Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине «Криптографические методы защиты информации»

Тема: «Программная реализация ЭЦП»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Литвинюк Т. В.

**Проверил:**

Хацкевич А. С.

Брест 2023

**Цель:** создать программу, которая реализует учебный вариант схем ЭЦП, используя алгоритмы с открытыми ключами.

**Ход работы:**

**Вариант 2**

Цифровая подпись на базе алгоритма RSA

Код сервера:

*import* socket

*from* threading *import* Thread

clients = []

def resend\_messages(client: socket.socket):

*while* True:

*try*:

            data = client.recv(512)

*except* ConnectionResetError:

            server.close()

*return*

*else*:

*if* "!exit" in data.decode():

                server.close()

*return*

*for* cl *in* clients:

*if* cl != client:

                    cl.send(data)

*if* \_\_name\_\_ == '\_\_main\_\_':

    server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server.bind(('localhost', 8080))

    server.listen(2)

*while* len(clients) != 2:

        client = server.accept()[0]

        print("user connected")

        clients.append(client)

        Thread(target=resend\_messages, args=(client,)).start()

    clients[0].send("!ready".encode())

    clients[1].send("!ready".encode())

Код клиента:

*import* socket

*from* threading *import* Thread

*from* encrypting *import* generate\_rsa\_keys, ripemd320

def send\_message(key):

    print("Команды:\n!exit - закрыть соединение\n!send - отправить обычное сообщение\n!S\_\_ - отправить подписанное сообщение")

*while* True:

        message = input()

*if* '!exit' in message:

            client.send(message.encode())

*return*

*elif* '!S\_\_' in message:

            client.send(message.encode())

            C = pow(ripemd320(message[5:]), key, NA)

            client.send(f"{C}".encode())

            print(f"{C=}, {ripemd320(message)}")

*else*:

            client.send(message.encode())

def receive\_message(key):

*while* True:

        message = client.recv(2048).decode()

*if* '!exit' in message:

*return*

*elif* '!S\_\_' in message:

            M = message[5:]

            C = int(client.recv(512))

            print(f"{C=}")

*if* pow(C, key, NB) == ripemd320(M) % NB:

                print("<Сообщение подписано> ", end='')

*else*:

                print("<Сообщение подписано не этим пользователем>")

        print(message)

*if* \_\_name\_\_ == '\_\_main\_\_':

    client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    client.connect(('localhost', 8080))

    print("Соединение с сервером установлено.")

    message = client.recv(512).decode()

*if* "!ready" in message:

        eA, dA, NA = generate\_rsa\_keys()

        client.send(f"{eA} {NA}".encode())

        eB, NB = map(int, client.recv(512).decode().split()) *# получение открытого ключа от другого пользователя*

        print(f"Обмен ключами произошел. {eA=}, {dA=}, {NA=}, {eB=}, {NB=}")

        send\_message\_thr = Thread(target=send\_message, args=(dA,))

        recv\_message\_thr = Thread(target=receive\_message, args=(eB,))

        send\_message\_thr.start()

        recv\_message\_thr.start()

        send\_message\_thr.join()

        client.close()

Код функций, необходимых для работы приложения:

*import* random

*import* ctypes

*# Функция для проверки простоты числа*

def is\_prime(n):

*if* n <= 1:

*return* False

*if* n <= 3:

*return* True

*if* n % 2 == 0 or n % 3 == 0:

*return* False

    i = 5

*while* i \* i <= n:

*if* n % i == 0 or n % (i + 2) == 0:

*return* False

        i += 6

*return* True

*# Функция для поиска наибольшего общего делителя*

def gcd(a, b):

*while* b:

        a, b = b, a % b

*return* a

*# Функция для нахождения обратного по модулю (расширенный алгоритм Евклида)*

def mod\_inverse(e, phi):

    d = 0

    x1, x2 = 0, 1

    y1, y2 = 1, 0

*while* phi != 0:

        quotient = e // phi

        e, phi = phi, e % phi

        x1, x2 = x2 - quotient \* x1, x1

        y1, y2 = y2 - quotient \* y1, y1

*if* x2 < 0:

        x2 += phi

*return* x2

def generate\_rsa\_keys():

