컴퓨터보안 보고서

실습과제 #11

강좌 명: 컴퓨터보안

교수님: 정준호 교수님

학과: 컴퓨터공학과

학년: 3학년

학번: 2017112138

이름: 정여준

전자서명

메시지 인증 코드의 한계에 메시지 인증 코드를 사용하면 메시지의 변경과 거짓 행세를 검출할 수 있지만 부인 방지에는 도움이 되지 않는 것이었다. 이런 것을 방지하기 위해서 디지털 서명은 개인적인 키를 써서 서명을 작성하고 수신자는 공개 키를 사용해서 서명을 검증하는 방식이다. 이렇게 메시지의 서명을 작성하는 행위는 서명용 개인 키와 검증용 공개 키로 나누어서 사용하는데 검증용 공개 키로는 서명을 작성할 수 없다. 여기서 보면 이전에 배웠던 공개키 암호와 비슷한데 순서가 반대이다. 즉, 디지털 서명은 공개키 암호를 역으로 사용함으로써 실현하는 프로그램을 말한다.

전자서명 방법

- 메시지에 직접 서명하는 방법
  1. 앨리스는 자신의 개인 키로 메시지를 암호화한다.
  2. 앨리스는 메시지와 서명을 밥에게 송신한다.
  3. 밥은 수신한 서명을 앨리스의 공개키로 복호화한다.
  4. 밥은 이제 서명을 복호화해서 얻어진 메시지와 앨리스로부터 직접 수신한 메시지를 비교한다.
- 메시지의 해시 값에 서명하는 방법
  1. 앨리스는 일 방향 해시 함수로 메시지의 해시 값을 계산한다.
  2. 앨리스는 자신의 개인 키로 해시 값을 암호화한다.
  3. 앨리스는 메시지와 서명을 밥에게 송신한다.
  4. 밥은 수신한 서명을 앨리스의 공개키로 복호화한다.
  5. 밥은 수신한 서명으로부터 얻어진 해시 값과 앨리스로부터 직접 수신한 메시지의 해시 값을 비교한다.

전자서명에 대한 공격

1. 중간자 공격

해결책: 공개 키가 정확한 상대의 것인지를 확인하는 것이 필요하다. 공개 키를 취급하는 소프트웨어는 공개 키의 해시 값을 표시하는 수단을 준비하는데 이 해시 값을 핑거프린트라고 한다.

2. 일 방향 해시 함수에 대한 공격

해결책: 전자서명에서 사용하는 일 방향 해시 함수는 충돌 내성을 가져야만 한다.

3. 전자 서명을 사용한 공개 키 암호 공격

해결책: 의미를 모르는 메시지에는 절대로 전자서명을 하면 안된다.

전자서명으로 해결할 수 없는 문제

전자서명은 기밀성을 보장하지 않는다. 서명검증을 할 공개키가 송신자의 공개 키인가에 대한 인증을 할 수 없다.

<소스코드>

```
#include<iostream>
#include<string>
#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 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;
}

int main()
{

        string input = "";
        cout << "메시지 입력: ";
        getline(cin, input);

        string output = hashfunc(input);
        cout << "입력의 해시 값: " << output << endl;
        _int64 sum = 0; //해시 값이 너무 크기 때문에 개인키로 암호화 처리하는 데 무리가 있어서
해시 값을 대체하는 값을 넣을 정수형 변수
        for (int i = 13; i <output.length(); i++)
        {
                sum += pow(10,output.length()-(i+1)) * ((int)output[i] - '0'); //해시 값에서 0의 자리부터
10만의 자리까지의 수를 떼어낸 수
        }
        cout << "대체된 해시 값: " << sum << endl;
        //개인키 (D,N) = (206141, 1022117)
        //공개키 (E,N) = (3125, 1022117)
        unsigned int N = 1022117;
        unsigned int D = 206141;
    //unsigned int E = 3125;
        _int64 result = 1;
        //(입력문)^D mod N 연산
        for (int i = 0; i < D; i++)
        {
                result = result * sum;
                result = result % N;
        }
        cout << "=========================송신자=======================" << endl;
        cout << "해시 값을 개인키로 암호화: " << result << endl;
        cout << "송신자에게 암호화한 값 전송..." << endl;
        _int64 result2 = 1;
        unsigned int E = 0;
        cout << "=========================수신자=======================" << endl;
        cout << "복호화 작업 공유 키 입력: ";
        cin >> E;
        //(암호문)^E mod N 연산
        for (int i = 0; i < E; i++)
        {
                result2 = result2 * result;
                result2 = result2 % N;
        }
        cout << "암호화한 값 공유키로 복호화: " << result2 << endl;
        if (result2 == sum)
                cout << "검증 성공" << endl;
        else
                cout << "검증 실패" << endl;

}
```

