Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

За пятый семестр

По дисциплине: «Криптографические методы защиты информации»

**Тема: «Криптографические хэш-функции»**

Выполнила:

Студент 3 курса

Группы ИИ-23

Макаревич Н.Р.

Проверила:

Хацкевич А. С.

Брест 2024

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

**Ход работы:**

**Вариант 11 Алгоритм RIPEMD–160.**

#include <iostream>

#include <vector>

#include <cstring>

#include <string>

#include <iomanip>

#include <fstream>

typedef unsigned char uint8;

typedef unsigned int uint32;

class RIPEMD160 {

public:

RIPEMD160() {

reset();

}

void update(const uint8\_t\* message, size\_t length) {

size\_t index = count[0] / 8 % 64;

count[0] += length \* 8;

if (count[0] < length \* 8) {

count[1]++;

}

count[1] += length >> 29;

size\_t firstPart = 64 - index;

size\_t i = 0;

if (length >= firstPart) {

memcpy(&buffer[index], message, firstPart);

processBlock(buffer);

for (i = firstPart; i + 63 < length; i += 64) {

processBlock(&message[i]);

}

index = 0;

}

memcpy(&buffer[index], &message[i], length - i);

}

void finalize(uint8\_t digest[20]) {

uint8\_t padding[64] = { 0x80 };

size\_t index = count[0] / 8 % 64;

size\_t padLen = index < 56 ? 56 - index : 120 - index;

update(padding, padLen);

uint8\_t length[8];

for (int i = 0; i < 8; i++) {

length[i] = (count[i / 4] >> ((3 - (i % 4)) \* 8)) & 0xFF;

}

update(length, 8);

for (int i = 0; i < 5; i++) {

digest[i \* 4] = (state[i] >> 24) & 0xFF;

digest[i \* 4 + 1] = (state[i] >> 16) & 0xFF;

digest[i \* 4 + 2] = (state[i] >> 8) & 0xFF;

digest[i \* 4 + 3] = state[i] & 0xFF;

}

}

private:

uint32 state[5];

uint32 count[2];

uint8\_t buffer[64];

static const int R1[80];

static const int R2[80];

static const int S1[80];

static const int S2[80];

uint32 f(uint32 j, uint32 x, uint32 y, uint32 z) {

if (j >= 0 && j <= 15)

return x ^ y ^ z;

if (j > 15 && j <= 31)

return (x & y) | (~x & z);

if (j > 31 && j <= 47)

return (x | ~y) ^ z;

if (j > 47 && j <= 63)

return (x & z) | (y & ~z);

if (j > 63 && j <= 79)

return x ^ (y | ~z);

return 0;

}

uint32\_t K1(int j) {

if (j < 16)

return 0x00000000;

else if (j < 32)

return 0x5A827999;

else if (j < 48)

return 0x6ED9EBA1;

else if (j < 64)

return 0x8F1BBCDC;

else

return 0xA953FD4E;

}

uint32\_t K2(int j) {

if (j < 16) return 0x00000000;

else if (j < 32) return 0x5C4DD124;

else if (j < 48) return 0x6D703EF3;

else if (j < 64) return 0xA6D9D649;

else return 0x00000000;

}

void reset() {

state[0] = 0x67452301;

state[1] = 0xEFCDAB89;

state[2] = 0x98BADCFE;

state[3] = 0x10325476;

state[4] = 0xC3D2E1F0;

count[0] = 0;

count[1] = 0;

memset(buffer, 0, sizeof(buffer));

}

void processBlock(const uint8\_t\* block) {

uint32 M[16];

for (int i = 0; i < 16; ++i) {

M[i] = (block[i \* 4] << 24) | (block[i \* 4 + 1] << 16) |

(block[i \* 4 + 2] << 8) | (block[i \* 4 + 3]);

}

uint32 h0 = state[0], h1 = state[1], h2 = state[2], h3 = state[3], h4 = state[4];

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

uint32 A2 = h0, B2 = h1, C2 = h2, D2 = h3, E2 = h4;

for (int j = 0; j < 80; ++j) {

uint32 T1 = A1 + f(j, B1, C1, D1) + M[R1[j]] + K1(j);

T1 = (T1 << S1[j]) | (T1 >> (32 - S1[j]));

E1 = D1;

D1 = (C1 << 10) | (C1 >> (32 - 10));

C1 = B1;

B1 = T1;

uint32 T2 = A2 + f(79 - j, B2, C2, D2) + M[R2[j]] + K2(j);

T2 = (T2 << S2[j]) | (T2 >> (32 - S2[j]));

E2 = D2;

D2 = (C2 << 10) | (C2 >> (32 - 10));

C2 = B2;

B2 = T2;

