МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Інститут **КНІТ**

Кафедра **ПЗ**



ЗВІТ

До лабораторної роботи №3

**З дисципліни:** *“Безпека програм та даних”*

**На тему:** *“Cтворення програмного засобу для забезпечення конфіденційності інформації”*

**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 програмного засобу для забезпечення конфіденційності інформації.

**Мета роботи**: ознайомитись з методами криптографічного забезпечення конфіденційності інформації, навчитись створювати комплексні програмні продукти для захисту інформації з використанням алгоритмів симетричного шифрування, хешування та генераторів псевдовипадкових чисел.

**Теоретичні відомості**

RC5 – це алгоритм симетричного шифрування, розроблений Роном Райвестом в середині 90-х років. При розробці RC5 ставилась задача досягнення наступних характеристик.

• Придатність для апаратної та програмної реалізації. В RC5 використовуються тільки елементарні обчислювальні операції, які зазвичай застосовуються в мікропроцесорах.

• Швидкість виконання. RC5 є простим алгоритмом, що працює з даними розміром в машинне слово. Усі основні операції передбачають також роботу з даними довжиною в слово.

• Адаптованість до процесорів з різною довжиною слова. Довжина слова в бітах є параметром RC5 – при зміні довжини слова змінюється сам алгоритм.

• Змінна кількість раундів. Кількість раундів є другим параметром RC5. Цей параметр дозволяє вибрати оптимальне співвідношення необхідної швидкості роботи і вимог до ступеня захисту.

• Змінна довжина ключа. Довжина ключа є третім параметром RC5. Як і в попередньому випадку, цей параметр дозволяє знайти прийнятний компроміс між швидкістю роботи та необхідним рівнем безпеки.

• Простота. Структура RC5 дуже проста не тільки для реалізації, але й для оцінки її криптоаналітичної стійкості.

• Низькі вимоги до пам'яті. Низькі вимоги до пам'яті роблять RC5 придатним для використання в смарт-картах та інших подібних пристроях з обмеженим об'ємом пам'яті.

• Високий ступінь захисту. RC5 покликаний забезпечити високий ступінь захисту за умови вибору відповідних значень параметрів. 17

• Залежність циклічних зсувів від даних. В RC5 використовуються циклічні зсуви, величина яких залежить від даних, що повинно підвищувати криптоаналітичну стійкість алгоритму.

Алгоритм RC5 вбудований в багатьох основних продуктах компанії RSA Data Security Inc., включаючи BSAFE, JSAFE та S/MAIL.

**Завдання до виконання роботи**

Згідно до варіанту, наведеного в таблиці, створити прикладну програму для шифрування інформації за алгоритмом RC5.

Програма повинна отримувати від користувача парольну фразу і, на її основі, шифрувати файли довільного розміру, а результат зберігати у вигляді файлу з можливістю подальшого дешифрування (при введенні тієї самої парольної фрази). Для перетворення парольної фрази у ключ шифрування використати алгоритм MD5, реалізований в лабораторній роботі № 2 – ключем шифрування повинен бути хеш парольної фрази. Якщо згідно варіанту довжина ключа становить 64 біти, беруться молодші 64 біти хешу; якщо довжина ключа повинна бути 256 бітів, то хеш парольної фрази стає старшими 128 бітами, а молодшими є хеш від старших 128 бітів (тобто, позначивши парольну фразу через P, отримаємо K=H(H(P))||H(P)). Для забезпечення можливості роботи створеного програмного продукту з відкритим текстом довільної довжини, програмну реалізацію здійснити в режимі RC5-CBC-Pad. В якості вектора ініціалізації (IV) використати генератор псевдовипадкових чисел, реалізований в лабораторній роботі № 1. Для кожного нового шифрованого повідомлення слід генерувати новий вектор ініціалізації. Вектор ініціалізації зашифровується в режимі ECB і зберігається в першому блоці зашифрованого файлу.

У звіті навести протокол роботи програми та зробити висновки про поєднання різних криптографічних примітивів для задач захисту інформації.

