Министерство образования республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Контрольная работа №1

по дисциплине «Методы защиты информации»

Студент 4 курса

Группы № 493551

Рыбак Артем Владимирович

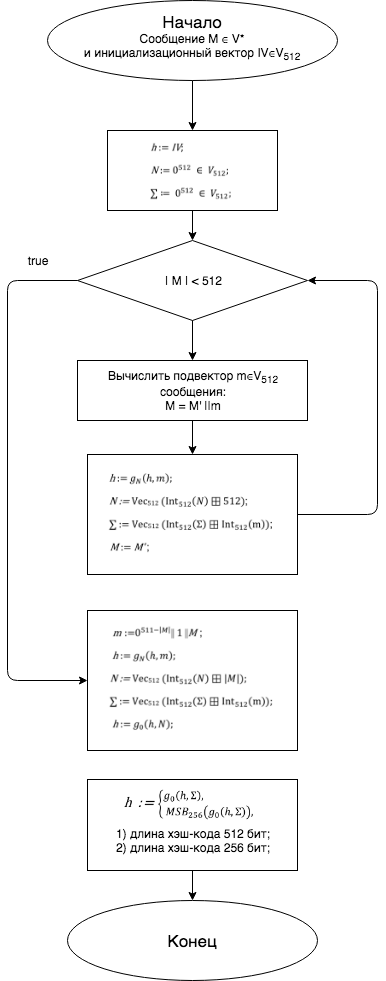
Минск, 2018

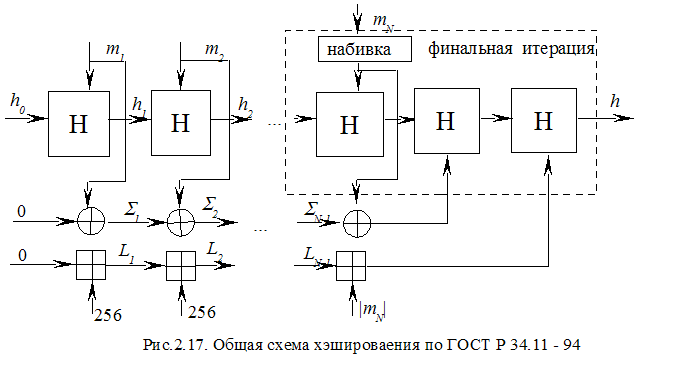
**Введение**

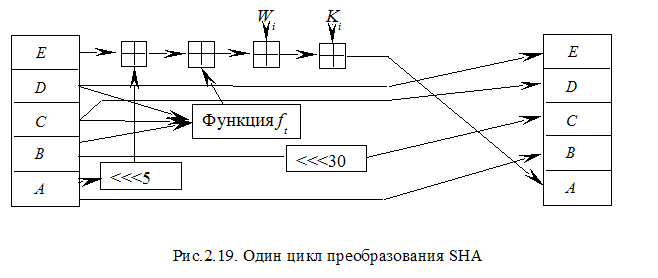
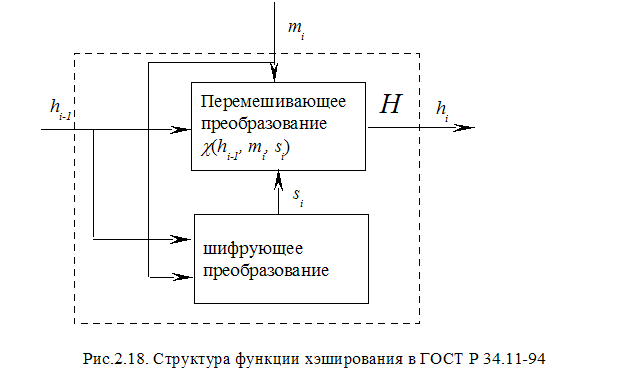
Контрольная работа № 1 подразумевает изучение и программную реализацию алгоритма формирования Хэш-функции ГОСТ 3411.

Целью работы является создание и тестирование алгоритма формирования Хэш-функции ГОСТ 3411 на языке высокого уровня.

