REPORT

프로젝트

과목명 :	컴퓨터보안
담당교수 :	정준호 교수님
제출일 :	2021 년 06 월 17 일
공과 대학	컴퓨터공학 과
학번 : 2016112154	이름: 정동구

목차

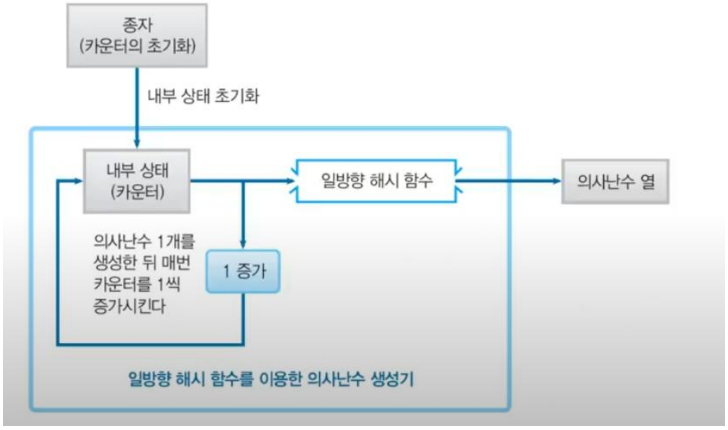
목표 및 설계

암호시스템에 사용 가능한 의사 난수 생성기 구현

난수의 성질은 3 가지로 분류 할 수 있다. '무작위성', '예측 불가능성', '재현 불가능성'. 무작위성은 통계적인 편중이 없이 수열이 무작위로 되어있는 성질, 예측 불가능성은 과거의 수열로부터 다음 수를 예측할 수 없는 성질이고, 마지막으로 재현 불가능성은 같은 수열의 재현이 불가능함을 말한다. 뒤로 갈수록 엄격한 조건이 된다. 이 때 예측 불가능성을 가지는 난수는 무작위성을 가지고, 재현 불가능성을 가지는 난수는 무작위성과 예측 불가능성을 가진다.

무작위성만 가지는 난수를 '약한 의사난수', 무작위성과 예측 불가능성을 가지는 난수를 '강한 의사난수', 재현 불가능성과 무작위성, 예측 불가능성을 가지는 난수를 '진성 난수'라 한다. 약한 의사난수는 암호기술에 사용 할 수 없고, 강한 의사난수와 진성 의사난수는 암호기술에 사용이 가능하다.

암호시스템에 사용 가능한 의사 난수 생성을 위해 일방향 해시 함수를 사용하는 방법을 택했다.(그림 1 참조)



[그림 1. 일방향 해시함수를 이용한 의사난수]

이 의사난수 생성기의 절차는 아래와 같다.

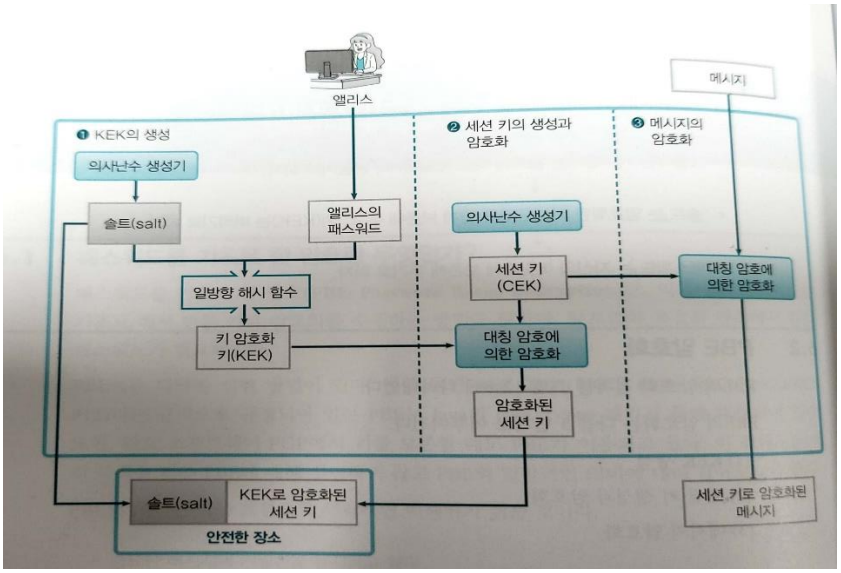
1. 의사난수의 종자를 사용하여 내부 상태(카운터)를 초기화한다.
2. 일방향 해시 함수를 사용하여 카운터의 해시 값을 얻는다.
3. 그 해시 값을 의사난수로서 출력한다.
4. 카운터를 1 증가시킨다.
5. 필요한 만큼의 의사난수가 얻어질 때 까지 2~4 를 반복한다.

