

REPORT



과목명 |

담당교수 |

학과 |

학년 | 3

학번 | 2017112138

이름 |

제출일 | 2021.06.17

1. 암호시스템에 사용 가능한 의사 난수 생성기 구현

난수의 용도는 아무리 강한 암호 알고리즘이라도 키가 공격자에게 알려지면 아무런 소용이 없기 때문에 적절한 난수를 사용해 키를 만들어 공격자에게 키를 간파 당하지 않도록 하기 위해서이다.

난수를 사용해서 만들 수 있는 것은 대칭암호나 메시지 인증코드 등에 사용되는 키를 만드는 것이나 공개키 암호나 디지털 서명 등에 사용하는 키 쌍을 만든 것이나 블록암호에서 사용하는 초기화 벡터를 생성할 때 또는 재전송 공격을 방지할 때 사용하는 비표를 생성할 때 또는 패스워드를 기초로 한 암호화에 사용되는 솔트를 생성할 때 그리고 일회용 패드를 생성할 때 사용된다.

그렇기 때문에 구현하는 의사 난수 생성기는 무작위성과 예측 불가능성 그리고 재현 불가능성을 가져야 한다.

지금부터 C++의 random_device를 사용해서 0~127까지의 정수를 난수로 생성하고 그에 맞는 아스키코드를 저장하여 무작위 키를 만들 것이다.

그렇게 되면 8자리의 키를 만들더라도 127^8 의 경우의 수를 가지기 때문에 전사공격으로 공격하기 힘든 난수를 생성하게 된다.

<코드>

```
#include<iostream>
#include<random>
#include<string>
#include<vector>
#include<cmath>
#include<fstream>
using namespace std;
int GetRand(int min, int max);

int GetRand(int min, int max)
{
    random_device rn;
    mt19937_64 rnd(rn());
    uniform_int_distribution<int> range(min, max);

    return range(rnd);
}

int main()
{
    string input = "My name is Jung yeo joon";
    string output = "";
    cout << "무작위 키 생성중..." << endl;
    char salt[8];
    for (int i = 0; i < 8; i++)
    {
        int a = GetRand(0, 127);
        salt[i] = a;
    }
}
```


```

        cout << "키 생성 완료!" << endl;
        cout << "키는 ";
        for (int i = 0; i < 8; i++)
            cout << salt[i];
        cout << "입니다." << endl;

        return 0;
}

```

방식은 random_device를 사용해서 0~127사이에 있는 정수를 중복되지 않도록 뽑고 이에 해당하는 아스키코드를 저장해서 키로 만든 방식이다. 경우의 수는 $127^{\text{키의 길이}}$ 가 된다.

 Microsoft Visual Studio 디버그 콘솔

```

무작위 키 생성중...
키 생성 완료!
키는 %2}8e}Rs입니다.

C:\Users\User\Documents\Visual Studio 2017\Projects\...
로 인해 종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.
    
```

2. 난수를 활용한 암호시스템 선정 및 설계

구현할 암호시스템: 메시지 인증코드

선정 이유의 타당성: 메시지 인증코드에서 마지막 MAC 값이 노출이 되더라도 공격자가 키를 추측하지 못하도록 해야 한다. 앞에서 생성한 키는 2^{127} (키의 길이) 즉 키의 길이가 8 이상만 되더라도 전사공격으로 뚫기 힘든 키를 생성할 수 있게 된다. 이를 통해서 메시지 인증코드 프로그램을 만들면 MAC 값과 입력 메시지가 노출이 되더라도 키를 예측할 수 없기 때문에 적절한 프로그램이라고 생각했다.

설계: 위에서 만든 방식으로 8자리 키를 무작위로 생성해준다. 키를 생성했으면 ipad와 opad를 사용할 것이다. 여기서 ipad와 opad는 생성된 키와 XOR 연산을 해줄 비트열이다. 키가 생성되면 ipad와 키를 XOR연산해서 값을 생성하고 그 값을 입력받은 메시지와 결합한다. 그리고 그렇게 결합된 메시지를 일방향 해시함수를 통해서 해시 값을 얻는다. 이어서 키를 opad와 XOR 연산해서 값을 생성하고 그 값을 앞에서 얻은 해시 값과 결합한다. 이렇게 얻은 값을 다시 일방향 해시함수를 통해서 해시 값을 얻으면 이것이 MAC 값이 된다. 일방향 해시 함수와 XOR연산과 결합을 적절하게 함으로써 MAC 값을 알더라도 그것을 통해서 키 값을 찾을 수 있는 알고리즘은 없다. 혹여나 전사공격으로 찾으려고 한다 하더라도 경우의 수가 너무나 많기 때문에 거의 불가능에 가깝다. 여기서 위험성을 줄이기 위해서 키의 길이를 조금만 늘리더라도 보안의 수준은 기하급수적으로 증가할 것이다.

