**МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Защита информации

Отчет по лабораторной работе № 4

Хэш-функция «Стрибог»

Выполнил студент группы М3О-409Б-20:

Мокшин И. Н.

Проверил, Ведьманов И. С.

Москва 2023 г.

Оглавление

[1. Код программы 3](#_Toc153353644)

[2. Результат работы 11](#_Toc153353645)

# Код программы

#pragma once

#include <stdint.h>

#include <string.h>

#include <malloc.h>

#define BLOCK\_SIZE 64

typedef uint8\_t uint\_block\_size[BLOCK\_SIZE];

/\*

\* значения для перестановки байт (преобразование P)

\*/

static const uint8\_t tau\_values[64] = {

0, 8, 16, 24, 32, 40, 48, 56,

1, 9, 17, 25, 33, 41, 49, 57,

2, 10, 18, 26, 34, 42, 50, 58,

3, 11, 19, 27, 35, 43, 51, 59,

4, 12, 20, 28, 36, 44, 52, 60,

5, 13, 21, 29, 37, 45, 53, 61,

6, 14, 22, 30, 38, 46, 54, 62,

7, 15, 23, 31, 39, 47, 55, 63

};

/\*

\* значения для нелинейного преобразования

\* множества двоичных векторов (преобразование S)

\*/

static const uint8\_t stri\_pi\_values[256] = {

252, 238, 221, 17, 207, 110, 49, 22,

251, 196, 250, 218, 35, 197, 4, 77,

233, 119, 240, 219, 147, 46, 153, 186,

23, 54, 241, 187, 20, 205, 95, 193,

249, 24, 101, 90, 226, 92, 239, 33,

129, 28, 60, 66, 139, 1, 142, 79,

5, 132, 2, 174, 227, 106, 143, 160,

6, 11, 237, 152, 127, 212, 211, 31,

235, 52, 44, 81, 234, 200, 72, 171,

242, 42, 104, 162, 253, 58, 206, 204,

181, 112, 14, 86, 8, 12, 118, 18,

191, 114, 19, 71, 156, 183, 93, 135,

21, 161, 150, 41, 16, 123, 154, 199,

243, 145, 120, 111, 157, 158, 178, 177,

50, 117, 25, 61, 255, 53, 138, 126,

109, 84, 198, 128, 195, 189, 13, 87,

223, 245, 36, 169, 62, 168, 67, 201,

215, 121, 214, 246, 124, 34, 185, 3,

224, 15, 236, 222, 122, 148, 176, 188,

220, 232, 40, 80, 78, 51, 10, 74,

167, 151, 96, 115, 30, 0, 98, 68,

26, 184, 56, 130, 100, 159, 38, 65,

173, 69, 70, 146, 39, 94, 85, 47,

140, 163, 165, 125, 105, 213, 149, 59,

7, 88, 179, 64, 134, 172, 29, 247,

48, 55, 107, 228, 136, 217, 231, 137,

225, 27, 131, 73, 76, 63, 248, 254,

141, 83, 170, 144, 202, 216, 133, 97,

32, 113, 103, 164, 45, 43, 9, 91,

203, 155, 37, 208, 190, 229, 108, 82,

89, 166, 116, 210, 230, 244, 180, 192,

209, 102, 175, 194, 57, 75, 99, 182

};

/\*

\* значения для линейного преобразования

\* множества двоичных векторов (преобразование L)