위의 방법을 활용하여 의사난수 생성기를 구현하고, 이 때 사용하는 일방향 해시 함수는 SHA-1 을 사용한다.

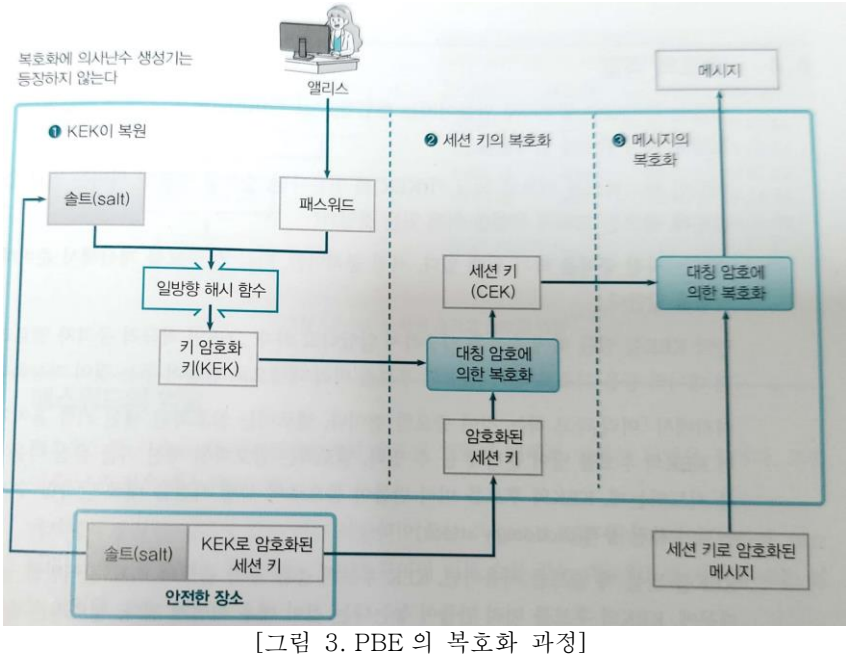
해당 방식을 활용한 의사난수 생성기는 일방향 해시함수의 성질에 의해 예측 불가능성을 보장하게 된다.

난수를 활용한 암호시스템 선정 및 설계

난수를 활용하는 암호시스템은 여러가지가 존재한다. 난수를 이용한 메시지 인증코드, 난수를 활용하는 블록암호모드, 난수를 키로 이용하는 단일치환 암호 등 여러 암호시스템에서 난수가 활용된다. 난수를 단순히 한번만 이용하는 것이 아닌 최대한 활용해 보고자 CTR 을 암호화 알고리즘으로 이용하는 패스워드를 기초로 한 암호(PBE)를 구현하고자 한다.