<코드>

```
#include<iostream>
#include<random>
#include<string>
#include<vector>
#include<cmath>
using namespace std;

bool isPrime(int num);
int GetRand(int min, int max);
int getPrime(long double sum);
string XOR(char input[], char key[]);
string hashfunc(string input);
string HMac(string input, char key[]);
string HMac(string input, char key[], int &count);

bool isPrime(int num) //소수인지 판별하는 함수
{
    if (num < 2)
        return false;
    int a = (int)sqrt(num);
    for (int i = 2; i <= a; i++)
    {
        if (num%i == 0)
            return false;
    }
    return true;
}

int getPrime(long double sum) //sum보다 큰 소수 중 가장 작은 소수를 반환하는 함수
{
    int num = (int)sum + 1;
    while (isPrime(num) == false)
    {
        num++;
    }
    return num;
}

string XOR(char input[], char key[])
{
    int length = strlen(input);
    char result[100] = "";
    for (int i = 0; i < length; i++) {
        result[i] = input[i] ^ key[i];
    }
    return result;
}

string hashfunc(string input)
{
    long double sum = 0;
    string output = "";
    cout << fixed;
    cout.precision(30); //소수점 30자리까지 출력하도록 한다.
```

```

for (int i = 0; i < input.length(); i++)
{
    long double a = input[i]; //아스키 코드로 변환
    sum += a; //아스키 코드로 변환한 값들을 모두 더해준다.
}
sum = sum / getPrime(sum); //더한 값을 더한 값보다 큰 소수 중 가장 작은 소수로
나눠준다.
//소수점 4자리부터 23자리까지의 수를 구하기
sum = sum * 1000;
sum = sum - floor(sum);
sum = sum * 100000000000000000000;
cout.precision(0); //소수점 출력 x
if (floor(sum) == 0) //0일 경우 sum을 임의의 수(나눠서 소수점 자리로 나오기 힘든 수)로
변환
{
    output = "1234567891011121314";
}
else
{
    char result[] = { '' };
    string middle = to_string(floor(sum));
    int k = 0;
    while (middle[k] != '.') {
        output += middle[k];
        k++;
    }
}
return output;
}

string HMac(string input, char key[])
{
    int a = 0;
    input += to_string(a);
    string ipadstr = "";
    string opadstr = "";
    char ipad[8] = { '0','0','1','1','0','1','1','0' }; //키와 XOR 연산을 할 비트
    char opad[8] = { '0','1','0','1','1','1','0','0' }; //키와 XOR 연산을 할 비트

    ipadstr = XOR(key, ipad); //키와 ipad xor연산
    input += ipadstr; //키와 ipad를 xor연산한 결과를 메시지와 결합
    input = hashfunc(input); //결합한 메시지의 해시 값 계산

    opadstr = XOR(key, opad); //키와 opad XOR
    input += opadstr; //키와 opad를 xor 연산한 결과를 해시 값과 결합
    input = hashfunc(input); //결합한 메시지 해시 값 계산

    return input; //마지막 결과물 즉, Mac 값 반환
}

string HMac(string input, char key[], int &count)
{
    input += to_string(count);
    string ipadstr = "";
    string opadstr = "";

```

```

char ipad[8] = { '0','0','1','1','0','1','1','0' }; //키와 XOR 연산을 할 비트
char opad[8] = { '0','1','0','1','1','1','0','0' }; //키와 XOR 연산을 할 비트

ipadstr = XOR(key, ipad); //키와 ipad xor연산
input += ipadstr; //키와 ipad를 xor연산한 결과를 메시지와 결합
input = hashfunc(input); //결합한 메시지의 해시 값 계산

opadstr = XOR(key, opad); //키와 opad XOR
input += opadstr; //키와 opad를 xor 연산한 결과를 해시 값과 결합
input = hashfunc(input); //결합한 메시지 해시 값 계산
count++; //재전송 공격을 방지하기 위해서 count를 1씩 증가시킨다.
return input; //마지막 결과물 즉, Mac 값 반환
}