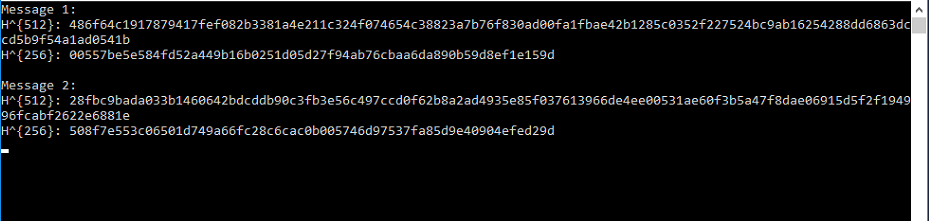
**Блок-схема алгоритма**







**Результаты ввода данных и исполнения программы**



**Программный код**

//Stribog.c

#include <stdio.h>

#include <string.h>

#include <conio.h>

#include "types.h"

#include "stribog.h"

void print\_hash(struct stribog\_ctx\_t \*ctx)

{

u8 i;

for (i = 0; i < (ctx->size ? OUTPUT\_SIZE\_512 : OUTPUT\_SIZE\_256); i++)

printf("%02x", ctx->h[i]);

putchar('\n');

}

int main(int argc, char \*argv[])

{

struct stribog\_ctx\_t ctx;

u8 message1[63] = {

0x32, 0x31, 0x30, 0x39, 0x38, 0x37, 0x36, 0x35,

0x34, 0x33, 0x32, 0x31, 0x30, 0x39, 0x38, 0x37,

0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30, 0x39,

0x38, 0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31,

0x30, 0x39, 0x38, 0x37, 0x36, 0x35, 0x34, 0x33,

0x32, 0x31, 0x30, 0x39, 0x38, 0x37, 0x36, 0x35,

0x34, 0x33, 0x32, 0x31, 0x30, 0x39, 0x38, 0x37,

0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30

};

u8 message2[72] = {

0xfb, 0xe2, 0xe5, 0xf0, 0xee, 0xe3, 0xc8, 0x20,

0xfb, 0xea, 0xfa, 0xeb, 0xef, 0x20, 0xff, 0xfb,

0xf0, 0xe1, 0xe0, 0xf0, 0xf5, 0x20, 0xe0, 0xed,

0x20, 0xe8, 0xec, 0xe0, 0xeb, 0xe5, 0xf0, 0xf2,

0xf1, 0x20, 0xff, 0xf0, 0xee, 0xec, 0x20, 0xf1,

0x20, 0xfa, 0xf2, 0xfe, 0xe5, 0xe2, 0x20, 0x2c,

0xe8, 0xf6, 0xf3, 0xed, 0xe2, 0x20, 0xe8, 0xe6,

0xee, 0xe1, 0xe8, 0xf0, 0xf2, 0xd1, 0x20, 0x2c,

0xe8, 0xf0, 0xf2, 0xe5, 0xe2, 0x20, 0xe5, 0xd1

};

printf("Message 1:\n");

init(&ctx, HASH512);

stribog(&ctx, message1, sizeof(message1));

printf("H^{512}: ");

print\_hash(&ctx);

init(&ctx, HASH256);

stribog(&ctx, message1, sizeof(message1));

printf("H^{256}: ");

print\_hash(&ctx);

printf("\nMessage 2:\n");

init(&ctx, HASH512);

stribog(&ctx, message2, sizeof(message2));

printf("H^{512}: ");

print\_hash(&ctx);

init(&ctx, HASH256);

stribog(&ctx, message2, sizeof(message2));

printf("H^{256}: ");

print\_hash(&ctx);

\_getch();

return 0;

}

//stribog.h

#ifndef \_STRIBOG\_H

#define \_STRIBOG\_H

#include <string.h>

#include "types.h"

#include "stribog\_data.h"

#define BLOCK\_SIZE 64

#define BLOCK\_BIT\_SIZE (BLOCK\_SIZE \* 8)

#define OUTPUT\_SIZE\_512 64

#define OUTPUT\_SIZE\_256 32

#define HASH256 0

#define HASH512 1

struct stribog\_ctx\_t {

u8 h[BLOCK\_SIZE];

u8 N[BLOCK\_SIZE];

u8 S[BLOCK\_SIZE];

u8 size;

};

void addmod512(u8 \*dst, u8 \*src, u8 \*add)

{

u8 i;

u16 overrun = 0;

for (i = BLOCK\_SIZE; i-- > 0;) {

overrun = src[i] + add[i] + (overrun >> 8);

dst[i] = (u8) overrun;

