**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:** *“Створення програмного засобу для забезпечення конфіденційності інформації”*

**Лектор:**

проф. каф. ПЗ

Сенів М.М.

**Виконав:**

ст. гр. ПЗ-45

Хруставчук М.Л.

**Прийняв:**

асист. каф. ПЗ

РабійчукІ.О.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2025 р.

∑= \_\_\_\_\_ .

Львів – 2025

**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.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

Конфіденційність інформації забезпечується за допомогою криптографічного шифрування, яке перетворює відкриті дані у вигляд, незрозумілий стороннім особам. Найпоширенішими для цього є симетричні алгоритми шифрування, в яких один і той самий секретний ключ використовується як для шифрування, так і для розшифрування повідомлення. Одним із таких алгоритмів є RC5 (Rivest Cipher 5), розроблений Р. Рівестом у 1994 році. Його головною особливістю є параметричність – користувач може вибирати довжину слова (*w*), кількість раундів (*r*) та довжину ключа (*b*), що дозволяє налаштовувати рівень безпеки й швидкодію під конкретне застосування.

Алгоритм RC5 складається з трьох основних етапів: розширення ключа, шифрування та дешифрування. Під час розширення ключа вихідний ключ користувача перетворюється у таблицю підключів *S[0...t−1]*, яка використовується у кожному раунді. Сам процес шифрування відбувається над двома словами (наприклад, по 32 біти кожне) і включає чергування операцій додавання по модулю 2ʷ, побітового XOR та циклічних зсувів, величина яких залежить від значень самих даних. Такі залежні зсуви створюють сильну нелінійність і високу дифузію – ключові властивості безпечного шифру.

RC5 може працювати в різних режимах шифрування: ECB (Electronic Codebook), CBC (Cipher Block Chaining), CBC-Pad і CTS (Ciphertext Stealing). Найбільш безпечним є режим CBC, який використовує вектор ініціалізації (IV) для уникнення повторів і підвищення криптостійкості. Алгоритм характеризується простотою реалізації, високою швидкістю, мінімальними вимогами до пам’яті та придатністю для апаратної або програмної реалізації.

Таким чином, RC5 залишається ефективним прикладом блочного симетричного шифру з відкритими параметрами, який демонструє принципи побудови сучасних криптографічних систем та дозволяє практично дослідити вплив розміру ключа, кількості раундів і режиму роботи на рівень захисту даних.

**ЗАВДАННЯ**

1. Згідно до варіанту, наведеного в таблиці, створити прикладну програму для шифрування інформації за алгоритмом RC5.
2. w = 16 (довжина слова, біт), r = 8 (к-сть раундів), b = 8 (довжина ключа, байт).

**ХІД ВИКОНАННЯ**

Файл RC5.cs

using System;

using System.Text;

namespace Lab\_03;

public class RC5

{

private const uint Pw = 0xB7E15163;

private const uint Qw = 0x9E3779B9;

private readonly int r, w;

private readonly uint[] S;

public RC5(int W, int rounds, byte[] key)

{

w = W;

r = rounds;

S = KeyExpansion(key);

}

// -------------------- ENCRYPT --------------------

public byte[] EncryptBlock(byte[] input)

{

if (input.Length != 8)

throw new ArgumentException("RC5 works on 64-bit blocks (8 bytes).");

uint A = BitConverter.ToUInt32(input, 0);

uint B = BitConverter.ToUInt32(input, 4);

A += S[0];

B += S[1];

for (int i = 1; i <= r; i++)

{

A = RotateLeft(A ^ B, (int)B) + S[2 \* i];

B = RotateLeft(B ^ A, (int)A) + S[2 \* i + 1];

}

byte[] output = new byte[8];

Array.Copy(BitConverter.GetBytes(A), 0, output, 0, 4);

Array.Copy(BitConverter.GetBytes(B), 0, output, 4, 4);

return output;

}

// -------------------- DECRYPT --------------------

public byte[] DecryptBlock(byte[] input)

{

if (input.Length != 8)

throw new ArgumentException("RC5 works on 64-bit blocks (8 bytes).");

uint A = BitConverter.ToUInt32(input, 0);

uint B = BitConverter.ToUInt32(input, 4);

for (int i = r; i >= 1; i--)