\*/

static const unsigned long long a\_values[64] = {

0x8e20faa72ba0b470,

0x47107ddd9b505a38,

0xad08b0e0c3282d1c,

0xd8045870ef14980e,

0x6c022c38f90a4c07,

0x3601161cf205268d,

0x1b8e0b0e798c13c8,

0x83478b07b2468764,

0xa011d380818e8f40,

0x5086e740ce47c920,

0x2843fd2067adea10,

0x14aff010bdd87508,

0x0ad97808d06cb404,

0x05e23c0468365a02,

0x8c711e02341b2d01,

0x46b60f011a83988e,

0x90dab52a387ae76f,

0x486dd4151c3dfdb9,

0x24b86a840e90f0d2,

0x125c354207487869,

0x092e94218d243cba,

0x8a174a9ec8121e5d,

0x4585254f64090fa0,

0xaccc9ca9328a8950,

0x9d4df05d5f661451,

0xc0a878a0a1330aa6,

0x60543c50de970553,

0x302a1e286fc58ca7,

0x18150f14b9ec46dd,

0x0c84890ad27623e0,

0x0642ca05693b9f70,

0x0321658cba93c138,

0x86275df09ce8aaa8,

0x439da0784e745554,

0xafc0503c273aa42a,

0xd960281e9d1d5215,

0xe230140fc0802984,

0x71180a8960409a42,

0xb60c05ca30204d21,

0x5b068c651810a89e,

0x456c34887a3805b9,

0xac361a443d1c8cd2,

0x561b0d22900e4669,

0x2b838811480723ba,

0x9bcf4486248d9f5d,

0xc3e9224312c8c1a0,

0xeffa11af0964ee50,

0xf97d86d98a327728,

0xe4fa2054a80b329c,

0x727d102a548b194e,

0x39b008152acb8227,

0x9258048415eb419d,

0x492c024284fbaec0,

0xaa16012142f35760,

0x550b8e9e21f7a530,

0xa48b474f9ef5dc18,

0x70a6a56e2440598e,

0x3853dc371220a247,

0x1ca76e95091051ad,

0x0edd37c48a08a6d8,

0x07e095624504536c,

0x8d70c431ac02a736,

0xc83862965601dd1b,

0x641c314b2b8ee083

};

