МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи №4
З дисципліни: *“Безпека програм та даних”*
На тему: *“Створення програмної реалізації алгоритму шифрування з відкритим ключем rsa з використанням microsoft cryptoapi”*

Лектор:
доцент каф. ПЗ
Сенів М. М.

Виконав:
ст. гр. ПЗ-43
Лесневич Є. Є.

Прийняв:
ст. викладач каф. ПЗ
Угриновський Б. В.

«_____»_____2024 р.

∑=_____ .....

Львів – 2024

Тема роботи: створення програмної реалізації алгоритму шифрування з відкритим ключем rsa з використанням microsoft cryptoapi.
Мета роботи: ознайомитись з методами і засобами криптографії з відкритим ключем, навчитись створювати програмні засоби з використанням криптографічних інтерфейсів.

Теоретичні відомості

Концепція криптографії з відкритим ключем була запропонована Уітфілдом Діффі (Whitfield Diffie) та Мартіном Хеллманом (Martin Hellman), і, незалежно, Ральфом Мерклом (Ralph Merkle). Основна ідея – використовувати ключі парами, що складаються з ключа шифрування та ключа дешифрування, які неможливо обчислити один з одного. Перша праця, присвячені цій проблемі вийшла у 1976 році, і з того часу було створено багато алгоритмів, що використовують концепцію відкритих ключів. Загальна схема виглядає наступним чином:

1. Кожен користувач генерує пару ключів: один для шифрування і один для дешифрування.

2. Кожен користувач публікує свій ключ шифрування, розміщує його у відкритому для всіх доступі. Другий ключ, парний до відкритого, зберігається в таємниці.

3. Якщо користувач А збирається надіслати повідомлення користувачеві В, він шифрує повідомлення відкритим ключем користувача В.

4. Коли користувач В отримує повідомлення, він дешифрує його за допомогою свого приватного ключа. Інший користувач не зможе дешифрувати повідомлення, оскільки приватний ключ В відомий тільки користувачеві В. Алгоритми шифрування з відкритим ключем розроблялися для того, щоб вирішити дві найбільш важкі задачі, що виникли при використанні симетричного шифрування. Першою задачею є розподіл ключа. При симетричному шифруванні потрібно, щоб обидві сторони вже мали спільний ключ, що якимось чином повинний бути їм заздалегідь переданий. Ця вимога заперечує всю суть криптографії, а саме можливість підтримувати загальну таємність при комунікаціях.

Другою задачею є необхідність створення таких механізмів, при використанні яких неможливо було б підмінити кого-небудь з учасників, тобто потрібен цифровий підпис.

Криптографія з відкритим ключем дозволяє вирішити набагато ширше коло задач, ніж криптографія класична. Однак існує ряд причин, з яких асиметричні алгоритми шифрування не можуть повноцінно замінити симетричні алгоритми

- По-перше, алгоритми з таємним ключем набагато простіше реалізовуються як програмно, так і апаратно. Через це за однакових характеристик продуктивності та стійкості складність, а значить і ціна апаратних засобів, що реалізують шифр з відкритим ключем помітно вища за ціну апаратури, що реалізує класичний

шифр, а при програмній реалізації на одному й тому ж типі процесора симетричні шифри працюють швидше асиметричних.

- По-друге, надійність алгоритмів з відкритим ключем на даний час обґрунтована набагато гірше, ніж надійність алгоритмів з таємним ключем і немає гарантії, що через деякий час вони не будуть розкриті, як це сталося з криптосистемою, заснованою на задачі про вкладання ранця.

Завдання до виконання роботи

З використання функцій CryptoAPI створити програмну реалізацію алгоритму шифрування RSA. Оцінити швидкість шифрування алгоритму RSA та порівняти її зі швидкістю шифрування алгоритму RC5, реалізованого в роботі № 3, зробити відповідні висновки та відобразити їх у звіті до лабораторної роботи

Код аглоритму

```
using Lab01GUI.Services.Interfaces;
using System.Net;
using System.Security.Cryptography;
using System.Text;
using System.Xml;

namespace Lab01GUI.Services.Implementation;

public class RSAEncryptionService : IRSAEncryptionService
{
    private int KeySize = 2048;

          public void SetKeySize(int keySize)
           => KeySize = keySize;

          public int GetKeySize()
                    => KeySize;

    public RSAKeyResult GenerateKeys()
          {
                    using var rsa = new RSACryptoServiceProvider(KeySize);

                    try
                    {
                              string publicKey = rsa.ToXmlString(false); // false means public key only
                              string privateKey = rsa.ToXmlString(true); // true means public and private key

                              return new RSAKeyResult(publicKey, privateKey);
                    }
                    finally
                    {
                              rsa.PersistKeyInCsp = false; // Avoid saving the keys in a key container
                    }
          }

          public byte[] Encrypt(string plainText, string publicKey)
          {
                    using var rsa = new RSACryptoServiceProvider();

                    try
                    {
                              publicKey = WebUtility.HtmlDecode(publicKey);
                              rsa.FromXmlString(publicKey);
                              var byteData = Encoding.UTF8.GetBytes(plainText);
                              var result = rsa.Encrypt(byteData, false);
                              return result; // OAEP (Optimal asymmetric encryption padding) padding - will use PKCS (Public Key Cryptography
Standards) #1 v1.5 padding if false
                    }
                    finally
                    {
                              rsa.PersistKeyInCsp = false;
                    }
          }

          public string Decrypt(string cipherText, string privateKey)
                    => Decrypt(Encoding.UTF8.GetBytes(cipherText), privateKey);

          public string Decrypt(byte[] cipherText, string privateKey)
          {
                    using var rsa = new RSACryptoServiceProvider();

                    try
                    {
                              privateKey = WebUtility.HtmlDecode(privateKey);
                              rsa.FromXmlString(privateKey);
                              var decryptedBytes = rsa.Decrypt(cipherText, false); // false for OAEP padding

                              return Encoding.UTF8.GetString(decryptedBytes);
```

```
                }
                finally
                {
                        rsa.PersistKeyInCsp = false;
                }
        }
}
public record RSAKeyResult(string PublicKey, string PrivateKey);
```

