

컴퓨터보안 보고서

실습과제 #5주차

강좌 명: 컴퓨터보안

교수님: 정준호 교수님

학과: 컴퓨터공학과

학년: 3학년

학번: 2017112138

이름: 정여준

문제 1. 나만의 OTP 구현하기

OTP

<정의>

One Time Password의 준말로 일회용 패스워드이다. 즉, 매번 다른 비밀번호로 사용자를 인증하는 일회용 비밀번호를 의미한다. 주로 높은 수준의 보안을 유지하며 사용자를 인증해야 할 필요가 있을 때 사용된다. 평문과 랜덤한 비트열과의 XOR연산을 취해서 암호화하는 방법의 암호 알고리즘. 어떤 암호화 방법에도 키 공간을 모두 탐색하는 전사공격을 수행하면 해독이 가능하다는 취약점이 있지만 OTP는 예외로 키공간을 모두 탐색해도 해독이 불가능하다는 특징이 있다.

<암호화/복호화 과정>

암호화 하는 방법은 입력받은 평문에 생성한 패드에 있는 비트열과 XOR연산을 취하는 방법으로 암호화한다. XOR연산은 특이하게 동일한 비트열끼리 XOR연산을 취할 경우 0을 반환한다. 이러한 특징을 활용해서 암호화된 문장에 다시 똑같은 키 값으로 XOR연산을 취해서 평문으로 되돌리는 복호화방법을 사용한다.

<문제점>

OTP 방식별 단점이다.

난수표 - 난수표에 대한 복제가 용이하다.

클라이언트용 SW프로그램 - PIN 값을 알고 있을 경우 시스템 접속이 가능하다.
S/key 방식의 경우 상대적으로 암호화 수준이 낮다.

MOTP - 전용 휴대폰이 필요하다. 시스템구축 비용이 타 방식에 비해 높다.

SMS - 핸드폰 발송을 위한 시스템구축 비용 및 문자 메시지 전송 시 지속적인 비용이 발생한다.

<소스코드>

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;

string encryption(char text[], string name) //암호화
{
    ifstream ifs(name + ".txt"); //입력받은 이름의 텍스트파일을 연다
    if (!ifs) //패드를 찾지 못할 경우
    {
        string k = "에러";
        return k; // "에러" 리턴
    }
    else
    {
        char result[100] = "";
        int length = strlen(text);
        for (int i = 0; i < length; i++)
        {
            result[i] = text[i] ^ ifs.get(); //xor연산으로 암호화
        }
        ifs.close();
        return result;
    }
}

string decryption(char encrpyted[], string name) //복호화
{
    ifstream ifs(name + ".txt"); //입력받은 이름의 텍스트파일을 연다
    if (!ifs) //패드를 찾지 못할 경우
    {
        string k = "에러";
        return k; // "에러" 리턴
    }
    else
    {
        char result[100] = "";
        int length = strlen(encrpyted);
        for (int i = 0; i < length; i++)
        {
            result[i] = encrpyted[i] ^ ifs.get(); //xor연산으로 복호화
        }
        ifs.close();
        return result;
    }
}

int main()
{
    char text[100] = "";
    char encrypted[100] = "";
    string name = "";
    int mod = 0;
    do
    {
        cout << "One Time Pad" << endl;
        cout << "모드를 입력하세요 1:암호화 2:복호화 3:패드생성 4:종료 =>";
        cin >> mod;
        switch (mod)
        {
            case 1: //암호화
                cout << "암호화할 평문을 입력하세요: ";
                cin >> text;
                cout << "불러올 패드를 입력하시오: ";
                cin >> name;
                if (encryption(text, name) == "에러") //암호화 함수가 "에러"를 리턴하는 경우는
```

패드를 찾지 못했을 경우