```

```

int GetRand(int min, int max)
{
    random_device rn;
    mt19937_64 rnd(rn());
    uniform_int_distribution<int> range(min, max);

    return range(rnd);
}

```

```

int main()
{
    string input = "My name is Jung yeo joon";
    string output = "";
    cout << "무작위 키 생성중..." << endl;
    char salt[8];
    for (int i = 0; i < 8; i++)
    {
        int a = GetRand(0, 127);
        salt[i] = a;
    }
    cout << "키 생성 완료!" << endl;
    cout << "키는 ";
    for (int i = 0; i < 8; i++)
        cout << salt[i];
    cout << "입니다." << endl;

    output = HMAC(input, salt);
    cout << "MAC: " << output << endl;

    string input2 = "";
    string input3 = "";
    string output2 = "";
    string output3 = "";
    int count = 0;
    char key2[8];
    char key3[8];
    cin.ignore();

    cout << "인증할 메시지 입력: ";
    getline(cin, input2);
}

```

```

cout << "키 입력(8자리): ";
for (int k = 0; k < 8; k++)
{
    cin >> key2[k];
}
output2 = HMac(input2, key2, count);
cout << "Mac: " << output2 << endl;

if (output == output2)
    cout << "올바른 Mac 값입니다." << endl;
else
    cout << "올바른 Mac 값이 아닙니다." << endl;
cout << endl;
cin.ignore();

cout << "인증할 메시지 입력: ";
getline(cin, input3);
cout << "키 입력(8자리): ";
for (int k = 0; k < 8; k++)
{
    cin >> key3[k];
}
output3 = HMac(input2, key2, count);
cout << "Mac: " << output3 << endl;

if (output == output3)
    cout << "올바른 Mac 값입니다." << endl;
else
    cout << "올바른 Mac 값이 아닙니다." << endl;

return 0;

```

```

}

```


3. 구현한 암호 시스템에 대한 성능 분석 설계 및 결과

```
Microsoft Visual Studio 디버그 콘솔
무작위 키 생성중...
키 생성 완료!
키는 $ @4^3dG입니다.
MAC: 2019034670291830528

인증할 메시지 입력: My name is Jung yeo joon
키 입력(8자리): $ @4^P3dG
Mac: 2019034670291830528
올바른 Mac 값입니다.

인증할 메시지 입력: My name is Jung yeo joon
키 입력(8자리): $ @4^P3dG
Mac: 2202918693536730624
올바른 Mac 값이 아닙니다.

C:\Users\user\Documents\Visual Studio 2017\Projects\Security_final\Debug\Security_final.exe(1816 프로세스)이
로 인해 종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]->[디버깅이 중지되면 자동으로 콘솔 닫기]
를 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.
```

일단 결과를 보면 키가 생성되고 인증할 메시지를 입력하고 올바른 키를 입력하면 MAC 값이 일치해서 올바른 MAC 값이라고 출력되는 것을 볼 수 있다. 하지만 메시지 인증코드에서 가장 취약한 부분이 재전송 공격이다. 즉, 공격자가 MAC 값을 보존해두고 그 MAC 값을 반복해서 송신하는 공격을 말한다. 이런 것에 대비하기 위해서 순서번호 즉 송신 메시지에 매회 1씩 증가하는 번호를 붙여서 위의 결과 화면에서 다시 한번 시도했을 때 올바른 메시지와 키를 입력했음에도 불구하고 올바른 MAC 값이 아니라고 출력되는 것을 확인할 수 있다. 이런 방법을 사용해서 재전송 공격에 대비하였다.

여기서 만든 MAC 값은 일방향 해시 함수에서 나온 최종 해시 값이기 때문에 이 MAC 값을 통해서 키를 추측할 수 없다. 따라서 공격할 수 있는 방법이 전사공격 밖에 없다. 따라서 전사공격 취약한지 강한지를 알아보는 실험을 진행해보았다.

