Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №2**

По дисциплине «Криптографические методы защиты информации»

Тема: «Криптографические хэш-функции»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Литвинюк Т. В.

**Проверил:**

Хацкевич А. С.

Брест 2023

**Цель:** изучить существующие алгоритмы вычисления дайджестов сообщений и написать программу, реализующую заданный алгоритм хэширования.

**Ход работы:**

**Вариант** 2

Алгоритм Ripemd320

#include <iostream>

#include <vector>

#include <bitset>

using namespace std;

uint32\_t f(int j, uint32\_t x, uint32\_t y, uint32\_t z) {

    if (j <= 15)

        return x ^ y ^ z;

    if (j <= 31)

        return (x & y) | (~x & z);

    if (j <= 47)

        return (x | ~y) ^ z;

    if (j <= 63)

        return (x & z) | (y & ~z);

    return x ^ (y | ~z);

}

uint32\_t K1(int j) {

    if (j <= 15)

        return 0x00000000;

    if (j <= 31)

        return 0x5A827999;

    if (j <= 47)

        return 0x6ED9EBA1;

    if (j <= 63)

        return 0x8F1BBCDC;

    return 0xA953FD4E;

}

uint32\_t K2(int j) {

    if (j <= 15)

        return 0x50A28BE6;

    if (j <= 31)

        return 0x5C4DD124;

    if (j <= 47)

        return 0x6D703EF3;

    if (j <= 63)

        return 0x7A6D76E9;

    return 0x00000000;

}

int R1 [] = {

    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,

    7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8,

    3, 10, 14,  4,  9, 15,  8,  1,  2,  7,  0,  6, 13, 11,  5, 12,

    1,  9, 11, 10,  0,  8, 12,  4, 13,  3,  7, 15, 14,  5,  6,  2,

    4,  0,  5,  9,  7, 12,  2, 10, 14,  1,  3,  8, 11,  6, 15, 13

}, R2 [] = {

    5, 14,  7,  0,  9,  2, 11,  4, 13,  6, 15,  8,  1, 10,  3, 12,

    6, 11,  3,  7,  0, 13,  5, 10, 14, 15,  8, 12,  4,  9,  1,  2,

    15,  5,  1,  3,  7, 14,  6,  9, 11,  8, 12,  2, 10,  0,  4, 13,

    8,  6,  4,  1,  3, 11, 15,  0,  5, 12,  2, 13,  9,  7, 10, 14,

    12, 15, 10,  4,  1,  5,  8,  7,  6,  2, 13, 14,  0,  3,  9, 11

}, S1 [] = {

    11, 14, 15, 12,  5,  8,  7,  9, 11, 13, 14, 15,  6,  7,  9,  8,

    7,  6,  8, 13, 11,  9,  7, 15,  7, 12, 15,  9, 11,  7, 13, 12,

    11, 13,  6,  7, 14,  9, 13, 15, 14,  8, 13,  6,  5, 12,  7,  5,

    11, 12, 14, 15, 14, 15,  9,  8,  9, 14,  5,  6,  8,  6,  5, 12,

    9, 15,  5, 11,  6,  8, 13, 12,  5, 12, 13, 14, 11,  8,  5,  6

}, S2 [] = {

    8,  9,  9, 11, 13, 15, 15,  5,  7,  7,  8, 11, 14, 14, 12,  6,

    9, 13, 15,  7, 12,  8,  9, 11,  7,  7, 12,  7,  6, 15, 13, 11,

    9,  7, 15, 11,  8,  6,  6, 14, 12, 13,  5, 14, 13, 13,  7,  5,

    15,  5,  8, 11, 14, 14,  6, 14,  6,  9, 12,  9, 12,  5, 15,  8,

    8,  5, 12,  9, 12,  5, 14,  6,  8, 13,  6,  5, 15, 13, 11, 11

};

class Bitset {

public:

    const int set\_size = 2000;

    bool set[2000];

    int length = 0;

    Bitset() {

        for (int i = 0; i < set\_size; i++)

            this->set[i] = false;

    }

    Bitset(const string & str) {

        this->extend(str);

    }

    void append(bool bit) {

        this->set[length++] = bit;

