

컴퓨터보안 보고서

실습과제 #6주차

강좌 명: 컴퓨터보안

교수님: 정준호 교수님

학과: 컴퓨터공학과

학년: 3학년

학번: 2017112138

이름: 정여준

CBC모드 CTR모드의 암호화 및 복호화가 가능한 암호 프로그램 구현하여 성능 및 차이점 등을 비교 분석해보자.

<CBC모드의 정의>

블록 암호 운용 방식은 하나의 키 아래에서 블록 암호를 반복적으로 안전하게 이용하게 하는 절차를 말한다. 블록 암호는 특정한 길이의 블록 단위로 동작하기 때문에, 가변 길이 데이터를 암호화하기 위해서는 먼저 이들을 단위 블록들로 나누어야 하며, 그 블록들을 어떻게 암호화할지 정해야 하는데, 이때 블록들의 암호화 방식을 운용방식이라 부른다. 이러한 운용 방식 중 하나인 CBC 암호 블록 체인 방식은 1976년 IBM에 의해 개발되었다. CBC 방식은 현재 널리 사용되는 운용 방식 중 하나이다. CBC는 암호화 입력 값이 이전 결과에 의존하기 때문에 병렬화가 불가능하지만 복호화의 경우 각 블록을 복호화 한 다음 이전 암호화 블록과 XOR하여 복구할 수 있기 때문에 병렬화가 가능하다.

<CBC모드 암호화/복호화 과정>

암호화는 평문 블록을 암호화하고 그 결과물을 다음 평문 블록과 XOR연산하는 식으로 작동한다. 처음에 암호화할 때는 IV(초기벡터)를 따로 넣어서 이 초기벡터와 첫번째 평문 블록을 XOR해서 암호화를 진행한다. 여기서 사용되는 초기 벡터는 반드시 송수신 양자가 모두 알고 있어야 하고 제 3자로부터 예측이 불가능 해야 한다.

복호화는 암호화한 반대 순서로 XOR하면 된다.

<CBC모드 장점과 단점>

장점: 복호화에서는 병렬처리 기능이 사용 가능하다. 임의의 암호문 블록을 복호화 가능하다. 평문의 반복은 암호문에 반영되지 않는다. 무결성과 인증을 제공할 수 있다.

단점: 암호화에서는 병렬처리 기능을 사용할 수 없다. 전송 도중 비트오류 시 대부분 비트에서 오류가 발생한다. 즉, 연쇄적으로 작동하기 때문에 그에 따른 위험이 있다.

<CTR모드의 정의>

카운터 방식은 블록 암호를 스트림 암호로 바꾸는 구조를 가진다. 카운터 방식에는 각 블록마다 현재 블록이 몇 번째인 값을 얻어 그 숫자와 nonce를 결합하여 블록 암호의 입력으로 사용한다. 그렇게 각 블록 암호에서 연속적인 난수를 얻은 다음 암호화하려는 문자열과 XOR한다. 카운터 모드는 각 블록의 암호화 및 복호화가 이전 블록에 의존하지 않기 때문에 병렬적으로 동작하는 것이 가능하다. 또는 암호화된 문자열에서 원하는 부분만 복호화 하는 것도 가능하다.

<CTR모드 암호화/복호화 과정>

암호화는 카운터 키를 사용하는데 단계마다 1씩 증가시키면서 암호화한다. 즉, 첫번째 블록에는 그냥 카운터 키를 사용하고 두번째 블록에는 첫번째 사용했던 카운터 키에 +1을 해준 형태의 키를 사용하여 XOR해준다.

복호화는 암호화와 동일한 방법으로 진행된다.

<CTR모드 장점과 단점>

장점: 패딩이 필요 없다. 암호 복호화의 사전 준비가 가능하다. 암호 복호화가 같은 구조를 가진다. 평문의 대응하는 비트만 에러가 난다. 암호 복호화 모두 병렬로 처리할 수 있다.

단점: 적극적 공격자가 암호문 블록의 비트를 반전시키면 대응하는 평문 블록의 비트가 반전된다.

