

컴퓨터보안 보고서

실습과제 #12

강좌 명: 컴퓨터보안

교수님: 정준호 교수님

학과: 컴퓨터공학과

학년: 3학년

학번: 2017112138

이름: 정여준

키

암호기술을 사용하려면 반드시 키라 불리는 대단히 큰 수가 필요한데 여기서 중요한 것은 수 그 자체의 크기보다는 키 공간의 크기가 중요하다. 키 공간의 크기는 키의 비트 길이로 결정한다. 키는 평문과 같은 가치를 가지며 도청자에게 키가 넘어가는 것은 평문이 넘어가는 것과 같다고 말한다. 따라서 키를 비밀로 하는 것은 당연하고 이에 의해서 기밀성이 지켜진다.

키의 종류

대칭암호: 암호화와 복호화에서 공동 키를 사용한다. 키는 송신자와 수신자만 공유한다. 양측이 공유 키를 비밀로 유지한다.

공개키 암호: 암호화와 복호화에 다른 키를 사용하고 개인 키를 비밀로 유지한다. 암호화할 때는 공개키를 사용하고 복호화 할 때 개인키를 사용한다.

메시지 인증 코드: 송신자와 수신자가 공동의 키를 사용해서 인증을 수행한다.

디지털 서명: 서명 작성과 서명 검증에 서로 다른 키를 사용한다.

세션 키: 통신 때마다 한 번만 사용되는 키

마스터 키: 반복적으로 사용되는 키

키 생성

난수를 이용한 키 생성: 다른 사람이 추측하기 어려워야 하는 성질을 충족하기 위해서 추측하기 힘든 난수가 키로 적합하다. 여기서 주의해야 할 점은 자신이 적당한 바이트 열을 만들면 안된다. 그 이유는 랜덤한 값이라고 스스로 생각할지라도 인위적인 편중이 존재할 수 있기 때문이다. 따라서 암호용으로 이용하는 의사난수 생성기를 사용한다.

키 갱신

공통 키를 사용하여 통신을 하고 있을 때 정기적으로 키를 교환하는 방법이다. 송신자와 수신자가 동시에 같은 방법으로 키를 교환해야 한다. 예시로는 1000문자를 통신할 때마다 현재 키의 해시 값을 다음 키로 사용하는 방법이 있을 수 있다. 이렇게 하면 키 노출 시 과거 통신의 복호화를 막을 수 있다는 장점이 있다.

패스워드를 이용한 키 생성

패스워드 혹은 패스 프레이즈로부터 키를 만든다. 패스워드를 일방향 해시 함수에 입력해서 얻어진 해시 값을 키로 이용하는 방법이 있다. 패스워드에 솔트라 불리는 난수를 부가해서 일방향 해시 함수에 입력하고 그 출력을 키로 사용하는 방법이 있다.

random_device를 사용해서 키 만들기

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

double getRand(double min, double max)
{
    random_device rn;
    mt19937_64 rnd(rn());
    uniform_real_distribution<double>range(min, max);

    return range(rnd);
}

int main()
{
    cout.precision(100);
    double a = getRand(0, 1);
    a *= pow(10, 100);
    cout << a << endl;
    string str = to_string(a);
    string key = "";
    int i = 0;
    while (1)
    {
        if (str[i] == '.')
            break;
        key += str[i];
        i++;
    }
    cout << key;
}
```

Microsoft Visual Studio 디버그 콘솔