```
{
    do{ //패드를 찾을때까지 패드 이름 입력
        cout << "패드를 찾을 수 없습니다." << endl;
        cout << "불러올 패드를 입력하십시오: ";
        cin >> name;
    } while (encryption(text, name) == "에러");
    cout << "암호문: " << encryption(text, name) << endl;
    cout << "===== " <<
endl;
}
else {
    cout << "암호문: " << encryption(text, name) << endl;
    cout << "===== " <<
endl;
}
break;
case 2: //복호화
    cout << "복호화할 암호문을 입력하세요: ";
    cin >> encrypted;
    cout << "불러올 패드를 입력하세요: ";
    cin >> name;
    if (decryption(encrypted, name) == "에러") //복호화 함수가 "에러"를 리턴하는
    경우는 패드를 찾지 못했을 경우
    {
        do { //패드를 찾을때까지 패드 이름 입력
            cout << "패드를 찾을 수 없습니다." << endl;
            cout << "불러올 패드를 입력하십시오: ";
            cin >> name;
        } while (decryption(encrypted, name) == "에러");
        cout << "암호문: " << decryption(encrypted, name) << endl;
        cout << "===== " <<
endl;
    }
    else {
        cout << "암호문: " << decryption(encrypted, name) << endl;
        cout << "===== " <<
endl;
    }
    break;
case 3: //패드생성
    {
        cout << "생성할 패드의 이름을 입력하세요: ";
        cin >> name;
        ofstream fout;
        fout.open(name + ".txt"); //입력한 이름의 텍스트파일 생성 => 생성된 텍스트파일
        패드 역할
        for (int i = 0; i < name.length(); i++)
        {
            fout.put(name[i]); //텍스트파일에 내용 삽입(텍스트파일 이름과 동일한 내용)
        }
        fout.close();
        cout << "패드 생성 완료!" << endl;
        cout << "===== " <<
endl;
        break;
    }
default: //예외
    cout << "다시 입력해주세요" << endl;
    cout << "===== " <<
endl;
    break;
} while (mod != 4); //4입력시 반복문 종료
cout << "프로그램이 종료됐습니다.";
return 0;
}
```

<소스코드 리뷰>

switch 문을 이용해서 사용하고자 하는 모드를 선택할 수 있게 구현하고, 패드를 생성할 때 .txt 확장자 파일을 생성해서 저장한다. 이러한 패드를 암호화와 복호화를 할 때 불러서 안에 있는 내용을 사용해서 암호화나 복호화를 진행한다. 만약 입력한 이름의 파일이 존재하지 않을 경우 에러가 발생해서 존재하는 패드의 이름을 찾을 때까지 반복해서 패드의 이름을 입력한다. 암호화 복호화 같은 경우에 암호화는 패드에 있는 내용과 평문에 XOR연산을 취해서 암호화하고 복호화 같은 경우에는 패드에 있는 내용과 암호문에 XOR연산을 취해서 복호화 한다.

<결과>

Microsoft Visual Studio 디버그 콘솔

```
One Time Pad
모드를 입력하세요 1:암호화 2:복호화 3:패드생성 4:종료 =>3
생성할 패드의 이름을 입력하세요: 19950701
패드 생성 완료!

=====

One Time Pad
모드를 입력하세요 1:암호화 2:복호화 3:패드생성 4:종료 =>1
암호화할 평문을 입력하세요: apple
암호화할 패드를 입력하시오: 19950701
암호문: P11YU

=====

One Time Pad
모드를 입력하세요 1:암호화 2:복호화 3:패드생성 4:종료 =>2
복호화할 암호문을 입력하세요: P11YU
복호화할 패드를 입력하시오: 19950701
복호문: apple

=====

One Time Pad
모드를 입력하세요 1:암호화 2:복호화 3:패드생성 4:종료 =>3
생성할 패드의 이름을 입력하세요: pad01
패드 생성 완료!