/\*

\* итерационные константы

\*/

static const uint8\_t c\_values[12][64] = {

{

0x07, 0x45, 0xa6, 0xf2, 0x59, 0x65, 0x80, 0xdd,

0x23, 0x4d, 0x74, 0xcc, 0x36, 0x74, 0x76, 0x05,

0x15, 0xd3, 0x60, 0xa4, 0x08, 0x2a, 0x42, 0xa2,

0x01, 0x69, 0x67, 0x92, 0x91, 0xe0, 0x7c, 0x4b,

0xfc, 0xc4, 0x85, 0x75, 0x8d, 0xb8, 0x4e, 0x71,

0x16, 0xd0, 0x45, 0x2e, 0x43, 0x76, 0x6a, 0x2f,

0x1f, 0x7c, 0x65, 0xc0, 0x81, 0x2f, 0xcb, 0xeb,

0xe9, 0xda, 0xca, 0x1e, 0xda, 0x5b, 0x08, 0xb1

},

{

0xb7, 0x9b, 0xb1, 0x21, 0x70, 0x04, 0x79, 0xe6,

0x56, 0xcd, 0xcb, 0xd7, 0x1b, 0xa2, 0xdd, 0x55,

0xca, 0xa7, 0x0a, 0xdb, 0xc2, 0x61, 0xb5, 0x5c,

0x58, 0x99, 0xd6, 0x12, 0x6b, 0x17, 0xb5, 0x9a,

0x31, 0x01, 0xb5, 0x16, 0x0f, 0x5e, 0xd5, 0x61,

0x98, 0x2b, 0x23, 0x0a, 0x72, 0xea, 0xfe, 0xf3,

0xd7, 0xb5, 0x70, 0x0f, 0x46, 0x9d, 0xe3, 0x4f,

0x1a, 0x2f, 0x9d, 0xa9, 0x8a, 0xb5, 0xa3, 0x6f

},

{

0xb2, 0x0a, 0xba, 0x0a, 0xf5, 0x96, 0x1e, 0x99,

0x31, 0xdb, 0x7a, 0x86, 0x43, 0xf4, 0xb6, 0xc2,

0x09, 0xdb, 0x62, 0x60, 0x37, 0x3a, 0xc9, 0xc1,

0xb1, 0x9e, 0x35, 0x90, 0xe4, 0x0f, 0xe2, 0xd3,

0x7b, 0x7b, 0x29, 0xb1, 0x14, 0x75, 0xea, 0xf2,

0x8b, 0x1f, 0x9c, 0x52, 0x5f, 0x5e, 0xf1, 0x06,

0x35, 0x84, 0x3d, 0x6a, 0x28, 0xfc, 0x39, 0x0a,

0xc7, 0x2f, 0xce, 0x2b, 0xac, 0xdc, 0x74, 0xf5

},

{

0x2e, 0xd1, 0xe3, 0x84, 0xbc, 0xbe, 0x0c, 0x22,

0xf1, 0x37, 0xe8, 0x93, 0xa1, 0xea, 0x53, 0x34,

0xbe, 0x03, 0x52, 0x93, 0x33, 0x13, 0xb7, 0xd8,

0x75, 0xd6, 0x03, 0xed, 0x82, 0x2c, 0xd7, 0xa9,

0x3f, 0x35, 0x5e, 0x68, 0xad, 0x1c, 0x72, 0x9d,

0x7d, 0x3c, 0x5c, 0x33, 0x7e, 0x85, 0x8e, 0x48,

0xdd, 0xe4, 0x71, 0x5d, 0xa0, 0xe1, 0x48, 0xf9,

0xd2, 0x66, 0x15, 0xe8, 0xb3, 0xdf, 0x1f, 0xef

},

{

0x57, 0xfe, 0x6c, 0x7c, 0xfd, 0x58, 0x17, 0x60,

0xf5, 0x63, 0xea, 0xa9, 0x7e, 0xa2, 0x56, 0x7a,

0x16, 0x1a, 0x27, 0x23, 0xb7, 0x00, 0xff, 0xdf,

0xa3, 0xf5, 0x3a, 0x25, 0x47, 0x17, 0xcd, 0xbf,

0xbd, 0xff, 0x0f, 0x80, 0xd7, 0x35, 0x9e, 0x35,

0x4a, 0x10, 0x86, 0x16, 0x1f, 0x1c, 0x15, 0x7f,

0x63, 0x23, 0xa9, 0x6c, 0x0c, 0x41, 0x3f, 0x9a,

0x99, 0x47, 0x47, 0xad, 0xac, 0x6b, 0xea, 0x4b

},

{

0x6e, 0x7d, 0x64, 0x46, 0x7a, 0x40, 0x68, 0xfa,

0x35, 0x4f, 0x90, 0x36, 0x72, 0xc5, 0x71, 0xbf,

0xb6, 0xc6, 0xbe, 0xc2, 0x66, 0x1f, 0xf2, 0x0a,

0xb4, 0xb7, 0x9a, 0x1c, 0xb7, 0xa6, 0xfa, 0xcf,

0xc6, 0x8e, 0xf0, 0x9a, 0xb4, 0x9a, 0x7f, 0x18,

0x6c, 0xa4, 0x42, 0x51, 0xf9, 0xc4, 0x66, 0x2d,

0xc0, 0x39, 0x30, 0x7a, 0x3b, 0xc3, 0xa4, 0x6f,