Результати роботи

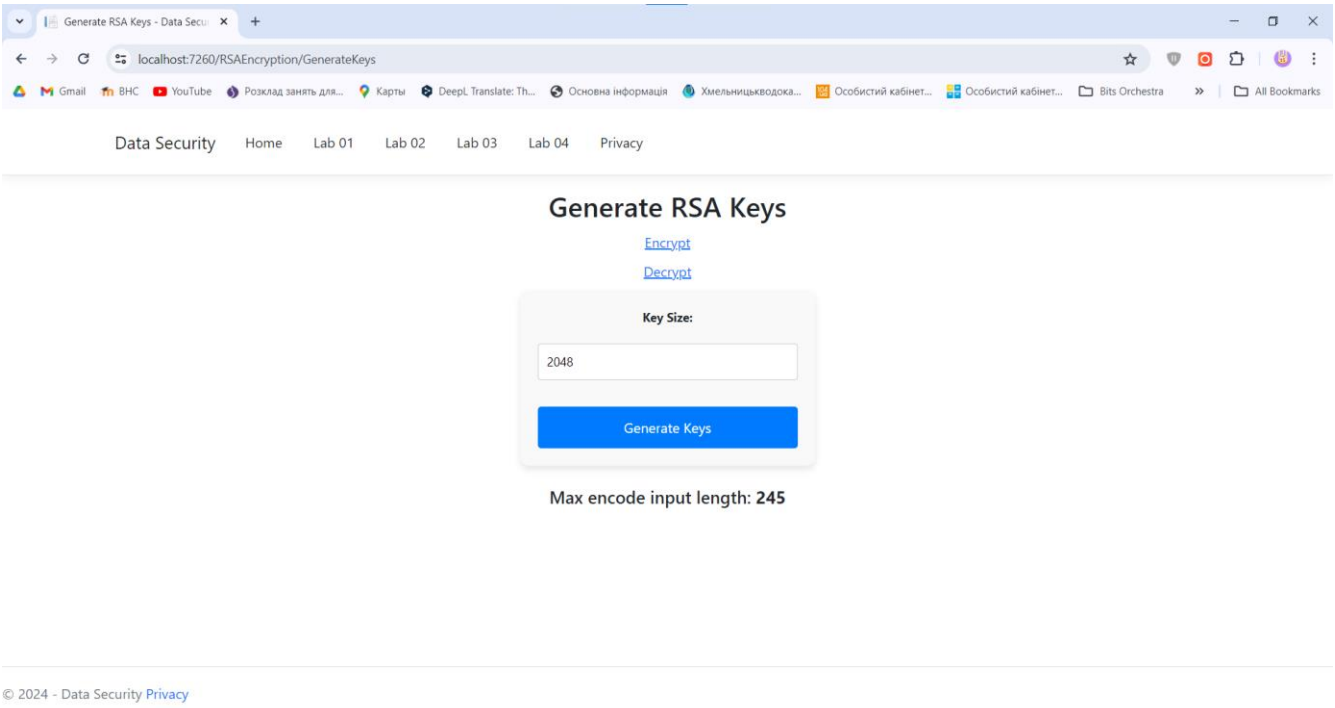

Рис. 1 Головне вікно програми

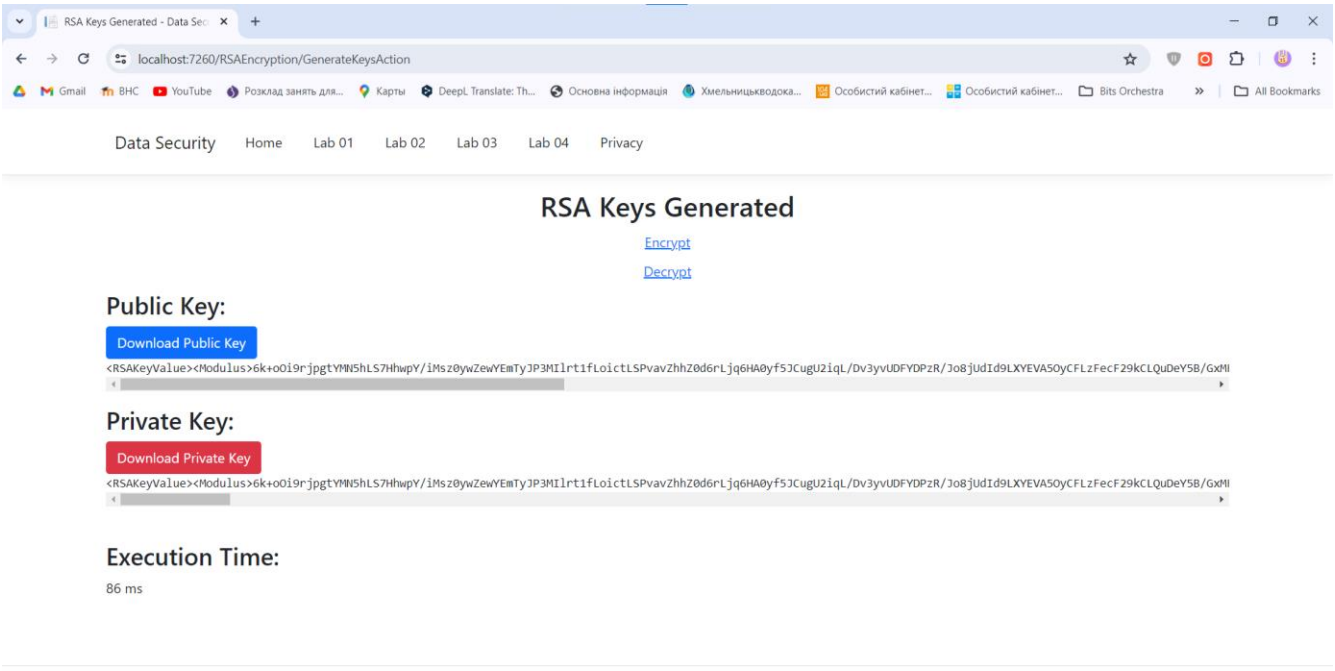

Рис. 2 Вхідні дані для шифрування

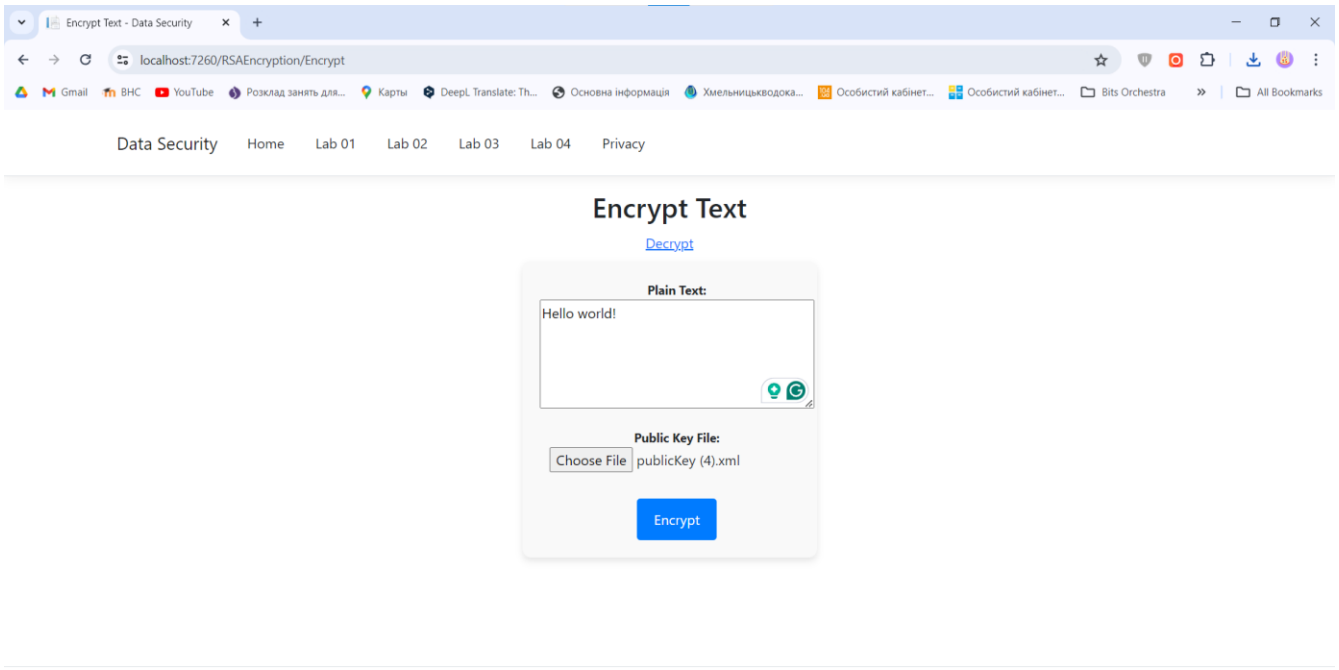