{

B = RotateRight(B - S[2 \* i + 1], (int)A) ^ A;

A = RotateRight(A - S[2 \* i], (int)B) ^ B;

}

B -= S[1];

A -= S[0];

byte[] output = new byte[8];

Array.Copy(BitConverter.GetBytes(A), 0, output, 0, 4);

Array.Copy(BitConverter.GetBytes(B), 0, output, 4, 4);

return output;

}

// -------------------- KEY EXPANSION --------------------

private uint[] KeyExpansion(byte[] key)

{

int u = w / 8;

int c = Math.Max(1, key.Length / u);

uint[] L = new uint[c];

// Little-endian завантаження ключа

for (int i = key.Length - 1; i >= 0; i--)

L[i / u] = (L[i / u] << 8) + key[i];

uint[] S = new uint[2 \* (r + 1)];

S[0] = Pw;

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

S[i] = S[i - 1] + Qw;

uint A = 0, B = 0;

int n = 3 \* Math.Max(S.Length, L.Length);

int si = 0, li = 0;

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

{

A = S[si] = RotateLeft(S[si] + A + B, 3);

B = L[li] = RotateLeft(L[li] + A + B, (int)(A + B));

si = (si + 1) % S.Length;

li = (li + 1) % L.Length;

}

return S;

}

// -------------------- UTILITIES --------------------

private static uint RotateLeft(uint value, int shift)

{

shift &= 31;

return (value << shift) | (value >> (32 - shift));

}

private static uint RotateRight(uint value, int shift)

{

shift &= 31;

return (value >> shift) | (value << (32 - shift));

}

// -------------------- CBC MODE --------------------

private static byte[] ApplyPadding(byte[] data)

{

int padLen = 8 - (data.Length % 8);

byte[] padded = new byte[data.Length + padLen];

Array.Copy(data, padded, data.Length);

for (int i = data.Length; i < padded.Length; i++)

padded[i] = (byte)padLen; // PKCS7

return padded;

}

private static byte[] RemovePadding(byte[] data)

{

int padLen = data[^1];

if (padLen <= 0 || padLen > 8)

throw new InvalidOperationException("Invalid padding");

byte[] unpadded = new byte[data.Length - padLen];

Array.Copy(data, unpadded, unpadded.Length);

return unpadded;

}

public byte[] EncryptCBC(byte[] data, byte[] iv)

{

byte[] padded = ApplyPadding(data);

byte[] prev = new byte[8];

Array.Copy(iv, prev, 8);

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

for (int i = 0; i < padded.Length; i += 8)

{

byte[] block = new byte[8];

Array.Copy(padded, i, block, 0, 8);

for (int j = 0; j < 8; j++)

block[j] ^= prev[j]; // XOR з попереднім блоком

byte[] cipher = EncryptBlock(block);

Array.Copy(cipher, 0, result, i, 8);

prev = cipher;

}

return result;

}

public byte[] DecryptCBC(byte[] cipher, byte[] iv)

{

if (cipher.Length % 8 != 0)

throw new ArgumentException("Ciphertext length must be multiple of block size.");

byte[] prev = new byte[8];

Array.Copy(iv, prev, 8);

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

for (int i = 0; i < cipher.Length; i += 8)

{

byte[] block = new byte[8];

Array.Copy(cipher, i, block, 0, 8);

byte[] plain = DecryptBlock(block);

for (int j = 0; j < 8; j++)

plain[j] ^= prev[j]; // XOR з попереднім шифротекстом

Array.Copy(plain, 0, result, i, 8);

prev = block;

}

return RemovePadding(result);

}

}

Файл RC5Encryptor.cs

using Lab\_03.Models;

using System;

using System.IO;

using System.Linq;

using System.Text;

namespace Lab\_03;

public class RC5Encryptor

{

private const int W = 16;

private const int R = 8;

private const int KEY\_BITS = 8;

private readonly RC5 \_rc5;

public RC5Encryptor(string password)

{

byte[] key = DeriveKeyFromPassword(password, KEY\_BITS);

\_rc5 = new RC5(W, R, key);