Індивідуальне завдання

Варіант 10

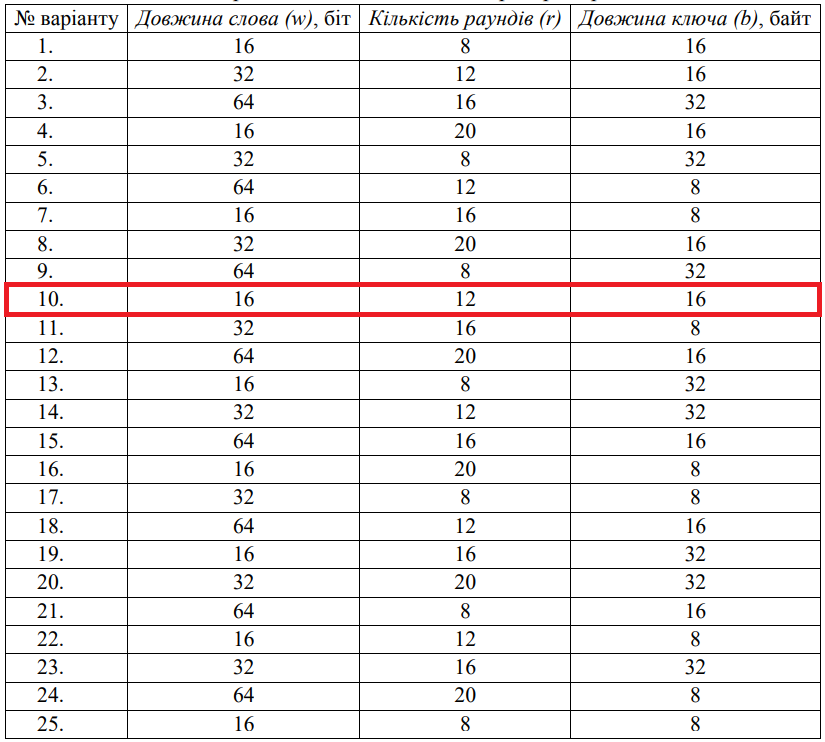


Рис. 1 Індивідуальне завдання

**Код аглоритму**

namespace Lab01GUI.Services.Implementation;

using Interfaces;

using System;

using System.Text;

public class RC5\_CBC\_PadService

{

private readonly IMD5Service \_md5;

private readonly IRandomNumberGeneratorService \_pseudoRandomGenerator;

private readonly int \_wordLengthInBytes;

private readonly int \_wordLengthInBits;

private readonly ulong \_wordBytesUsage;

private readonly ulong \_arrP;

private readonly ulong \_arrQ;

private readonly int \_numberOfRounds;

private readonly int \_secretKeyLengthInBytes;

private ulong[] \_s;

public RC5\_CBC\_PadService(WordLength wordLength, int numberOfRounds, int secretKeyLengthInBytes)

{

\_md5 = new MD5Service();

\_pseudoRandomGenerator = new RandomNumberGeneratorService();

\_wordLengthInBits = wordLength.Length;

\_wordLengthInBytes = wordLength.Length / 8;

\_wordBytesUsage = wordLength.BytesUsage;

\_arrP = wordLength.P;

\_arrQ = wordLength.Q;

\_numberOfRounds = numberOfRounds;

\_secretKeyLengthInBytes = secretKeyLengthInBytes;

\_s = [];

}

public byte[] Encrypt(byte[] message, string password)

{

\_s = GenerateArrayS(password);

byte[] extendedMessage = AddMessagePadding(message);

ulong[] words = SplitArrayToWords(extendedMessage);

byte[] result = new byte[\_wordLengthInBytes \* 2 + extendedMessage.Length];

// Calculate IV

ulong[] iv = GenerateIv();

byte[] ivArr = new byte[\_wordLengthInBytes \* 2];

Array.Copy(UlongToByteArray(iv[0]), 0, ivArr, 0, \_wordLengthInBytes);

Array.Copy(UlongToByteArray(iv[1]), 0, ivArr, \_wordLengthInBytes, \_wordLengthInBytes);

byte[] encryptedIv = EncryptEcb(ivArr);

Array.Copy(encryptedIv, 0, result, 0, encryptedIv.Length);

ulong preA = iv[0];

ulong preB = iv[1];

for (int i = 0; i < words.Length; i += 2)

{

ulong wordA = words[i] ^ preA;

ulong wordB = words[i + 1] ^ preB;

ulong[] twoWordsEncrypted = EncryptTwoWords(wordA, wordB);

Array.Copy(UlongToByteArray(twoWordsEncrypted[0]), 0, result, ivArr.Length + i \* \_wordLengthInBytes, \_wordLengthInBytes);

Array.Copy(UlongToByteArray(twoWordsEncrypted[1]), 0, result, ivArr.Length + (i + 1) \* \_wordLengthInBytes, \_wordLengthInBytes);

preA = twoWordsEncrypted[0];

preB = twoWordsEncrypted[1];

}

return result;

}

public byte[] Decrypt(byte[] message, string password)

{

\_s = GenerateArrayS(password);

// Calculate IV

byte[] ivArr = new byte[\_wordLengthInBytes \* 2];

Array.Copy(message, 0, ivArr, 0, ivArr.Length);

byte[] decryptedIv = DecryptEcb(ivArr);

byte[] ivA = new byte[\_wordLengthInBytes];

byte[] ivB = new byte[\_wordLengthInBytes];

Array.Copy(decryptedIv, 0, ivA, 0, \_wordLengthInBytes);

Array.Copy(decryptedIv, \_wordLengthInBytes, ivB, 0, \_wordLengthInBytes);

ulong preA = ByteArrayToUlong(ivA);

ulong preB = ByteArrayToUlong(ivB);

// Resolve message

byte[] messageWithoutIv = new byte[message.Length - \_wordLengthInBytes \* 2];

Array.Copy(message, \_wordLengthInBytes \* 2, messageWithoutIv, 0, message.Length - \_wordLengthInBytes \* 2);

int extendedMessageLength = (messageWithoutIv.Length / \_wordLengthInBytes + messageWithoutIv.Length % \_wordLengthInBytes);

extendedMessageLength += extendedMessageLength % 2;

extendedMessageLength \*= \_wordLengthInBytes;

byte[] extendedMessage = new byte[extendedMessageLength];