소스코드 분석

일 방향 해시함수는 이전에 직접 구현한 일 방향 해시함수를 사용했다. 여기서 나온 해시 값은 19자리의 수인데 이를 개인 키로 암호화하고 공유 키로 복호화할 경우에 자료형의 범위를 벗어나 오버플로우가 발생했기 때문에 처음에는 해시 값을 온전히 사용한 것이 아닌 해시 값에서 나온 값에서 각 자리의 수들을 합한 수를 해시 값을 대체할 수로 했지만 충돌내성이 없어 디지털 서명을 한 해시 값과 같은 해시 값을 갖는 다른 메시지가 나올 가능성이 있기 때문에 다른 방법을 찾다가 해시 값을 1의 자리부터 10만의 자리까지 잘라서 사용하는 방법 즉, 해시 값이 5027349577324003328로 나올 경우 4003328을 해시 값을 대체할 수로 사용했다. 이렇게 될 경우 아예 충돌이 안 일어나지는 않겠지만 웬만하면 일어나지 않는 것을 확인했다. 해시 값을 10만자리수로 했기 때문에 RSA 암호 알고리즘으로 암호화하기 위해서 N은 10만보다 큰 100만이상의 수를 사용했다.

p = 1009, q = 1013으로 N = 1022117을 사용하였고

L = (p-1, q-1)의 최소공배수로 255024이고

RSA 암호 알고리즘에 맞게 E와 D를 구한 결과 E는 3125, D는 206141을 사용하게 됐다.

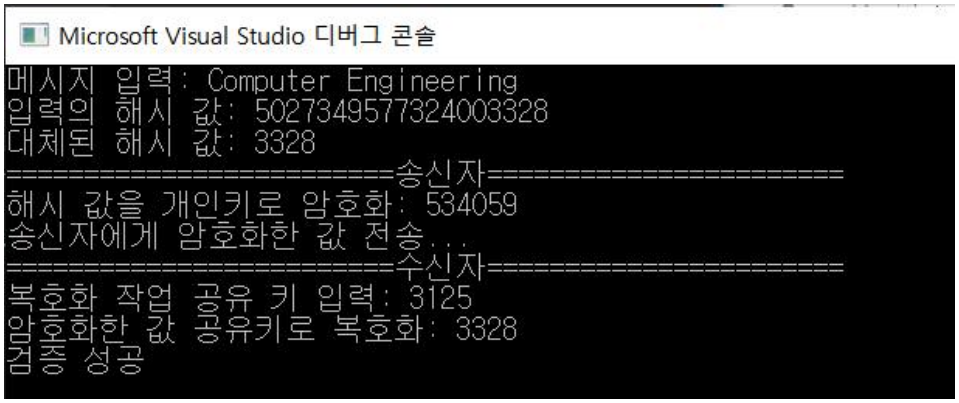
N과 D와 E를 unsigned int 형태로 저장하고 암호화할 때는 반복문을 사용해서

(입력문)^D mod N 연산을 했다. 여기서 (입력문)^D 연산을 먼저 할 경우 오버플로우가 발생하기 때문에 반복문에서 한번 계산할 때마다 mod N 연산을 해주었다.

