n%s", middes[index1].key_name,
midaes[index2].key_name);
			break;
		}
		else if (check > 0)
		{
			index2++;
		}
		else if (check < 0)
		{
```

```
                        index1++;
                }
        }

        //파일 스트림 닫기
        fclose(hashtable);
        fclose(plcitxt);
        fclose(result);

        //동적할당 해제
        free(middes);
        free(midaes);

        free(plntxt);
        free(cphtxt);
}
```