Array.Copy(messageWithoutIv, 0, extendedMessage, 0, messageWithoutIv.Length);

ulong[] words = SplitArrayToWords(extendedMessage);

byte[] result = new byte[extendedMessage.Length];

for (int i = 0; i < words.Length; i += 2)

{

ulong[] twoWordsDecrypted = DecryptTwoWords(words[i], words[i + 1]);

twoWordsDecrypted[0] ^= preA;

twoWordsDecrypted[1] ^= preB;

Array.Copy(UlongToByteArray(twoWordsDecrypted[0]), 0, result, i \* \_wordLengthInBytes, \_wordLengthInBytes);

Array.Copy(UlongToByteArray(twoWordsDecrypted[1]), 0, result, (i + 1) \* \_wordLengthInBytes, \_wordLengthInBytes);

preA = words[i];

preB = words[i + 1];

}

return RemoveMessagePadding(result);

}

private ulong[] GenerateArrayS(string password)

{

byte[] arrK = GenerateSecretKey(password);

ulong[] arrL = SplitArrayToWords(arrK);

ulong[] arrS = InitArrayS();

int i = 0;

int j = 0;

ulong a = 0;

ulong b = 0;

int t = Math.Max(arrL.Length, 2 \* \_numberOfRounds + 2);

for (int s = 1; s < t \* 3; s++)

{

arrS[i] = ((arrS[i] + a + b) << 3) & \_wordBytesUsage;

a = arrS[i];

i = (i + 1) % t;

arrL[j] = ((arrL[j] + a + b) << (int)(a + b)) & \_wordBytesUsage;

b = arrL[j];

j = (j + 1) % arrL.Length;

}

return arrS;

}

private byte[] GenerateSecretKey(string password)

{

byte[] hash = \_md5.GetHash(Encoding.UTF8.GetBytes(password));

if (hash.Length > \_secretKeyLengthInBytes)

{

byte[] result = new byte[\_secretKeyLengthInBytes];

Array.Copy(hash, hash.Length - \_secretKeyLengthInBytes, result, 0, \_secretKeyLengthInBytes);

return result;

}

if (hash.Length < \_secretKeyLengthInBytes)

{

byte[] result = new byte[\_secretKeyLengthInBytes];

for (int i = 0; i < \_secretKeyLengthInBytes / hash.Length + \_secretKeyLengthInBytes % hash.Length; i++)

{

Array.Copy(hash, 0, result, \_secretKeyLengthInBytes - (i + 1) \* hash.Length,

Math.Min(\_secretKeyLengthInBytes - (i + 1) \* hash.Length, hash.Length));

hash = \_md5.GetHash(hash);

}

return result;

}

return hash;

}

private ulong[] SplitArrayToWords(byte[] byteArray)

{

int numberOfWords = byteArray.Length / \_wordLengthInBytes + byteArray.Length % \_wordLengthInBytes;

ulong[] wordList = new ulong[numberOfWords];

for (int i = 0; i < numberOfWords; i++)

{

int offset = i \* \_wordLengthInBytes;

int numberOfBytes = Math.Min(\_wordLengthInBytes, byteArray.Length - offset);

byte[] value = new byte[\_wordLengthInBytes];

Array.Copy(byteArray, offset, value, 0, numberOfBytes);

wordList[i] = ByteArrayToUlong(value);

}

return wordList;

}

private ulong ByteArrayToUlong(byte[] byteArray)

{

ulong value = 0L;

int offset = 0;

foreach (byte b in byteArray)

{

value += (ulong)(b & 0xFF) << offset;

offset += 8;

}

return value;

}

private byte[] UlongToByteArray(ulong value)

{

byte[] byteArray = new byte[\_wordLengthInBytes];

for (int i = 0; i < byteArray.Length; i++)

{

byteArray[i] = (byte)(value & 0xFF);

value >>= 8;

}

return byteArray;

}

private ulong[] InitArrayS()

{

ulong[] arrS = new ulong[2 \* \_numberOfRounds + 2];

arrS[0] = \_arrP;

for (int i = 1; i < arrS.Length; i++)

{

arrS[i] = (arrS[i - 1] + \_arrQ) & \_wordBytesUsage;

}

return arrS;

}

private byte[] AddMessagePadding(byte[] message)

{

if (message.Length % (\_wordLengthInBytes \* 2) == 0)

{

return message.ToArray();

}

int bytesToAdd = \_wordLengthInBytes \* 2 - message.Length % (\_wordLengthInBytes \* 2);

byte[] result = new byte[message.Length + bytesToAdd];

Array.Copy(message, 0, result, 0, message.Length);

for (int i = 0; i < bytesToAdd; i++)

{

result[message.Length + i] = (byte)bytesToAdd;

}

return result;

}

private byte[] RemoveMessagePadding(byte[] message)

{

byte lastByte = message[^1];

for (int i = 0; i < lastByte; i++)

{

if (message[message.Length - 1 - i] != lastByte)

{

return message;

}

}

byte[] messageWithoutPadding = new byte[message.Length - lastByte];