[그림 2.PBE 의 암호화 과정]



[그림 3. PBE 의 복호화 과정]

패스워드를 기초로 한 암호(PBE)는 그림 2.3 과 같은 과정을 거쳐 구현된다. 이를 총 3 가지 단계로 구분하면

1. KEK 생성

2. 세션 키 생성과 암호화
3. 메시지 암호화

와 같은 과정을 거치게 된다. 세 과정에서 모두 의사난수가 활용된다. KEK 생성과정에서 솔트(salt)라 불리는 난수를 입력한 패스워드 뒤에 붙여 일방향 해시함수에 입력한다. 세션 키(CEK)는 의사난수 생성기를 이용하여 생성한다. 메시지 암호화를 위해서 사용할 암호화 알고리즘은 블록암호모드 중 CTR 로 CTR 모드는 카운터를 만들기 위해 nonce(비표)를 활용하는데 이 때 nonce 는 의사난수 생성기를 이용하여 생성하게 된다. 따라서 CTR 을 암호화 알고리즘을 사용하는 PBE 를 구현 하게 되면 전 과정에 걸쳐 의사난수를 이용하게 된다.

구현한 암호시스템에 대한 성능분석 설계 및 결과

구현된 암호시스템에 대한 성능 분석은 크게 두가지로 나누어 진행한다. 첫째 구현한 의사난수 생성기가 암호시스템에 사용 가능하도록 무작위성과 예측불가능성을 보장하는지, 둘째 구현된 암호시스템의 암호화 복호화가 정상적으로 일어나는지 여부와 암호화와 복호화를 하는데 걸리는 시간을 분석하고자 한다.

의사난수 생성기에 대한 분석은 출력되는 난수의 범위를 좁게 한 뒤, 다량의 난수를 생성하고, 생성 가능한 숫자 별 빈도의 분석과 생성된 난수를 순서쌍으로 나타내어 좌표평면에 점을 찍어보는 방식으로 무작위성이 충족되는지 확인하려 한다. 예측 불가능성은 일방향 해시 함수를 이용하기 때문에 보장된다.

구현된 암호시스템의 암호화 복호화의 정상적 실행 여부는 직접 시행해 보는 것으로 확인한다. 해당 과정에 소요되는 시간은 첫번째로는 구현된 암호시스템을 일반적으로 수행하여 시간을 측정하고 두번째로는 CTR 이 병렬처리가 가능하기 때문에 병렬처리를 한 후 시간을 측정하려 한다. CTR 암호를 병렬처리 함으로서 효율성이 얼마나 증가하는지 분석 할 것이다.

코드 및 결과

일방향 해시 함수를 이용한 의사난수 생성기

```
class HashRandom:
    def __init__(self,size,seed):#size == 출력할 16 진수형 난수의 길이
        self.size=size/2
        self.seed = int(seed)
    def getRand(self):#16 진수 형태의 정수형 난수 생성
        seed_str = str(self.seed)
        seed_enc = seed_str.encode()
        random_hash = hash-
lib.shake_128(seed_enc).hexdigest(int(self.size)) #size 자리 해시값 생성
        self.seed = self.seed + 1
        return str(random_hash)
```

일방향 해시함수를 이용하여 의사난수를 생성하기 위한 클래스이다. 클래스 생성자로는 생성할 의사난수의 길이와 seed 를 받는다. getRand 함수를 통해 16 진수 형태의 난수를 생성하게 된다. 입력받은 seed 를 문자열 형태로 변환한 뒤 SHA-1 을 이용하여 해시 값을 얻는다. 해당 코드에서 사용한 shake_128 은 SHA-1 일방향 해시 함수로 출력될 해시값의 길이를 지정 할 수 있어 사용하였다. 해시값을 통해 난수를 얻은 이후에는 카운터를 +1 하여 다음에 생성할 난수에 적용 할 수 있도록 한다.

난수 생성기를 이용한 비표와 키의 생성

```
nonce = HashRandom(8,time.time())
rnd = HashRandom(16,time.time_ns())

#비표 생성
with open("nonce.txt", 'wt',encoding = 'utf-8') as f:
    f.write(nonce.getRand().upper())

#salt 생성
salt = rnd.getRand().upper()
with open("salt.txt",'wt',encoding='utf-8') as f:
    f.write(salt)
#세션 키 생성
CEK = rnd.getRand().upper()
with open("CEK.txt", 'wt',encoding = 'utf-8') as f:
    f.write(CEK)

#KEK 생성
passwd = input('enter password :')
passwd +=salt
```

