REPORT

일방향 해시 함수

과목명 :	컴퓨터보안
담당교수 :	정준호 교수님
제출일 :	2021 년 05 월 11 일
공과 대학	컴퓨터공학 과
학번 : 2016112154	이름: 정동구

1. 일방향 해시 함수 개요 및 특징.

구현된 일방향 해시 함수는 총 두 단계를 거친다. 우선 입력받은 문자열을 전처리 한 후 multiplication Method 를 활용하여 해싱을 하였다.

입력받은 문자열을 전처리 하는 과정은 MD5 와 SHA256 에서 사용하는 방식을 참고 하였다. 우선 문자열을 입력 받은 뒤 입력받은 문자열의 길이를 동일하게 하기 위해 남는 공간만큼 0 을 채워주고 마지막 64 비트의 공간은 입력받은 문자열의 길이가 들어가게 하였다. 이후 각 글자를 정수형태로 바꾼 후, 이진수로 변환하여 string 형태로 저장 한 뒤, 그대로 이어붙인다. 하나의 이진수 형태의 문자열이 된 값을 다시 정수로 변환하여 전처리를 마친다. 이어 붙일경우 숫자가 많이 커지기 때문에 unsigned long long 을 사용하였따.

전처리된 문자열은 multiplication Method 를 이용한 해시 함수를 거쳐 결과가 나온다. Multiplication Method 의 경우 $h(k)=(kA \bmod 1) \times m$ 의 식을 활용하여 해시값을 구한다. 이때 k 는 입력값, A 는 0 과 1 사이의 소수 m 은 2 의 제곱수를 사용한다. 위 식에서 mod1 을 하는 이유는 소수 부분만 사용하기 위해서 이다.

2. 충돌 내성 테스트 및 개선방안

```
x : a
A : 0.1
hash result : 1433.6
x : b
A : 0.1
hash result : 614.4
x : c
A : 0.1
hash result : 204.8
x : d
A : 0.1
hash result : 204.8
x : e
A : 0.1
hash result : 1843.2
x : f
A : 0.1
hash result : 1024
x : g
A : 0.1
hash result : 614.4
x : h
A : 0.1
hash result : 1024
x :
```

$h(k)=(kA \bmod 1) \times m$ 에서 A 를 0.1 로 하였을 때 의 결과이다. 충돌이 굉장히 빈번하게 일어남을 알 수 있다. K 의 값은 입력받은 문자열과 그 길이에 따라 달라지기 때문에

같은 경우가 거의 발생할 수 없다. 따라서 충돌이 일어나는 이유는 A 의 값에 있다고 추측 할 수 있다. 수식에서도 mod1 을 사용하는 것은 소수부분을 사용하기 위해서 이기 때문에 A 의 값, 즉 k*A 연산을 하였을 때 소수점 아래 자릿수에 따라 테이블의 크기가 결정 됨을 알 수 있다. A 의 크기가 0.1 일 경우 mod 을 취했을 때 가능한 값은 0.0-0.9 까지로 총 10 개밖에 되지 않는다. 해시 함수를 거친 이후 나올 수 있는 결과의 경우가 10 가지 밖에 되지 않으므로 당연히 충돌이 빈번하게 일어날 수 밖에 없다.

따라서 A 값을 변화시키는 방법으로 충돌이 줄어드는지 확인 해 보았다. A 값의 소수점 아래 자릿수를 늘렸을 때

```
x : b
A : 0.01
hash result : 1085.44
x : c
A : 0.01
hash result : 839.68
x : d
A : 0.01
hash result : 1658.88
x : e
A : 0.01
hash result : 1413.12
x : f
A : 0.01
hash result : 512
x : g
A : 0.01
hash result : 266.24
x : h
A : 0.01
hash result : 921.6
x : i
A : 0.01
hash result : 675.84
x : j
A : 0.01
hash result : 1822.72
x : k
A : 0.01
hash result : 1576.96
x : l
A : 0.01
hash result : 348.16
x : m
A : 0.01
hash result : 102.4
x : n
A : 0.01
hash result : 1249.28
x : o
A : 0.01
hash result : 1003.52
x : p
A : 0.01
hash result : 1576.96
```

```
x : b
A : 0.001
hash result : 1542.14
x : c
A : 0.001
hash result : 83.968
x : d
A : 0.001
hash result : 1394.69
x : e
A : 0.001
hash result : 1984.51
x : f
A : 0.001
hash result : 1689.6
x : g
A : 0.001
hash result : 231.424
x : aa
A : 0.001
hash result : 1005.57
x : bb
A : 0.001
hash result : 1873.92
x : cc
A : 0.001
hash result : 137.216
x : dd
A : 0.001
hash result : 1284.1
x : ee
A : 0.001
hash result : 1595.39
x : ff
A : 0.001
hash result : 415.744
x : h
A : 0.001
hash result : 296.96
x : i
A : 0.001
hash result : 886.784
x : j
A : 0.001
hash result : 591.872
x : k
A : 0.001
hash result : 1181.7
x : kk
A : 0.001
hash result : 432.128
x : jj
A : 0.001
hash result : 120.832
x : hh
A : 0.001
hash result : 989.184
x : zz
A : 0.001
hash result : 268.288
```

와 같은 결과가 나왔다. A 가 0.01 일 때에는 0.1 일 때와 비교하였을 때 중복이 덜 일어나지만 몇 번 시행하지 않고 중복이 나왔음이 보인다. 하지만 0.001 일 때에는 여러가지 입력을 해보아도 충돌이 나는 경우가 거의 없었다. A 의 값이 충분히 작아지면 충돌 횟수가 0 에 가까워져 사실상 충돌이 없는 것 과 마찬가지일 것으로 생각 된다.