Array.Copy(message, 0, messageWithoutPadding, 0, messageWithoutPadding.Length);

return messageWithoutPadding;

}

private ulong LoopLeftShift(ulong value, ulong bits)

{

bits %= (uint)\_wordLengthInBits;

ulong copyValue = value;

value <<= (int)bits;

value &= \_wordBytesUsage;

copyValue >>= (int)((uint)\_wordLengthInBits - bits);

copyValue &= \_wordBytesUsage;

return value | copyValue;

}

private ulong LoopRightShift(ulong value, ulong bits)

{

bits %= (uint)\_wordLengthInBits;

ulong copyValue = value;

value >>= (int)bits;

value &= \_wordBytesUsage;

copyValue <<= (int)((uint)\_wordLengthInBits - bits);

copyValue &= \_wordBytesUsage;

return value | copyValue;

}

private ulong[] EncryptTwoWords(ulong a, ulong b)

{

a = (a + \_s[0]) & \_wordBytesUsage;

b = (b + \_s[1]) & \_wordBytesUsage;

for (int i = 1; i <= \_numberOfRounds; i++)

{

a ^= b;

a = LoopLeftShift(a, b);

a = (a + \_s[2 \* i]) & \_wordBytesUsage;

b ^= a;

b = LoopLeftShift(b, a);

b = (b + \_s[2 \* i + 1]) & \_wordBytesUsage;

}

return [a, b];

}

private ulong[] DecryptTwoWords(ulong a, ulong b)

{

for (int i = \_numberOfRounds; i >= 1; i--)

{

b = (b - \_s[2 \* i + 1]) & \_wordBytesUsage;

b = LoopRightShift(b, a);

b ^= a;

a = (a - \_s[2 \* i]) & \_wordBytesUsage;

a = LoopRightShift(a, b);

a ^= b;

}

b = (b - \_s[1]) & \_wordBytesUsage;

a = (a - \_s[0]) & \_wordBytesUsage;

return [a, b];

}

private ulong[] GenerateIv() => [

(ulong)\_pseudoRandomGenerator.Next(),

(ulong)\_pseudoRandomGenerator.Next()

];

private byte[] EncryptEcb(byte[] message)

{

byte[] extendedMessage = AddMessagePadding(message);

ulong[] words = SplitArrayToWords(extendedMessage);

byte[] result = new byte[extendedMessage.Length];

for (int i = 0; i < words.Length; i += 2)

{

ulong wordA = words[i];

ulong wordB = words[i + 1];

ulong[] twoWordsEncrypted = EncryptTwoWords(wordA, wordB);

Array.Copy(UlongToByteArray(twoWordsEncrypted[0]), 0, result, i \* \_wordLengthInBytes, \_wordLengthInBytes);

Array.Copy(UlongToByteArray(twoWordsEncrypted[1]), 0, result, (i + 1) \* \_wordLengthInBytes, \_wordLengthInBytes);

}

return result;

}

private byte[] DecryptEcb(byte[] message)

{

int extendedMessageLength = (message.Length / \_wordLengthInBytes + message.Length % \_wordLengthInBytes);

extendedMessageLength += extendedMessageLength % 2;

extendedMessageLength \*= \_wordLengthInBytes;

byte[] extendedMessage = new byte[extendedMessageLength];

Array.Copy(message, 0, extendedMessage, 0, message.Length);

ulong[] words = SplitArrayToWords(extendedMessage);

byte[] result = new byte[extendedMessage.Length];

for (int i = 0; i < words.Length; i += 2)

{

ulong[] twoWordsDecrypted = DecryptTwoWords(words[i], words[i + 1]);

Array.Copy(UlongToByteArray(twoWordsDecrypted[0]), 0, result, i \* \_wordLengthInBytes, \_wordLengthInBytes);

Array.Copy(UlongToByteArray(twoWordsDecrypted[1]), 0, result, (i + 1) \* \_wordLengthInBytes, \_wordLengthInBytes);

}

return result;

}

public class WordLength

{

public int Length { get; }

public ulong BytesUsage { get; }

public ulong P { get; }

public ulong Q { get; }

public WordLength(int length)

{

switch (length)

{

case 16:

Length = 16;

BytesUsage = 0x000000000000FFFFL;

P = 0x000000000000B7E1L;

Q = 0x0000000000009E37L;

break;

case 32:

Length = 32;

BytesUsage = 0x00000000FFFFFFFFL;

P = 0x00000000B7E15163L;

Q = 0x000000009E3779B9L;

break;

case 64:

Length = 64;

BytesUsage = 0xFFFFFFFFFFFFFFFFL;

P = 0xB7E151628AED2A6BL;

Q = 0x9E3779B97F4A7C15L;

break;

default:

Length = 16;

BytesUsage = 0x000000000000FFFFL;

P = 0x000000000000B7E1L;

Q = 0x0000000000009E37L;

break;

}

}

public WordLength(int length, ulong bytesUsage, ulong p, ulong q)

{

this.Length = length;

this.BytesUsage = bytesUsage;

this.P = p;

this.Q = q;