}

}

void addmod512\_u32(u8 \*dst, u8 \*src, u32 add)

{

u8 i;

for (i = BLOCK\_SIZE; i-- > 0;) {

add = (u32) src[i] + add;

dst[i] = add;

add >>= 8;

}

}

void xor512(u8 \*dst, u8 \*a, u8 \*b)

{

u8 i;

for (i = 0; i < BLOCK\_SIZE; i++) {

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

}

}

void S(u8 \*vect)

{

u8 i;

for (i = 0; i < BLOCK\_SIZE; i++) {

vect[i] = sbox[vect[i]];

}

}

void LP(u8 \*vect)

{

u8 i, j, k;

u8 tmp[64];

memcpy(tmp, vect, 64);

u64 c;

/\*

\* subvectors of 512-bit vector (64\*8 bits)

\* an subvector is start at [j\*8], its componenst placed

\* with step of 8 bytes (due to this function is composition

\* of P and L) and have length of 64 bits (8\*8 bits)

\*/

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

c = 0;

/\*

\* 8-bit components of 64-bit subvectors

\* components is placed at [j\*8+i]

\*/

for (j = 0; j < 8; j++) {

/\* bit index of current 8-bit component \*/

for (k = 0; k < 8; k++) {

/\* check if current bit is set \*/

if (tmp[j\*8+i] & 0x80 >> k)

c ^= A[j\*8+k];

}

}

for (j = 0; j < 8; j++) {

vect[i\*8+j] = c >> (7 - j) \* 8;

}

}

}

void X(u8 \*dst, u8 \*a, u8 \*b)

{

xor512(dst, a, b);

}

void E(u8 \*dst, u8 \*k, u8 \*m)

{

u8 i;

u8 K[64];

memcpy(K, k, BLOCK\_SIZE);

X(dst, K, m);

for (i = 1; i < 13; i++) {

S(dst);

LP(dst);

/\* next K \*/

X(K, K, C[i-1]);

S(K);

LP(K);

X(dst, K, dst);

}

}

void g\_N(u8 \*h, u8 \*N, u8 \*m)

{

u8 hash[BLOCK\_SIZE];

memcpy(hash, h, BLOCK\_SIZE);

xor512(h, h, N);

S(h);

LP(h);

E(h, h, m);

xor512(h, h, hash);

xor512(h, h, m);

}

void g\_0(u8 \*h, u8 \*m)

{

u8 hash[64];

memcpy(hash, h, BLOCK\_SIZE);

S(h);

LP(h);

E(h, h, m);

xor512(h, h, hash);

xor512(h, h, m);

}

void stribog(struct stribog\_ctx\_t \*ctx, u8 \*message, u64 len)

{

u8 m[BLOCK\_SIZE];

u8 padding;

while (len >= BLOCK\_SIZE) {

memcpy(m, message + len - BLOCK\_SIZE, BLOCK\_SIZE);

g\_N(ctx->h, ctx->N, m);

len -= BLOCK\_SIZE;

addmod512\_u32(ctx->N, ctx->N, BLOCK\_BIT\_SIZE);

addmod512(ctx->S, ctx->S, m);

}

padding = BLOCK\_SIZE - len;

if (padding) {

memset(m, 0x00, padding - 1);

memset(m + padding - 1, 0x01, 1);

memcpy(m + padding, message, len);

}

g\_N(ctx->h, ctx->N, m);

addmod512\_u32(ctx->N, ctx->N, len\*8);

addmod512(ctx->S, ctx->S, m);

g\_0(ctx->h, ctx->N);

g\_0(ctx->h, ctx->S);

}

void init(struct stribog\_ctx\_t \*ctx, u8 size)

{

memset(ctx->N, 0x00, BLOCK\_SIZE);

memset(ctx->S, 0x00, BLOCK\_SIZE);

ctx->size = !!size;

if (ctx->size)

memcpy(ctx->h, iv512, BLOCK\_SIZE);

else

memcpy(ctx->h, iv256, BLOCK\_SIZE);