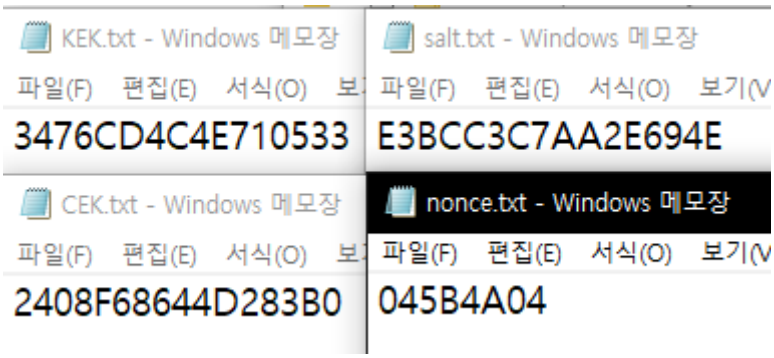
```
passwd = passwd.encode()
KEK = hashlib.shake_128(passwd).hexdigest(8)

with open("KEK.txt", 'wt',encoding = 'utf-8') as f:
    f.write(KEK.upper())
```

앞서 생성한 의사난수 클래스를 선언하여 비표와 세션 키(CEK)를 생성하고 텍스트 파일 형태로 출력한다. KEK 를 얻기 위해 패스워드를 입력하고, 이를 salt 와 결합하고 일방향 해시함수를 거친 뒤 마찬가지로 텍스트파일 형태로 출력한다.

비표 및 키 생성 결과

```
(venv) PS S:\repos\security\TermProject> python getRandom.py
enter password :CompuerSecurity
```

[그림 4. 파이썬 실행]



[그림 5. 생성된 키와 비표]

그림 4 와 5 에서 확인 할 수 있듯 비밀번호를 입력하고 이를 통해 생성된 KEK 와 세션 키(CEK) 그리고 salt 와 nonce(비표) 가 정상적으로 16 진수의 형태로 생성되었음을 알 수 있다.

패스워드를 기초로 한 암호

```
int main()
{
    clock_t start, end;
    double result;
    std::ifstream readFile("test1.txt");//평문
    std::string plain_text;
    std::getline(readFile, plain_text);

    std::ifstream readKEK("../KEK.txt");//KEK
    std::string KEK;
    std::getline(readKEK, KEK);
```

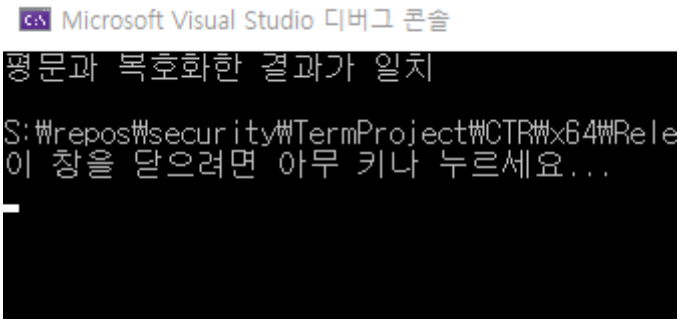
```
    std::ifstream readCEK("../CEK.txt");//CEK
    std::string CEK;
    std::getline(readCEK, CEK);

    std::ifstream readNonce("../nonce.txt");//비표
    std::string nonce;
    std::getline(readNonce, nonce);

    std::string CEK_encrypted;//CEK 암호화 하여 저장
    DES EncCEK(CEK, KEK, MODE::ENCRYPTION);
    CEK_encrypted = EncCEK.encryption();

    std::ofstream writeCEK("CEK_encrypted.txt");
    writeCEK << CEK_encrypted;

    CTR enc(plain_text, CEK,nonce);//암호화
    enc.execute();
    std::string enc_str = enc.getResult();
    std::ofstream writeEnc("enc_str.txt");
    writeEnc << enc_str;

    CTR dec(enc_str, CEK, nonce);//복호화
    dec.execute();
    std::string dec_str = dec.getResult();
    std::ofstream writeDec("dec_str.txt");
    writeDec << dec_str;

    if (plain_text.compare(dec_str) == 0)
    {
        std::cout << "평문과 복호화한 결과가 일치\n";
    }
    else
        std::cout << "복호화가 제대로 이루어지지 않음\n";
```