}

}

}

**Результати роботи**

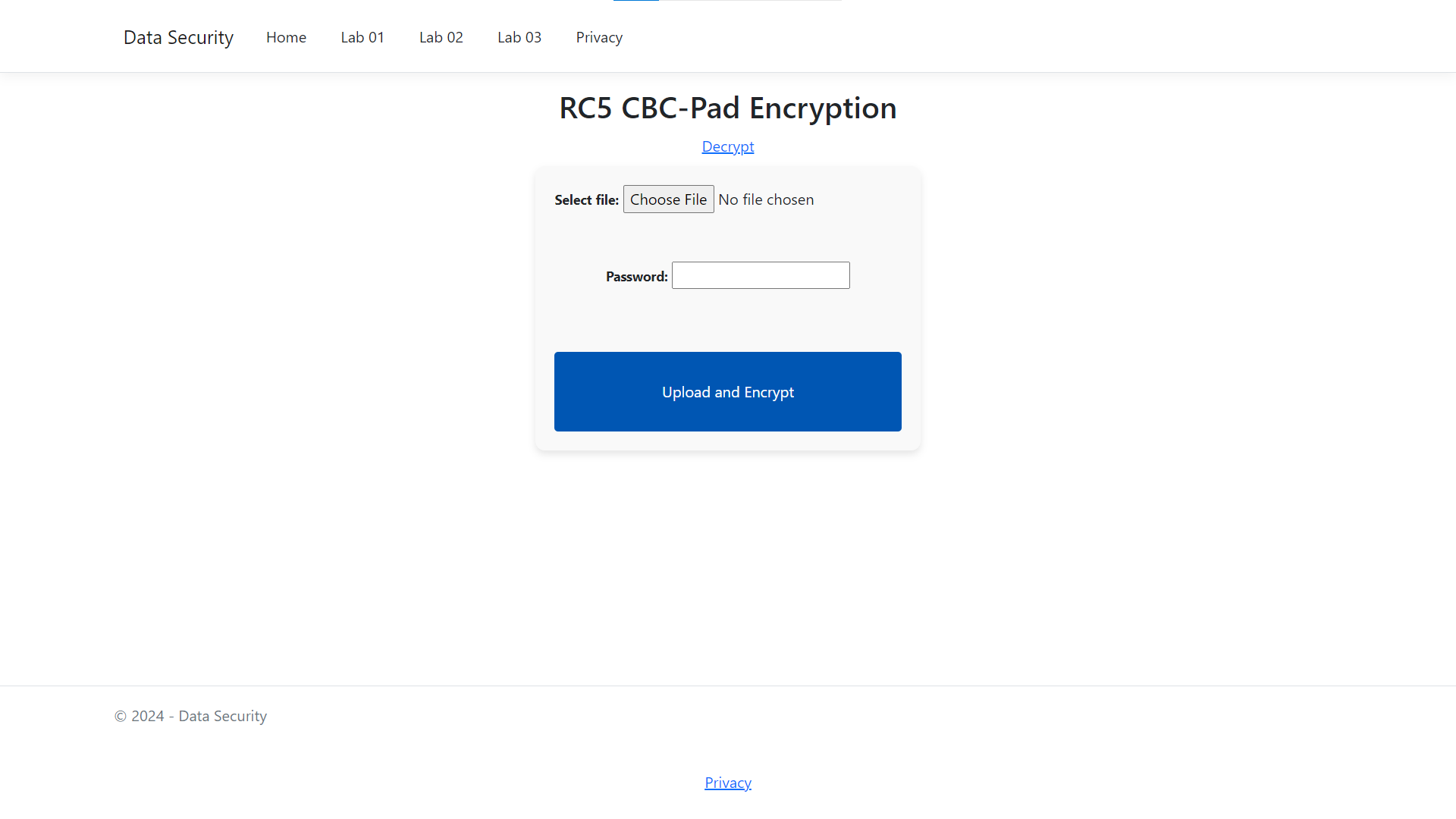


Рис. 2 Головне вікно програми

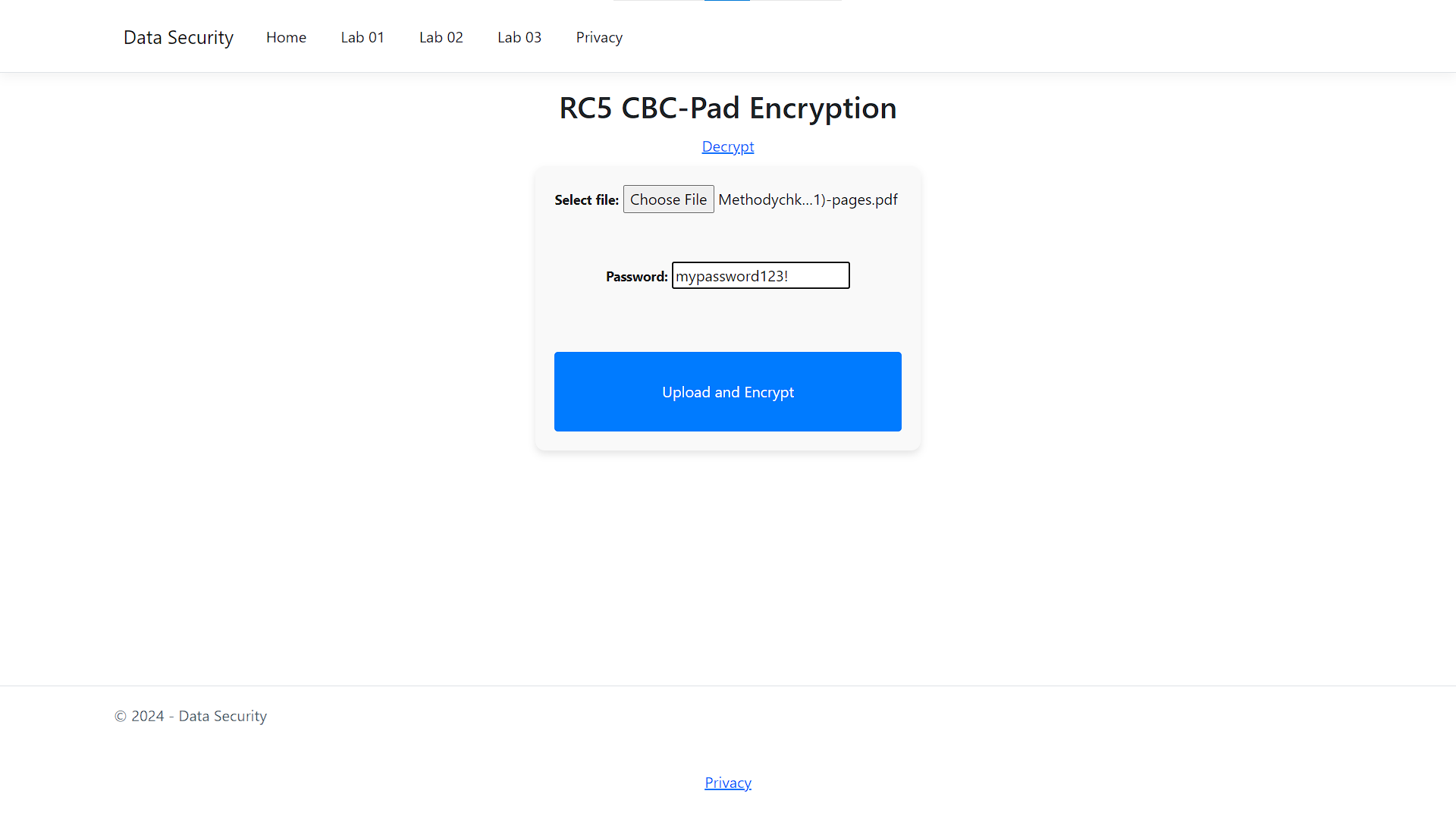


Рис. 3 Вхідні дані для шифрування

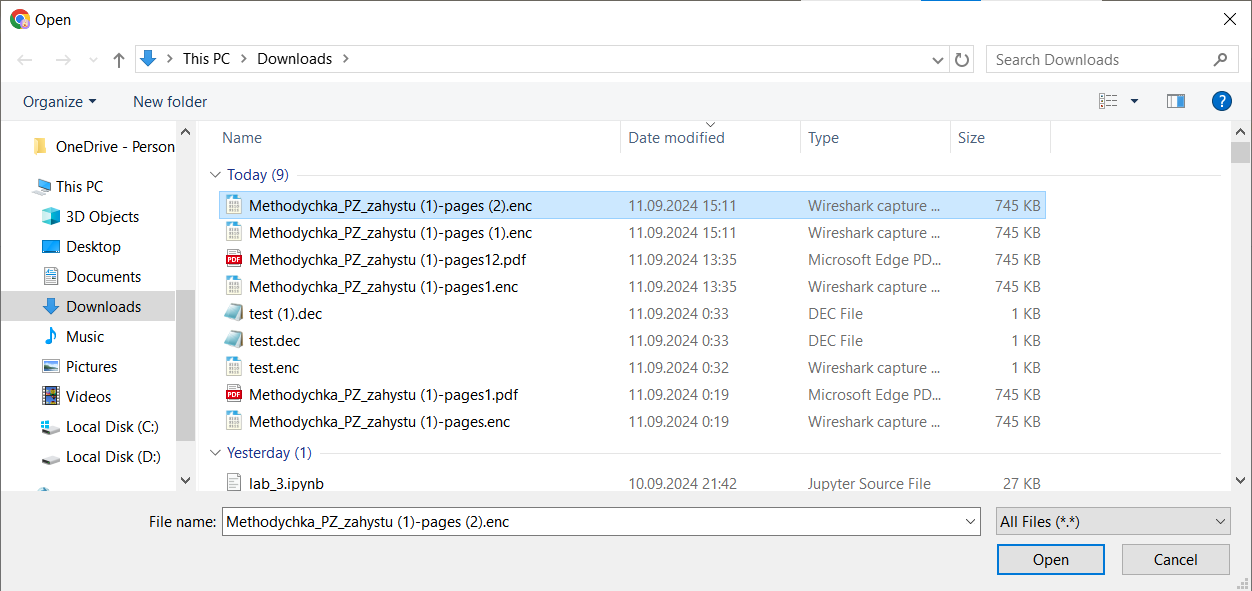