```
6373403091917882969803569200390938322232105283906306267276738826607186721516231177180326722891939840
C:\Users\user\Documents\Visual Studio 2017\Projects\Security13\Debug\Security13.exe(21740 프로세스)이
종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]->[디버깅이 중지되면 자동으로 콘솔
을 닫습니다]를 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.
```

C++의 <random> 헤더파일을 사용해서 random_device를 사용해서 이전에 나온 난수가 반복되지 않도록 하고 0과 1사이의 실수를 난수로 생성하고 이를 소수점 100자리까지 표현하고 다시 10^{100} 을 곱해서 큰 수의 키로 하여 예측하기 힘들고 반복되기 어려운 키를 생성할 수 있다.

문자열이 아닌 정수형이나 실수형 자료형을 사용할 경우 오버플로우가 발생하기 때문에 문자열을 사용해서 더욱 더 긴 키를 만들어내고 이를 저장하고 쓸 수 있게 한다.

Password와 Salt를 사용해서 Key 생성하기

Salt는 해시함수를 돌리기 전에 원문에 임의의 문자열을 덧붙이는 것을 말한다. 단어 뜻 그대로 임의의 문자열을 붙인다는 의미의 소금 친다는 것이다. 이렇게 할 경우 생성한 key를 추측하기는 더 어려워질 것이고 원문으로 해독하기도 더욱 어려워질 것이다.

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

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 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]; //아스키코드로 변환
        sum += a; //아스키코드로 변환한 값들을 모두 더해준다.
    }
    sum = sum / getPrime(sum); //더한 값을 더한 값보다 큰 소수 중 가장 작은 소수로 나뉜다.
    //소수점 4자리부터 23자리까지의 수를 구하기
    sum = sum * 1000;
    sum = sum - floor(sum);
    sum = sum * 10000000000000000000;
    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;
}

double getRand(double min, double max)
{
    random_device rn;
    mt19937_64 rnd(rn());
    uniform_real_distribution<double>range(min, max);

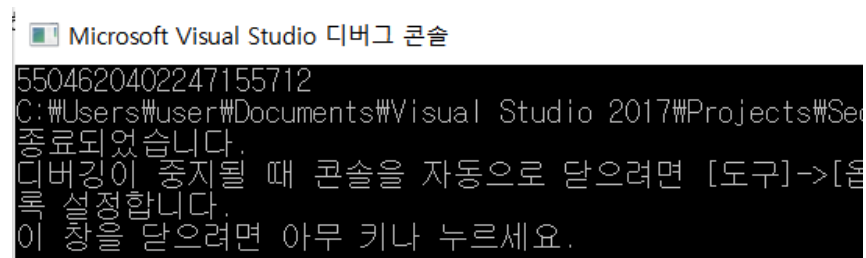
    return range(rnd);
}

int main()
{
    string password = "123456";
    cout.precision(100);
    double a = getRand(0, 1);
    a *= pow(10, 100);
    string str = to_string(a);
    string salt = "";
    int i = 0;
    while (1)
    {
        if (str[i] == '.')
            break;
        salt += str[i];
        i++;
    }

    string input = password + salt;

    string key = hashfunc(input);
    cout << key;
}

```



Microsoft Visual Studio 디버그 콘솔

```

5504620402247155712
C:\Users\User\Documents\Visual Studio 2017\Projects\Sec
종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[콘솔]
를 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.

```

생성한 password 문자열에 임의의 문자열 salt 문자열을 더해준다. 이 상태에서 직접 만든 일방향 해시함수에 넣어 해시 값을 얻어 이 해시 값을 키로 한다.

위에서 설명한 두 방법으로 키를 만든 이후에 이전 과제에서 실행했던 OTP를 사용해서 암호화 복호화를 진행해본다. 이전 과제에서는 키를 .txt 파일에 저장한 값을 받아와서 그 값으로 암호화와 복호화를 진행했다. 이럴 경우에는 사실 이 .txt 파일만 찾으면 키를 찾아내는 것이기 때문에 기밀성 부분에 있어서 좋지 못하다고 느꼈다. 그래서 위의 두 방법으로 키를 만들어서 이를 통해서 암호화와 복호화를 진행해본다. 일단 난수를 이용해서 만든 키로 암호화 및 복호화를 진행하는 과정이다.