기존에 생성 했던 비표와 각종 키를 텍스트 형태로 읽어 와 PBE 에 활용한다. 이 때 PBE 에서 CEK 와 salt 는 따로 보관하기 때문에 CEK 를 암호화 하여 별도의 파일로 다시 저장한다. 대칭 암호로는 DES 기반의 CTR 을 이용하였기 때문에 해당 객체를 생성하여 입력받은 평문을 키를 이용하여 암호화 하고 암호화된 메시지를 통해 복호화 까지 진행하였다.

[그림 6. test.1]

[그림 7. 암호화된 평문]

[그림 8. 복호화된 평문]



[그림 9. 평문과 복호화된 결과가 일치함]

CTR 블록암호모드

```
class CTR
{
public:
    CTR(std::string input, std::string key_, std::string nonce_);//생성자
    void setKey(std::string key_);//key setter
    void parseString();//문자열 64 비트 단위로 잘라 저장
    std::string XORBlock(std::string prev_enc_, std::string block_);//xor 연산
    std::string counterStream(int in-
dex);//nonce 와 블록 번호를 결합하여 카운터 생성
    void encdec(int x, int n);//암호화 복호화
    int getThreadNum();//적절한 멀티스레드 개수 반환
    void execute();//실행
    void nonMulti();//단일 스레드 실행
    std::string getResult();// 암호화 복호화 결과 리턴
    std::string dec2hex(int d);//10 진수 16 진수 변환
private:
    std::string input_string; //입력 문자열
    std::vector<std::string> str_vec;//파싱한 문자열
    std::vector<std::string> result_vec;//암호화 복호화 한 결과
    std::string nonce;//비표
    std::string arr = "0123456789ABCDEF";
    int last_block_length;//마지막 블록 문자열 길이
    std::string key;//키
    std::vector<std::thread> threads;//멀티스레드 벡터
};
```

CTR 블록암호 모드에 이용되는 함수와 변수들이다. CTR 클래스의 생성자에서 평문, 키, 비표를 입력받고 초기화 한다.

parseString()함수를 이용해 입력받은 평문을 64 비트 크기로 잘라서 블록을 만든다. 마지막 블록 크기가 64 비트가 안될 경우 '0'으로 채운다. 이는 패딩을 하기 위함이 아닌 xor 연산을 하기 위함으로 xor 연산 이후 추가한 만큼 즉시 삭제한다.

counterStream() 함수에서는 nonce 와 블록 번호를 결합하여 카운터를 생성한다.

