컴퓨터보안 보고서

실습과제 #10

강좌 명: 컴퓨터보안

교수님: 정준호 교수님

학과: 컴퓨터공학과

학년: 3학년

학번: 2017112138

이름: 정여준

메시지 인증 코드

메시지 인증 코드는 해시 알고리즘으로 수정 또는 변경을 검출할 수는 있지만 거짓 행세를 검출하는 것은 불가능하기 때문에 무결성 외에 인증이라는 절차가 필요하게 됐다. 메시지 인증 코드는 데이터가 변조되었는지를 검증할 수 있도록 데이터에 덧붙이는 코드이다. 원래의 데이터로만 생성할 수 있는 값을 데이터에 덧붙여서 확인하도록 하는 것이 필요하고, 변조된 데이터에 대해서 MAC을 생성하여 MAC도 바꿔 치기 할 가능성이 있으므로 MAC의 생성과 검증은 반드시 비밀키를 사용하여 수행해야만 한다.

메시지 인증 코드를 이용한 인증 순서

1. 송신자 앨리스와 수신자 밥은 사전에 키를 공유한다.
2. 송신자 앨리스는 송금 의로 메시지를 기초로 해서 MAC 값을 계산한다. (공유키 사용)
3. 송신자 앨리스는 수신자 밥에게 송금 의로 메시지와 MAC 값을 보낸다.
4. 수신자 밥은 수신한 송금 의뢰 메시지를 기초로 해서 MAC 값을 계산한다. (공유키 사용)
5. 수신자 밥은 앨리스로부터 수신한 MAC 값과 계산으로 얻어진 MAC 값을 비교한다.
6. 수신자 밥은 2개의 MAC 값이 동일하면 송금 의뢰가 틀림없이 앨리스로부터 온 것이라고 판단한다. 동일하지 않다면 앨리스로부터 온 것이 아니라고 판단한다.

메시지 인증 코드에 대한 공격

1. 재전송 공격: 도청, 보존해 둔 메시지와 MAC 값을 반복 송신하는 공격 방법이다.
해결책: 순서번호(송신 메시지에 매회 1씩 증가하는 번호를 넣는 기법), 타임스태프(송신 메시지에 현재 시각을 넣는 기법), 비표(메시지를 수신하기에 앞서 수신자는 송신자에게 일회용의 랜덤한 값을 건네 주는 기법)

2. 키의 추측에 의한 공격: 메시지 인증 코드에 대해서도 전사 공격과 생일 공격이 가능하다. 공격자에 의해 송수신에서 사용된 키를 추측 당해서는 안된다. 메시지 인증 코드에서 사용하는 키를 생성할 때에는 암호학적으로 안전하고 강한 의사 난수 생성기를 사용해야 한다.

메시지 인증 코드로 해결할 수 없는 문제

1. 제 3자에 대한 증명
앨리스로부터 메시지를 받은 밥이 '이 메시지는 앨리스가 보낸 것이다' 라는 것을 제 3자인 검증자 빅터에게 증명하고 싶다고 하자. 그러나 메시지 인증 코드로는 그 증명을 할 수 없다.

2. 부인 방지
밥이 MAC 값이 딸린 메시지를 받았다고 하자 이 MAC 값은 앨리스와 밥이 공유하고 있는 키를 사용해서 계산한 것이다. 밥은 '이 메시지는 앨리스로부터 온 것이다' 라고 알 수 있다. 그러나 위에서 말한 것처럼 그것을 검증자 빅터에게 증명할 수 없다.

