Лабораторная работа № 3 по курсу криптографии

Выполнил студент группы М8О-308Б *Куликов Алексей.*

Условие

1. Строку в которой записано своё ФИО подать на вход в хеш-функцию ГОСТ Р 34.11-2012 (Стрибог). Младшие 4 бита выхода интерпретировать как число, которое в дальнейшем будет номером варианта. Процесс выбора варианта требуется отразить в отчёте.

2. Программно реализовать один из алгоритмов функции хеширования в соответствии с номером варианта. Алгоритм содержит в себе несколько раундов.

3. Модифицировать оригинальный алгоритм таким образом, чтобы количество раундов было настраиваемым параметром программы. в этом случае новый алгоритм не будет являться стандартом, но будет интересен для исследования.

4. Применить подходы дифференциального криптоанализа к полученным алгоритмам с разным числом раундов.

5. Построить график зависимости количества раундов и возможности различения отдельных бит при количестве раундов 1,2,3,4,5,... .

6. Сделать выводы.

Выбор варианта

```
$ python2
Python 2.7.17 (default, Nov  7 2019, 10:07:09)
[GCC 9.2.1 20191008] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from pygost import gost34112012256
>>> gost34112012256.new("Куликов Алексей Владимирович").digest()
'\xc7\\\xdc\xb6[6G\xf1\xdf\xe1\xad\x89\xa05[\x07\xe9b\te\xcf8\xe8@\xda\xcb
\xe7\x1f\xfd\xf5\xcf\xd3'
```

Таким образом, заданием является реализация и анализ алгооритма BLAKE. BLAKE – это целове семейство алгоритмов, отличающееся в основном разрядностью. Мною был выбран алгоритм BLAKE-256.

Метод решения

Реализация любого подобного алгоритма, скорее всего, начинается с изучения спецификации. Мне удалось найти оффициальный сопровождающий документ от авторов алгоритма для конкурса в NIST. В нем даны подробное описание алгоритма, его свойства и несколько примеров работы.

Сомневаюсь, что стоит приводить здесь подробное описание алгоритма т.к. он отлично описан в вышеуказанном документе, и я вряд ли смогу что-то добавть. Описание так же переведено на русский язык и находится на Википедии. Приложу лишь ссылки:

- SHA-3 proposal BLAKE
- BLAKE (хеш-функция)

Воспользовавшись этими материалами, не без приключений, но все же удалось реализовать алгоритм, который на предоставленных примерах выдает те же результаты, что и эталонный. Поэтому, вероятно, в этом он отвечает стандарту.

Анализ

Для исследования алгоритма были применены базовые знания дифференциального криптоанализа.

Берется некая строка, в ней по очереди меняется первый бит каждого байта, и вычисляются хеши от исходной и измененной строки. Далее подсчитывается количество несовпавших в них бит. Полученный результат усредняется по количеству байт в строке, поэтому для достаточно длинной строки можно получить довольно точные данные. Такая операция проводится для числа раундов в алгоритме от 1 до 20, и строится зависимость числа измененных бит от кол-ва раундов. Получены следующие данные:

Кол-во раундов	**Кол-во измененный бит**
1	128.326
5	128.093
10	127.659
15	127.016
20	128.163

Так же были подсчитаны приблезительные вероятности изменения бита в выходной последовательности. Для числа раундов они имет следующие значения:

Кол-во раундов	**Вероятность**
1	0.500617
5	0.499092
10	0.499235
15	0.498338
20	0.500961

