ДОДАТОК А «ЛІСТИНГ ПРОГРАМИ»

```
#include "cryptlib.h"
#include "integer.h"
#include "nbtheory.h"
#include "osrng.h"
#include "rsa.h"
#include "sha.h"
#include <iostream>
#include <stdexcept>

int main(int argc, char* argv[])
{
    using namespace CryptoPP;

    // Bob artificially small key pair
    AutoSeededRandomPool prng;
    RSA::PrivateKey privKey;

    privKey.GenerateRandomWithKeySize(prng, 64);
    RSA::PublicKey pubKey(privKey);

    /// Integer description
    /// Multiple precision integer with arithmetic operations
    /// The Integer class can represent positive and negative integers
    /// with absolute value less than (256**sizeof(word))*(256**sizeof(int)).
    /// Internally, the library uses a sign magnitude representation, and the class
    ///  has two data members. The first is a IntegerSecBlock (a SecBlock<word>) and it
is
    ///  used to hold the representation. The second is a Sign (an enumeration), and it
is
    ///  used to track the sign of the Integer.
    /// For details on how the Integer class initializes its function pointers using
    // Convenience
    const Integer& n = pubKey.GetModulus();
    const Integer& e = pubKey.GetPublicExponent();
    const Integer& d = privKey.GetPrivateExponent();

    // Print params
    std::cout << "Pub mod: " << std::hex << n << std::endl;
    std::cout << "Pub exp: " << std::hex << e << std::endl;
    std::cout << "Priv mod: " << std::hex << privKey.GetModulus() << std::endl;
    std::cout << "Priv exp: " << std::hex << d << std::endl;

    // For sizing the hashed message buffer. This should be SHA256 size.
    // Bytes number for the SHA256
    const size_t SIG_SIZE = UnsignedMin(SHA256::BLOCKSIZE, n.ByteCount());

    // Scratch
    SecByteBlock buff1, buff2, buff3;

    // Alice original message to be signed by Bob
    SecByteBlock orig((const byte*)"secret", 6);

    // Bob's message
    Integer m(orig.data(), orig.size());
    std::cout << "Message: " << std::hex << m << std::endl;

    // Hash message
    buff1.resize(SIG_SIZE);
```

```
    SHA256 hash1;

       /// Updates the hash with additional input and computes the hash of the current
message
       /// digest a pointer to the buffer to receive the hash
       /// digestSize the length of the truncated hash, in bytes
       /// input the additional input as a buffer
       /// length the size of the buffer, in bytes
       /// Use this if your input is in one piece and you don't want to call Update()
       ///  and CalculateDigest() separately.
       /// CalculateTruncatedDigest() restarts the hash for the next message.
       /// digestSize <= DigestSize() or digestSize <= HASH::DIGESTSIZE ensures
       ///  the output byte buffer is a valid size.
    hash1.CalculateTruncatedDigest(buff1, buff1.size(), orig, orig.size());

    // H(m) as Integer
    Integer hm(buff1.data(), buff1.size());
    std::cout << "H(m): " << std::hex << hm << std::endl;

    // Alice blinding
    Integer r;
    do {
        /// Integer::One
        /// Integer representing 1
              /// an Integer representing 1
              /// One() avoids calling constructors for frequently used integers
        /// prng - used for generation random numbers
        /// Integer::One(), - minimal bound for randomization
        /// n - Integer::One() - maximal bound for randomization
        r.Randomize(prng, Integer::One(), n - Integer::One());

    /// Determine relative primality
    /// a the first term
    /// b the second term
    /// true if a and b are relatively prime, false otherwise.
    }
    while (!RelativelyPrime(r, n));

    // Blinding factor
    /// Modular exponentiation
       /// x a reference to the base
       /// e a reference to the exponent
       /// m a reference to the modulus
       /// an Integer (a ^ b) % m.
    Integer b = a_exp_b_mod_c(r, e, n);
    std::cout << "Random: " << std::hex << b << std::endl;

    // Alice blinded message
    /// Modular multiplication