0xd9, 0xd3, 0x3a, 0x1d, 0xae, 0xae, 0x4f, 0xae

},

{

0x93, 0xd4, 0x14, 0x3a, 0x4d, 0x56, 0x86, 0x88,

0xf3, 0x4a, 0x3c, 0xa2, 0x4c, 0x45, 0x17, 0x35,

0x04, 0x05, 0x4a, 0x28, 0x83, 0x69, 0x47, 0x06,

0x37, 0x2c, 0x82, 0x2d, 0xc5, 0xab, 0x92, 0x09,

0xc9, 0x93, 0x7a, 0x19, 0x33, 0x3e, 0x47, 0xd3,

0xc9, 0x87, 0xbf, 0xe6, 0xc7, 0xc6, 0x9e, 0x39,

0x54, 0x09, 0x24, 0xbf, 0xfe, 0x86, 0xac, 0x51,

0xec, 0xc5, 0xaa, 0xee, 0x16, 0x0e, 0xc7, 0xf4

},

{

0x1e, 0xe7, 0x02, 0xbf, 0xd4, 0x0d, 0x7f, 0xa4,

0xd9, 0xa8, 0x51, 0x59, 0x35, 0xc2, 0xac, 0x36,

0x2f, 0xc4, 0xa5, 0xd1, 0x2b, 0x8d, 0xd1, 0x69,

0x90, 0x06, 0x9b, 0x92, 0xcb, 0x2b, 0x89, 0xf4,

0x9a, 0xc4, 0xdb, 0x4d, 0x3b, 0x44, 0xb4, 0x89,

0x1e, 0xde, 0x36, 0x9c, 0x71, 0xf8, 0xb7, 0x4e,

0x41, 0x41, 0x6e, 0x0c, 0x02, 0xaa, 0xe7, 0x03,

0xa7, 0xc9, 0x93, 0x4d, 0x42, 0x5b, 0x1f, 0x9b

},

{

0xdb, 0x5a, 0x23, 0x83, 0x51, 0x44, 0x61, 0x72,

0x60, 0x2a, 0x1f, 0xcb, 0x92, 0xdc, 0x38, 0x0e,

0x54, 0x9c, 0x07, 0xa6, 0x9a, 0x8a, 0x2b, 0x7b,

0xb1, 0xce, 0xb2, 0xdb, 0x0b, 0x44, 0x0a, 0x80,

0x84, 0x09, 0x0d, 0xe0, 0xb7, 0x55, 0xd9, 0x3c,

0x24, 0x42, 0x89, 0x25, 0x1b, 0x3a, 0x7d, 0x3a,

0xde, 0x5f, 0x16, 0xec, 0xd8, 0x9a, 0x4c, 0x94,

0x9b, 0x22, 0x31, 0x16, 0x54, 0x5a, 0x8f, 0x37

},

{

0xed, 0x9c, 0x45, 0x98, 0xfb, 0xc7, 0xb4, 0x74,

0xc3, 0xb6, 0x3b, 0x15, 0xd1, 0xfa, 0x98, 0x36,

0xf4, 0x52, 0x76, 0x3b, 0x30, 0x6c, 0x1e, 0x7a,

0x4b, 0x33, 0x69, 0xaf, 0x02, 0x67, 0xe7, 0x9f,

0x03, 0x61, 0x33, 0x1b, 0x8a, 0xe1, 0xff, 0x1f,

0xdb, 0x78, 0x8a, 0xff, 0x1c, 0xe7, 0x41, 0x89,

0xf3, 0xf3, 0xe4, 0xb2, 0x48, 0xe5, 0x2a, 0x38,

0x52, 0x6f, 0x05, 0x80, 0xa6, 0xde, 0xbe, 0xab

},

{

0x1b, 0x2d, 0xf3, 0x81, 0xcd, 0xa4, 0xca, 0x6b,

0x5d, 0xd8, 0x6f, 0xc0, 0x4a, 0x59, 0xa2, 0xde,

0x98, 0x6e, 0x47, 0x7d, 0x1d, 0xcd, 0xba, 0xef,

0xca, 0xb9, 0x48, 0xea, 0xef, 0x71, 0x1d, 0x8a,

0x79, 0x66, 0x84, 0x14, 0x21, 0x80, 0x01, 0x20,

0x61, 0x07, 0xab, 0xeb, 0xbb, 0x6b, 0xfa, 0xd8,

0x94, 0xfe, 0x5a, 0x63, 0xcd, 0xc6, 0x02, 0x30,

0xfb, 0x89, 0xc8, 0xef, 0xd0, 0x9e, 0xcd, 0x7b

},

{

0x20, 0xd7, 0x1b, 0xf1, 0x4a, 0x92, 0xbc, 0x48,

0x99, 0x1b, 0xb2, 0xd9, 0xd5, 0x17, 0xf4, 0xfa,

0x52, 0x28, 0xe1, 0x88, 0xaa, 0xa4, 0x1d, 0xe7,

0x86, 0xcc, 0x91, 0x18, 0x9d, 0xef, 0x80, 0x5d,

0x9b, 0x9f, 0x21, 0x30, 0xd4, 0x12, 0x20, 0xf8,

0x77, 0x1d, 0xdf, 0xbc, 0x32, 0x3c, 0xa4, 0xcd,

0x7a, 0xb1, 0x49, 0x04, 0xb0, 0x80, 0x13, 0xd2,

0xba, 0x31, 0x16, 0xf1, 0x67, 0xe7, 0x8e, 0x37

}

};