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

double getRand(double min, double max)
{
    random_device rn;
    mt19937_64 rnd(rn());
    uniform_real_distribution<double>range(min, max);

    return range(rnd);
}

string OTP(string input, string key, int len)
{
    char c1[100000] = "";
    char c2[100000] = "";
    for (int i = 0; i < len; i++)
    {
        c1[i] = input[i];
        c2[i] = key[i];
    }

    char result[100000] = "";

    for (int i = 0; i < len; i++)
        result[i] = c1[i] ^ c2[i];

    return result;
}

int main()
{
    cout.precision(100);
    double a = getRand(0, 1);
    a *= pow(10, 100);
    string str = to_string(a);
    string key = "";
    int i = 0;
    while (1)
    {
        if (str[i] == '.')
            break;
        key += str[i];
        i++;
    }
    string input;
```

```

cout << "평문 입력: ";
cin >> input;

string output = OTP(input, key, input.length());
cout << "암호화: " << output << endl;
cout << "복호화: " << OTP(output, key, output.length());
}

```

```

Microsoft Visual Studio 디버그 콘솔
평문 입력: abcdefghijklmnopqrstuvwxyz,./;\'[]<>?:\"{}=-09876!#$%^&)%+_)(*&#|
암호화: PSPRUQWXX^[Z^###HJGA@GCOMJ↔↔i i j ㅈ ㅊ ㅌ JKㅍL
복호화: abcdefghijklmnopqrstuvwxyz,./;\'[]<>?:\"{}=-09876!#$%^&)%+_)(*&#|
C:\Users\User\Documents\Visual Studio 2017\Projects\Security13\Debug\Security
종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]->[디버깅
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.

```

이렇게 될 경우 키가 너무나 큰 난수이기 때문에 전사공격으로도 알 수 없고 저장되어 있는 파일이 아니고 컴퓨터 상에서 직접 랜덤하게 만들어낸 수이기 때문에 알 방법이 없다.

다음은 password와 salt를 사용해서 만든 키를 통해서 암호화와 복호화 작업을 해본다.

패스워드를 입력받은 후에 난수를 생성한다. 생성한 난수를 입력받은 password 뒤에 salt하고 이렇게 만들어진 문자열을 일방향 해시함수를 통해 해시 값을 얻는다. 이렇게 생성한 해시 값을 키로 하여 암호화 복호화 작업을 한다.

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

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 OTP(string input, string key, int len)
{
    char c1[100000] = "";
    char c2[100000] = "";
    for (int i = 0; i < len; i++)
    {
        c1[i] = input[i];
        c2[i] = key[i];
    }

    char result[100000] = "";

    for (int i = 0; i < len; i++)
        result[i] = c1[i] ^ c2[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]; //아스키코드로 변환
    sum += a; //아스키코드로 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 * 10000000000000000000;
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;
}

double getRand(double min, double max)
{
    random_device rn;
    mt19937_64 rnd(rn());
    uniform_real_distribution<double>range(min, max);

    return range(rnd);
}

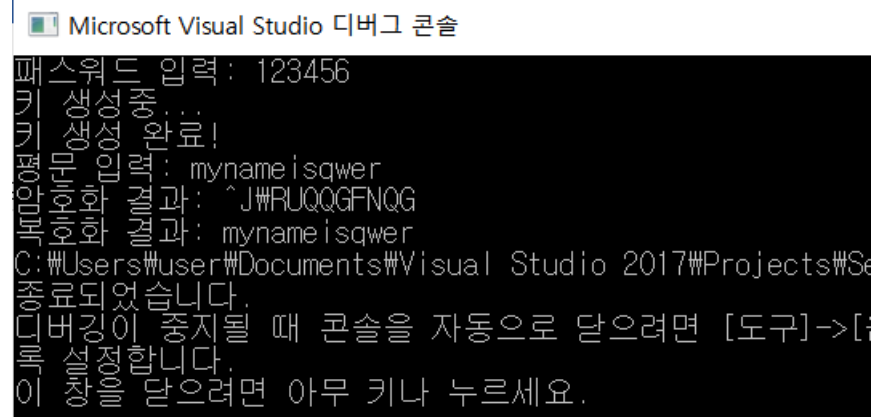
int main()
{
    string password;
    cout << "패스워드 입력: ";
    cin >> password;
    cout << "키 생성중..." << endl;
    cout.precision(100);
    double a = getRand(0, 1);
    a *= pow(10, 100);
    string str = to_string(a);
    string salt = "";
    int i = 0;
    while (1)
    {
        if (str[i] == '.')
            break;
        salt += str[i];
        i++;
    }
    string key = hashfunc(password+salt);
    cout << "키 생성 완료!" << endl;
}