*while* True:

        p = random.randint(1000000, 10000000)

*if* is\_prime(p):

*break*

*while* True:

        q = random.randint(1000000, 10000000)

*if* is\_prime(q) and q != p:

*break*

*# Вычисление N и функции Эйлера phi(N)*

    N = p \* q

    phi\_N = (p - 1) \* (q - 1)

*# Выбор числа e (обычно простого, но не обязательно)*

    e = random.randint(2, phi\_N)

*while* gcd(e, phi\_N) != 1:

        e = random.randint(2, phi\_N)

*# Вычисление числа d, обратного e по модулю phi(N)*

    d = mod\_inverse(e, phi\_N)

*# Вывод открытого и закрытого ключей*

*return* e, d, N

lib = ctypes.WinDLL(r"E:\Studing\KMZI\LAB3\ripemd320.dll")

lib.ripemd320.restype = ctypes.c\_char\_p

def ripemd320(message: str) -> int:

*return* int(lib.ripemd320(message), 16)

Код функции ripemd320:

*#include* "ripemd320.h"

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

};

*#define* BITSET\_SIZE 2000

typedef struct {

    int set\_size;

    int set[BITSET\_SIZE];

    int length;

} Bitset;

void initBitset(Bitset \*bitset) {

    bitset->set\_size = BITSET\_SIZE;

    bitset->length = 0;

    memset(bitset->set, 0, BITSET\_SIZE \* sizeof(int));

}

void appendBit(Bitset \*bitset, int bit) {

*if* (bitset->length < bitset->set\_size) {

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

    }

}

void appendChar(Bitset \*bitset, char chr) {

*for* (int i = 7; i >= 0; i--) {

        int bit = (chr >> i) & 1;

        appendBit(bitset, bit);

    }

}

void extendString(Bitset \*bitset, const char \*str) {

*for* (int i = 0; str[i] != '\0'; i++) {

        appendChar(bitset, str[i]);

    }

}

void addValues(Bitset \*bitset, int bit, int size) {

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

        appendBit(bitset, bit);

    }

}

char \*ripemd320(const char \*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;

    initBitset(&bitarray);

    int b = strlen(message) \* 8;

*// добавление битов сообщения*

    extendString(&bitarray, message);

*// добавление дополнительных битов*

    appendBit(&bitarray, 1);

    addValues(&bitarray, 0, (448 - (b + 1) % 512) % 512);

*// добавление исходной длины сообщения*

*for* (int i = 0; i < 32; i++) {

        int bit = (b >> i) & 1;

        appendBit(&bitarray, bit);

    }

*for* (int i = 0; i < 32; i++) {

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

        appendBit(&bitarray, 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.set[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.set[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;

    }

    char hexString[100];

    snprintf(hexString, sizeof(hexString), "%X%X%X%X%X%X%X%X%X", h1, h2, h3, h4, h5, h6, h7, h8, h9);

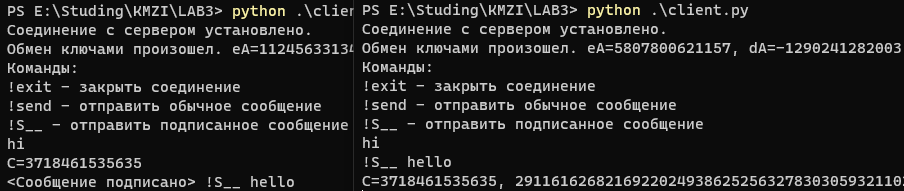
    char\* res = (char\*)malloc(100 \* sizeof(char));

*for* (int i = 0; i < 100; i++)

        res[i] = hexString[i];

*return* res;

}



**Вывод:** в ходе лабораторной работы я изучил работу ЭЦП, создал ее на базе алгоритма RSA.