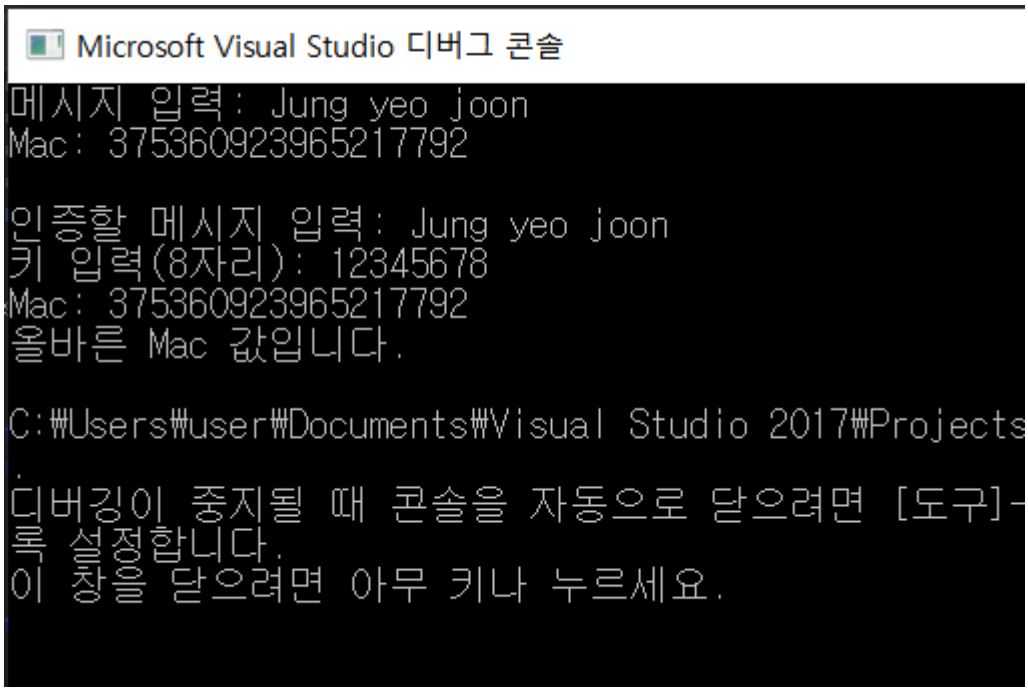
<소스코드>

```
#include<iostream>
#include<string>
#include<cmath>
#include<vector>
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 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]; //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;
}

string HMac(string input,char key[])
{
	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
}


int main()
{
	string input = "";
	string input2 = "";
	string output = "";
	string output2 = "";
	char key[8] = { '1','2','3','4','5','6','7','8' }; //사전에 공유한 공유키
	char key2[8];
	cout << "메시지 입력: ";
	getline(cin, input);
	output = HMac(input, key);
	cout << "Mac: " << output << endl;
	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);
	cout << "Mac: " << output2 << endl;

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

}
```

일단 사전에 공유한 키를 12345678이라고 설정해 놓았다. 여기서 메시지를 입력하면 키와 메시지를 이용해서 MAC값을 구한다. 구하는 방법은 다음과 같다. 일단 기존에 설정해 둔 ipad와 opad 문자열이 있다. 사전에 공유한 key를 ipad와 XOR연산한 후에 메시지와 결합한다. 그리고 결합한 메시지를 일 방향 해시함수를 이용해서 해시 값을 구한다. 그리고 공유 키와 opad를 XOR연산해 주고 앞에서 구한 해시 값과 결합해준다. 그리고 다시 일 방향 해시함수를 사용해서 해시 값을 구한다. 이 값이 MAC 값이 된다. 이렇게 해서 입력한 메시지와 키가 공유키와 동일하다면 동일한 MAC 값이 나온다.



하지만 이렇게 될 경우 재전송 공격에 취약해지게 된다. 재전송 공격이란 도청 보존해 둔 메시지와 MAC 값을 다시 송신해서 공격하는 방법이다. 즉, 송금의뢰 메시지와 MAC 값을 도청해서 저장하고 다시 똑같이 보냄으로써 도청자가 송금의뢰를 해서 또 송금을 하도록 하는 방법이다. 이를 해결하기 위해서 나는 메시지를 입력하고 키를 입력했을 때 인증을 한번 한 메시지는 count를 하나 늘려서 전혀 다른 MAC 값이 출력되도록 하는 방법이다.