       /// x a reference to the first term
       /// y a reference to the second term
       /// m a reference to the modulus
       /// an Integer (a * b) % m.
    Integer mm = a_times_b_mod_c(hm, b, n);
    std::cout << "Blind msg: " << std::hex << mm << std::endl;

    // Bob sign
    /// Calculates the inverse of an element
       /// rng a RandomNumberGenerator derived class
       /// x the element
       /// the inverse of the element in the group
    Integer ss = privKey.CalculateInverse(prng, mm);
    std::cout << "Blind sign: " << ss << std::endl;
```

```
    // Alice checks s(s'(x)) = x
    /// Applies the trapdoor
       /// x the message on which the encryption function is applied
       /// the message x encrypted under the public key
       /// ApplyFunction is a generalization of encryption under a public key
       ///   cryptosystem. Derived classes must implement it.
    Integer ck = pubKey.ApplyFunction(ss);
    std::cout << "Check sign: " << ck << std::endl;
    if (ck != mm)
        throw std::runtime_error("Alice cross-check failed");


    // Alice remove blinding
    //InverseMod
    /// Calculate multiplicative inverse
       /// n a reference to the modulus
       /// an Integer *this % n.
       /// InverseMod returns the multiplicative inverse of the Integer *this
       ///  modulo the Integer n. If no Integer exists then Integer 0 is returned.
       /// a_times_b_mod_c() and a_exp_b_mod_c()

    /// Modular multiplication
       /// x a reference to the first term
       /// y a reference to the second term
       /// m a reference to the modulus
       /// an Integer (a * b) % m
    Integer s = a_times_b_mod_c(ss, r.InverseMod(n), n);
    std::cout << "Unblind sign: " << std::hex << s << std::endl;


    /// Applies the trapdoor
       /// x the message on which the encryption function is applied
       /// the message x encrypted under the public key
       /// ApplyFunction is a generalization of encryption under a public key
       ///   cryptosystem. Derived classes must implement it.
    Integer v = pubKey.ApplyFunction(s);
    std::cout << "Verify: " << std::hex << v << std::endl;

    // Convert to a string
    /// Minimum number of bytes to encode this integer
    /// sign enumeration indicating Signedness
    /// The MinEncodedSize() of 0 is 1.
    size_t req = v.MinEncodedSize();
    buff2.resize(req);

    /// Encode in big-endian format
    /// output big-endian byte array
    /// outputLen length of the byte array
    /// sign enumeration indicating Signedness
    /// Unsigned means encode absolute value, signed means encode two's complement if
negative.
    /// outputLen can be used to ensure an Integer is encoded to an exact size (rather
than a
    /// minimum size). An exact size is useful, for example, when encoding to a field
element size.
    v.Encode(&buff2[0], buff2.size());

    // Hash message
    buff3.resize(SIG_SIZE);
    SHA256 hash2;

    /// Updates the hash with additional input and computes the hash of the current
message
       /// digest a pointer to the buffer to receive the hash
       /// digestSize the length of the truncated hash, in bytes
```

```
      /// input the additional input as a buffer
      /// length the size of the buffer, in bytes
      /// Use this if your input is in one piece and you don't want to call Update()
      ///  and CalculateDigest() separately.
      /// CalculateTruncatedDigest() restarts the hash for the next message.
      /// digestSize <= DigestSize() or digestSize <= HASH::DIGESTSIZE ensures
      ///  the output byte buffer is a valid size.
    hash2.CalculateTruncatedDigest(buff3, buff3.size(), orig, orig.size());

    // Compare
    bool equal = buff2.size() == buff3.size() && VerifyBufsEqual(buff2.data(),
buff3.data(), buff3.size());

    if (!equal)
        throw std::runtime_error("Verified failed");

    std::cout << "Verified signature" << std::endl;

    return 0;
}
```