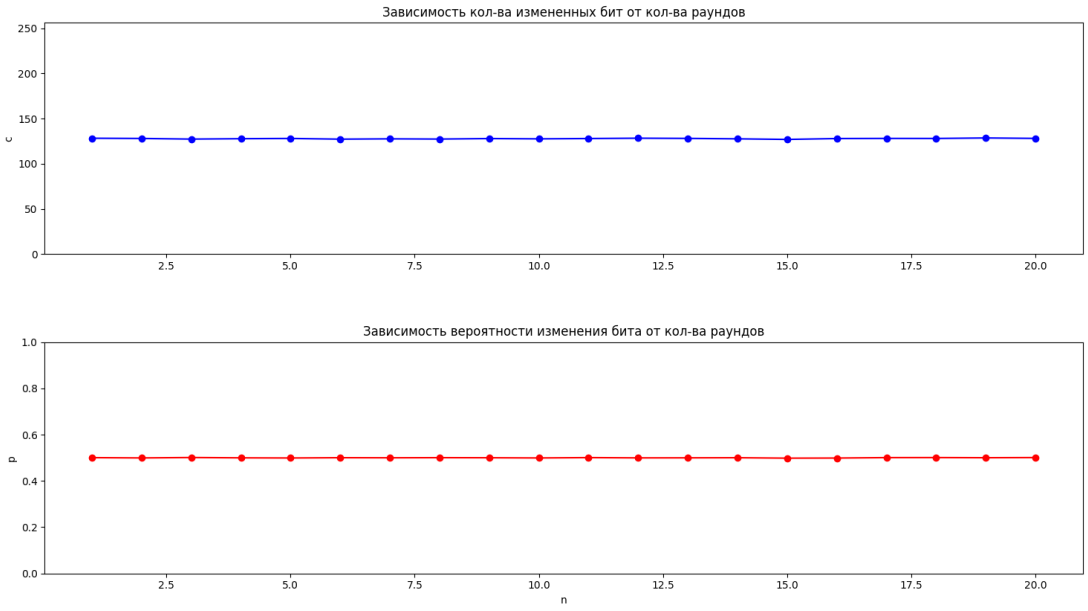
Из графика видно, что уже на первом раунде число измененных бит находится в районе половины (128 бит) от числа бит хеша на выходе функции (256). Это говорит о выполнении лавинного критерия.

Более того из второй зависимости видно, что вероятность изменения бита на выходе хеш-функции при изменении одного бита на входе приблизительно равна 0.5. Это говорит о выполнении строгого лавинного критерия.

Отсюда можно сделать выводы:

1. При использовании данного метода криптоаналитик не может сделать никаких предположений о виде входной информации, основываясь на выходной информации.

2. Т.к. лавинный эффект, судя по всему, достигается уже при числе раундов равном одному, значит, используемые в стандарте 14 раундов нужны для какой-то другой цели (?).



Выводы

Алгоритмы хеширования, подобные рассмотренному, находят массу применений в современном мире (например те же проверка или восстановление пароля в веб-сервисах, проверка целостности файлов с помощью контрольной суммы и т.д и т.п.) и было бы неплохо знать хотя бы основы основ в этой области. Именно поэтому в ходе выполнения данной работы я собственноручно реализовал один из криптографических алгоритмов хеширования и на его примере познакомился с базовыми приемами дифференциального криптоанализа.