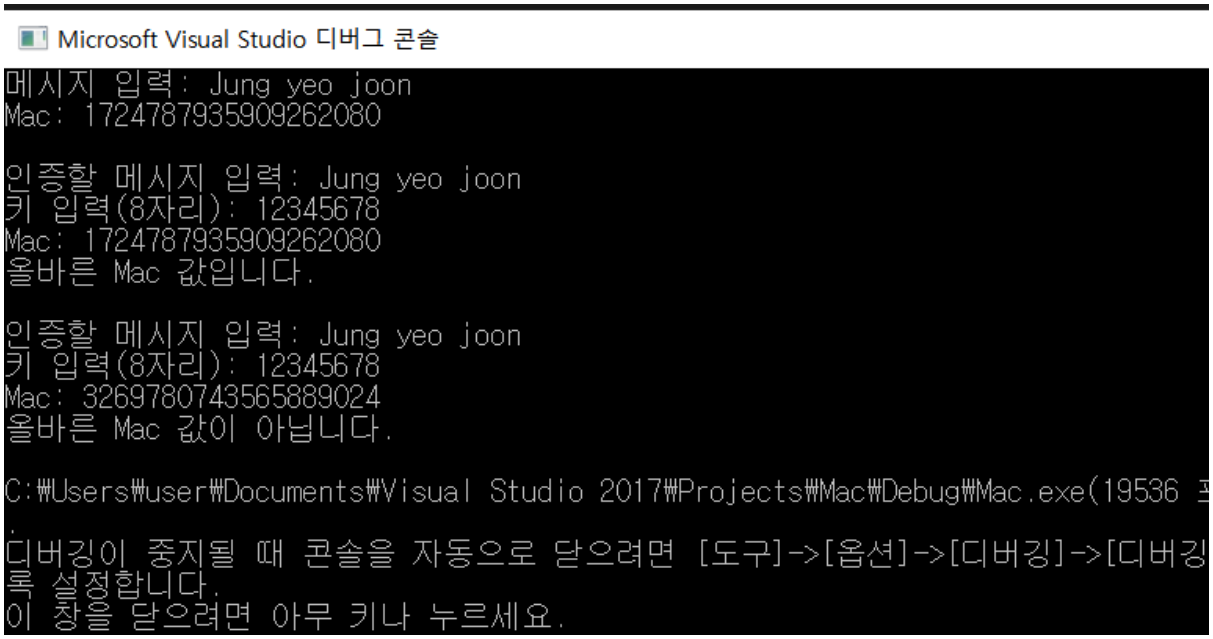
<소스코드>

```
#include<iostream>
#include<string>
#include<cmath>
#include<vector>
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 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]; //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;
}

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 main()
{
        string input = "";
        string input2 = "";
        string input3 = "";
        string output = "";
        string output2 = "";
        string output3 = "";
        int count = 0;
        char key[8] = { '1','2','3','4','5','6','7','8' }; //사전에 공유한 공유키
        char key2[8];
        char key3[8];
        cout << "메시지 입력: ";
```

```
	getline(cin, input);
	output = HMac(input, key);
	cout << "Mac: " << output << endl;

	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;

}
```



위의 실행화면과 같이 처음 인증할 경우 입력한 메시지가 서로 같고 입력한 키가 공유키와 같을 경우 같은 MAC 값이 나온다. 하지만 두번째 인증할 때부터는 count가 되기 때문에 입력한 메시지와 입력한 키가 동일하더라도 아예 다른 MAC 값이 출력되는 것을 확인할 수 있다. 이를 통해서 재전송 공격을 막을 수 있다.

<소감>

이번에 메시지 인증 코드를 직접 구현해보면서 지난번에 제가 작성했던 일 방향 해시함수를 사용하게 되었습니다. 다시 한번 제가 작성한 알고리즘을 사용하면서 보완해야 할 부분들을 찾게 되었고 이를 통해 조금 더 나아진 일 방향 해시함수를 구현할 수 있게 되었습니다. 또한 직접 메시지 인증 코드가 어떤 것인지 수업시간에 배운 것과 비교하면서 검색하고 공부하면서 조금 더 메시지 인증 코드에 대한 지식을 쌓을 수 있어 좋았습니다. 또한 메시지 인증 코드가 취약한 중간자 공격에 직접 대응하는 코드를 구현하면서 메시지 인증 코드에 대한 이해도가 한층 더 높아졌고 저의 코딩 실력도 조금 더 성장한 것 같아서 기분이 좋았습니다.