컴퓨터보안 보고서

실습과제 #9

강좌 명: 컴퓨터보안

교수님: 정준호 교수님

학과: 컴퓨터공학과

학년: 3학년

학번: 2017112138

이름: 정여준

일방향 해시 함수

일방향 해시 함수는 바로 파일의 지문을 채취하는 기술로 일방향 해시 함수가 만들어내는 해시 값이 메시지의 지문에 해당한다. 일방향 해시 함수는 입력과 출력이 각각 1개씩 있다. 입력이 메시지 출력이 해시 값이다. 여기서 특징은 해시 값의 길이는 메시지의 길이와는 상관이 없이 고정된 해시 값을 출력한다. 또한 메시지가 다르면 해시 값도 다르고 일방향성이기 때문에 암호화는 가능하지만 복호화는 불가능하다는 특징이 있다. 해시 함수에서 충돌이라는 것은 다른 메시지가 같은 해시 값을 갖는 상황을 말한다.

일방향 해시 함수 구현

<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;
}

int main()
{
	cout << fixed;
	cout.precision(30);  //소수점 30자리까지 출력하도록 한다.

	string input;
	long double sum = 0;
	cout << "평문을 입력하세요: ";
	getline(cin, input);
```

```
        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;
        long long output = floor(sum); //소수점 4자리부터 23자리까지 값을 결과 값에 저장

        cout << output;
}
```

설명: 평문으로 입력 받은 문자열의 길이만큼 반복문을 활용해서 각 문자열의 문자를 모두 아스키코드 10진수로 변환해준다. 그 이후 거기에서 나온 값들을 모두 더해준다. 모두 더한 값을 더한 값보다 큰 소수 중 가장 작은 수로 나눠준다. 소수점 4자리부터 23자리까지 값을 결과 값에 저장한다. 이렇게 되면 어떤 문자를 입력한다 해도 변환된 값과 평문 사이에서 어떠한 수학적인 관계도 찾기 힘들다. 그리고 어떤 문자를 입력한다 해도 똑같은 길이의 출력 값이 나오게 된다.

하지만 여기서 문제가 존재하는데 문제는 만약 더한 값이 같을 경우 충돌이 발생한다. 이런 경우를 해결할 방법을 생각해본 결과 평문을 입력하면서 평문에 대한 값이 나왔을 때 그 값을 배열에 저장한다. 그리고 이전에 똑같은 값이 나온 경우를 확인하고 있을 경우에 값에 1을 더해주거나 값을 변화해줘서 충돌을 없애는 방법이 있을 것으로 예상된다. 그래서 한번 구현해봤다.

<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;
}

int main()
{
	cout << fixed;
	cout.precision(30);  //소수점 30자리까지 출력하도록 한다.

	vector<long double> v; //sum값을 넣을 벡터 생성
	string input;
	long double sum = 0;
	long long output = 0;
	do{
		input = "";
		sum = 0;
		output = 0;
		cout << "평문을 입력하세요(아무것도 입력 안하면 종료): ";
		getline(cin, input);
		for (int i = 0; i < input.length(); i++)
		{
			long double a = input[i]; //아스키코드로 변환
			sum += a; //아스키코드로 변환한 값들을 모두 더해준다.
		}
		for (int i = 0; i < v.size(); i++) //벡터에 같은 값이 있는 경우 sum++를
진행해서 같은 값이 나오는 것을 없앤다.
		{
```

```
                        if (v[i] == sum)
                        {
                                sum++;
                        }
                }
                v.push_back(sum);
                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;
                output = floor(sum); //소수점 4자리부터 23자리까지 값을 결과 값에 저장
                cout << output << endl;
        } while (input != ""); //아무것도 입력 안된 경우 반복문 종료
}
```

이렇게 반복문을 통해서 입력 받은 평문에 대해서 sum 값을 벡터에 계속 저장해 가면서 같은 값이 나올 경우 값을 1씩 증가하면서 기존에 나왔던 해시 값이 나오지 않도록 즉 충돌이 일어나는 것을 줄였다. 하지만 구하기는 힘들지만 다른 값을 다른 값으로 나누었을 때 소수점 4자리부터 23자리까지 모두 같은 값이 나올 가능성도 배제할 수 없다. 하지만 이런 경우는 극히 일부라고 생각하고 찾기 매우 어려워 직접 구현한 일방향 해시 함수는 강한 충돌 내성을 가지고 있다고 생각한다. 그런 경우가 있을 수도 있는데 보통 나누었을 때 소수점 4자리까지 가지 않고 나누어 떨어지는 경우이다. 예를 들어 합이 1인 경우 다음 소수는 2이므로 1을 2로 나누면 0.5로 나누어 떨어져 출력 값이 0이 된다. 이러한 경우가 1을 제외하고도 나오기 때문에 이런 경우를 보완해야 한다고 생각한다. 일단 0이 되는 것을 제외하고 출력 값이 동일한 것이 나오는 경우 충돌 발생이라는 문구가 나올 수 있도록 구현하여 0이 나오는 것을 제외하고 충돌이 일어나는지 확인해본다.