Листинг кода

`blake.h`

```
#ifndef BLAKE_H
#define BLAKE_H

#include <iostream>
#include <fstream>
#include <algorithm>

#include <limits.h>
#include <stdint.h>

#include <bitset>

#define UINT_BITS (sizeof(unsigned int) * 8)
#define UINT_BITS_M1 (UINT_BITS - 1)

unsigned int iv[8] = {
    0x6A09E667, 0xBB67AE85,
    0x3C6EF372, 0xA54FF53A,
    0x510E527F, 0x9B05688C,
    0x1F83D9AB, 0x5BE0CD19};

unsigned int c[16] = {
    0x243F6A88, 0x85A308D3,
    0x13198A2E, 0x03707344,
    0xA4093822, 0x299F31D0,
    0x082EFA98, 0xEC4E6C89,
    0x452821E6, 0x38D01377,
    0xBE5466CF, 0x34E90C6C,
    0xC0AC29B7, 0xC97C50DD,
    0x3F84D5B5, 0xB5470917};

unsigned char sgm[160] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
    14, 10, 4, 8, 9, 15, 13, 6, 1, 12, 0, 2, 11, 7, 5, 3,
    11, 8, 12, 0, 5, 2, 15, 13, 10, 14, 3, 6, 7, 1, 9, 4,
    7, 9, 3, 1, 13, 12, 11, 14, 2, 6, 5, 10, 4, 0, 15, 8,
    9, 0, 5, 7, 2, 4, 10, 15, 14, 1, 11, 12, 6, 8, 3, 13,
    2, 12, 6, 10, 0, 11, 8, 3, 4, 13, 7, 5, 15, 14, 1, 9,
    12, 5, 1, 15, 14, 13, 4, 10, 0, 7, 6, 3, 9, 2, 8, 11,
    13, 11, 7, 14, 12, 1, 3, 9, 5, 0, 15, 4, 8, 6, 2, 10,
    6, 15, 14, 9, 11, 3, 0, 8, 12, 2, 13, 7, 1, 4, 10, 5,
    10, 2, 8, 4, 7, 6, 1, 5, 15, 11, 9, 14, 3, 12, 13, 0};

unsigned char sigma(unsigned int r, unsigned int i) {
    return sgm[r * 16 + i];
}
```

```
unsigned int cons(unsigned int i) {
    return c[i];
}

#define P2_32 4294967296
#define BYTES_TO_READ (sizeof(unsigned int) * 16)

unsigned int summod2_32(unsigned int a, unsigned int b) {
    return ((unsigned long int)a + b) % P2_32;
}

uint32_t rotleft(uint32_t value, unsigned int count) {
    const unsigned int mask = CHAR_BIT * sizeof(value) - 1;
    count &= mask;
    return (value << count) | (value >> (-count & mask));
}

uint32_t rotright(uint32_t value, unsigned int count) {
    const unsigned int mask = CHAR_BIT * sizeof(value) - 1;
    count &= mask;
    return (value >> count) | (value << (-count & mask));
}

void G(unsigned int *m, unsigned int &a, unsigned int &b, unsigned int &c,
    unsigned int &d, unsigned int r, unsigned int i) {
    unsigned char s = sigma(r % 10, 2 * i), s1 = sigma(r % 10, 2 * i + 1);
    a = summod2_32(a, summod2_32(b, m[s] ^ cons(s1)));
    d = rotright(d ^ a, 16);
    c = summod2_32(c, d);
    b = rotright(b ^ c, 12);
    a = summod2_32(a, summod2_32(b, m[s1] ^ cons(s)));
    d = rotright(d ^ a, 8);
    c = summod2_32(c, d);
    b = rotright(b ^ c, 7);
}

void round(unsigned int *m, unsigned int *v, unsigned int r) {
    G(m, v[0], v[4], v[8], v[12], r, 0);
    G(m, v[1], v[5], v[9], v[13], r, 1);
    G(m, v[2], v[6], v[10], v[14], r, 2);
    G(m, v[3], v[7], v[11], v[15], r, 3);
    G(m, v[0], v[5], v[10], v[15], r, 4);
    G(m, v[1], v[6], v[11], v[12], r, 5);
    G(m, v[2], v[7], v[8], v[13], r, 6);
    G(m, v[3], v[4], v[9], v[14], r, 7);
}

void compress(unsigned int *h, unsigned int *m, unsigned int *s, unsigned int *t,
    unsigned int *res, unsigned int round_count=14) {
    unsigned int v[16];

    for (int i = 0; i < 8; ++i)
        v[i] = h[i];
    for (int i = 0; i < 4; ++i) {
        v[i + 8] = s[i] ^ c[i];
    }
    v[12] = t[0] ^ c[4];
    v[13] = t[0] ^ c[5];
    v[14] = t[1] ^ c[6];
    v[15] = t[1] ^ c[7];

    for (unsigned int i = 0; i < round_count; ++i) {
        round(m, v, i);
    }
```

```
    for (int i = 0; i < 8; ++i) {
        res[i] = h[i] ^ s[i % 4] ^ v[i] ^ v[i + 8];