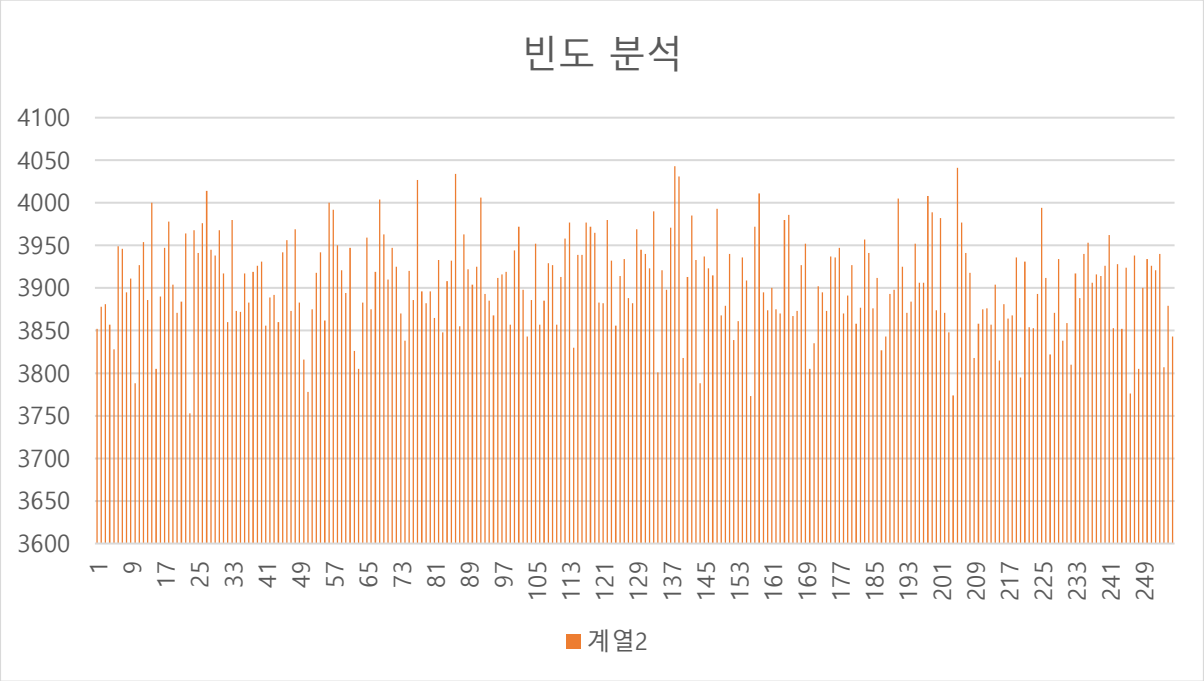
encdec(int x, int n) 함수에서는 암호화와 복호화를 진행한다. 블록 번호에 맞는 카운터를 DES 를 이용하여 암호화 한 뒤 평문 블록과 XOR 연산을 하여 암호화와 복호화를 한다. 마지막 블록의 경우 크기가 64 가 아니었을 경우 추가한 '0'글자를 전부 삭제한 후 결과에 저장한다. int x 는 스레드 번호를 int n 은 총 스레드 개수이다. 사용할 스레드 개수는 getThreadNum()을 이용하여 구한다.

Execute()함수는 멀티스레딩을 이용하여 암호화 복호화를 실행하는 함수이다. DES 와 CTR 은 이전 과제에서 구현한 부분을 활용하였기 때문에 모든 코드를 직접 설명하지는 않겠다.

DES,CTR 을 포함한 전체 코드 : security/TermProject at master · dsaf2007/security (github.com)

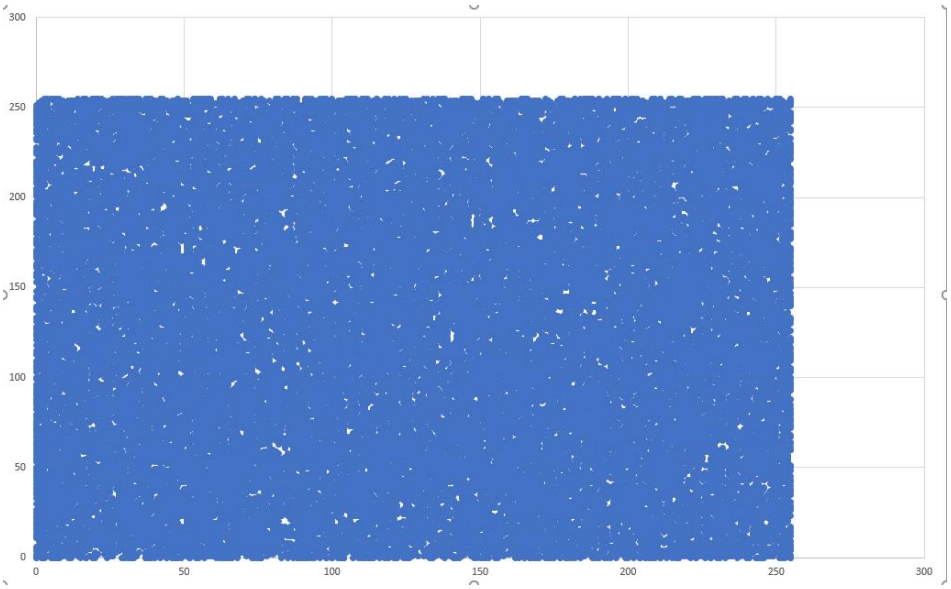
결과 분석

일방향 해시함수로 구현한 의사난수 생성기 분석



[그래프 1. 난수의 빈도 분석]