또한 암호화된 값을 복호화할 때는 반복문을 사용해서 (암호문)^E mod N 연산을 했다. 동일하게 오버플로우를 피하기 위해서 계산할 때마다 mod N 연산을 해주었다.

이렇게 해서 입력된 문자를 일 방향 해시 함수를 통해 해시 값을 구하고 구한 해시 값을 통해 대체할 해시 값으로 변환한 이후에 개인 키로 암호화해서 송신하고 수신자는 공유 키를 통해서 검증을 하는 형태로 구현하였다.

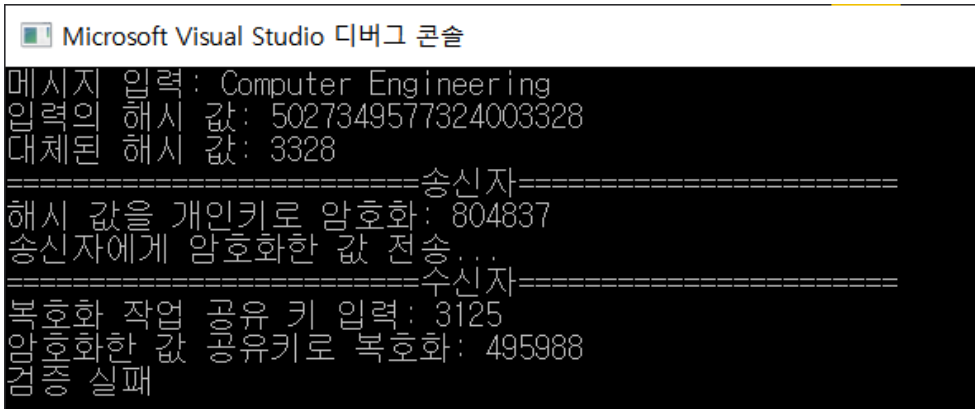
실행화면



임의로 설정한 올바른 개인키 (D,N) = (206141, 1022117)로 해시 값을 암호화해서 전송한다. 그럼 이 개인 키에 맞는 공개 키 (E,N) = (3125, 1022117)로 복호화해서 해시 값을 비교한다.

여기서 다른 사람이 개인키가 아닌 다른 키로 해시 값을 암호화해서 보내면 서명의 검증에 실패하기 때문에 올바른 사람이 보낸 것이 아니라고 판단 가능하다.

즉, 아래의 그림처럼 개인 키에서 D를 219401로 조정하고 다시 실행해보도록 하겠다.



위와 같이 올바르지 않은 개인 키로 암호화해서 송신할 경우 수신자가 공유 키로 검증했을 때 검증이 실패하기 때문에 진위를 알 수 있다.

소감

Visual Studio C++ 언어를 통해서 구현을 하는데 RSA알고리즘을 사용해서 해시 값을 암호화 복호화 작업을 하려고 했는데 계속되는 오버플로우로 인해서 좀 힘들었습니다. 그래서 처음에는 해시 값으로 얻은 값에서 각 자리에 있는 수를 모두 더하는 방식으로 구현하였으나 그렇게 될 경우 충돌내성이 없어서 디지털 서명을 한 해시 값과 같은 해시 값을 갖는 다른 메시지가 나올 가능성이 있기 때문에 오버플로우가 발생하지 않고 충돌내성을 최대한 있게 구현하기 위해서 노력했습니다. 결국 해시 값에서 10만자리까지의 정수를 떼어와서 해시 값을 대체하는 형태로 구현했습니다. 충돌이 아예 발생하지 않는 것은 아니겠지만 제가 실험해 본 결과 많은 충돌이 발생하지는 않을 것으로 판단해 이 방법으로 구현하게 되었습니다. 이를 통해서 다시 자료형의 크기 같은 것을 조사하고 공부하면서 조금 더 언어에 대한 지식을 높일 수 있어서 좋았습니다. 그리고 제가 만들었던 일 방향 해시함수를 사용하니 성취감도 있고 좋았습니다.