Лабораторна робота № 3 Асиметричне шифрування. Алгоритм RSA

Група КН-М922в

Автор Савичев О.В.

Мета

Дослідити і реалізувати механізм асиметричного алгоритму шифрування RSA.

Завдання

Розробити додаток обміну таємними посиланнями між двома клієнтами за допомогою алгоритму шифрування RSA

- Реалізувати алгоритм генерації ключів (public / private keys) для алгоритму RSA. Створити ключі заданої довжини (напр. 1024 біт)
- Реалізувати та продемонструвати роботу алгоритму шифрування та дешифрування повідомлення RSA
- Підтвердити роботу реалізованого алгоритму шляхом порівняння результату кодування з існуючим алгоритмом (наприклад, використовуючи утиліту openssl або вбудовані системи шифрування обраної мови програмування)

Хід роботи

RSA — криптографічний алгоритм з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел.

RSA став першим алгоритмом такого типу, придатним і для шифрування, і для цифрового підпису. Алгоритм застосовується до великої кількості криптографічних застосунків.

Алгоритм RSA складається з 4 етапів: генерації ключів, шифрування, розшифрування та розповсюдження ключів.

Безпека алгоритму RSA побудована на принципі складності факторизації цілих чисел. Алгоритм використовує два ключі — відкритий (public) і секретний (private), разом відкритий і відповідний йому секретний ключі утворюють пари ключів (keypair). Відкритий ключ не потрібно зберігати в таємниці, він використовується для шифрування даних. Якщо повідомлення було зашифровано відкритим ключем, то розшифрувати його можна тільки відповідним секретним ключем.

Метод генерації ключів

```
public void GenerateKeys(out BigInteger N, out BigInteger publicE, out BigInteger privateD)
{
    // 1. Вибираються два великі прості числа p і q 512 біт завдовжки кожне
    BigInteger P;
    BigInteger Q;
    const int bits = 1024;
    const int bytes = bits / 8;
    do
    {
        P = Helper.Instance.GetPrime(bytes / 2);
        Q = Helper.Instance.GetPrime(bytes / 2);

        N = P * Q; // Обчислюється їх добуток N = P*Q - модуль
    } while (P == Q || N.ToByteArray().Length != bytes);
    Console.WriteLine($"P = {P}");
    Console.WriteLine($"Q = {Q}");
    Console.WriteLine($"N = {N}");

    // Обчислюється функція Ейлера phi(n) = (P-1)(Q-1)
    BigInteger PHI = (P - 1) * (Q - 1);

    // Вибирається ціле число E, таке, що 1 < E <PHI(n), E взаємно просте з phi(n)
    // Для підвищення швидкості шифрування відкритий показник E, вибирається невеликим,
    // звичайно 3, 17, 257 або 65537
    publicE = 65537;
    Console.WriteLine($"E = {publicE}");

    // За допомогою розширеного алгоритму Евкліда знаходиться число d, таке, що ed = 1 mod phi(n)
    var gcd = Helper.Instance.ExtendedEuclidean(publicE, PHI);
    var r = gcd.Item2 % PHI;
    privateD = r < 0 ? r + PHI : r;
    Console.WriteLine($"D = {privateD}");
}
```