typedef struct stribog\_hash\_context

{

uint\_block\_size buffer; // буффер

size\_t buffer\_size; // размер оставшейся части

uint\_block\_size result; // результат (хэш)

uint\_block\_size v\_0; // Инициализационный вектор

uint\_block\_size v\_512; // Инициализационный вектор

uint\_block\_size h; // Промежуточный результат вычислений

uint\_block\_size sigma; // Контрольная сумма

uint\_block\_size N;

} THashContext;

void hash\_init(THashContext\* stribog\_ctx)

{

memset(stribog\_ctx, 0x00, sizeof(THashContext));

memset(stribog\_ctx->h, 0x01, BLOCK\_SIZE);

stribog\_ctx->v\_512[1] = 0x02;

}

// Побитовое xor (512)

void hash\_xor(const uint8\_t\* a, const uint8\_t\* b, uint8\_t\* c)

{

int i;

int internal = 0;

for (i = 0; i < 64; i++)

{

internal = a[i] + b[i] + (internal >> 8);

c[i] = internal & 0xff;

}

}

// Двоичное сложение по модулю 2

void hash\_add\_mod(const uint8\_t\* a, const uint8\_t\* b, uint8\_t\* c)

{

int i;

for (i = 0; i < 64; i++)

c[i] = a[i] ^ b[i];

}

// преобразование перестановки

void hash\_p(uint8\_t\* state)

{

int i;

uint\_block\_size internal;

for (i = 63; i >= 0; i--)

internal[i] = state[tau\_values[i]];

memcpy(state, internal, BLOCK\_SIZE);

}

// преобразование замены или подстановки

void hash\_s(uint8\_t\* state)

{

int i;

uint\_block\_size internal;

for (i = 63; i >= 0; i--)

internal[i] = stri\_pi\_values[state[i]];

memcpy(state, internal, BLOCK\_SIZE);

}

// линейное преобразование

void hash\_l(uint8\_t\* state)

{

uint64\_t\* internal\_in = (uint64\_t\*)state;

uint64\_t internal\_out[8];

memset(internal\_out, 0x00, BLOCK\_SIZE);

int i, j;

for (i = 7; i >= 0; i--)

{

for (j = 63; j >= 0; j--)

if ((internal\_in[i] >> j) & 1)

internal\_out[i] ^= a\_values[63 - j];

}

memcpy(state, internal\_out, 64);

}

// Получаем раундовые ключи

void hash\_getkey(uint8\_t\* K, int i)

{

hash\_add\_mod(K, c\_values[i], K);

hash\_s(K);

hash\_p(K);

hash\_l(K);

}

// ф-ция преобразования

void hash\_e(uint8\_t\* K, const uint8\_t\* m, uint8\_t\* state)

{

int i;

memcpy(K, K, BLOCK\_SIZE);

hash\_add\_mod(m, K, state);

for (i = 0; i < 12; i++)

{

hash\_s(state);

hash\_p(state);

hash\_l(state);

hash\_getkey(K, i);

hash\_add\_mod(state, K, state);

}

}

// ф-ция сжатия

void hash\_g(uint8\_t\* h, uint8\_t\* N, const uint8\_t\* m)

{

uint\_block\_size K, internal;

hash\_add\_mod(N, h, K);

hash\_s(K);

hash\_p(K);

hash\_l(K);

hash\_e(K, m, internal);

hash\_add\_mod(internal, h, internal);

hash\_add\_mod(internal, m, h);

