**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 2

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

**На тему:** *“Створення програмного засобу для забезпечення*

*цілісності 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

цілісності інформації.

**Мета роботи:** Ознайомитись з методами криптографічного забезпечення цілісності інформації, навчитись створювати програмні засоби для забезпечення цілісності інформації з використанням алгоритмів хешування.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

У сучасних інформаційних системах надзвичайно важливим є захист даних від несанкціонованих змін та пошкоджень. Одним із основних механізмів забезпечення цілісності є використання криптографічних хеш-функцій. Хеш-функція – це одностороннє відображення довільного вхідного повідомлення у бітовий рядок фіксованої довжини. Криптографічна хеш-функція повинна задовольняти кілька ключових властивостей: стійкість до знаходження прообразу (неможливість відновити повідомлення за відомим хешем), стійкість до колізій (неможливість знайти два різні повідомлення з однаковим хешем) та лавиноподібний ефект (незначна зміна у вхідних даних повинна радикально змінювати результат).

Однією з найвідоміших хеш-функцій є MD5 (Message Digest 5), розроблена Р. Рівестом у 1991 році. MD5 перетворює повідомлення довільної довжини у 128-бітний дайджест, який зазвичай подається у вигляді 32-символьного шістнадцяткового рядка. Алгоритм MD5 складається з кількох етапів:

1. Додавання доповнення: до повідомлення приписується біт «1» та стільки нулів, щоб довжина стала конгруентною 448 за модулем 512.
2. Додавання довжини повідомлення: у кінець блоку записується 64-бітне значення довжини вхідного повідомлення у бітах.
3. Ініціалізація буфера: використовується чотири 32-бітові змінні A, B, C, D з фіксованими початковими значеннями.
4. Обробка блоків: повідомлення ділиться на 512-бітові блоки, кожен з яких проходить через чотири раунди перетворень. Кожен раунд складається з 16 операцій над поточними словами блоку з використанням нелінійних логічних функцій, циклічних зсувів та додавання констант, що обчислюються на основі функції синуса.
5. Формування результату: після обробки всіх блоків значення A, B, C, D об’єднуються у 128-бітний.

**ЗАВДАННЯ**

1. Створити програмну реалізацію алгоритму хешування MD5.
2. Програмна реалізація повинна виводити значення хешу як для рядка, заданого в полі вводу, так і для файлу. Результат роботи програми повинен відображатись на екрані з можливістю наступного запису в файл. Крім того програма повинна мати можливість перевірити цілісність будь-якого файлу за наявним файлом з MD5 хешем, записаним у шістнадцятковому форматі.
3. У звіті навести протокол тестування і роботи програми та зробити висновки.

**ХІД ВИКОНАННЯ**

Файл MyMd5.cs

namespace Lab\_02.Services;

public static class MyMd5

{

private const int INIT\_A = 0x67452301;

private const int INIT\_B = unchecked((int)0xEFCDAB89u);

private const int INIT\_C = unchecked((int)0x98BADCFEu);

private const int INIT\_D = 0x10325476;

private static readonly int[] SHIFT\_AMTS = {

7, 12, 17, 22,

5, 9, 14, 20,

4, 11, 16, 23,

6, 10, 15, 21

};

private static readonly int[] TABLE\_T = BuildTableT();

private static int[] BuildTableT()

{

var t = new int[64];

for (int i = 0; i < 64; i++)

{

double v = Math.Abs(Math.Sin(i + 1.0)) \* Math.Pow(2.0, 32.0);

t[i] = unchecked((int)(long)v);

}

return t;

}

public static byte[] CalculateHash(byte[] message)

{

int messageLenBytes = message.Length;

int numBlocks = ((messageLenBytes + 8) >> 6) + 1;

int totalLen = numBlocks << 6;

var paddingBytes = new byte[totalLen - messageLenBytes];

paddingBytes[0] = 0x80;

long messageLenBits = (long)messageLenBytes << 3;

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

{

paddingBytes[paddingBytes.Length - 8 + i] = (byte)messageLenBits;

messageLenBits = (long)((ulong)messageLenBits >> 8);

}

int a = INIT\_A;

int b = INIT\_B;

int c = INIT\_C;

int d = INIT\_D;

var buffer = new int[16];

for (int block = 0; block < numBlocks; block++)

{

int index = block << 6;

Array.Clear(buffer, 0, buffer.Length);

for (int j = 0; j < 64; j++, index++)

{

int bVal = (index < messageLenBytes)

? message[index]

: paddingBytes[index - messageLenBytes];

buffer[j >> 2] = (int)((bVal & 0xFF) << 24) | ((int)((uint)buffer[j >> 2] >> 8));

}

int originalA = a;

int originalB = b;

int originalC = c;

int originalD = d;

for (int j = 0; j < 64; j++)

{

int div16 = j >> 4;

int f = 0;

int bufferIndex = j;

switch (div16)

{

case 0:

f = (b & c) | (~b & d);

break;

case 1:

f = (b & d) | (c & ~d);

bufferIndex = (bufferIndex \* 5 + 1) & 0x0F;

break;

case 2:

f = b ^ c ^ d;

bufferIndex = (bufferIndex \* 3 + 5) & 0x0F;

break;

case 3:

f = c ^ (b | ~d);

bufferIndex = (bufferIndex \* 7) & 0x0F;

break;