일방향 해시함수로 출력되는 난수의 최대 크기를 FF 로 제한하고 난수를 총 100 만개 생성하여 0 ~ 255 까지의 숫자들의 출현 빈도를 분석하였다. 100 만을 256 으로 나눌경우 약 3906 이 나온다. 위의 그래프의 결과에 의하면 각 숫자들은 최대 4043 최소 3753 으로 완전히 균일하게 3906 개씩 나오지는 않았으나 오차 범위 +- 150 이내에서 빈도를 가짐을 알 수 있다. 이는 난수 생성 횟수를 늘리면 늘릴수록



3906 에 가까운 숫자만큼 생성될 것으로 보인다.

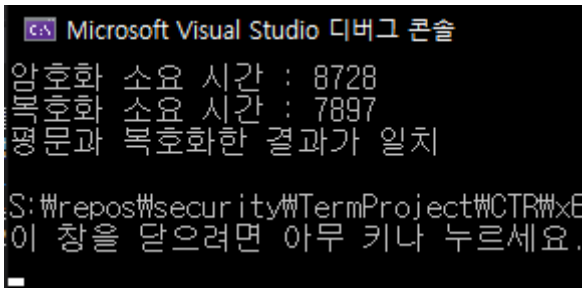
[그래프 2. 좌표평면에 나타낸 순서쌍]

의사난수 생성기로 난수를 5 만개씩 생성하여 순서쌍으로 나타낸 값을 좌표평면에 나타낸 것이 그래프 2 이다. 그래프에서 알 수 있듯 난수가 생성 될 수 있는 범위인 사각형 내부를 크게 빈 부분 없이 채우고 있음을 볼 수 있다. 그래프 1 과 그래프 2 의 결과를 분석해 보아 구현한 의사난수 생성기가 무작위성을 만족함을 알 수 있다.

강한 의사난수의 요건인 예측 불가능성은 seed 의 카운트를 1 씩 증가시켜 일방향 해시 함수를 거쳐 해시값을 얻는 것으로 만족한다. 이 때 생성된 난수를 다시 seed 로 이용하지 않는 이유는 이렇게 할 경우 예측이 가능해지기 때문이다. 이로 미루어 보아 생성한 의사난수 생성기는 강한 의사난수임을 알 수 있다.

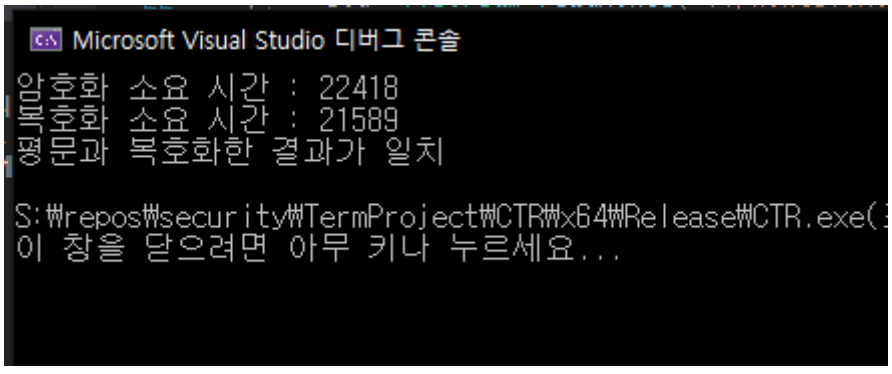
암호화 복호화 성능 분석

DES,CTR 기반의 PBE 암호시스템이 정확하게 작동함은 2.의 결과에서 알 수 있다.