}

// ========================= MAIN METHODS =========================

public void EncryptFile(string inputPath, string outputPath)

{

byte[] data = File.ReadAllBytes(inputPath);

// Генеруємо IV через власний генератор

var a = (uint)Math.Pow(9, 3);

var m = (uint)Math.Pow(2, 22) - 1;

uint c = 233, x0 = 5;

var rng = new RandomNumber(a, c, m);

var randomValues = rng.GenerateNumbers(x0, 2);

byte[] iv = BitConverter.GetBytes(randomValues[0])

.Concat(BitConverter.GetBytes(randomValues[1]))

.Take(8)

.ToArray();

byte[] cipher = \_rc5.EncryptCBC(data, iv);

byte[] ivEncrypted = \_rc5.EncryptBlock(iv);

File.WriteAllBytes(outputPath, ivEncrypted.Concat(cipher).ToArray());

}

public void DecryptFile(string inputPath, string outputPath)

{

byte[] full = File.ReadAllBytes(inputPath);

byte[] ivEncrypted = full[..8];

byte[] iv = \_rc5.DecryptBlock(ivEncrypted);

byte[] cipher = full[8..];

byte[] plain = \_rc5.DecryptCBC(cipher, iv);

File.WriteAllBytes(outputPath, plain);

}

// ========================= KEY DERIVATION =========================

private static byte[] DeriveKeyFromPassword(string password, int keyBits)

{

var pwdBytes = Encoding.UTF8.GetBytes(password);

var h1 = MyMd5.CalculateHash(pwdBytes);

if (keyBits == 64)

return h1[..8];

if (keyBits == 256)

{

var h2 = MyMd5.CalculateHash(h1);

byte[] key = new byte[32];

Array.Copy(h2, 0, key, 0, 16);

Array.Copy(h1, 0, key, 16, 16);

return key;

}

return h1;