}

uint32 T = h1 + C1 + D2;

h1 = h2 + D1 + E2;

h2 = h3 + E1 + A2;

h3 = h4 + A1 + B2;

h4 = h0 + B1 + C2;

h0 = T;

state[0] += h0;

state[1] += h1;

state[2] += h2;

state[3] += h3;

state[4] += h4;

}

};

const int RIPEMD160::R1[80] = {

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

};

const int RIPEMD160::R2[80] = {

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

};

const int RIPEMD160::S1[80] = {

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

};

const int RIPEMD160::S2[80] = {

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

};

int main() {

std::string msg1 = "Hello";

RIPEMD160 ripemd160;

ripemd160.update(reinterpret\_cast<const uint8\_t\*>(msg1.c\_str()), msg1.length());

uint8\_t digest[20];

ripemd160.finalize(digest);

std::cout << "RIPEMD160(\"" << msg1 << "\") = ";

for (const auto& byte : digest) {

std::cout << std::hex << std::setw(2) << std::setfill('0') << (int)byte;

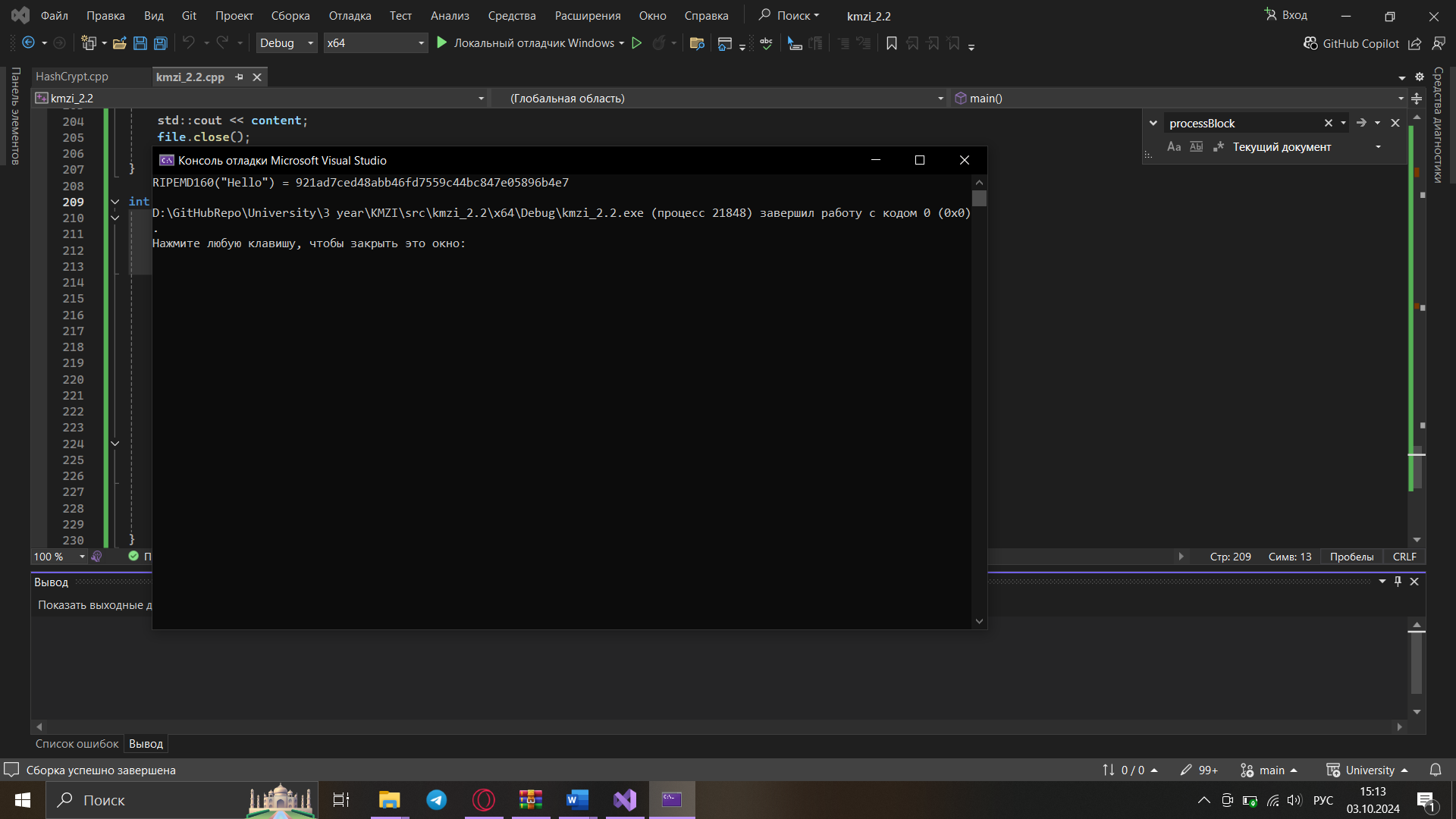
}

std::cout << std::endl;