<소스코드>

```
#include <iostream>
#include<string>
#include<stdlib.h>
#include<time.h>
using namespace std;

string SDESEncryption(string, string, int);
string SDESDecryption(string, string, int);
string findKey(string, int);
string functionF(string, string);
string XOR(string, string);
string SUM(string);
string S1Box(string);
string S2Box(string);
string CBCencrypt(string, string, string, int);
string CBCdecrypt(string, string, string, int);

string SUM(string ctrkey) //CTR모드에서 키값을 하나씩 더해줄 함수
{
    string y = "";
    string z = "";
    int num = 0;
    for (int i = 0; i < ctrkey.length(); i++) //2진수 형태의 문자열을 10진수형태의 정수로
바꾸는 과정
    {
        num += (ctrkey[i] - '0')*(2048 >> i);
    }
    do {
        if (num % 2 == 1)
        {
            num = num / 2;
            y += "1";
        }
        else
        {
            num = num / 2;
            y += "0";
        }
    } while (num != 0);
    while (y.length() != 12)
    {
        y += "0";
    }
    for (int i = y.length() - 1; i >= 0; i--) //뒤집기
    {
        z += y[i];
    }
    return z;
}

string CBCencrypt(string key, string plaintext, string IV, int rounds) //CBC기법으로 암호화
{
    string ct1, ct2, ct3, ct4; //평문을 12비트의 크기로 나눈다
```

```

ct1.append(plaintext, 0, 12);
ct2.append(plaintext, 12, 12);
ct3.append(plaintext, 24, 12);
ct4.append(plaintext, 36, 12);

ct1 = XOR(ct1, IV); //초기벡터와 첫 블록 XOR연산

for (int i = 1; i <= rounds; i++)
{
    ct1 = SDESEncryption(key, ct1, i);
}

ct2 = XOR(ct2, ct1);

for (int i = 1; i <= rounds; i++)
{
    ct2 = SDESEncryption(key, ct2, i);
}

ct3 = XOR(ct3, ct2);

for (int i = 1; i <= rounds; i++)
{
    ct3 = SDESEncryption(key, ct3, i);
}

ct4 = XOR(ct4, ct3);

for (int i = 1; i <= rounds; i++)
{
    ct4 = SDESEncryption(key, ct4, i);
}

return (ct1 + ct2 + ct3 + ct4); //암호화한 문장 리턴
}

string CBCdecrypt(string key, string ciphertext, string IV, int rounds) //CBC기법으로 복호화
{
    string pt1, pt2, pt3, pt4; //암호문을 12비트의 4개의 문장으로 나눈다

    pt1.append(ciphertext, 0, 12);
    pt2.append(ciphertext, 12, 12);
    pt3.append(ciphertext, 24, 12);
    pt4.append(ciphertext, 36, 12);

    //암호화 한 반대방향으로 동일하게 진행하면서 복호화
    for (int i = rounds; i > 0; i--)
    {
        pt4 = SDESDecryption(key, pt4, i);
    }

    pt4 = XOR(pt3, pt4);

    for (int i = rounds; i > 0; i--)

```

```

    {
        pt3 = SDESDecryption(key, pt3, i);
    }

    pt3 = XOR(pt2, pt3);

    for (int i = rounds; i > 0; i--)
    {
        pt2 = SDESDecryption(key, pt2, i);
    }

    pt2 = XOR(pt1, pt2);

    for (int i = rounds; i > 0; i--)
    {
        pt1 = SDESDecryption(key, pt1, i);
    }

    pt1 = XOR(IV, pt1);

    return (pt1 + pt2 + pt3 + pt4);
}

string CTRdecrypt(string key, string ctrkey, string ciphertext, int rounds)//CTR기법으로 복호화
{
    string pt1, pt2, pt3, pt4; //평문을 4개의 12비트 문자열로 저장
    string a1, a2, a3, a4; //key를 +1해주면서 저장해줄 문자열
    pt1.append(ciphertext, 0, 12);
    pt2.append(ciphertext, 12, 12);
    pt3.append(ciphertext, 24, 12);
    pt4.append(ciphertext, 36, 12);
    a1 = ctrkey;
    a2 = SUM(a1);
    a3 = SUM(a2);
    a4 = SUM(a3);

    for (int i = 1; i <= rounds; i++)
    {
        a1 = SDESEncryption(key, a1, i);
        a2 = SDESEncryption(key, a2, i);
        a3 = SDESEncryption(key, a3, i);
        a4 = SDESEncryption(key, a4, i);
    }
    pt1 = XOR(a1, pt1);
    pt2 = XOR(a2, pt2);
    pt3 = XOR(a3, pt3);
    pt4 = XOR(a4, pt4);

    return (pt1 + pt2 + pt3 + pt4);
}

string CTRencrypt(string key, string ctrkey, string plaintext, int rounds)//CTR기법으로 암호화
{
    string ct1, ct2, ct3, ct4;
    string a1, a2, a3, a4;
    ct1.append(plaintext, 0, 12);