}

int s = SHIFT\_AMTS[(div16 << 2) | (j & 3)];

int temp = b + RotateLeft(a + f + buffer[bufferIndex] + TABLE\_T[j], s);

a = d;

d = c;

c = b;

b = temp;

}

a += originalA;

b += originalB;

c += originalC;

d += originalD;

}

var md5 = new byte[16];

int count = 0;

for (int i = 0; i < 4; i++)

{

int n = (i == 0) ? a : (i == 1) ? b : (i == 2) ? c : d;

for (int j = 0; j < 4; j++)

{

md5[count++] = (byte)n;

n = (int)((uint)n >> 8);

}

}

return md5;

}

private static int RotateLeft(int value, int bits)

{

unchecked

{

return (int)(((uint)value << bits) | ((uint)value >> (32 - bits)));

}

}

}

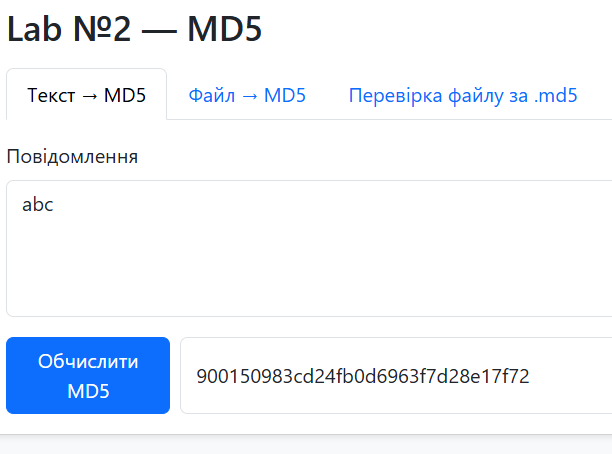


Рис. 1. Результат функції хешування для стрічки “abc”

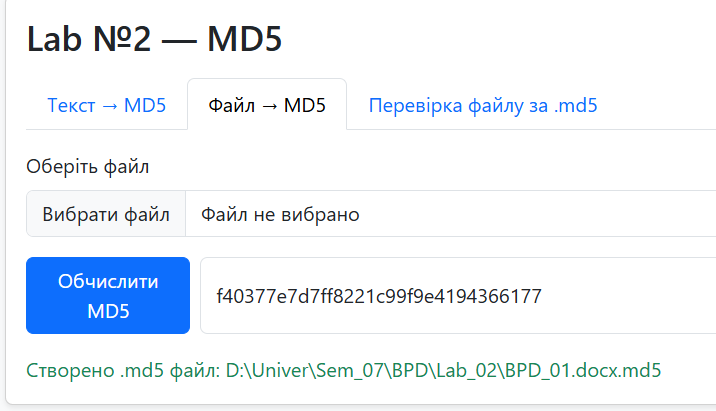


Рис. 2. Результат функції хешування для файлу

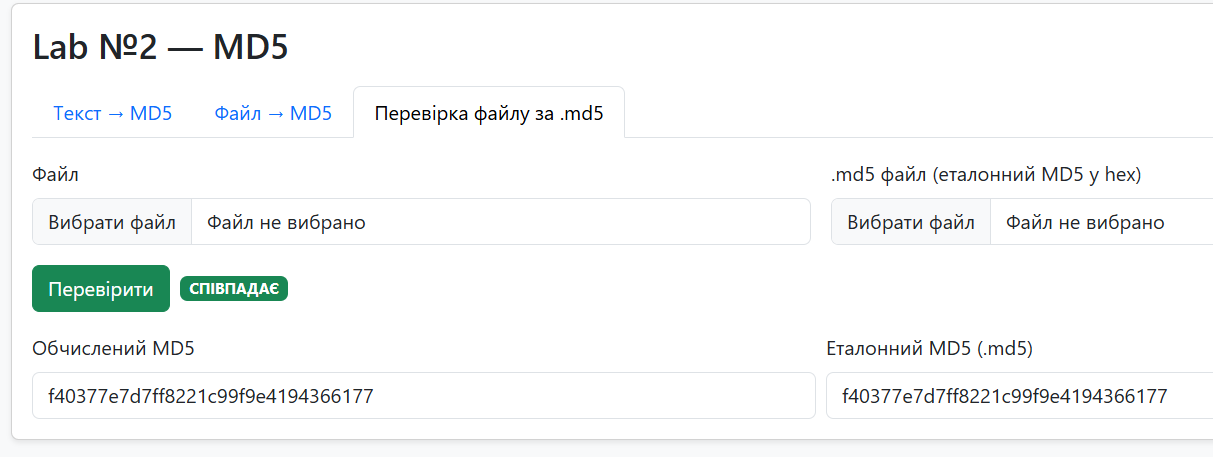


Рис. 3. Перевірка зашифрованого файлу

**ВИСНОВКИ**

На цій лабораторній роботі я ознайомився з алгоритмом хешування даних MD5, а також застосував отримані знання на практиці по реалізації алгоритму мовою С#.