ДОДАТОК Б «МАТЕРІАЛИ ДЛЯ НАУКОВОЇ КОНФЕРЕНЦІЇ»

Назва конференції: «І Всеукраїнська студентська наукова конференція «НАУКОВИЙ ПРОСТІР: АНАЛІЗ, СУЧАСНИЙ СТАН, ТРЕНДИ ТА ПЕРСПЕКТИВИ»»

Тема доповіді: «Сучасні механізми захисту конфіденційності інформації в децентралізованих системах»

Текст доповіді: «Сьогодні використання аналітичних можливостей дозволяє компаніям не лише враховувати звички користувачів, а й впливати на їхню поведінку. Все це стало можливим завдяки обробці даних про покупки та інші дії користувачів у мережі. Ці дані приносять компаніям великі доходи. Більше того, якщо проаналізувати отриману компаніями інформацію, то більшість з неї можливо віднести до конфіденційної. Тому питання приватності у цифровому світі займає одне з перших місць серед тих, на які слід звертати увагу. А враховуючи розвиток технологій та аналітичних інструментів, це питання стає дедалі актуальнішим.

При вивченні децентралізованих систем на рівні протоколу відразу стає зрозуміло, що вони більше орієнтовані на конфіденційність, ніж традиційні цифрові платіжні системи. Рівень конфіденційності, який забезпечується, сильно змінюється залежно від вибору користувачем криптовалюти та використання допоміжних технологій.

Можна виділити декілька основних технологічних аспектів щодо забезпечення конфіденційності користувачів, які впроваджені в сучасних децентралізованих системах [1]:

– підходи, засновані на комбінуванні даних. Дані підходи використовують вхідні та вихідні дані про різні операції з подальшим їх об'єднанням в єдину велику операцію для приховування зв'язків між адресами відправників та одержувачів. Такі підходи до підвищення конфіденційності реалізовано в протоколах: CoinJoin, Mimblewimble та Monero.

– конфіденційність, заснована на нульовому розголошенні. Це технологія, в якій користувачі надають докази з нульовим розголошенням – дані, які свідчать про знання якоїсь інформації без розкриття самої інформації. При правильному використанні дана криптографічна технологія здатна забезпечувати конфіденційність транзакцій та станів мережі. Прикладом реалізації є методи Blind Signature, zk-SNARK.

– протоколи другого рівня, такі як Lightning Network, State Channels або Plasma, що функціонують поверх базового рівня криптовалюти, дозволяють невеликим групам користувачів здійснювати між собою операції поза блокчейном. Це означає, що всі проміжні стани блокчейну зберігаються такими користувачами, а в основний блокчейн вносяться лише періодичні записи про зміну стану мережі. В результаті проміжні стани є невидимими для зовнішніх спостерігачів, оскільки вони взагалі ніколи не з'являються в основному блокчейні.

Також не можна залишати без уваги використання допоміжних технологій з метою забезпечення додаткового рівня захисту конфіденційності даних користувачів децентралізованих систем [2]. До таких технологій належать:

– досвід користувачів. Навіть при використанні криптовалют без будь-яких додаткових функцій конфіденційності користувачі можуть мінімізувати деякі з наслідків аналізу мережі та блокчейну. Для запобігання застосуванню метаданих мережі для деанонімізації можна приховувати місце проведення операцій за допомогою таких технологій як Tor або I2P. Для запобігання аналізу блокчейну рекомендується використовувати нову адресу для кожного нового платежу. Такі криптовалюти, як Monero і Verge, пропонують цю функцію за замовчуванням.

– довірче середовище виконання (Trusted Execution Environment, TEE) – це процесор (як, наприклад, Intel SGX), який претендує на забезпечення криптографічного захисту цілісності та конфіденційності даних та коду в них. Деякі протоколи, зокрема Ekiden, пропонують використовувати TEE.

Наприклад, дані про залишки на рахунках можуть бути зашифровані приватним ключем, який зберігається в TEE, і таким чином можна розшифрувати або змінити тільки в TEE. В даному випадку гарантії конфіденційності перекладаються на TEE.

У висновку можна сказати, що технологія блокчейн широко використовується в різних областях завдяки таким властивостям як децентралізація, незмінність даних та надійність. Однак через прозорість та децентралізацію виникає нова проблема – захист конфіденційності користувачів, що робить збереження конфіденційності в блокчейні важливою темою для подальших досліджень.»

Список використаних джерел:

1. Збереження конфіденційності в блокчейні. [Електронний ресурс] – режим доступу до ресурсу https://www.sciencedirect.com/science/article/pii/S2352864819303827.

2. Аналіз та оцінка політики конфіденційності. Анонімні криптовалюти на основі блокчейну. [Електронний ресурс] – режим доступу до ресурсу https://arxiv.org/pdf/2012.10563.pdf.

Рисунок – Сертифікат учасника конференції