```

```

string input;
cout << "평문 입력: ";
cin >> input;
int len = input.length();
string output = OTP(input, key, len);
cout << "암호화 결과: " << output << endl;
cout << "복호화 결과: " << OTP(output, key, len);
}

```



Microsoft Visual Studio 디버그 콘솔

```

패스워드 입력: 123456
키 생성 중...
키 생성 완료!
평문 입력: mynameisqwer
암호화 결과: ^J#RUQQGFNQG
복호화 결과: mynameisqwer
C:\Users\User\Documents\Visual Studio 2017\Projects\Se
종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.

```

앞에서 첫번째 방법으로 만든 키보다 훨씬 복잡하고 예측하기 어렵다. 그 이유는 일단 패스워드를 알아야 하고 이 패스워드와 결합한 엄청 큰 난수를 알아내야 하고 이렇게 만들어진 문자열을 일방향 해시함수를 통해 나온 해시 값을 알아내야 하기 때문에 키 값을 찾는 것은 거의 불가능에 가깝다고 볼 수 있다. 이를 더 업그레이드하는 방법은 일방향 해시 함수를 반복적으로 돌려서 더욱 더 알기 힘든 해시 값을 만드는 방법도 있다.

분석

이번에 만든 두개의 키는 모두 대칭 암호이다. 이는 지난번 전자서명 과제에서 사용한 공개키 암호에 반해 엄청나게 빠른 속도를 자랑한다. 첫번째 만든 키는 일반적인 대칭 암호 키와 비슷하다면 두번째로 만든 키는 메시지 인증코드에서 사용하는 키와 유사하다고 볼 수 있다. 이는 일방향 해시함수의 사용 유무로 판단한 것인데 첫번째는 단순히 엄청나게 큰 난수를 생성해서 그 값을 키로 사용하는 거인 반면에 두번째로 만든 키는 첫번째와 동일하게 만든 수를 입력받은 패스워드에 salt 작업을 하고 이렇게 만든 문자열을 일방향 해시 함수를 통해서 해시 값을 얻고 이 해시 값을 키로 사용하는 방법이다. 한눈에 봐도 훨씬 복잡하게 설계되어 있기 때문에 기밀성이 더욱 보장되는 것을 확인할 수 있다. 또한 두번째의 키 같은 경우에는 주기적으로 일방향 해시 함수를 이용하여 키를 갱신할 수 있어 좋은 방법이라고 생각된다.

소감

이번에 키에 대해서 공부하면서 이전 과제를 진행하면서 생성했던 키에 대한 공부를 다시 할 수 있었고 새로운 방법으로 키를 생성하고 이를 암호화와 복호화에 적용하면서 새로 만든 키에 대한 장점과 발전 가능성에 대한 것을 알 수 있어 좋았습니다. 또한 각 키들의 차이점과 각각의 장점과 단점을 다시 한번 상기시킬 수 있어 좋았습니다. 또한 이번 과제에서 키를 크게 만들기 위해서 정수나 실수형 데이터가 아닌 문자열로 저장하여 적용하기 쉬운 OTP 알고리즘으로 암호화와 복호화를 했는데 문자열이 아닌 정수형이나 실수형 데이터로 저장하여 블록형 암호 알고리즘이나 다양한 알고리즘에 응용할 수 있을 것 같은 가능성도 보았기 때문에 좋았습니다. 이번학기 배웠던 알고리즘들을 전체적으로 볼 수 있어서 과제를 하면서 공부가 되는 느낌을 받아 좋았습니다.