Рис. 3 Шифрування повідомлення відкритим ключем

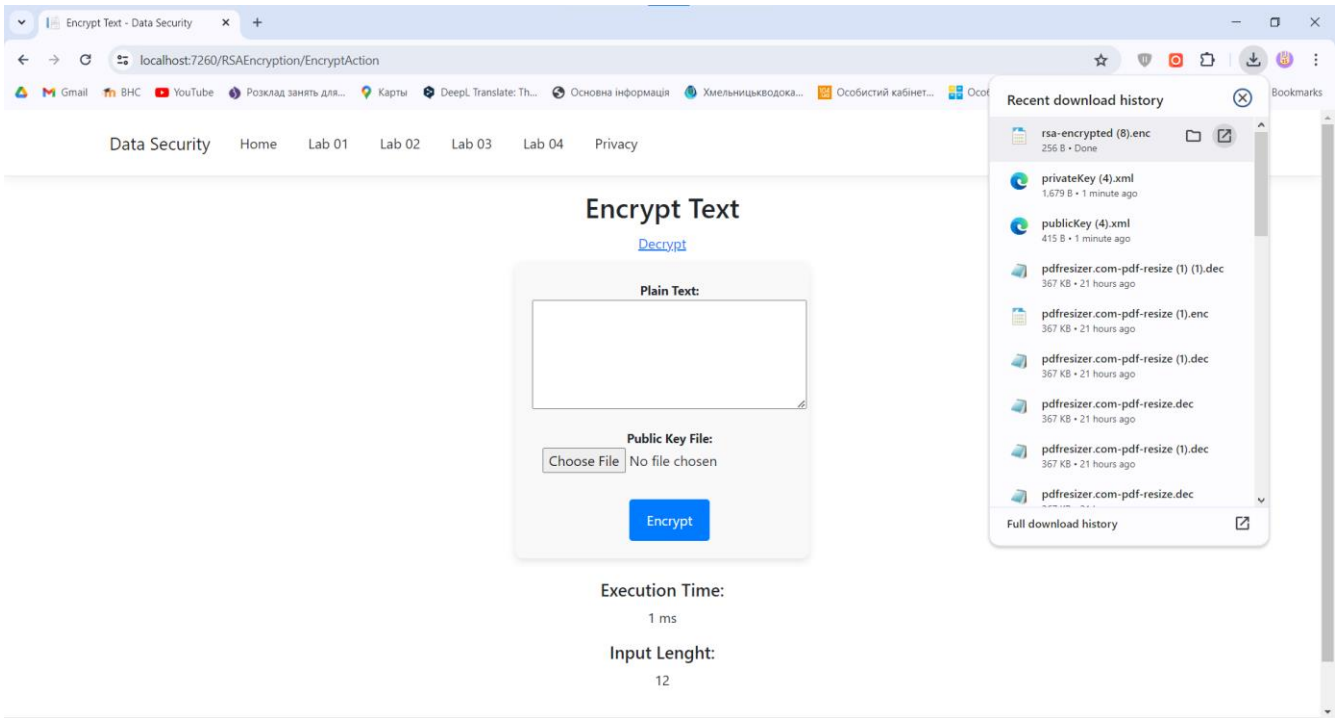

Рис. 4 Результат шифрування

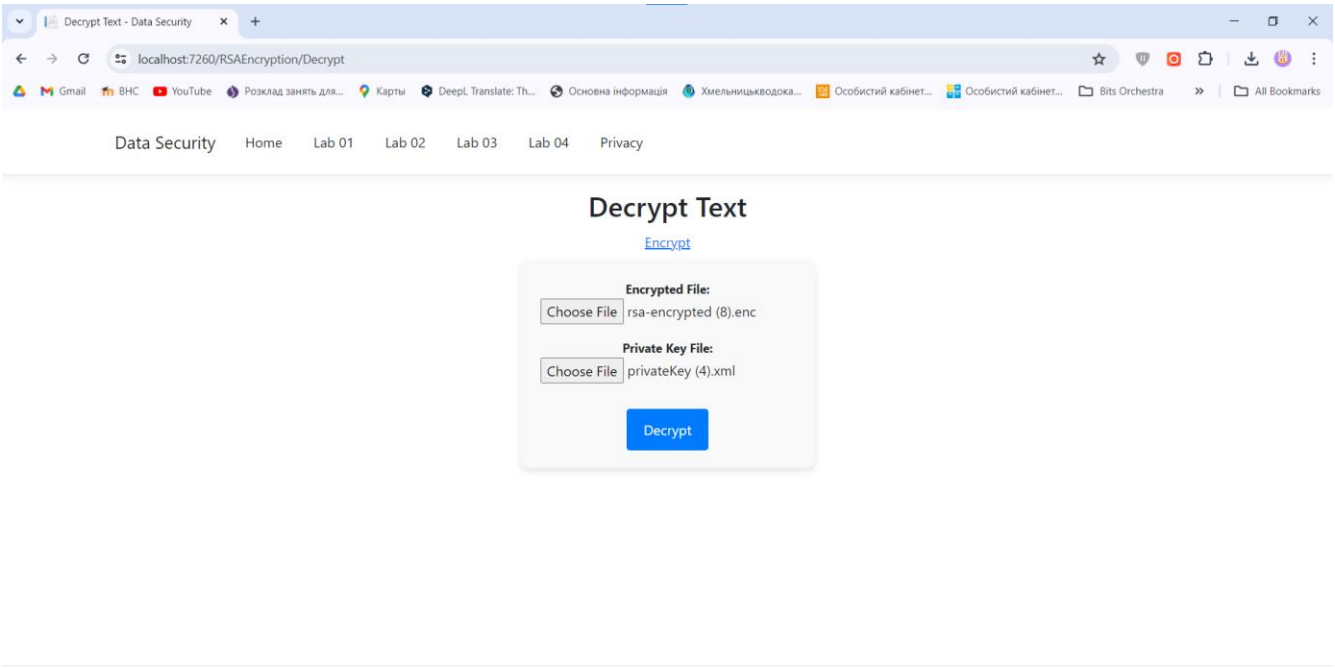

Рис. 5 Дешифрування повідомлення закритим ключем