```

```

ct2.append(plaintext, 12, 12);
ct3.append(plaintext, 24, 12);
ct4.append(plaintext, 36, 12);
a1 = ctrkey;
a2 = SUM(a1);
a3 = SUM(a2);
a4 = SUM(a3);

for (int i = 1; i <= rounds; i++)
{
    a1 = SDESEncryption(key, a1, i);
    a2 = SDESEncryption(key, a2, i);
    a3 = SDESEncryption(key, a3, i);
    a4 = SDESEncryption(key, a4, i);
}
ct1 = XOR(a1, ct1);
ct2 = XOR(a2, ct2);
ct3 = XOR(a3, ct3);
ct4 = XOR(a4, ct4);

return (ct1 + ct2 + ct3 + ct4);
}
string SDESEncryption(string key, string plaintext, int round) //Simple Des로 암호화하는 함수
{
    string Li;
    string Ri;
    string Ln;
    string Rn;
    string K;
    string f;

    K = findKey(key, round); //라운드 키 생성

    Li.append(plaintext, 0, 6);
    Ri.append(plaintext, 6, 6);

    Ln = Ri;

    f.append(functionF(Ri, K));

    Rn.append(f);
    Rn = XOR(Li, f); //xor연산

    return (Ln + Rn);
}

string SDESDecryption(string key, string ciphertext, int round) //Simple Des로 복호화하는 함수
{
    string Li;
    string Ri;
    string Ln;
    string Rn;
    string K;
    string f;

```

```

    K = findKey(key, round); //라운드 키 생성

    Li.append(ciphertext, 0, 6);
    Ri.append(ciphertext, 6, 6);

    Rn = Li;

    f.append(functionF(Rn, K));

    Ln.append(f);
    Ln = XOR(Ri, f);

    return (Ln + Rn);
}

string findKey(string key, int round) //라운드 키를 찾는 함수
{
    string K;

    if (round == 1)
    {
        K.append(key, 0, 8);
    }
    else if (round == 2)
    {
        K.append(key, 1, 8);
    }
    else if (round == 3)
    {
        K.append(key, 2, 7);
        K.append(key, 0, 1);
    }
    else if (round == 4)
    {
        K.append(key, 3, 6);
        K.append(key, 0, 2);
    }
    return K;
}

string functionF(string R, string K) //Simple DES 함수
{
    char tmp;
    string s1;
    string s2;

    R.append(R, 4, 2);
    tmp = R[3];
    R[5] = R[2];
    R[4] = tmp;
    R[3] = R[2];
    R[2] = tmp;

    R = XOR(R, K);
    s1.append(R, 0, 4);

```



```

        s2.append(R, 4, 4);

        return S1Box(s1) + S2Box(s2);
    }

string XOR(string x, string y) //비트열 XOR연산 함수
{
    for (int i = 0; i < x.length(); i++)
    {
        if (x[i] == y[i])
        {
            x[i] = '0';
        }
        else if (x[i] != y[i])
        {
            x[i] = '1';
        }
    }

    return x;
}

string S1Box(string s1) //Simple DES에서 S1박스
{
    string row1[8] = { "101", "010", "001", "110", "011", "100", "111", "000" };
    string row2[8] = { "001", "100", "110", "010", "000", "111", "101", "011" };

    int column = 0;

    if (s1[0] == '0')
    {
        if (s1[1] == '1')
        {
            column += 4;
        }
        if (s1[2] == '1')
        {
            column += 2;
        }
        if (s1[3] == '1')
        {
            column += 1;
        }

        return row1[column];
    }
    else if (s1[0] == '1')
    {
        if (s1[1] == '1')
        {
            column += 4;
        }
        if (s1[2] == '1')
        {
            column += 2;
        }
    }
}

```

```

    }
    if (s1[3] == '1')
    {
        column += 1;
    }

    return row2[column];
}
else
{
    return "ERROR";
}
}

```

```

string S2Box(string s2)
{
    string row1[8] = { "100", "000", "110", "101", "111", "001", "011", "010" };
    string row2[8] = { "101", "011", "000", "111", "110", "010", "001", "100" };

    int column = 0;

    if (s2[0] == '0')
    {
        if (s2[1] == '1')
        {
            column += 4;
        }
        if (s2[2] == '1')
        {
            column += 2;
        }
        if (s2[3] == '1')
        {
            column += 1;
        }

        return row1[column];
    }
    else if (s2[0] == '1')
    {
        if (s2[1] == '1')
        {
            column += 4;
        }
        if (s2[2] == '1')
        {
            column += 2;
        }
        if (s2[3] == '1')
        {
            column += 1;
        }

        return row2[column];
    }
}