<코드>

```
#include<iostream>
#include<random>
#include<string>
#include<vector>
#include<cmath>
#include<ctime>
using namespace std;

int GetRand(int min, int max)
{
    random_device rn;
    mt19937_64 rnd(rn());
    uniform_int_distribution<int> range(min, max);

    return range(rnd);
}

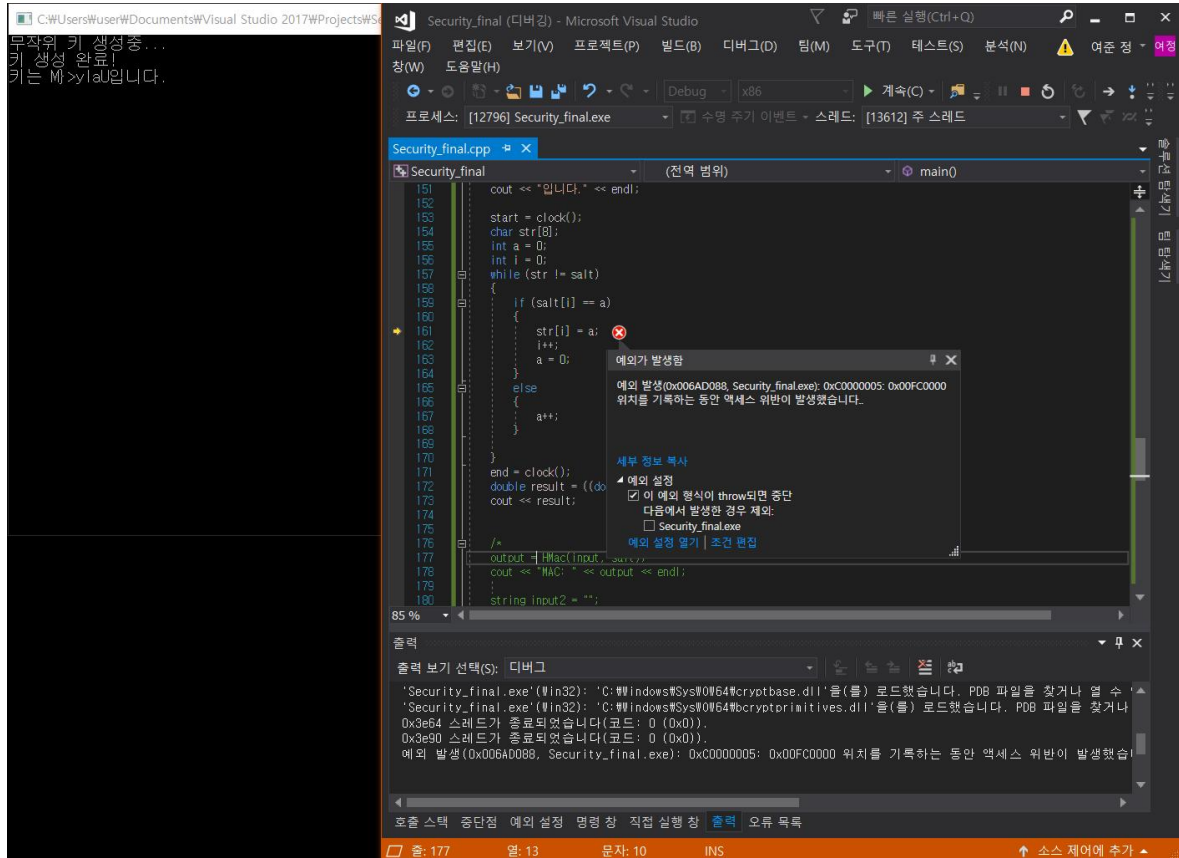
int main()
{
    time_t end, start;
    string input = "My name is Jung yeo joon";
    string output = "";
    cout << "무작위 키 생성중..." << endl;
    char salt[8];
    for (int i = 0; i < 8; i++)
    {
        int a = GetRand(0, 127);
        salt[i] = a;
    }
    cout << "키 생성 완료!" << endl;
    cout << "키는 ";
    for (int i = 0; i < 8; i++)
        cout << salt[i];
    cout << "입니다." << endl;

    start = clock();
    char str[8];
    int a = 0;
    int i = 0;
    while (str != salt)
    {
        if (salt[i] == a)
        {
            str[i] = a;
            i++;
            a = 0;
        }
        else
        {
            a++;
        }
    }
    end = clock();
    double result = ((double)start - (double)end) / CLOCKS_PER_SEC;
```

```
cout << result;
```

```
return 0;
```

```
}
```



위의 화면과 같이 전사공격을 진행하면 오류가 발생하는 것을 확인할 수 있다. 근데 여기서는 키를 안다고 가정을 하고 전사공격을 실행한 것이지만 키를 모르는 경우 경우의 수가 너무나도 많기 때문에 알 방법이 없다. 따라서 전사공격에도 안전한 모습을 볼 수 있다.