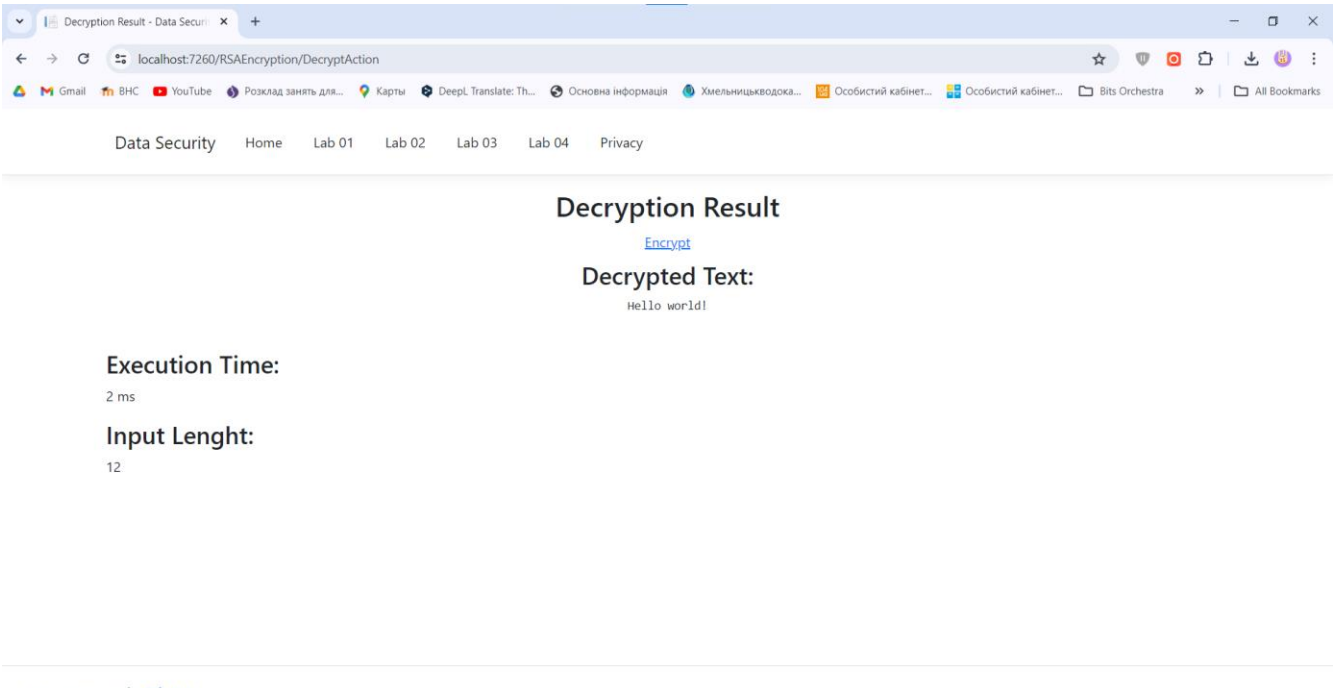

Рис. 6 Отримане дешифроване повідомлення

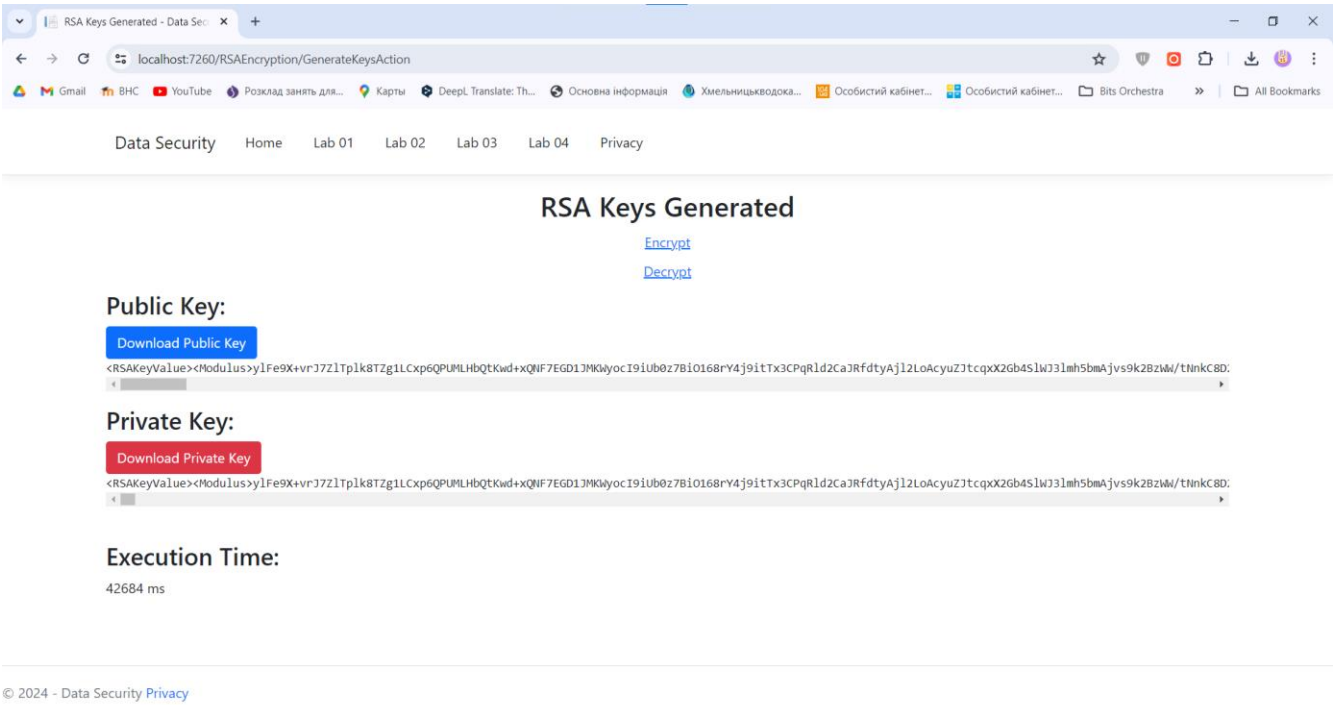

Рис. 7 Генерація ключів довжиною 16384 біт

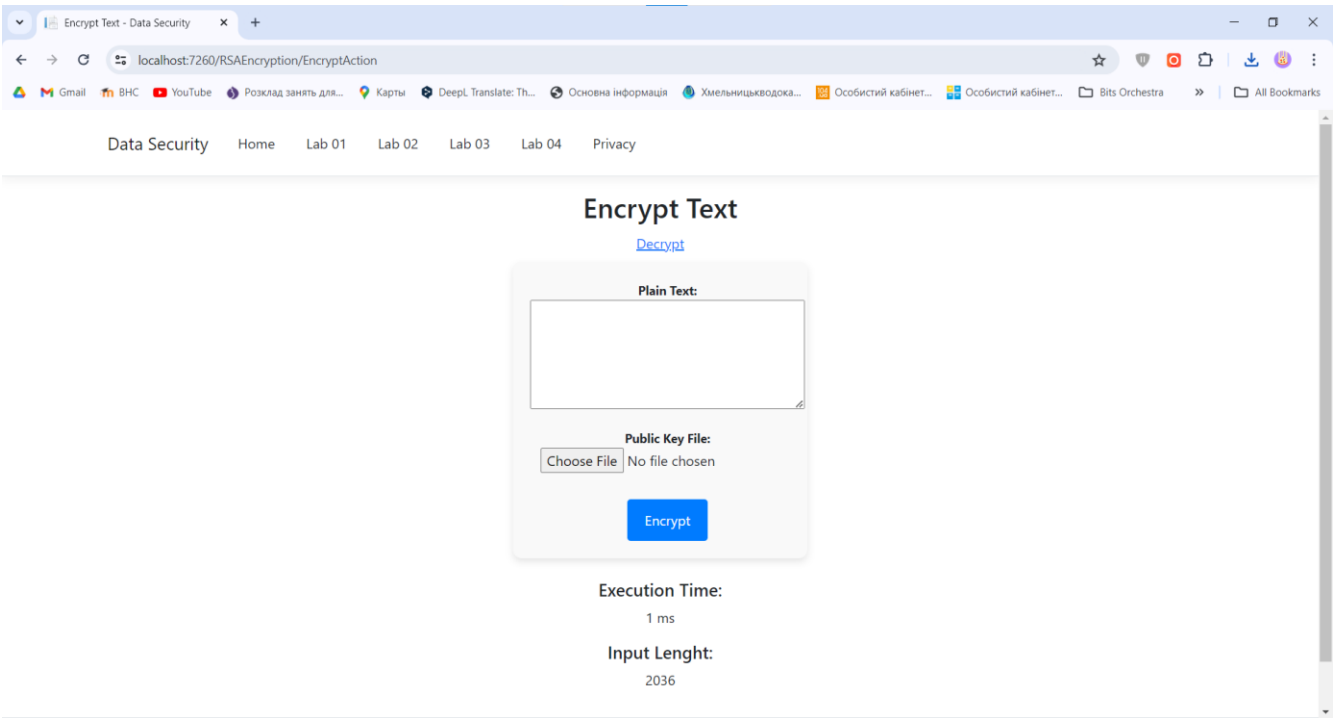

