Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

За пятый семестр

По дисциплине: «Криптографические методы защиты информации»

**Тема: «Программная реализация ЭЦП»**

Выполнила:

Студент 3 курса

Группы ИИ-23(II)

Романюк А. П.

Проверила:

Хацкевич А. С.

Брест 2024

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

**Ход работы:**

**Вариант 1** (Цифровая подпись Эль-Гамаля)

**Задание:**

#include <iostream>

#include <vector>

#include <openssl/bn.h>

#include <openssl/evp.h>

#include <openssl/rand.h>

using namespace std;

const int SHA256\_DIGEST\_LENGTH = 32;

const int r1[64] = {

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,

};

const int r2[64] = {

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

};

const int s1[64] = {

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

};

const int s2[64] = {

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

};

uint32\_t K1(unsigned int j) {

return

j < 16 ? 0x00000000 :

j < 32 ? 0x5A827999 :

j < 48 ? 0x6ED9EBA1 :

0x8F1BBCDC;

}

uint32\_t K2(unsigned int j) {

return

j < 16 ? 0x50A28BE6 :

j < 32 ? 0x5C4DD124 :

j < 48 ? 0x6D703EF3 :

0x00000000;

}

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

if (round < 16) return (x ^ y ^ z);

if (round < 32) return ((x & y) | (~x & z));

if (round < 48) return ((x | ~y) ^ z);

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

}

void ripemd256(const uint8\_t\* input, size\_t length, uint8\_t\* output) {

uint32\_t h[8] = { 0x67452301,0xEFCDAB89,0x98BADCFE,0x10325476,

0x76543210,0xFEDCBA98,0x89ABCDEF,0x01234567 };

vector<uint8\_t> padded\_data(length + 64);

memcpy(padded\_data.data(), input, length);

padded\_data[length] = 0x80;

size\_t padded\_length = length + 1;

while (padded\_length % 64 != 56) {

padded\_data[padded\_length++] = 0x00;

}

uint64\_t bit\_length = length \* 8;

padded\_data[padded\_length++] = bit\_length & 0xFF;

padded\_data[padded\_length++] = (bit\_length >> 8) & 0xFF;

padded\_data[padded\_length++] = (bit\_length >> 16) & 0xFF;

padded\_data[padded\_length++] = (bit\_length >> 24) & 0xFF;

padded\_data[padded\_length++] = (bit\_length >> 32) & 0xFF;

padded\_data[padded\_length++] = (bit\_length >> 40) & 0xFF;

padded\_data[padded\_length++] = (bit\_length >> 48) & 0xFF;

padded\_data[padded\_length++] = (bit\_length >> 56) & 0xFF;

for (size\_t i = 0; i < padded\_length; i += 64) {

uint32\_t w[16];

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

w[j] = padded\_data[i + j \* 4 + 0] |

(padded\_data[i + j \* 4 + 1] << 8) |

(padded\_data[i + j \* 4 + 2] << 16) |

(padded\_data[i + j \* 4 + 3] << 24);

}

uint32\_t A1 = h[0];

uint32\_t B1 = h[1];

uint32\_t C1 = h[2];

uint32\_t D1 = h[3];

uint32\_t A2 = h[4];

uint32\_t B2 = h[5];

uint32\_t C2 = h[6];

uint32\_t D2 = h[7];

uint32\_t T;

for (int j = 0; j < 64; j++) {

T = (A1 + f(B1, C1, D1, j) + w[r1[j]] + K1(j)) & 0xFFFFFFFF;

T = (T << s1[j]) | (T >> (32 - s1[j])); // Rotate left by s1[j]

A1 = D1;

D1 = C1;

C1 = B1;

B1 = T;

T = (A2 + f(B2, C2, D2, 63 - j) + w[r2[j]] + K2(j)) & 0xFFFFFFFF;

T = (T << s2[j]) | (T >> (32 - s2[j])); // Rotate left by s2[j]

A2 = D2;

D2 = C2;

C2 = B2;

B2 = T;

if (j == 15) { std::swap(A1, A2); }

if (j == 31) { std::swap(B1, B2); }

if (j == 47) { std::swap(C1, C2); }

if (j == 63) { std::swap(D1, D2); }

}

h[0] = h[0] + A1;

h[1] = h[1] + B1;

h[2] = h[2] + C1;

h[3] = h[3] + D1;

h[4] = h[4] + A2;

h[5] = h[5] + B2;

h[6] = h[6] + C2;

h[7] = h[7] + D2;

}

memcpy(output, h, 32);

}

void hashMessage(const unsigned char\* msg, size\_t msgLen, unsigned char\* hash) {

ripemd256(msg, msgLen, hash); // Используем RIPEMD-256

}

// Генерация ключей для системы Эль-Гамаля

void generateKeys(BIGNUM\* p, BIGNUM\* g, BIGNUM\* x, BIGNUM\* y, BN\_CTX\* ctx) {

// Генерация секретного ключа x (x < p)

BN\_rand\_range(x, p);

cout << "Секретный ключ x: " << BN\_bn2dec(x) << endl;

// Вычисление открытого ключа y: y = g^x mod p

BN\_mod\_exp(y, g, x, p, ctx);

cout << "Открытый ключ y: " << BN\_bn2dec(y) << endl;

}

// Подписание сообщения с использованием Эль-Гамаля

void signMessage(const BIGNUM\* p, const BIGNUM\* g, const BIGNUM\* x, const unsigned char\* M\_hash, BIGNUM\* a, BIGNUM\* b, BN\_CTX\* ctx) {