Метод шифрування

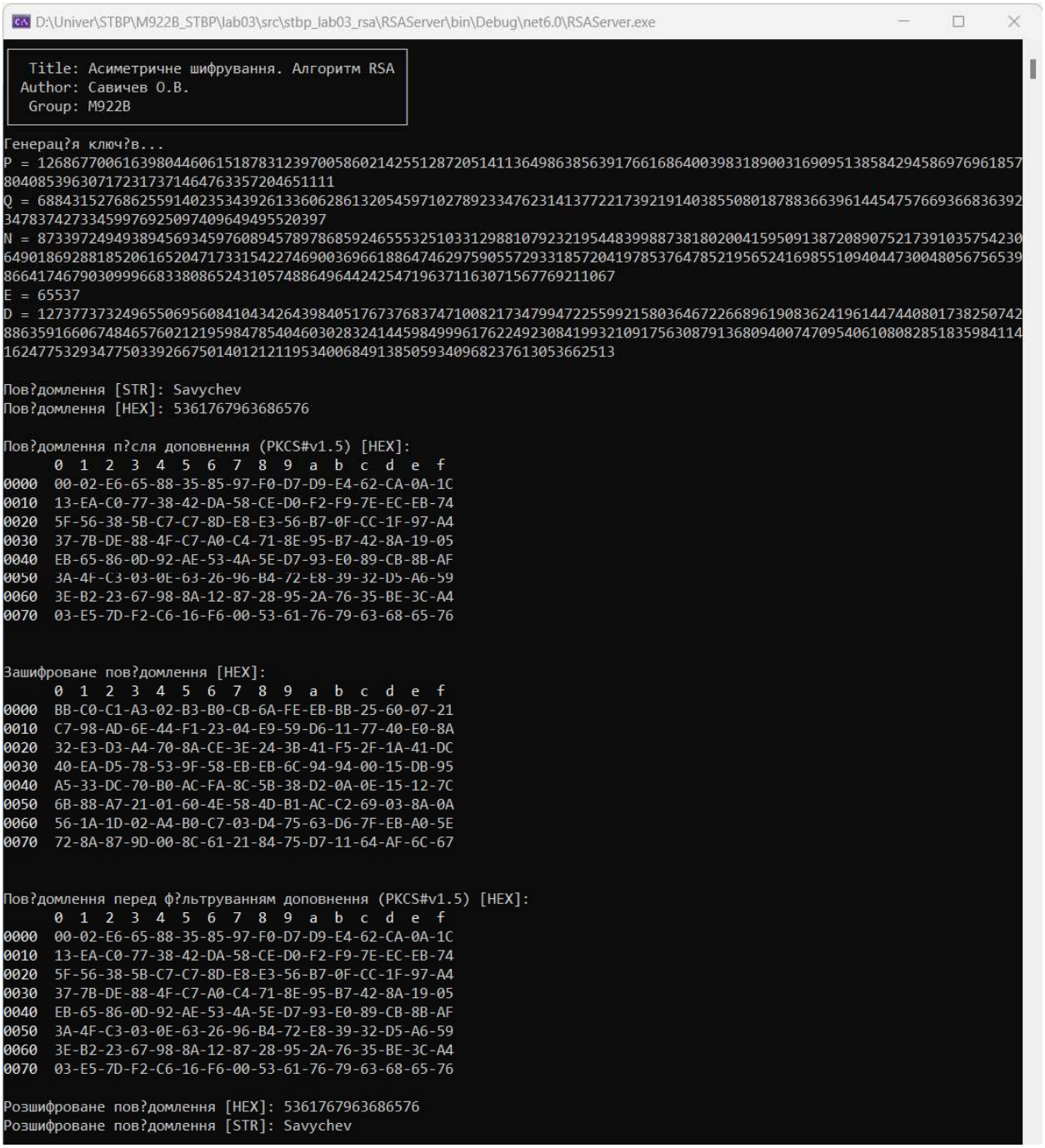
```
public byte[] Encode(byte[] data, BigInteger publicE, BigInteger N)
{
    var blockSize = N.ToByteArray().Length;
    var msgAsBytes = _PaddingBlock(data, blockSize);
    var result = new byte[msgAsBytes.Length];
    var blocks = msgAsBytes.Length / blockSize;
    for (int i = 0; i < blocks; i++)
    {
        byte[] chunk = new byte[blockSize];
        Buffer.BlockCopy(msgAsBytes, i * blockSize, chunk, 0, blockSize);

        BigInteger chunkAsBigInteger = new BigInteger(chunk, true);
        var encodedChunk = BigInteger.ModPow(chunkAsBigInteger, publicE, N);
        var encodedBytes = encodedChunk.ToByteArray();
        Buffer.BlockCopy(encodedBytes, 0, result, i * blockSize, blockSize);
    }
    return result;
}
```