=====

One Time Pad
모드를 입력하세요 1:암호화 2:복호화 3:패드생성 4:종료 =>1
암호화할 평문을 입력하세요: apple
암호화할 패드를 입력하시오: pad01
암호문: ◀◀T

=====

One Time Pad
모드를 입력하세요 1:암호화 2:복호화 3:패드생성 4:종료 =>2
복호화할 암호문을 입력하세요: "Q" T#T
복호화할 패드를 입력하시오: pad01
복호문: apple

=====

One Time Pad
모드를 입력하세요 1:암호화 2:복호화 3:패드생성 4:종료 =>4
다시 입력해주세요

=====

프로그램이 종료했습니다.
C:\Users\User\Documents\Visual Studio 2017\Projects\OTP1\Debug\OTP1.exe(23844 프로세스)이(가) 0 코드로 인해 종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]->[디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.
```

문제 2. 페이스텔 암호 알고리즘 구현

DES

<정의>

1997년에 미국의 연방 정보처리 표준 규격을 채택된 대칭형 암호이다. 즉, 암호화 및 복호화 키가 동일하다. 그래서 비대칭형 암호보다 암호화와 복호화에 있어서 속도가 빠르다. 데이터 암호화 표준은 64비트 평문을 64비트 암호문으로 암호화하는 대칭키 암호 알고리즘이다. 키와 비트의 길이는 56비트이다. 64비트 평문을 하나의 단위로 암호화한다. 이것은 블록 암호의 특징을 갖고 있는 것이다. 블록 암호는 블록 단위로 처리를 하는 알고리즘을 뜻한다. DES는 페이스텔 네트워크의 변형된 형태로 라운드 횟수는 16회이다. 56비트짜리 원래 키로부터 16개의 서브키를 생성하고 그 서브키를 각 라운드에서 사용한다. 최근 반도체 칩 기술의 발달로 DES암호는 쉽게 해독이 가능하다. 그래서 2중 3중 DES 사용을 권고하고 있다.

<암호화/복호화 과정>

암호화 과정은 두 개의 전치와 16개의 페이스텔 라운드 함수로 구성된다. 라운드는 초기전치 박스의 출력 값 또는 이전 라운드 함수의 출력 값 $L(i-1)$ 과 $R(i-1)$ 을 입력 받아 다음 라운드의 입력 값 $L(i)$ 와 $R(i)$ 를 생성한다. 16개의 라운드 함수 중에 사용된 두 개의 P박스 중 하나는 초기 전치, 다른 하나는 최종 전치라고 한다. 초기 전치박스와 최종 전치박스는 서로 역의 관계에 있는 단순 P박스이다. 최종 전치는 DES에 있어서 암호학적으로 중요하지 않다. 각 라운드는 라운드 키 생성기에 의해 암호키로부터 생성된 48비트 라운드 키를 사용한다. 64비트의 평문을 초기전치 테이블을 통해서 평문의 문자를 재배열하고 라운드 함수를 16라운드 반복하여 암호화를 한다. 복호화는 암호화한 과정을 역순으로 진행하면 된다.

<문제점>

DES는 56비트의 키를 사용하므로 최대 키 공간의 크기가 2^{56} 이 되고, 이는 개발 당시에 전사공격하기 굉장히 어려운 수였지만 현재 계산능력이 훨씬 발전하여서 지금은 전사공격에도 안전하지 못한 방법이 되었다. 또한 대칭키에도 문제점이 있는데 키 관리 문제가 있다. 만약 사람 A, B, C가 비밀 키 암호통신을 할 경우 사용하는 비밀 키는 각각 달라야한다. n 사람이 암호화 통신을 할 때 필요한 키 개수가 $n(n-1)/2$ 로 관리하기 힘들다.

<소스코드>

```
#include<iostream>
#include <stdlib.h>
#define CYPHER 0
#define DECYPHER 1
using namespace std;

unsigned long long des(unsigned long long plainTxt, unsigned long long key, int mode);
static unsigned char parity_drop[56] =
{
    57, 49, 41, 33, 25, 17, 9,
    1, 58, 50, 42, 34, 26, 18,
    10, 2, 59, 51, 43, 35, 27,
    19, 11, 3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14, 6, 61, 53, 45, 37, 29,
    21, 13, 5, 28, 20, 12, 4
};