    }
}

union block_t{
    unsigned char c[64];
    unsigned int i[16];
    unsigned long int li[8];
};

void blake_hash(std::ifstream &in, unsigned int *res, unsigned int round_count=14){
    unsigned int h0[8], h1[8];

    for (int i = 0; i < 8; ++i)
        h0[i] = iv[i];

    unsigned int s[4] = {0, 0, 0, 0};
    unsigned long int t = 0;
    block_t m;

    unsigned int k = 0;
    while (true) {
        if (!in)
            break;

        in.read((char *)m.i, BYTES_TO_READ);
        unsigned int count = in.gcount();

        t += count * 8;
        if (count != BYTES_TO_READ) {
            m.c[count] = (1 << 7);
            unsigned int i = count + 1;
            for (; i < BYTES_TO_READ - 9; ++i) {
                m.c[i] = 0;
            }
            m.c[i] = 1;

            for (int j = 0; j < 14; ++j) {
                unsigned char *b = (unsigned char *)&m.i[j];
                std::swap(b[0], b[3]);
                std::swap(b[1], b[2]);
            }

            m.li[7] = t;
            std::swap(m.i[14], m.i[15]);
        }
        else{
            for (int j = 0; j < 16; ++j) {
                unsigned char *b = (unsigned char *)&m.i[j];
                std::swap(b[0], b[3]);
                std::swap(b[1], b[2]);
            }
        }

        if (k++ % 2 == 0) {
            compress(h0, m.i, s, (unsigned int *)&t, h1, round_count);
        } else {
            compress(h1, m.i, s, (unsigned int *)&t, h0, round_count);
        }
    }

    if (k % 2) {
        for (int i = 0; i < 8; ++i) {
```

```
            res[i] = h1[i];
        }
    } else {
        for (int i = 0; i < 8; ++i) {
            res[i] = h0[i];
        }
    }
}

void blake_hash(const char *data, unsigned int n,
    unsigned int *res, unsigned int round_count=14) {
    unsigned int h0[8], h1[8];

    for (int i = 0; i < 8; ++i)
        h0[i] = iv[i];

    unsigned int s[4] = {0, 0, 0, 0};
    unsigned long int t = 0;
    block_t m;

    unsigned int k = 0;
    while (k * BYTES_TO_READ < n) {
        unsigned int count = 0;

        for(; count < BYTES_TO_READ; ++count){
            unsigned int j = k * BYTES_TO_READ + count;
            if(j >= n)
                break;
            m.c[count] = data[j];
        }

        t += count * 8;
        if (count != BYTES_TO_READ) {
            m.c[count] = (1 << 7);
            unsigned int i = count + 1;
            for (; i < BYTES_TO_READ - 9; ++i) {
                m.c[i] = 0;
            }
            m.c[i] = 1;

            for (int j = 0; j < 14; ++j) {
                unsigned char *b = (unsigned char *)&m.i[j];
                std::swap(b[0], b[3]);
                std::swap(b[1], b[2]);
            }

            ++i;

            m.li[7] = t;
            std::swap(m.i[14], m.i[15]);
        }
        else{
            for (int j = 0; j < 16; ++j) {
                unsigned char *b = (unsigned char *)&m.i[j];
                std::swap(b[0], b[3]);
                std::swap(b[1], b[2]);
            }
        }

        if (k++ % 2 == 0) {
            compress(h0, m.i, s, (unsigned int *)&t, h1, round_count);
        } else {
            compress(h1, m.i, s, (unsigned int *)&t, h0, round_count);
        }
    }
```

```
    if (k % 2) {
        for (int i = 0; i < 8; ++i) {
            res[i] = h1[i];
        }
    } else {
        for (int i = 0; i < 8; ++i) {
            res[i] = h0[i];
        }
    }
}

#endif
```