}

void hash\_padding(THashContext\* stribog\_ctx)

{

uint\_block\_size internal;

if (stribog\_ctx->buffer\_size < BLOCK\_SIZE)

{

memset(internal, 0x00, BLOCK\_SIZE);

memcpy(internal, stribog\_ctx->buffer, stribog\_ctx->buffer\_size);

internal[stribog\_ctx->buffer\_size] = 0x01;

memcpy(stribog\_ctx->buffer, internal, BLOCK\_SIZE);

}

}

// хешируем блок

void hash\_block(THashContext\* stribog\_ctx, const uint8\_t\* data)

{

hash\_g(stribog\_ctx->h, stribog\_ctx->N, data);

hash\_xor(stribog\_ctx->N, stribog\_ctx->v\_512, stribog\_ctx->N);

hash\_xor(stribog\_ctx->sigma, data, stribog\_ctx->sigma);

}

// хешируем остаток и считаем контрольную сумму

void hash\_last(THashContext\* stribog\_ctx)

{

uint\_block\_size internal;

memset(internal, 0x00, BLOCK\_SIZE);

internal[1] = ((stribog\_ctx->buffer\_size \* 8) >> 8) & 0xff;

internal[0] = (stribog\_ctx->buffer\_size \* 8) & 0xff;

hash\_padding(stribog\_ctx);

hash\_g(stribog\_ctx->h, stribog\_ctx->N, (const uint8\_t\*)&(stribog\_ctx->buffer));

hash\_xor(stribog\_ctx->N, internal, stribog\_ctx->N);

hash\_xor(stribog\_ctx->sigma, stribog\_ctx->buffer, stribog\_ctx->sigma);

hash\_g(stribog\_ctx->h, stribog\_ctx->v\_0, (const uint8\_t\*)&(stribog\_ctx->N));

hash\_g(stribog\_ctx->h, stribog\_ctx->v\_0, (const uint8\_t\*)&(stribog\_ctx->sigma));

memcpy(stribog\_ctx->result, stribog\_ctx->h, BLOCK\_SIZE);

}

// хешируем блоки

void hash\_blocks(THashContext\* stribog\_ctx, const uint8\_t\* data, size\_t len)

{

size\_t chk\_size;

while ((len > 63) && (stribog\_ctx->buffer\_size) == 0)

{

hash\_block(stribog\_ctx, data);

data += 64;

len -= 64;

}

while (len)

{

chk\_size = 64 - stribog\_ctx->buffer\_size;

if (chk\_size > len)

chk\_size = len;

memcpy(&stribog\_ctx->buffer[stribog\_ctx->buffer\_size], data, chk\_size);

stribog\_ctx->buffer\_size += chk\_size;

len -= chk\_size;

data += chk\_size;

if (stribog\_ctx->buffer\_size == 64)

{

hash\_block(stribog\_ctx, stribog\_ctx->buffer);

stribog\_ctx->buffer\_size = 0;

}

}

}

void hash\_final(THashContext\* stribog\_ctx)

{

hash\_last(stribog\_ctx);

stribog\_ctx->buffer\_size = 0;

}

void lr4(std::string str) {

std::cout << "Message: " << str << std::endl;

THashContext\* stribog\_ctx = new THashContext();

hash\_init(stribog\_ctx);

hash\_blocks(stribog\_ctx, (uint8\_t\*)str.c\_str(), str.size());

hash\_final(stribog\_ctx);

// Преобразуем в шестнадцатеричное представление

std::cout << "Hash: ";

for (int i = 32; i < 64; i++)

std::cout << std::setw(2) << std::setfill('0') << std::hex << int{ stribog\_ctx->result[i] };

std::cout << std::endl;

}

# Результат работы

Используется длина хэша 256 бит

Message: This is very long test string.

Hash: 2c1b7a5964c82a5868303a6169158a7b780578732c1d00932da4b409f8e6cf8e.

Сообщение было успешно хэшировано алгоритмом Стрибог. Была произведена проверка на альтернативной, проверенной реализации этого алгоритма.ы