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

static unsigned char initial_permutation[64] = //초기 전치테이블
{
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
};

static unsigned char expansion_pbox[48] = //확장 p-박스
{
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1
};

static unsigned char sbox[8][4][16] = //s-박스 8개
{
    { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7, },
      { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8, },
      { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0, },
      { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13, }, },
    { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10, },
      { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5, },
      { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15, },
      { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9, }, },
    { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8, },
      { 11, 1, 14, 4, 13, 10, 7, 0, 9, 5, 6, 12, 8, 3, 15, },
      { 13, 15, 9, 2, 6, 1, 10, 4, 0, 5, 3, 12, 8, 14, 7, },
      { 7, 13, 11, 2, 4, 14, 15, 10, 1, 13, 12, 7, 11, 4, 2, 8, }, },
    { { 14, 1, 10, 4, 13, 15, 8, 12, 0, 5, 9, 3, 11, 6, 7, 2, },
      { 11, 14, 4, 9, 13, 8, 5, 1, 10, 6, 12, 15, 9, 3, 7, },
      { 15, 13, 8, 11, 10, 14, 9, 7, 12, 4, 0, 6, 1, 3, 15, },
      { 13, 15, 9, 2, 6, 1, 10, 4, 0, 5, 3, 12, 8, 14, 7, }, },
    { { 14, 1, 10, 4, 13, 15, 8, 12, 0, 5, 9, 3, 11, 6, 7, 2, },
      { 11, 14, 4, 9, 13, 8, 5, 1, 10, 6, 12, 15, 9, 3, 7, },
      { 15, 13, 8, 11, 10, 14, 9, 7, 12, 4, 0, 6, 1, 3, 15, },
      { 13, 15, 9, 2, 6, 1, 10, 4, 0, 5, 3, 12, 8, 14, 7, }, },
    { { 14, 1, 10, 4, 13, 15, 8, 12, 0, 5, 9, 3, 11, 6, 7, 2, },
      { 11, 14, 4, 9, 13, 8, 5, 1, 10, 6, 12, 15, 9, 3, 7, },
      { 15, 13, 8, 11, 10, 14, 9, 7, 12, 4, 0, 6, 1, 3, 15, },
      { 13, 15, 9, 2, 6, 1, 10, 4, 0, 5, 3, 12, 8, 14, 7, }, }
};
```

```
{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1, },
{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7, },
{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12, }, },
{ 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15, }, },
{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9, }, },
{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4, }, },
{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14, }, },
{ 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9, }, },
{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6, }, },
{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14, }, },
{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3, }, }, },
{ 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11, }, },
{ 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8, }, },
{ 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6, }, },
{ 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13, }, }, },
{ 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1, }, },
{ 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6, }, },
{ 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2, }, },
{ 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12, }, }, },
{ 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7, }, },
{ 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2, }, },
{ 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8, }, },
{ 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } } };
```

```
static unsigned char straight_pbox[32] = { //단순 p-박스
```

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

```
};
```

```
static unsigned char final_permutation[64] = {
```

```
40, 8, 48, 16, 56, 24, 64, 32,
39, 7, 47, 15, 55, 23, 63, 31,
38, 6, 46, 14, 54, 22, 62, 30,
37, 5, 45, 13, 53, 21, 61, 29,
36, 4, 44, 12, 52, 20, 60, 28,
35, 3, 43, 11, 51, 19, 59, 27,
34, 2, 42, 10, 50, 18, 58, 26,
33, 1, 41, 9, 49, 17, 57, 25
```

```
};
```

```
int main() {
```

```
    unsigned long long painTxt, key, result;
```

```
    cout<<"64비트 평문을 입력하세요: ";
```

```
    cin >> painTxt;
```

```
    cout << "64비트 key를 입력하세요: ";
```

```
    cin >> key;
```

```
    result = des(painTxt, key, CYPERER);
```

```
    cout << "암호화한 문장: " << result << endl;
```

```
    cout << "복호화한 문장: " << des(result, key, DECYPER);
```

```
    return 0;
```

```
}
```

```
unsigned long long des(unsigned long long plainTxt, unsigned long long key, int mode) {
```