Метод дешифрування

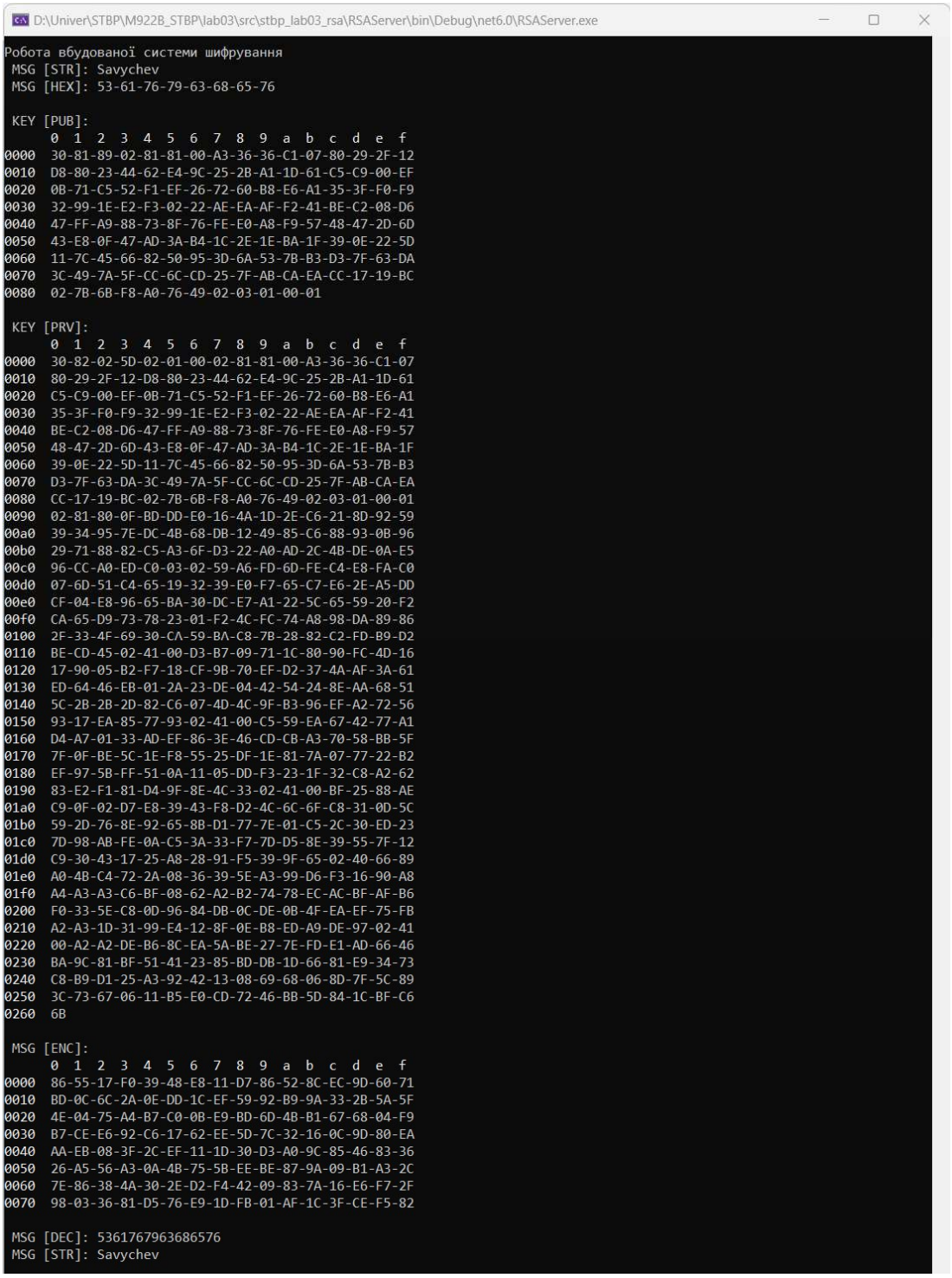
```
public byte[] Decode(byte[] message, BigInteger privateD, BigInteger N)
{
    var blockSize = N.ToByteArray().Length;
    var decoded = new byte[message.Length];
    var blocks = message.Length / blockSize;
    for (int i = 0; i < blocks; i++)
    {
        byte[] chunk = new byte[blockSize];
        Buffer.BlockCopy(message, 0, chunk, i * blockSize, blockSize);

        BigInteger chunkAsBigInteger = new BigInteger(chunk, true);
        var decodedChunk = BigInteger.ModPow(chunkAsBigInteger, privateD, N);
        var decodedBytes = decodedChunk.ToByteArray();
        Buffer.BlockCopy(decodedBytes, 0, decoded, i * blockSize, blockSize);
    }

    if (decoded[0] == 0x00 && decoded[1] == 0x02) // Чи присутня сигнатура росширення
    {
        for (int i = 2; i < decoded.Length; i++)
        {
            if (decoded[i] == 0x00) // Пошук початку повідомлення
            {
                var size = decoded.Length - i - 1;
                var result = new byte[size];
                Buffer.BlockCopy(decoded, i + 1, result, 0, size);
                return result;
            }
        }
        return decoded;
    }
    else
    {
        return decoded;
    }
}
```

Результати роботи

Результати роботи моєї реалізації алгоритму RSA



Результати роботи вбудованох системи шифрування алгоритму RSA



Висновок

Дослідив і реалізував механізм асиметричного алгоритму шифрування RSA.