Рис. 8 Шифрування повідомлення довжиною 2036 біт

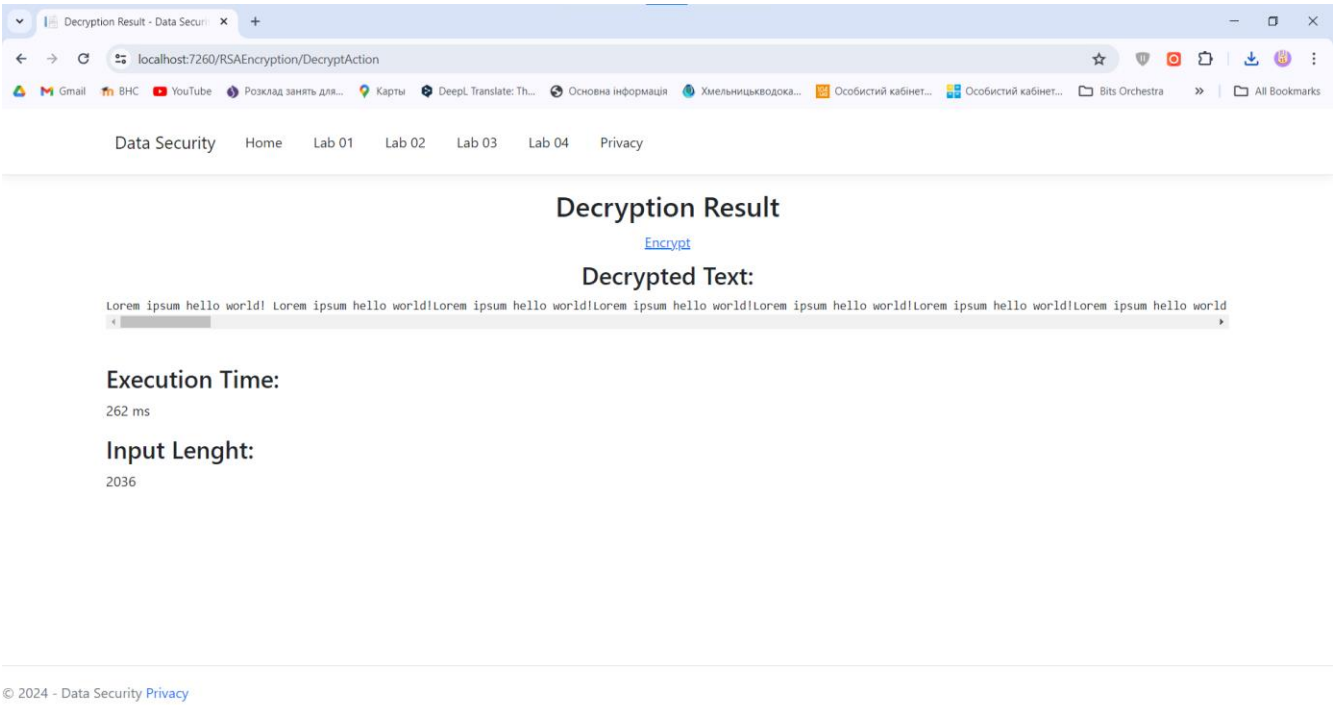

Рис. 9 Дешифрування повідомлення довжиною 2036 біт

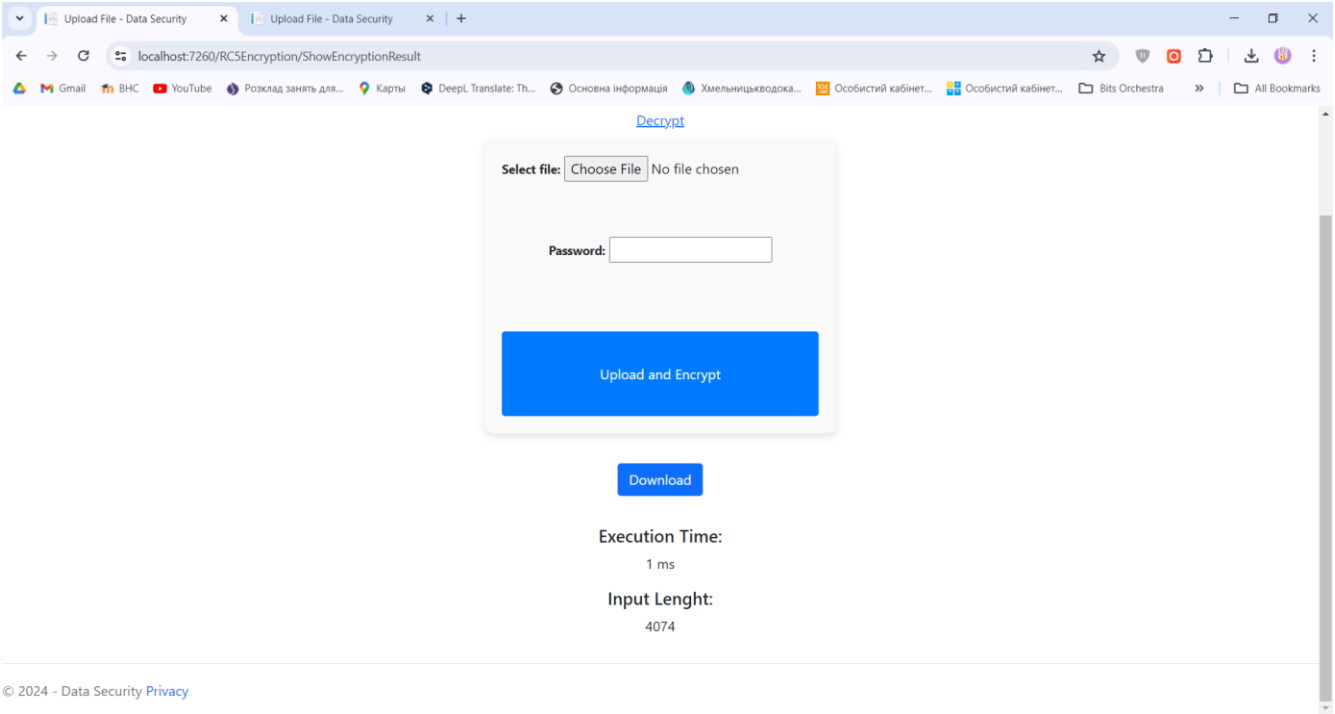