```
    unsigned long long roundKey[16];
```

```
    unsigned long long cd = 0;
```

```
    unsigned long long temp = 0;
```

```
    unsigned long long result = 0;
```



```

unsigned long long epOutput48 = 0; //확대전치 과정을 거친 입력
unsigned int sbboxOutput32 = 0, pboxOut32 = 0;

unsigned int l = 0, r = 0;
unsigned int c = 0, d = 0;

unsigned char sdata = 0, row = 0, col = 0;
int j = 0, i = 0, round = 0;

for (i = 1; i <= 16; i++) { //라운드 키 초기화
roundKey[i - 1] = 0;
}

// 키 생성
for (j = 27; j >= 0; j--) {
c = c ^ (((key >> (64 - parity_drop[(27 - j)])) & 0x1) << j);
d = d ^ (((key >> (64 - parity_drop[(55 - j)])) & 0x1) << j);
}

// 비트 시프팅
for (i = 1; i <= 16; i++) {
if ((i == 1) | (i == 2) | (i == 9) | (i == 16)) { //1,2,9,16번째는
왼쪽으로 한번 쉬프트
c = ((c << 1) | (c >> 27)) & 0xFFFFFFFF;
d = ((d << 1) | (d >> 27)) & 0xFFFFFFFF;
}
else { //나머지는 왼쪽으로 2비트 쉬프트
c = ((c << 2) | (c >> 26)) & 0xFFFFFFFF;
d = ((d << 2) | (d >> 26)) & 0xFFFFFFFF;
}
cd = 0; // 초기화
cd = ((cd ^ c) << 28) ^ d;

for (j = 47; j >= 0; j--) { //라운드 키 생성
roundKey[i - 1] = roundKey[i - 1] ^ (((cd >> (56 - compression_table[(47 - j)]))
& 0x1) << j);
}
}

//초기 순열
for (j = 31; j >= 0; j--) {
l = l ^ (((plainTxt >> (64 - initial_permutation[(31 - j)])) & 0x1) << j);
r = r ^ (((plainTxt >> (64 - initial_permutation[(63 - j)])) & 0x1) << j);
}

// 16 라운드
for (round = 0; round < 16; round++)
{
epOutput48 = 0;
for (j = 47; j >= 0; j--) {
epOutput48 = epOutput48 ^ ((long long)((r >> (32 - expansion_pbox[(47 - j)])) &
0x1) << j); //확대전치
}

if (mode == CYPHER) {
해준다
epOutput48 = epOutput48 ^ roundKey[round]; // 암호화 라운드키를 xor연산
}
if (mode == DECYPHER) {
xor연산해준다
epOutput48 = epOutput48 ^ roundKey[15 - round]; // 복호화 라운드 키를
}

sbboxOutput32 = 0;
for (i = 7; i >= 0; i--) {
row = 0;
col = 0;
}

```

```

        sdata = 0;
        sdata = (epOutput48 >> (i * 6)) & 0x3F;
        row = row ^ (sdata & 0x1);
        row = row ^ (((sdata >> 5) & 0x1) << 1);
        col = (sdata >> 1) & 0x0F;
        sbboxOutput32 = sbboxOutput32 ^ ((int)(sbox[7 - i][row][col] << (4 * i)));
    }

    pboxOut32 = 0;
    for (j = 31; j >= 0; j--) {
        pboxOut32 = pboxOut32 ^ (((sboxOutput32 >> (32 - straight_pbox[(31 - j)])) & 0x1)
<< j);
    }
    pboxOut32 = pboxOut32 ^ l;

    l = r;
    r = pboxOut32;
}

temp = 0;
temp = ((temp ^ r) << 32) ^ l;

result = 0;
for (j = 63; j >= 0; j--) {
    result = result ^ (((temp >> (64 - final_permutation[(63 - j)])) & 0x1) << j);
}