<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
```

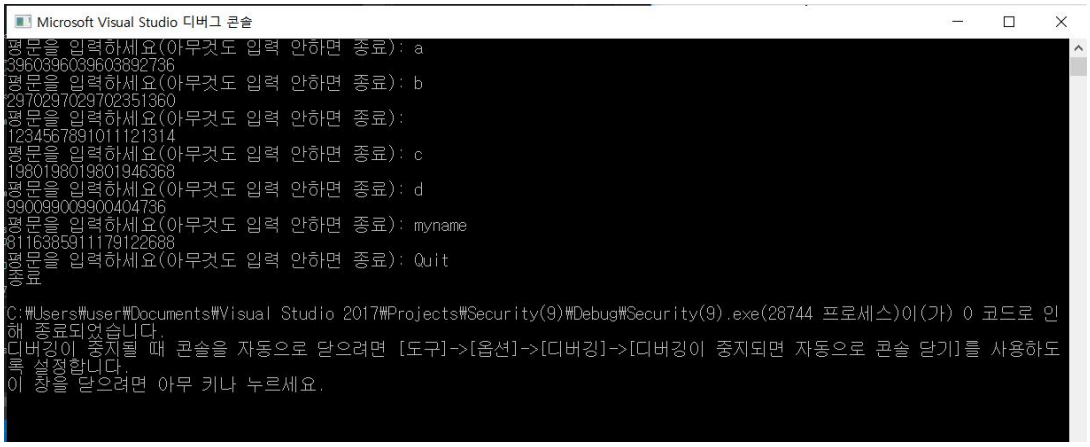
```
{
	int num = (int)sum + 1;
	while (isPrime(num) == false)
	{
		num++;
	}
	return num;
}

int main()
{


	vector<long double> v; //output을 넣을 벡터
	long double sum = 0;

	for (int i = 0; i < 10000; i++) {
		cout << fixed;
		cout.precision(30);  //소수점 30자리까지 출력하도록 한다.
		sum = i;
		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;
		for (int j = 0; j < v.size(); j++)
		{
			if (v[j] == floor(sum) && floor(sum) != 0) //0이 아닌 같은 결과 값이 나올 경우
			{
				cout << "충돌 발생" << endl;
				break;
			}
		}
		v.push_back(floor(sum));
		cout.precision(0);
		cout << floor(sum) << endl;
	}
}
```

0부터 10000까지 반복문을 수행해본 결과 출력 값이 0으로 나오는 것을 제외하고는 결과 값이 충돌하는 경우는 없는 것으로 나오는 것을 확인할 수 있다. 이제 0이 나온 것을 보완하면 된다. 즉, 나누었을 때 소수점 4자리부터 23자리가 모두 0인 경우를 보완해야한다. 이 경우에는 나누었을 때 소수점 4자리부터 23자리가 되기 힘든 수로 직접 지정해주는 것을 생각해봤다. 예를 들어, 합계가 1이고 2로 나누어 0.5가 되는 경우 위처럼 문제가 생기게 되는데 이런 경우 그냥 결과 값을 1234567891011121314 이런 식으로 지정해주는 것이다. 그리고 또 나올 경우 여기서 1을 더해준 12345678910111213315가 저장되는 식으로 진행해준다. 다음은 최종으로 충돌을 최대한 줄인 일방향 해시 함수 코드이다.

<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;
}

int main()
{
	vector<long double> v; //sum값을 넣을 벡터 생성
	string input;
	long double sum = 0;
	int count = 0;
	while (1) {
		cout << fixed;
		cout.precision(30);  //소수점 30자리까지 출력하도록 한다.
		input = "";
		sum = 0;
```

```
		cout << "평문을 입력하세요(아무것도 입력 안하면 종료): ";
		getline(cin, input);
		if (input == "Quit" )//Quit 입력 시 반복문 종료
		{
			cout << "종료" << endl;
			break;
		}

		for (int i = 0; i < input.length(); i++)
		{
			long double a = input[i]; //아스키코드로 변환
			sum += a; //아스키코드로 변환한 값들을 모두 더해준다.
		}
		for (int i = 0; i < v.size(); i++) //벡터에 같은 값이 있는 경우 sum++를
진행해서 같은 값이 나오는 것을 없앤다.
		{
			if (v[i] == sum)
			{
				sum++;
			}
		}
		v.push_back(sum);
		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
		{
			cout << 1234567891011121314 + count << endl;
			count++; //0이 나올 때마다 count 증가 시켜 충돌 방지

		}
		else
			cout << floor(sum) << endl;
	}
}
```



내가 직접 구현한 충돌을 최대한 줄인 해시함수 구현은 위와 같다.

<소감문>

일방향 해시 함수가 어떤 것인지 수업에서 배운 것을 직접 구현해보고 찾아보면서 제대로 이해할 수 있게 되었습니다. 또한 충돌이 언제 일어날지 생각해보고 그 때 어떤 방식으로 충돌을 최대한 줄일 수 있을지 생각해보고 구현해 보는 것이 재밌고 좋았습니다.