[그림 10. 병렬처리 수행 결과]

암호화 성능을 분석하기 위한 평문은 10Mb 크기의 텍스트 파일로 약 1050 만 글자로 이루어져 있다. 그림 10. 은 병렬처리로 암호화 복호화를 수행 한 결과로 ms 단위로 측정되었다. 약간의 차이가 있지만 10Mb 크기의 텍스트를 암호화, 복호화 하는데 약 8 초 정도가 소비 되었다. 암호화와 복호화가 거의 비슷한 수치가 나와야 하지만 다르게 나온 것은 컴퓨터에서 수행되고 있는 다른 작업에 CPU 자원이 할당되면서 생긴 문제로 보인다.



[그림 11. 병렬처리를 하지 않은 결과]

병렬처리를 하지 않고 암호화 복호화를 진행하였을 경우 병렬처리를 하였을 때보다 약 3 배정도 오래 걸렸다. 작업관리자의 리소스 모니터를 통하여 모니터링 해보았을 때 병렬처리를 하였을 때에는 11 스레드를 병렬처리를 하지 않았을 때에는 4 스레드로 프로그램이 실행 되었다. 해당 프로그램에서는 1 ~ 8 개의 스레드로 병렬처리를 하게 구현해 놓았고 평문의 길이가 10500001 이라 7 개로 나누어 병렬처리를 하여 11 스레드 만큼 사용 된 것으로 보인다. 평문의 길이에 적합한 다른 수로 나누어 병렬 처리 할 경우 더 빠른 속도로 암호화와 복호화가 진행 될 수 있을 것으로 보인다.

통상적으로 RSA 보다는 대칭암호시스템이 속도가 더 빠른 것으로 알려져 있고 블록암호모드 중 에서도 CTR 만 병렬처리가 가능하기 때문에 구현 가능한 범위 내의 암호시스템 중에서는 병렬처리를 하는 CTR 이 가장 빠를 것으로 예상된다.

소감

한 학기 동안 컴퓨터 보안 수업을 통해 배운 모든 것을 이용하여 진행 할 수 있는 프로젝트였다. 비록 일방향 해시함수는 라이브러리를 이용하였지만 그를 제외한 대부분을 직접 구현 하여 사용할 수 있었다. 특히 학기 초반에 구현한 DES 는 정말 학기 중에 계속해서 이용 할 수 있었 던 것 같다. 이번 프로젝트에서는 난수생성을 위해 일방향 해시 함수를, PBE 를 구현하기 위해 DES 와 CTR 을 사용하였다. 의사난수 생성기로 ANSI x9.17 를 구현 해 보고 싶었으나 난이도가 높아 제대로 해보지 못하였다. 일방향 해시 함수로 구현한 의사난수 생성기에서도 아쉬운 부분은 일반적으로 사용하는 의사난수 생성 알고리즘처럼 생성할 난수의 범위를 마음대로 지정할 수 있도록 구현하지 못하였다는 것이다. 수식을 통해 구현 해 보려 하였으나, 소수점 아래 자리의 문제 때문에 무작위성이 제대로 충족되지 않는 모습을 보여 포기하였다. DES 와 CTR 은 이전 과제에서 구현하였던 코드를 적극적으로 활용하여 구현하였다. DES 와 CTR 을 활용한 PBE 를 구현하리라 마음을 먹었던 것도 DES 와 CTR 이 가장 구현이 잘 되어있는 부분이라 생각 하였기 때문이다.

한 학기 컴퓨터 보안 수업을 진행하면서 나의 전체적인 실력이 많이 늘었다는 생각이 든다. 복학 후 전체적으로 실력이 많이 부족함을 느꼈었는데 보안 수업을 어렵게라도 따라가기 위해 고군분투 하다보니 학기 초 보다 많이 실력이 향상되었음을 느낀다. 또한 컴퓨터 보안이라는 분야가 굉장히 매력적임을 느낄 수 있는 한 학기였다.

참고문헌

[1] 히로시 유키, “알기 쉬운 정보보호개론 3 판”, 인피니티북스, 2021