    }

    void append(char chr) {

        for (int i = 7; i >= 0; i--)

            this->set[length++] = (chr >> i) & 1;

    }

    void extend(const string & str) {

        for (char chr: str)

            this->append(chr);

    }

    void add\_values(bool bit, int size) {

        for (int i = 0; i < size; i++)

            this->set[length++] = bit;

    }

    bool operator[](int i) {

        return this->set[i];

    }

    friend std::ostream& operator<<(std::ostream& stream, const Bitset& bitset) {

        for (int i = 0; i < bitset.length; i++)

            stream << (bitset.set[i] ? '1' : '0');

        return stream;

    }

};

uint32\_t \* ripemd320(const string & message) {

    uint32\_t h0 = 0x67452301,

    h1 = 0xEFCDAB89,

    h2 = 0x98BADCFE,

    h3 = 0x10325476,

    h4 = 0xC3D2E1F0,

    h5 = 0x76543210,

    h6 = 0xFEDCBA98,

    h7 = 0x89ABCDEF,

    h8 = 0x01234567,

    h9 = 0x3C2D1E0F;

    Bitset bitarray(message);

    int b = bitarray.length;

    // добавление дополнительных битов

    bitarray.append(true);

    bitarray.add\_values(false, (448 - (b+1) % 512) % 512);

    // добавление исходной длины сообщения

    for (int i = 0; i < 32; i++) {

        bool bit = (b >> i) & 1;

        bitarray.append(bit);

    }

    for (int i = 0; i < 32; i++) {

        bool bit = (b >> (i + 32)) & 1;

        bitarray.append(bit);

    }

    for(int i = 0; i < bitarray.length / 16; i++) {

        uint32\_t A1 = h0,  B1 = h1, C1 = h2, D1 = h3, E1 = h4,

        A2 = h5, B2 = h6, C2 = h7, D2 = h8, E2 = h9, T;

        for(int j = 0; j < 80; j++) {

            T = (A1 + f(j, B1, C1, D1) + bitarray[i \* 16 + R1[j]] + K1(j)) << S1[j] + E1;

            A1 = E1;   E1 = D1;   D1 = C1 << 10;   C1 = B1;   B1 = T;

            T = (A2 + f(79-j, B2, C2, D2) + bitarray[i \* 16 + R2[j]] + K2(j)) << S2[j] + E2;

            A2 = E2;   E2 = D2;   D2 = C2 << 10;   C2 = B2;   B2 = T;

            if (j == 15)

                T = B1; B1 = B2; B2 = T;

            if (j == 31)

            T = D1; D1 = D2; D2 = T;

            if (j == 47)

                T = A1; A1 = A2; A2 = T;

            if (j == 63)

                T = C1; C1 = C2; C2 = T;

            if (j == 79)

                T = E1; E1 = E2; E2 = T;

        }

        h0 += A1;   h1 += B1;   h2 += C1;   h3 += D1;

        h4 += E1;   h5 += A2;   h6 += B2;   h7 += C2;

        h8 += D2;   h9 += E2;

    }

    uint32\_t\* result = new uint32\_t[10];

    result[0] = h0;

    result[1] = h1;

    result[2] = h2;

    result[3] = h3;

    result[4] = h4;

    result[5] = h5;

    result[6] = h6;

    result[7] = h7;

    result[8] = h8;

    result[9] = h9;

    return result;

}

int main() {

    uint32\_t \*hash = ripemd320("abc");

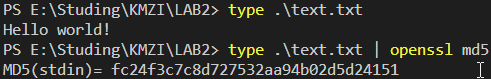
    cout << hex << hash[0] << " " << hex << hash[1] << " " << hex << hash[2] << " " << hex << hash[3] << " " << hex << hash[4] << " " << hex << hash[5] << " " << hex << hash[6] << " " << hex << hash[7] << " " << hex << hash[8] << " " << hex << hash[9];

    delete [] hash;

}



Реализация подсчета хэша средствами openssl:



**Вывод:** в ходе лабораторной работы я изучил алгоритмы хэширования, реализовал алгоритм ripemd320, реализовал подсчет хэша с помощью openssl.