Рис. 4 Отриманий зашифрований файл

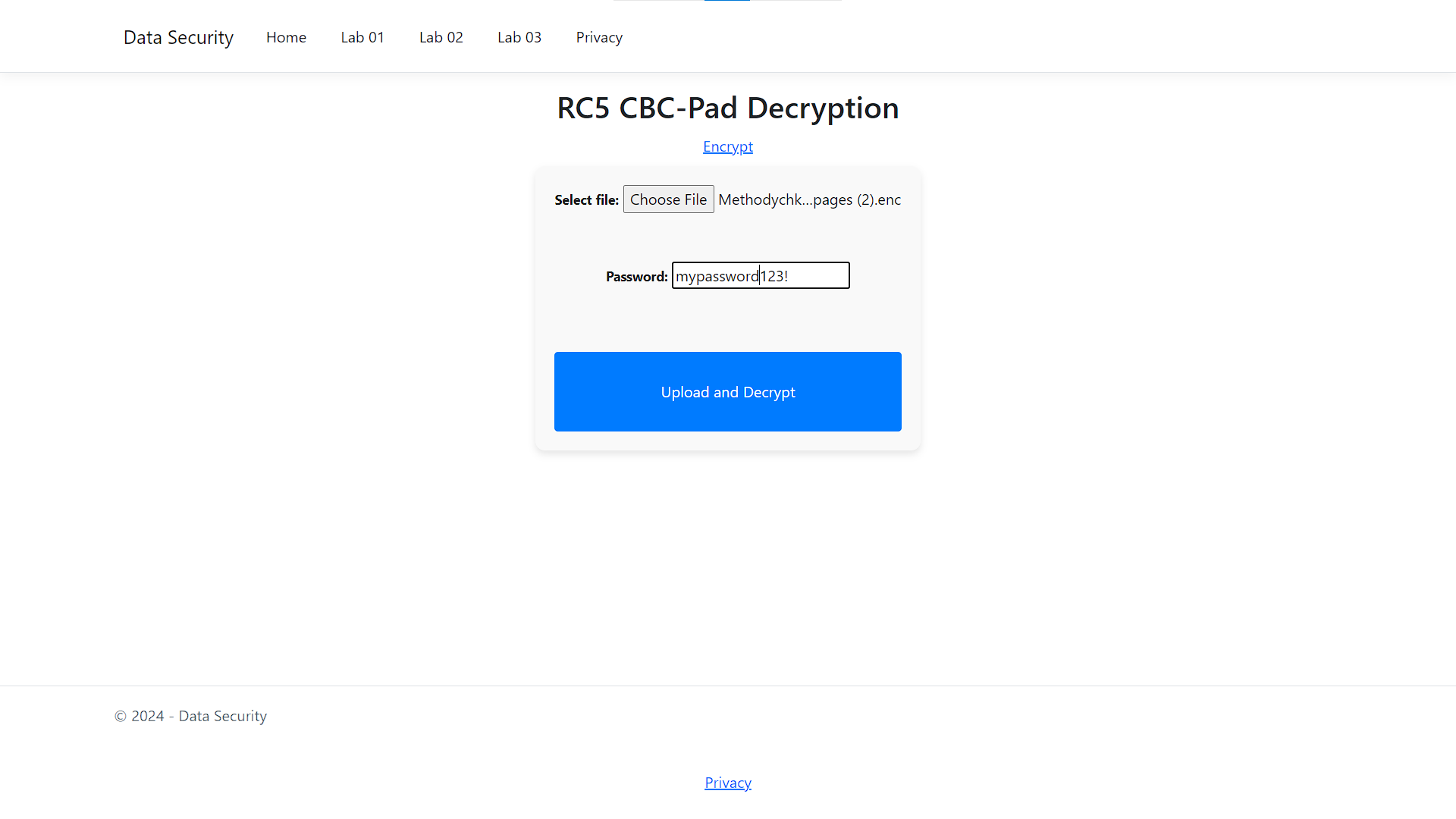


Рис. 5 Вхідні дані для дешифрування

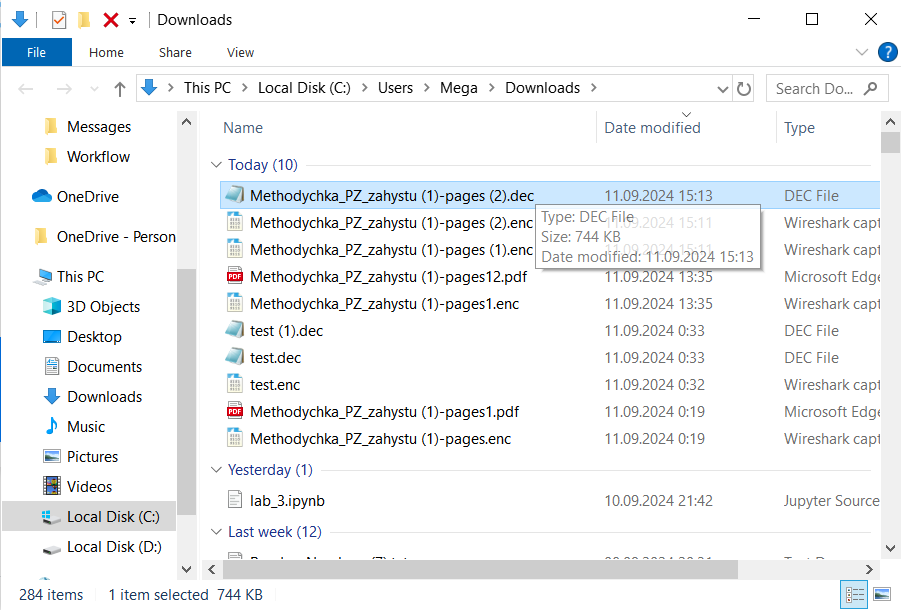
****

Рис. 6 Отриманий дешифрований файл

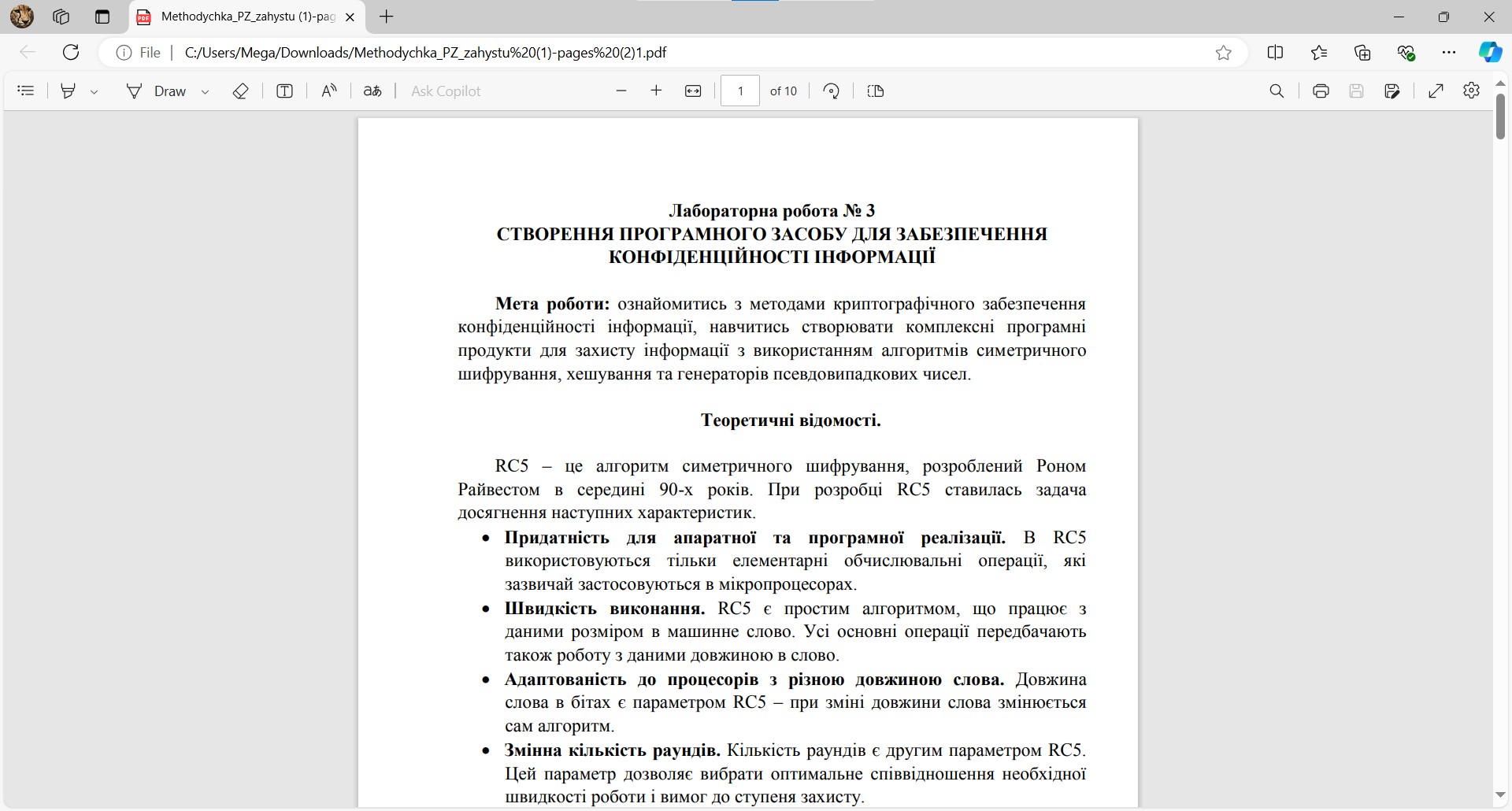


Рис. 7 Перевірка цілісності дешифрованого файлу

**Висновки**

Отже, під час виконання даної лабораторної роботи я ознайомивс з методами криптографічного забезпечення конфіденційності інформації, навчився створювати комплексні програмні продукти для захисту інформації з використанням алгоритмів симетричного шифрування, хешування та генераторів псевдовипадкових чисел. Створив програмну реалізацію алгоритму шифрування RC5-CBC-Pad.