Рис. 10 Шифрування повідомлення довжиною 2036 біт ключем довжиною 16384 біт алгоритмом RC5-CBC Pad

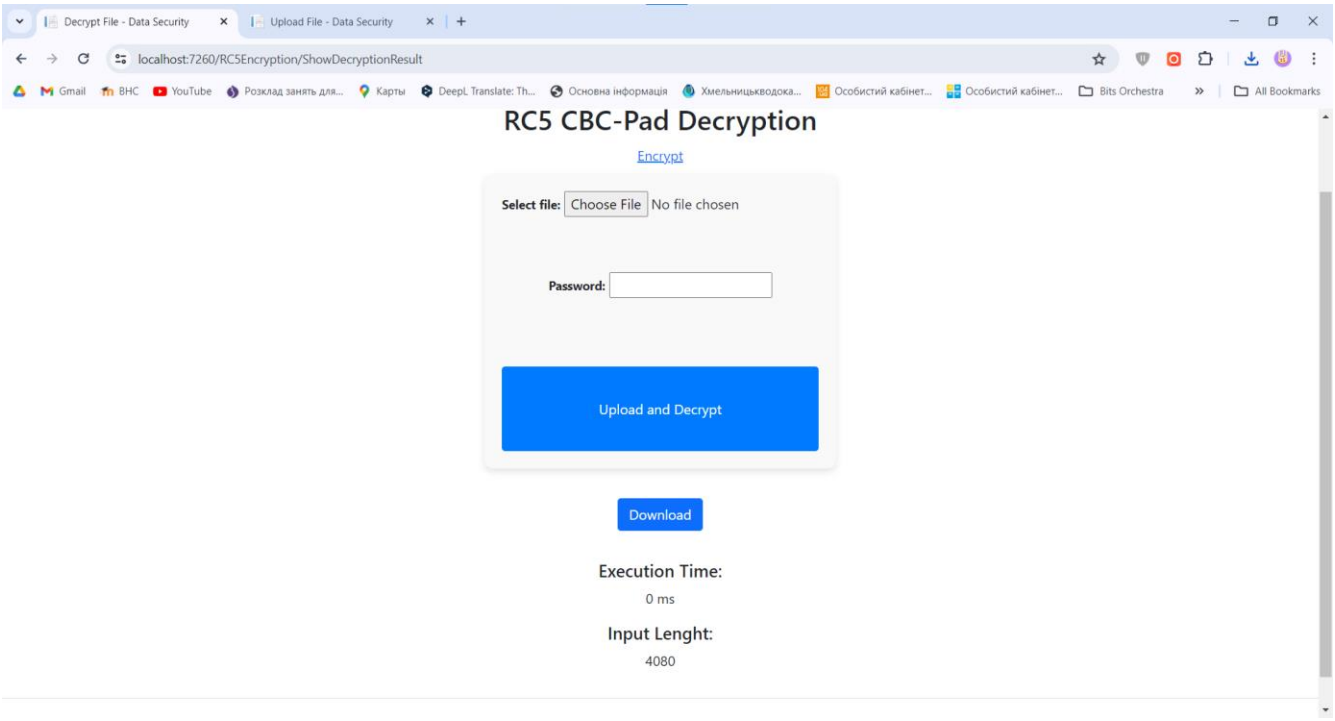

Рис. 11 Дешифрування повідомлення довжиною 2036 біт ключем довжиною 16384 біт алгоритмом RC5-CBC Pad

Отже, проаналізувавши отримані результати, можна зробити висновок, що алгоритм асиметричного шифрування працює в декілька разів повільніше, ніж алгоритм симетричного шифрування.

Висновки

Отже, під час виконання даної лабораторної роботи я ознайомився з методами і засобами криптографії з відкритим ключем, а також навчився створювати програмні засоби з використанням криптографічних інтерфейсів.