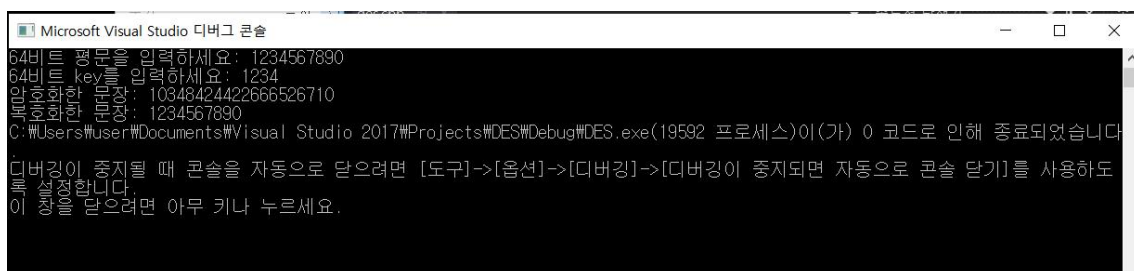
    return result;
}

```

<소스코드 리뷰>

#define으로 암호화와 복호화 모드를 생성해 각각 암호화는 0 복호화는 1로 설정한다. 그리고 des하는 과정에서 필요한 배열들을 설정한다. 초기 전치테이블 배열 확장된 p-박스 배열 s-박스 배열 8개 단순 p-박스 등을 설정해준다. 그리고 메인함수에서는 평문을 unsigned long long의 변수로 생성을 하고 입력하도록 코드를 짠다. 또한 암호화와 복호화에 필요한 key 값도 unsigned long long형으로 설정하고 입력 받는다. des함수에서는 키를 생성하고 라운드 키를 생성하고 반복문을 통해서 16라운드를 각각 xor연산을 해준다. 이렇게 16라운드를 거친 평문은 암호문이 되고 16라운드를 거친 암호문은 복호화한 평문이 된다.

<결과>



```

Microsoft Visual Studio 디버그 콘솔
64비트 평문을 입력하세요: 1234567890
64비트 key를 입력하세요: 1234
암호화한 문장: 10348424422666526710
복호화한 문장: 1234567890
C:\Users\User\Documents\Visual Studio 2017\Projects\DES\Debug\DES.exe(19592 프로세스)이(가) 0 코드로 인해 종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]->[디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.

```

<소감>

이번 과제를 통해서 XOR연산을 통해서 암호를 간편하게 만들 수 있구나 라는 것을 알 수 있었다. OTP같은 경우에는 간편하게 설정한 패드로 암호화하고 같은 패드로 복호화 하는 것이 아주 편리하고 좋은 점이 많았다. 간단한 암호를 설정할 때 편리한 방법인 거 같다고 느꼈다. 하지만 이에도 문제가 있다는 것을 공부를 통해 알 수 있었다. 그래서 중요한 보안에 OTP암호를 사용하는 것은 어느 정도 무리가 있을 것으로 판단됐다. 또한 DES같은 경우에는 필요한 배열이 많고 그 배열들을 반복문을 통해서 바꾸고 확장하고 XOR연산해야하기 때문에 코드를 짜는데 굉장히 어려움이 많았다. 그렇지만 그렇게 고민을 하면서 코드를 짰 결과 블록형 암호에 대한 전반적인 이해와 DES 암호가 가지는 단점이 왜 발생하는지 이것을 보완한 다른 블록형 암호들은 뭐가 다른지에 대해서 수업 때 들었던 것이 더 와 닿았다. 그래서 개념이 조금 더 확실히 잡힌 것 같은 느낌이 들어서 굉장히 좋았고 이번 과제를 통해서 기존에 알았던 암호 방식이 아닌 처음 보는 암호에 대해서 배우고 직접 구현해 볼 수 있어서 굉장히 좋았습니다. 또한 굉장히 복잡한 과정을 직접 코드로 구현하면서 암호에 대한 지식뿐만이 아니고 컴퓨터 프로그래밍에 대한 이해도와 숙련도가 더 올라갈 수 있었던 과제인 것 같아서 좋았습니다. 여러모로 제가 많이 배울 수 있도록 도와준 과제인 것 같습니다.