충돌 내성 완화 기법으로 체이닝,선형조사, 이중해싱 등의 여러 방법이 있지만, 애초에 중복이 일어나지 않게 만드는 것 또한 충돌 내성 완화 기법의 하나라고 생각한다.SHA 256 도 2^512 만큼의 경우의 수로 해시값이 결과로 출력되기 때문에 사실상 충돌이 일절 일어나지 않는 것으로 생각하기 때문에 가장 원초적이고 간단한 충돌을 줄이는 방법은 해시 결과 값의 경우의 수를 충분히 늘리는 것이라 생각한다.

3. 코드

```
unsigned long long preProcessing(std::vector<int>& input)//입력 값 전처리
{
    std::string output = "";
    input.push_back(0b10000000);//입력 값 뒤에 1 1개 추가
    unsigned long long binary=0;
    int L = input.size();

    int k = 64 - (((L % 64) + 9) % 64);//크기를 맞추기 위해
    if (k == 64) k = 0;

    for (int i = 0; i < k; i++)//부족한 만큼 뒤에 0추가
    {
        input.push_back(0);
    }

    uint64_t bitLengthInBigEndian = changeEndian(L * 8);
    auto ptr = reinterpret_cast<uint8_t*>(&bitLengthInBigEndian);

    input.insert(std::end(input), ptr, ptr + 8);//마지막 8비트에 문자열 크기 입력

    for (int i = 0; i < input.size(); i++)//이진수 형태의 문자열로 변환하여 합친다.
    {
        output += toBinary(input[i]);
    }

    for (int i = 0; i < output.length(); i++)
    {
        binary = (binary << 1) + output[i] - '0';//합쳐진 이진수 정수로 변환
    }

    return binary;
}
```

문자열 전처리 함수. 입력 값을 일정한 크기로 변경한다. 마지막 8 비트에는 입력 문자열의 길이를 넣는다. 각 자리의 값을 이진수형태의 문자열로 변환한 뒤 이어붙여 하나의 이진수를 만들고 다시 정수로 변환한다.

```
double hashing(double A,int m,unsigned long long x)//multiplication Method
{
    double hash = m * ((x*A)-(long long)(x*A));//m 2의 제곱수, A 0과1사이의 소수 , x 입력값.

    return hash;
}
```

Multiplication Method 를 활용한 해시 함수. Mod1 역할을 (x*A)-(long long)(x*A) 이 수행한다. Mod1 이 소수부만 남기기 위함으로 double 형태에서 정수 형태를 빼는것으로 연산한다.

```cpp
int main()
{
    while(1)
    {std::string x; double A=0.0000001;
    std::cout << "x : ";
    std::cin >> x;
    std::cout << "A : "<<A<<std::endl;

    std::vector<int> initial;

    for (int i = 0; i < x.length(); i++)
    {
        initial.push_back(((int)x[i]));//문자 ascii코드의 코드로 숫자로 변환
    }

    unsigned long long x_pre=0;
    x_pre=preProcessing(initial);

    double result;

    result = hashing(A, 2048, x_pre);//해싱

    std::cout << "hash result : " << result << std::endl;
    }


    return 0;
}
```

main 함수. 값을 입력받고 이를 전처리 한 후 해싱하여 결과를 출력한다.

전체 코드 : [security/main.cpp at master · dsaf2007/security (github.com)]

4. 소감

해시 함수 자체를 이해하는 것이 좀 어려웠다. 처음에는 SHA256 을 참고하여 비슷하게 구현을 해보려 하였는데, 전처리 과정도 어려웠고 이후 과정도 너무 어려웠다. 결국 곱셈 방식을 이용하여 구현을 하였는데, 충돌 테스트를 해보고 개선방안을 떠올리는 것이 그리 쉽지 않았다. 딱 떠오르는 방식이 크기를 늘리는 방안 밖에 없어서 해당 방식을 택했다. 실험은 한글자나 두글자 짜리 문자열로 하였지만 A 값이 충분히 작아진다면 그보다 훨씬 긴 문자열을 입력하여도 충돌이 일어나는 경우가 매우 적을 것으로 보였다. 개념 자체가 어려워서인지 여태 했던 과제중에 가장 힘들었다. 또한 근래에 여러 안 좋은 일들이 있어서 과제에 온전히 집중하기가 어려워 더 힘들었을 수 있다는 생각도 든다.