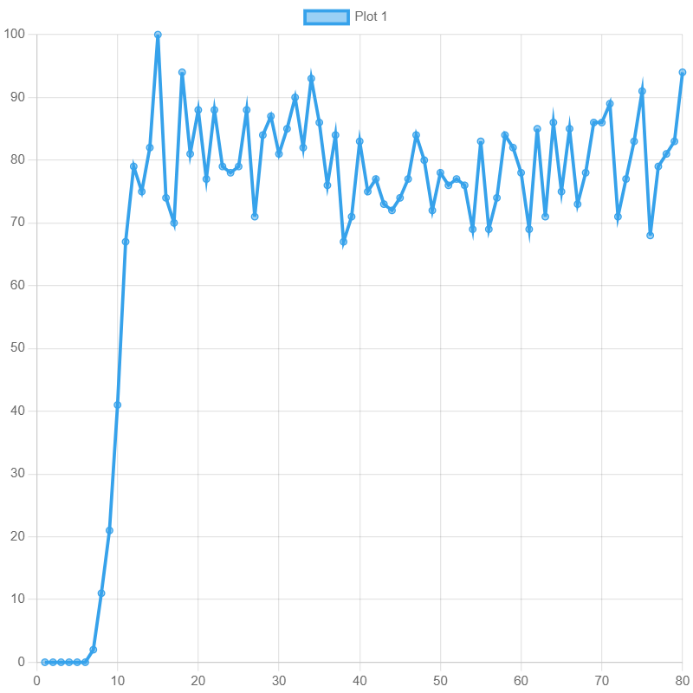
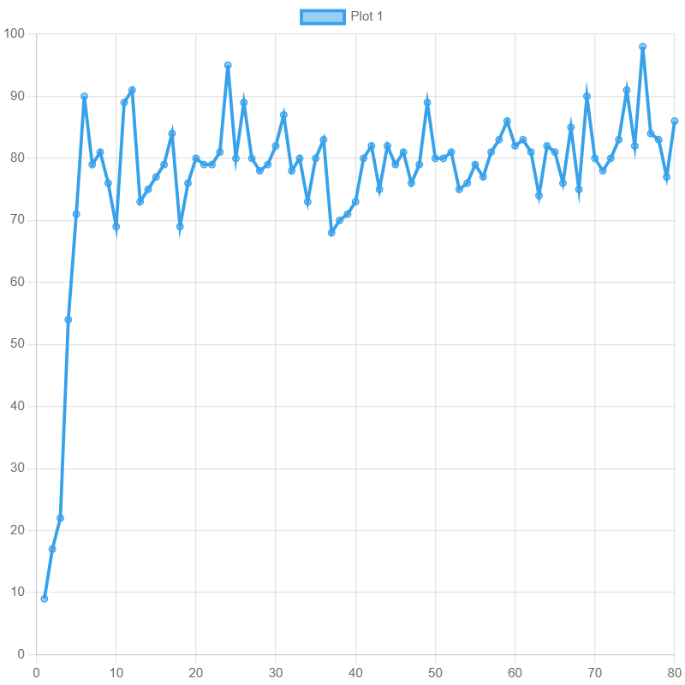
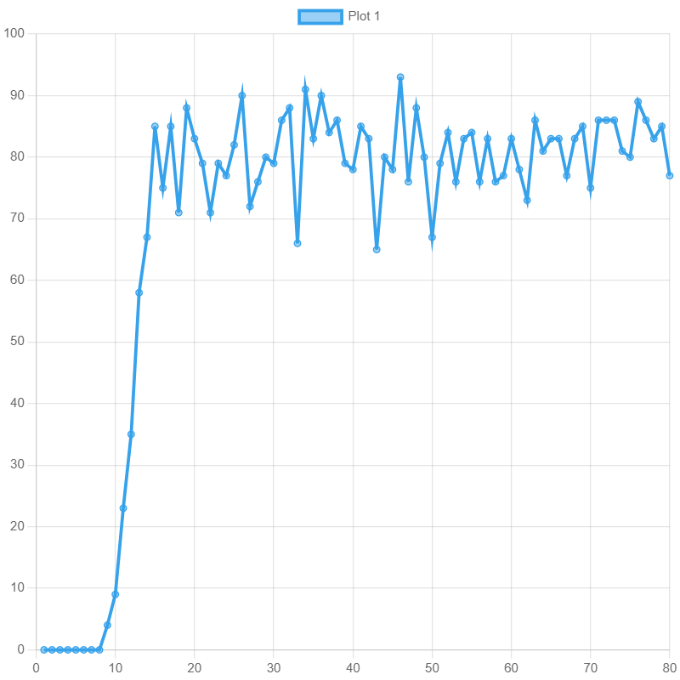
return 0;

}

Результат:



Графики лавинного эффекта:



**Вывод:** в ходе лабораторной работы изучила криптографические хэш-функции.