```

```

        else
        {
            return "ERROR";
        }
    }

int main()
{
    string plaintext = "101010101010101010101010101010100110101010110101010"; //CBC기법으로
암호화할 평문
    string key = "010011001"; //CBC기법에서 key
    string IV = "010111101000"; //CBC기법에서 초기벡터
    int numrounds = 4;

    string ciphertext = "";
    string decryption = "";

    ciphertext = CBCencrypt(key, plaintext, IV, numrounds);
    decryption = CBCdecrypt(key, ciphertext, IV, numrounds);

    cout << "CBC" << endl;
    cout << "평문: " << plaintext << endl
        << "키: " << key << endl << "초기벡터: " << IV << endl << endl;
    cout << "CBC모드로 암호화" << endl;
    cout << ciphertext << endl << endl;
    cout << "CBC모드로 복호화" << endl;
    cout << decryption << endl << endl;

    cout << "===== " << endl;

    string plaintext2 = "1111001010100011000011111111001010101010101100"; //CTR 기법에서
암호화할 평문
    string ctrkey = "010010100101"; //CTR 기법에서 초기 key

    string decryption2 = "";
    string ciphertext2 = "";

    ciphertext2 = CTRencrypt(key, ctrkey, plaintext2, numrounds);
    decryption2 = CTRdecrypt(key, ctrkey, ciphertext2, numrounds);

    cout << "CTR" << endl;
    cout << "평문2: " << plaintext2 << endl;
    cout << "키: " << key << endl;
    cout << "카운터 키: " << ctrkey << endl << endl;

    cout << "CTR모드로 암호화" << endl;
    cout << ciphertext2 << endl << endl;
    cout << "CTR모드로 복호화" << endl;
    cout << decryption2 << endl << endl;

    cout << "===== " << endl;

    return 0;
}

```

<소스코드 리뷰>

CBC같은 경우에는 암호화할 때 문자열로 입력 받은 key와 평문, 초기벡터를 매개변수로 입력 받고 정수형 변수로 진행할 rounds 수를 매개변수로 입력 받는다. 라운드는 4번 평문의 길이는 48비트이기 때문에 12비트씩 끊어서 저장하고 암호화를 진행한다. 첫번째 블록을 초기벡터와 XOR연산해주고 이후에 DES암호화를 4라운드로 해준다. 여기서 나온 암호문이 이제 키가 되어 이 암호문과 두번째 평문 블록을 XOR연산해준다. 이러한 동작을 반복해서 모두 암호화를 완료하면 문자열을 합쳐서 반환해준다. 복호화 같은 경우에는 암호화할 때의 순서와 반대로 동일하게 진행해주면 된다.

CTR같은 경우에는 카운터를 생성해야 한다. 나는 이것을 ctrkey로 설정했다. 암호화할 때 문자열로 입력 받은 key와 ctrkey 그리고 평문을 매개변수로 입력 받고 정수형 변수로 진행할 rounds 수를 매개변수로 입력 받는다. 라운드는 4번이고 평문의 길이는 48비트이기 때문에 12비트씩 끊어서 저장하고 암호화를 진행한다. 첫번째에는 입력 받은 ctrkey를 DES로 암호화하고 이 것을 첫번째 평문 블록에 XOR해준다. 그 이후 입력 받은 ctrkey에 +1연산을 해준다. 이후에 동일한 동작을 한다. 세번째 평문에서는 두번째에 사용한 ctrkey에 +1 연산을 하고 동일하게 진행한다. 이런 동작을 반복한다. 복호화 같은 경우에는 그냥 암호화하는 방법과 동일하게 동일한 순서로 진행하면 된다. 이러한 것을 병렬적으로 처리 가능하다.