}

}

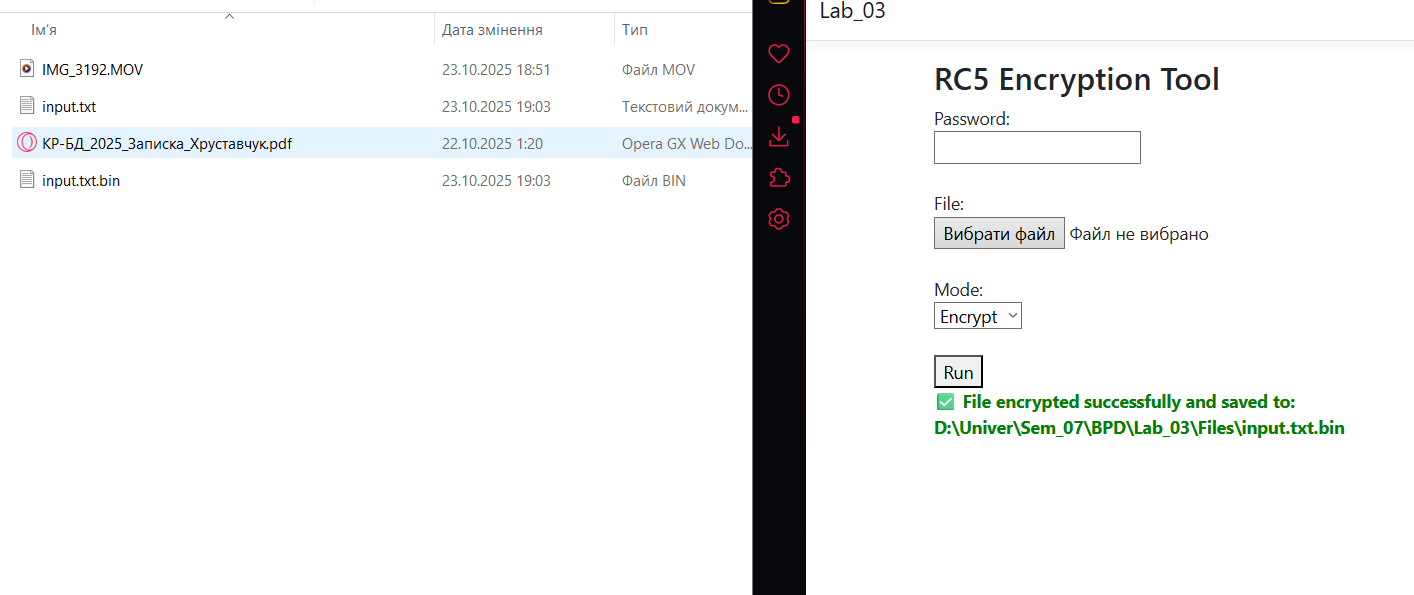


Рис. 1. Результат шифрування файлу

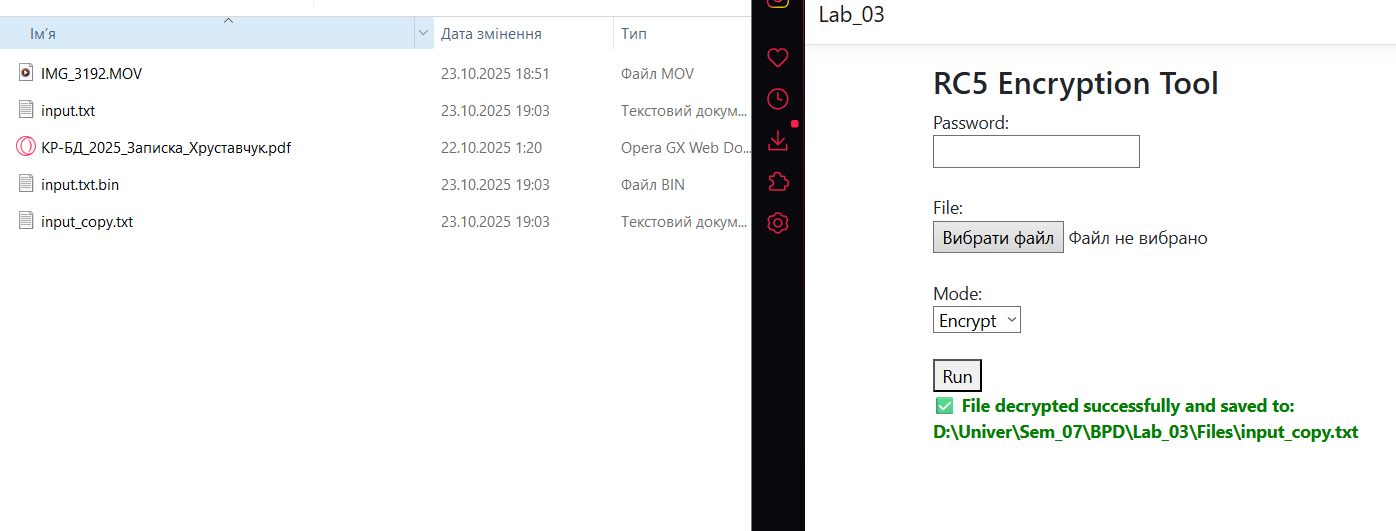


Рис. 2. Результат дешифрування файлу

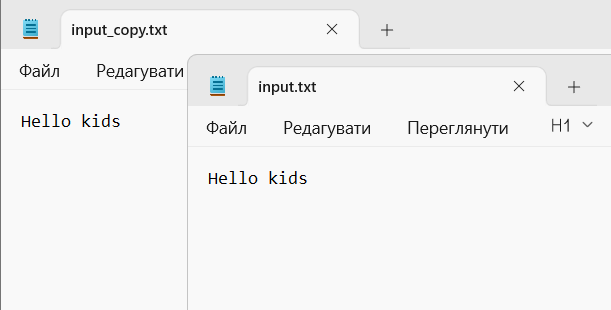


Рис. 3. Порівняння шифрованого та дешифрованого файлу

**ВИСНОВКИ**

У ході роботи було досліджено принципи симетричного шифрування та реалізовано алгоритм RC5, який забезпечує конфіденційність інформації шляхом використання змінних параметрів ключа і раундів. Практично підтверджено, що збільшення кількості раундів і довжини ключа підвищує стійкість шифру, але впливає на швидкодію. Алгоритм RC5 відзначається простою структурою, високою ефективністю та може бути використаний як основа для побудови більш складних криптосистем