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:** *“Cтворення програмного засобу для забезпечення цілісності 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 цілісності інформації.

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

**Теоретичні відомості**

Хеш функція – це функція, що відображає вхідне слово скінченної довжини у скінченному алфавіті в слово заданої, зазвичай фіксованої довжини. Таким чином, функція хешування отримує на вхід повідомлення M довільної довжини, а на вихід видає хеш-код H(M) фіксованого розміру, який іноді називають профілем повідомлення. Хеш-код є функцією усіх бітів повідомлення і забезпечує можливість контролю помилок: зміна будь-якої кількості бітів повідомлення призводить до зміни хеш-коду.

Основні області використання хеш функцій – аутентифікація інформації та цифровий підпис. Практичне використання функцій хешування накладає на них ряд вимог, наведених нижче:

1. Хеш-функція H повинна застосовуватися до блоку даних будь-якої довжини.
2. Хеш-функція H створює вихід фіксованої довжини.
3. H(M) відносно легко (за поліноміальний час) обчислюється для будьякого значення M, а алгоритм обчислення повинен бути практичним з погляду як апаратної, так і програмної реалізації.
4. Для будь-якого даного значення хеш-коду h обчислювально неможливо знайти M таке, що H(M)=h. Таку властивість іноді називають односторонністю.
5. Для будь-якого даного блоку x обчислювально неможливо знайти y≠x, для якого H(x)=H(y). Таку властивість іноді називають слабкою опірністю колізіям.
6. Обчислювально неможливо знайти довільну пару різних значень x та y, для яких H(x)=H(y). Таку властивість іноді називають сильною опірністю колізіям.

Перші три властивості описують вимоги, що забезпечують можливість практичного застосування функції хешування для аутентифікації повідомлень.

Четверту властивість визначає вимога односторонності хеш-функції: легко створити хеш-код за даним повідомленням, але неможливо відновити повідомлення за даним хеш-кодом. П'ята властивість гарантує, що неможливо знайти інше повідомлення, значення хеш-функції якого збігалося б зі значенням хеш-функції даного повідомлення. Це запобігає підробці аутентифікатора при використанні зашифрованого хеш-коду. Шоста властивість захищає проти класу атак, побудованих на парадоксі задачі про дні народження. Вона унеможливлює знаходження двох довільних різних повідомлень з однаковим хешем.

**Завдання до виконання роботи**

Створити програмну реалізацію алгоритму хешування MD5. Тестування створеної програми провести з використання наступних тестових значень хешу (згідно RFC 1321):

H() = D41D8CD98F00B204E9800998ECF8427E

H(a) = 0CC175B9C0F1B6A831C399E269772661

H(abc) = 900150983CD24FB0D6963F7D28E17F72

H(message digest) = F96B697D7CB7938D525A2F31AAF161D0

H(abcdefghijklmnopqrstuvwxyz) = C3FCD3D76192E4007DFB496CCA67E13B

H(ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789) = D174AB98D277D9F5A5611C2C9F419D9F

H(12345678901234567890123456789012345678901234567890123456789012345678901 234567890) = 57EDF4A22BE3C955AC49DA2E2107B67A

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. У звіті навести протокол тестування і роботи програми та зробити висновки.

**Код аглоритму**

public class MD5Service : IMD5Service

{

// Constants

private const int BlockSize = 64; // 512 bits

private const int PaddingModLength = 56; // BlockSize - 8 bytes for length

// Round shift values

private static readonly int[] S =

{

7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,

5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,

4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,

6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21

};

// Constant K Values

private static readonly uint[] K =

{

0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,

0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,

0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,

0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,

0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,

0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,

0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,

0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,

0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,

0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,

0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,

0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,

0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,

0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,

0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,

0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391

};

public string GetHash(string input)

{

byte[] inputBytes = Encoding.UTF8.GetBytes(input);

return GetHash(inputBytes);

}

public string GetHash(byte[] input)

{

uint[] h = { 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476 };

byte[] paddedInput = PadInput(input);

for (int i = 0; i < paddedInput.Length / BlockSize; ++i)

{

uint[] M = new uint[16];

Buffer.BlockCopy(paddedInput, i \* BlockSize, M, 0, BlockSize);

uint[] currentHash = (uint[])h.Clone();

ProcessChunk(M, currentHash);

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

{

h[j] += currentHash[j];

}

}

return string.Concat(h.Select(x => BitConverter.ToString(BitConverter.GetBytes(x)).Replace("-", "").ToLower()));