BIGNUM\* k = BN\_new(); // Случайное число k

BIGNUM\* k\_inv = BN\_new(); // Обратное k

BIGNUM\* temp = BN\_new(); // Временное значение

BIGNUM\* p\_1 = BN\_new(); // p-1

BN\_sub(p\_1, p, BN\_value\_one()); // Вычисляем p - 1

// Генерация случайного k (1 < k < p - 1), такое что gcd(k, p - 1) = 1

do {

BN\_rand\_range(k, p\_1);

BN\_gcd(temp, k, p\_1, ctx); // Вычисляем GCD

} while (!BN\_is\_one(temp)); // Проверяем, что GCD равен 1

cout << "Случайное k: " << BN\_bn2dec(k) << endl;

// Вычисляем подпись a: a = g^k mod p

BN\_mod\_exp(a, g, k, p, ctx);

cout << "Подпись a: " << BN\_bn2dec(a) << endl;

// Вычисляем обратное k^(-1) mod (p - 1)

BN\_mod\_inverse(k\_inv, k, p\_1, ctx);

BIGNUM\* M = BN\_new();

// Преобразуем хэш сообщения в BIGNUM

BN\_bin2bn(M\_hash, SHA256\_DIGEST\_LENGTH, M);

// Вычисляем подпись b: b = (M - x \* a) \* k^(-1) mod (p - 1)

BN\_mod\_mul(temp, x, a, p\_1, ctx); // temp = x \* a

BN\_mod\_sub(temp, M, temp, p\_1, ctx); // temp = M - x \* a

BN\_mod\_mul(b, temp, k\_inv, p\_1, ctx); // b = (M - x \* a) \* k^(-1)

cout << "Подпись b: " << BN\_bn2dec(b) << endl;

// Освобождение ресурсов

BN\_free(k);

BN\_free(k\_inv);

BN\_free(temp);

BN\_free(p\_1);

BN\_free(M);

}

// Проверка подписи

bool verifySignature(const BIGNUM\* p, const BIGNUM\* g, const BIGNUM\* y, const unsigned char\* M\_hash, const BIGNUM\* a, const BIGNUM\* b, BN\_CTX\* ctx) {

BIGNUM\* lhs = BN\_new(); // Левая часть (y^a \* a^b mod p)

BIGNUM\* rhs = BN\_new(); // Правая часть (g^M mod p)

BIGNUM\* y\_a = BN\_new(); // y^a mod p

BIGNUM\* a\_b = BN\_new(); // a^b mod p

BIGNUM\* M = BN\_new(); // Хэш сообщения

// Преобразуем хэш сообщения в BIGNUM

BN\_bin2bn(M\_hash, SHA256\_DIGEST\_LENGTH, M);

// Вычисляем левую часть: lhs = (y^a \* a^b) mod p

BN\_mod\_exp(y\_a, y, a, p, ctx);

BN\_mod\_exp(a\_b, a, b, p, ctx);

BN\_mod\_mul(lhs, y\_a, a\_b, p, ctx);

// Вычисляем правую часть: rhs = g^M mod p

BN\_mod\_exp(rhs, g, M, p, ctx);

// Проверка: является ли подпись действительной

bool isValid = BN\_cmp(lhs, rhs) == 0;

cout << "Левая часть (y^a \* a^b): " << BN\_bn2dec(lhs) << endl;

cout << "Правая часть (g^M): " << BN\_bn2dec(rhs) << endl;

// Освобождение ресурсов

BN\_free(lhs);

BN\_free(rhs);

BN\_free(y\_a);

BN\_free(a\_b);

BN\_free(M);

return isValid;

}

int main() {

BN\_CTX\* ctx = BN\_CTX\_new();

BIGNUM\* p = BN\_new();

BIGNUM\* g = BN\_new();

// Генерация простого числа p

BN\_generate\_prime\_ex(p, 32, 1, NULL, NULL, NULL); // Меняйте длину при необходимости

BN\_set\_word(g, 2); // Установка генератора g

BIGNUM\* x = BN\_new(); // Секретный ключ

BIGNUM\* y = BN\_new(); // Открытый ключ

generateKeys(p, g, x, y, ctx);

// Сообщение для подписи

const char\* message = "Hello, world!ввввввввввв";

unsigned char M\_hash[SHA256\_DIGEST\_LENGTH];

hashMessage(reinterpret\_cast<const unsigned char\*>(message), strlen(message), M\_hash);

cout << "Хэш сообщения M: ";

for (int i = 0; i < SHA256\_DIGEST\_LENGTH; i++)

cout << hex << static\_cast<int>(M\_hash[i]);

cout << endl;

BIGNUM\* a = BN\_new(); // Первая часть подписи

BIGNUM\* b = BN\_new(); // Вторая часть подписи

signMessage(p, g, x, M\_hash, a, b, ctx);

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание // Проверка подписи

if (verifySignature(p, g, y, M\_hash, a, b, ctx)) {

cout << "Подпись верна!" << endl;

}

else {

cout << "Подпись не прошла проверку!" << endl;

}

BN\_free(p);

BN\_free(g);

BN\_free(x);

BN\_free(y);

BN\_free(a);

BN\_free(b);

BN\_CTX\_free(ctx);

return 0;

}

**Вывод:** создал программу, которая реализует алгоритм криптографической системы Рабина.