<결과>

```
Microsoft Visual Studio 디버그 콘솔

CBC
평문: 101010101010101010101010101010100011010101010101010
키: 010011001
초기벡터: 010111101000

CBC모드로 암호화
111000100010010010011011010001100011001110011110

CBC모드로 복호화
101010101010101010101010101010100011010101010101010

=====

CTR
평문2: 1111001010100011000011111111001010101010101100
키: 010011001
카운터 키: 010010100101

CTR모드로 암호화
0001100000011110110100110000110000011010000000101

CTR모드로 복호화
1111001010100011000011111111001010101010101100

=====

C:\Users\user\Documents\Visual Studio 2017\Projects\Securtiy(5)\Debug\Securtiy(5).exe(18396 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.
```

<성능 비교>

시간 측정을 위해서 메인 함수를 다음과 같이 바꿔준다.

```
int main()
{
    clock_t s_cbc, e_cbc; //CBC모드 시간 측정
    clock_t s_ctr, e_ctr; //CTR모드 시간 측정
    srand((unsigned)time(NULL));
    string key = "010011001"; //CBC기법에서 key
    string IV = "010111101000"; //CBC기법에서 초기벡터
    int numrounds = 4;

    //암호화 복호화 하는데 걸린 시간 측정
    s_cbc = clock();
    string plaintext; //CBC기법으로 암호화할 평문
    for (int i = 0; i < 48; i++)
    {
        int random = rand() % 2;
        if (random == 0)
            plaintext += '0';
        else
            plaintext += '1';
    }

    string ciphertext = "";
    string decryption = "";
    for (int i = 0; i < 5000; i++) {
        ciphertext = CBCencrypt(key, plaintext, IV, numrounds);
        decryption = CBCdecrypt(key, ciphertext, IV, numrounds);
    }
    e_cbc = clock();
    //시간 측정 종료

    cout << "걸린 시간: " << (float)((e_cbc - s_cbc) / 5000) / CLOCKS_PER_SEC << endl;
    //시간을 초 단위로 계산
    cout << "===== " << endl;

    //암호화 복호화 하는데 걸린 시간 측정
    s_ctr = clock();
    string plaintext2; //CTR 기법에서 암호화할 평문
    for (int i = 0; i < 48; i++)
    {
        int random2 = rand() % 2;
        if (random2 == 0)
            plaintext2 += '0';
        else
            plaintext2 += '1';
    }
    string ctrkey = "010010100101"; //CTR 기법에서 초기 key

    string decryption2 = "";
    string ciphertext2 = "";
    for (int i = 0; i < 5000; i++) {
```

```

    ciphertext2 = CTRencrypt(key, ctrkey, plaintext2, numrounds);
    decryption2 = CTRdecrypt(key, ctrkey, ciphertext2, numrounds);
}
e_ctr = clock();
//시간 측정 종료


cout << "걸린 시간:" << (float)((e_ctr - s_ctr) / 5000) / CLOCKS_PER_SEC << endl;
//시간을 초 단위로 계산
cout << "===== " << endl;

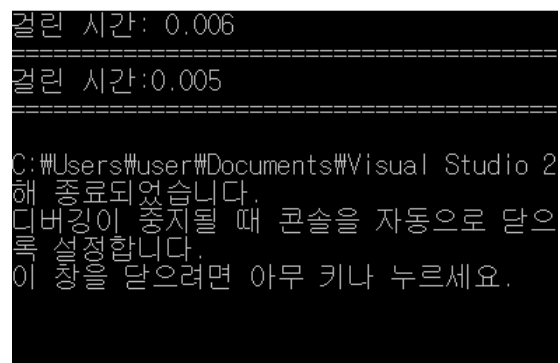
return 0;

}

```

총 5000회씩 실행해서 평균 값을 나타냈다.

 Microsoft Visual Studio 디버그 콘솔



```

걸린 시간: 0.006
=====
걸린 시간:0.005
=====
C:\Users\User\Documents\Visual Studio 2
해 종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.

```

CTR모드는 병렬처리로 실행했기 때문에 조금 더 좋은 성능을 보였다.

<소감>

이번 과제를 통해서 블록 암호 방식에 있는 두가지 방식 CBC 모드와 CTR 모드를 직접 구현해볼 수 있었다. 지난번 과제로 DES를 수행했기 때문에 이해가 안되는 부분은 따로 있지는 않았다. 하지만 몇 가지 변화로 DES보다 훨씬 해독하기 어려운 암호를 만들었다는 것을 느꼈다. 각 암호방식의 특징들이나 장점, 단점에 대한 것을 공부하는 데 도움이 됐고 직접 구현해보면서 각 모드 별로 어떤 것이 핵심인지 알게 되었다. 수업시간에 배웠던 부분에 대해서 살짝 의아했던 부분들도 모두 해결된 것 같아서 굉장히 좋다. 또한 직접 코드를 작성했기 때문에 프로그래밍 실력에도 도움이 된 것 같아서 매우 좋았다.