}

private static byte[] PadInput(byte[] input)

{

List<byte> padded = new List<byte>(input) { 0x80 };

while (padded.Count % BlockSize != PaddingModLength)

{

padded.Add(0x00);

}

padded.AddRange(BitConverter.GetBytes((long)input.Length \* 8));

return padded.ToArray();

}

private static void ProcessChunk(uint[] M, uint[] h)

{

uint A = h[0], B = h[1], C = h[2], D = h[3];

for (uint k = 0; k < BlockSize; ++k)

{

uint F = 0, g = 0;

if (k < 16)

{

F = (B & C) | (~B & D);

g = k;

}

else if (k < 32)

{

F = (D & B) | (~D & C);

g = (5 \* k + 1) % 16;

}

else if (k < 48)

{

F = B ^ C ^ D;

g = (3 \* k + 5) % 16;

}

else

{

F = C ^ (B | ~D);

g = 7 \* k % 16;

}

uint temp = D;

D = C;

C = B;

B += LeftRotate(A + F + K[k] + M[g], S[k]);

A = temp;

}

h[0] = A;

h[1] = B;

h[2] = C;

h[3] = D;

}

private static uint LeftRotate(uint x, int c)

{

return (x << c) | (x >> (32 - c));

}

}

**Результати роботи**

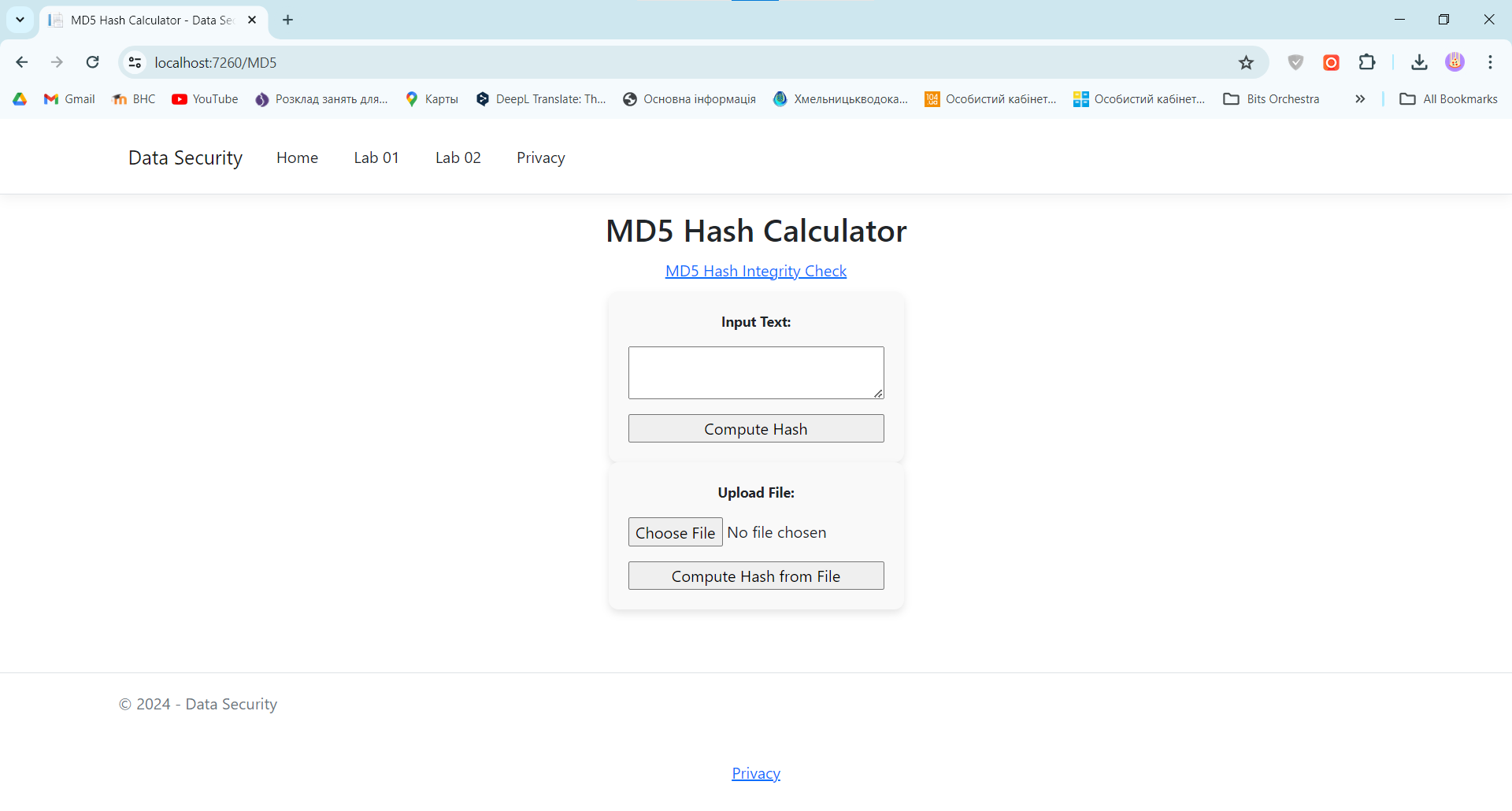


Рис. 1 Головне вікно програми

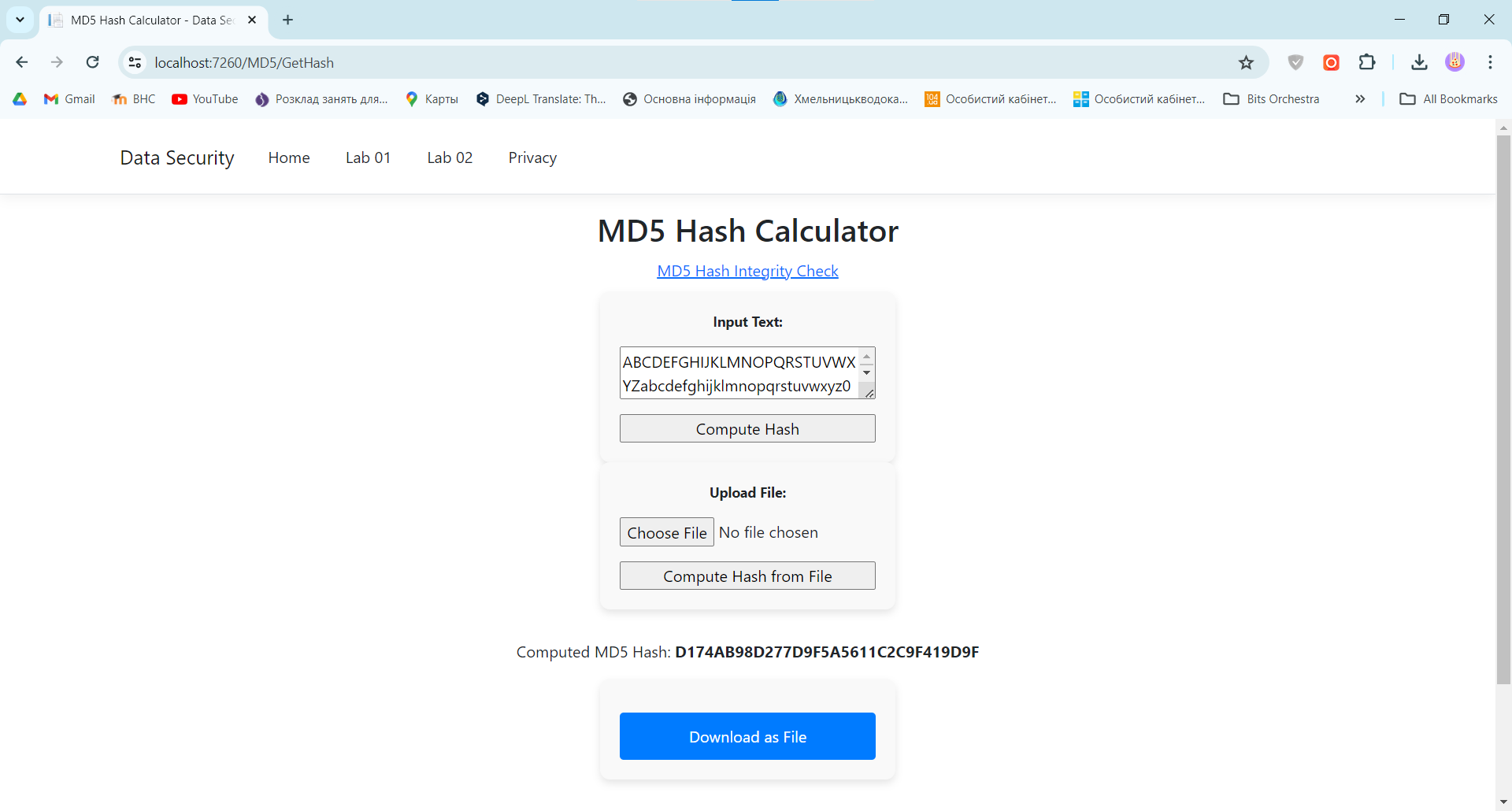


Рис. 2 Отриманий хеш введеного тексту

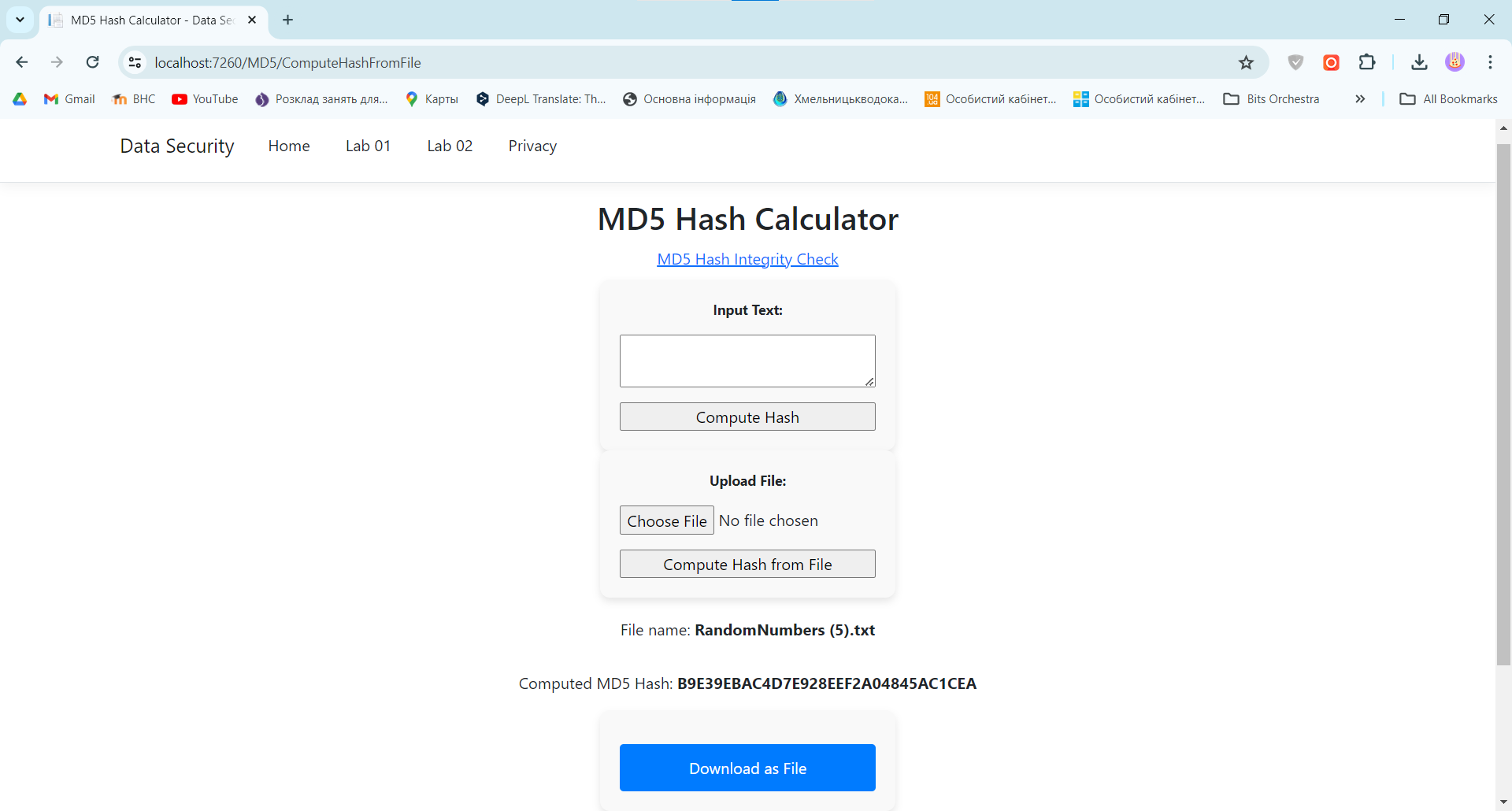


Рис. 3 Отриманий хеш обраного файлу

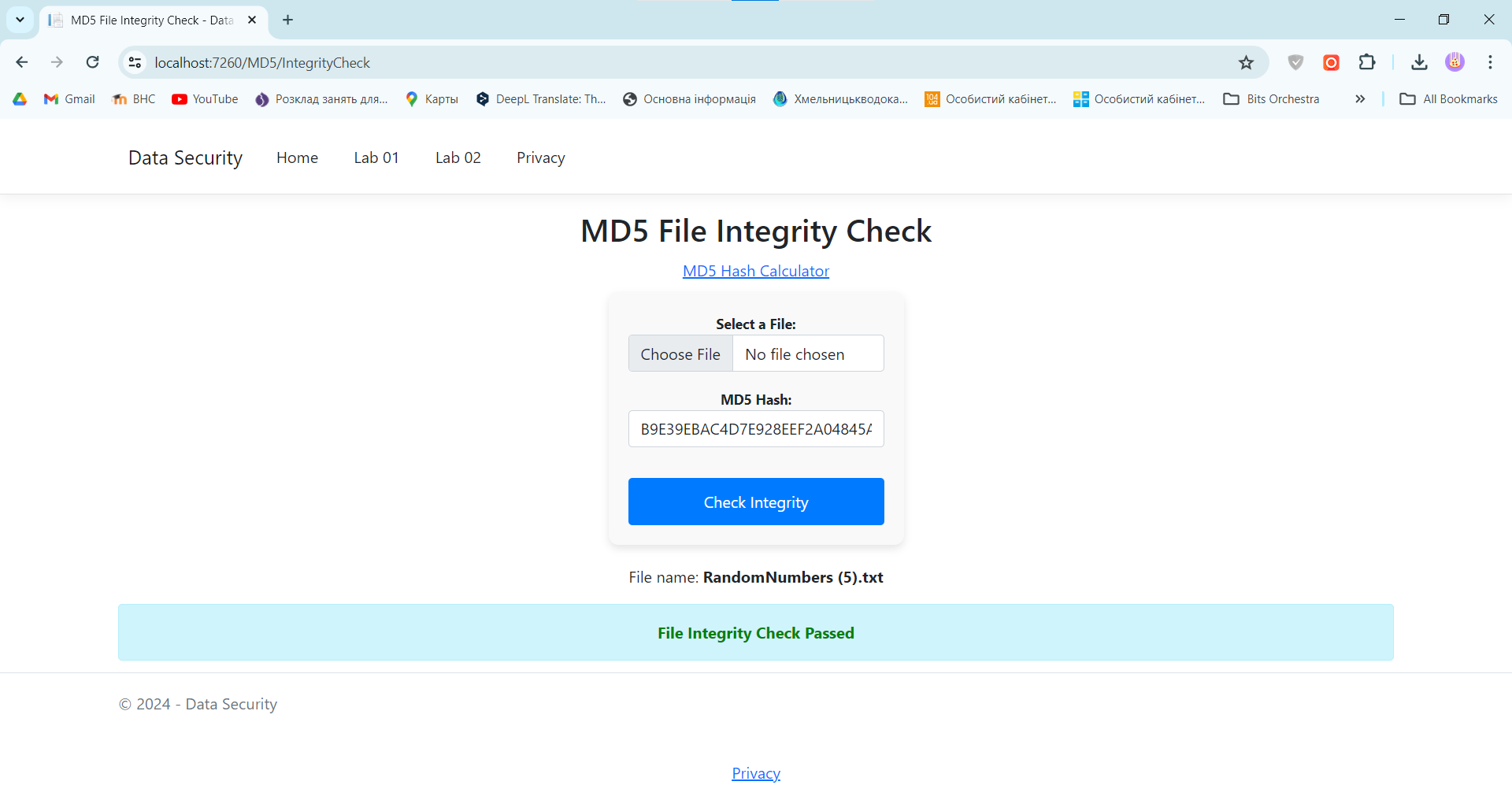


Рис. 4 Перевірка цілісності файлу за допомогою його MD5 хешу

Отже, провівши тестування створеної програми з використанням заданих тестових значень хешу (згідно RFC 1321), можна зробити висновок, що програма працює коректно.

**Висновки**

Отже, під час виконання даної лабораторної роботи я ознайомився з методами криптографічного забезпечення цілісності інформації, навчився створювати програмні засоби для забезпечення цілісності інформації з використанням алгоритмів хешування. Створив програмну реалізацію алгоритму хешування MD5.