}

#endif

//stribog\_data.h

#ifndef \_STRIBOG\_DATA\_H

#define \_STRIBOG\_DATA\_H

#include "types.h"

const u8 iv512[64] = {

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

};

const u8 iv256[64] = {

0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,

0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,

0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,

0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,

0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,

0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,

0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,

0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01

};

const u8 sbox[256] = {

0xfc, 0xee, 0xdd, 0x11, 0xcf, 0x6e, 0x31, 0x16,

0xfb, 0xc4, 0xfa, 0xda, 0x23, 0xc5, 0x04, 0x4d,

0xe9, 0x77, 0xf0, 0xdb, 0x93, 0x2e, 0x99, 0xba,

0x17, 0x36, 0xf1, 0xbb, 0x14, 0xcd, 0x5f, 0xc1,

0xf9, 0x18, 0x65, 0x5a, 0xe2, 0x5c, 0xef, 0x21,

0x81, 0x1c, 0x3c, 0x42, 0x8b, 0x01, 0x8e, 0x4f,

0x05, 0x84, 0x02, 0xae, 0xe3, 0x6a, 0x8f, 0xa0,

0x06, 0x0b, 0xed, 0x98, 0x7f, 0xd4, 0xd3, 0x1f,

0xeb, 0x34, 0x2c, 0x51, 0xea, 0xc8, 0x48, 0xab,

0xf2, 0x2a, 0x68, 0xa2, 0xfd, 0x3a, 0xce, 0xcc,

0xb5, 0x70, 0x0e, 0x56, 0x08, 0x0c, 0x76, 0x12,

0xbf, 0x72, 0x13, 0x47, 0x9c, 0xb7, 0x5d, 0x87,

0x15, 0xa1, 0x96, 0x29, 0x10, 0x7b, 0x9a, 0xc7,

0xf3, 0x91, 0x78, 0x6f, 0x9d, 0x9e, 0xb2, 0xb1,

0x32, 0x75, 0x19, 0x3d, 0xff, 0x35, 0x8a, 0x7e,

0x6d, 0x54, 0xc6, 0x80, 0xc3, 0xbd, 0x0d, 0x57,

0xdf, 0xf5, 0x24, 0xa9, 0x3e, 0xa8, 0x43, 0xc9,

0xd7, 0x79, 0xd6, 0xf6, 0x7c, 0x22, 0xb9, 0x03,

0xe0, 0x0f, 0xec, 0xde, 0x7a, 0x94, 0xb0, 0xbc,

0xdc, 0xe8, 0x28, 0x50, 0x4e, 0x33, 0x0a, 0x4a,

0xa7, 0x97, 0x60, 0x73, 0x1e, 0x00, 0x62, 0x44,

0x1a, 0xb8, 0x38, 0x82, 0x64, 0x9f, 0x26, 0x41,

0xad, 0x45, 0x46, 0x92, 0x27, 0x5e, 0x55, 0x2f,

0x8c, 0xa3, 0xa5, 0x7d, 0x69, 0xd5, 0x95, 0x3b,

0x07, 0x58, 0xb3, 0x40, 0x86, 0xac, 0x1d, 0xf7,

0x30, 0x37, 0x6b, 0xe4, 0x88, 0xd9, 0xe7, 0x89,

0xe1, 0x1b, 0x83, 0x49, 0x4c, 0x3f, 0xf8, 0xfe,

0x8d, 0x53, 0xaa, 0x90, 0xca, 0xd8, 0x85, 0x61,

0x20, 0x71, 0x67, 0xa4, 0x2d, 0x2b, 0x09, 0x5b,

0xcb, 0x9b, 0x25, 0xd0, 0xbe, 0xe5, 0x6c, 0x52,

0x59, 0xa6, 0x74, 0xd2, 0xe6, 0xf4, 0xb4, 0xc0,

0xd1, 0x66, 0xaf, 0xc2, 0x39, 0x4b, 0x63, 0xb6

};

const u64 A[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

};

static u8 C[12][64] = {

{

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}, {

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

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

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

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

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

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

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

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

}

};

#endif

//type.h

#ifndef \_TYPES\_H

#define \_TYPES\_H

#include <stdint.h>

typedef uint8\_t u8;

typedef uint16\_t u16;

typedef uint32\_t u32;

typedef uint64\_t u64;

#endif

**Вывод**

Данный алгоритм соответствуюет современным требованиям к криптографической стойкости. Размер хеша — 256 или 512 бит; размер блока входных данных — 512 бит.

Хэш-функция использует изученные конструкции и преобразования.Алгоритм не содержит лишних преобразований, усложняющих конструкцию. Вычисление хэш-функции эффективно и не занимает много времени.