main.cpp

```
#include <iostream>
#include <fstream>
#include <cstring>

#include "blake.h"

using namespace std;

#define MAX_ROUND 20

unsigned int compare(unsigned int a, unsigned int b){
    unsigned int sum = 0;
    unsigned int t = a ^ b;
    for(int i = 0; i < 32; ++i)
        sum += (t >> i) & 1;
    return sum;
}

double strong_avalanche_effect(const string &s, int round_count){
    int n = s.length();
    unsigned int hash[8];
    unsigned int hash1[8];

    vector<double> v(32 * 8, 0);

    for(int i = 0; i < n * 8; ++i){
        string s1 = s;
        s1[i / 8] ^= (1 << (i % 8));

        blake_hash(s.c_str(), n, hash, round_count);
        blake_hash(s1.c_str(), n, hash1, round_count);

        for(int j = 0; j < 8; ++j){
            unsigned int t = hash[j] ^ hash1[j];
            for(int k = 0; k < 32; ++k){
                if((t >> k) & 1){
                    v[j*32 + k]++;
                }
            }
        }
    }
    double sum = 0.0;
    for(int i = 0; i < 32*8; ++i){
        v[i] /= n * 8;
        sum += v[i];
    }
    return sum / (32 * 8);
}
```

```
void bits_changed(const string &s){
    int n = s.length();

    unsigned int hash[8];
    unsigned int hash1[8];

    vector<double> c(MAX_ROUND + 1, 0);

    for(int k = 0; k < n; ++k){
        string s1 = s;
        s1[k] ^= 1;

        for(int i = 1; i <= MAX_ROUND; ++i){
            blake_hash(s.c_str(), n, hash, i);
            blake_hash(s1.c_str(), n, hash1, i);

            int count = 0;

            for(int j = 0; j < 8; ++j){
                count += compare(hash[j], hash1[j]);
            }
            c[i] += count;
        }
    }

    cout << "i = [";
    for(int i = 1; i < MAX_ROUND + 1; ++i){
        if(i > 1)
            cout << ", ";
        cout << i;
    }
    cout << "]" << endl;

    cout << "c = [";
    for(int i = 1; i < MAX_ROUND + 1; ++i){
        if(i > 1)
            cout << ", ";
        cout << c[i] / n;
    }
    cout << "]" << endl;
}

int main() {
    string s = "ggfhfdghehteswhngfhfgbfhfgjgkhgcfhdgesgrhrtjhtydrthrthbrtefbervhbleigbv
    ilegirebglergvbeliagrkdjbfdljsnldslsdfnegvaebasergkelragae";

    bits_changed(s);

    vector<double> p(MAX_ROUND+1, 0);
    for(int i = 1; i <= MAX_ROUND; ++i){
        p[i] = strong_avalanche_effect(s, i);
    }

    cout << "p = [";
    for(int i = 1; i < MAX_ROUND + 1; ++i){
        if(i > 1)
            cout << ", ";
        cout << p[i];
    }
    cout << "]" << endl;
}
```

hash_file.cpp

```
#include <iostream>
```

```
#include <fstream>

#include "blake.h"

using namespace std;

int main(int argc, char **argv) {

    if(argc < 2){
        cout << "usage blake_hash <file_to_hash>" << endl;
        return 0;
    }

    ifstream in(argv[1], std::fstream::binary);
    if(!in.is_open()){
        cout << "can't open file: " << argv[1] << endl;
    }

    unsigned int hash[8];
    blake_hash(in, hash);

    cout << hex;
    for (int i = 0; i < 8; ++i) {
        cout.width(8);
        cout.fill('0');
        cout << hash[i] << ' ';
    }
    cout << endl;
}
```

hash_string.cpp

```
#include <iostream>
#include <fstream>
#include <cstring>

#include "blake.h"

using namespace std;

int main(int argc, char **argv) {

    if(argc < 2){
        cout << "usage blake_hash <string>" << endl;
        return 0;
    }

    unsigned int hash[8];
    blake_hash(argv[1], strlen(argv[1]), hash);

    cout << hex;
    for (int i = 0; i < 8; ++i) {
        cout.width(8);
        cout.fill('0');
        cout << hash[i] << ' ';
    }
    cout << endl;
}
```