<https://github.com/bolshakovartem/osnovi-bezpeki.git>

Звіт

Лабораторна робота №5

Тема: «Безпечне зберігання паролів»

Большаков Артем, МІТ-21

Завдання 2

**1) Зберігання у відкритому вигляді**

Є найбільш небезпечним методом зберігання паролів. Зловмиснику не потрібно виконувати ніяких складних дій, щоб викрасти пароль.

**2) Шифрування паролів**

Є більш кращим способом зберігання паролю, ніж у відкритому вигляді. Дозволяє виконувати зворотню дію – розшифровку. Але у його використанні з’являються інші труднощі. Наприклад, де зберігати та використовувати секретний ключ безпечно?

**3) Використання хешів для зберігання паролів**

Спосіб зберігання паролів коли він переходить у хеш-дані. Спосіб не дозволяє зворотної дій. Для цього способу є дві загрозливі атаки від шахраїв: атака грубої сили та атака з використанням райдужних таблиць.

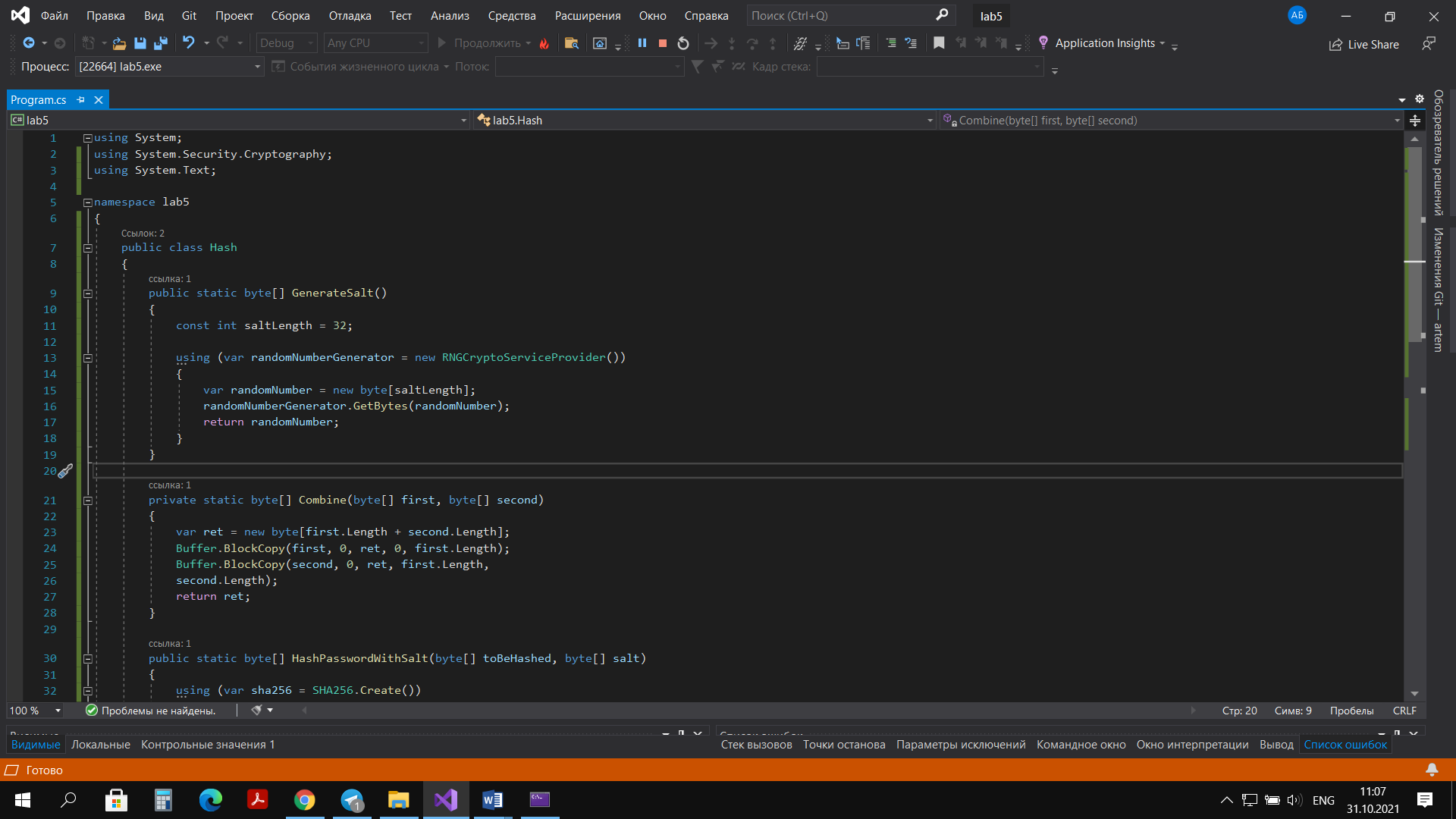
**4) Використання хешів із сіллю для зберігання паролів**

«Сіль» додається до паролю перед хешування, як набір випадкових цифр. Так як «сіль» це рандомний набір цифр, то кожен раз хеш паролю буде змінюватися.

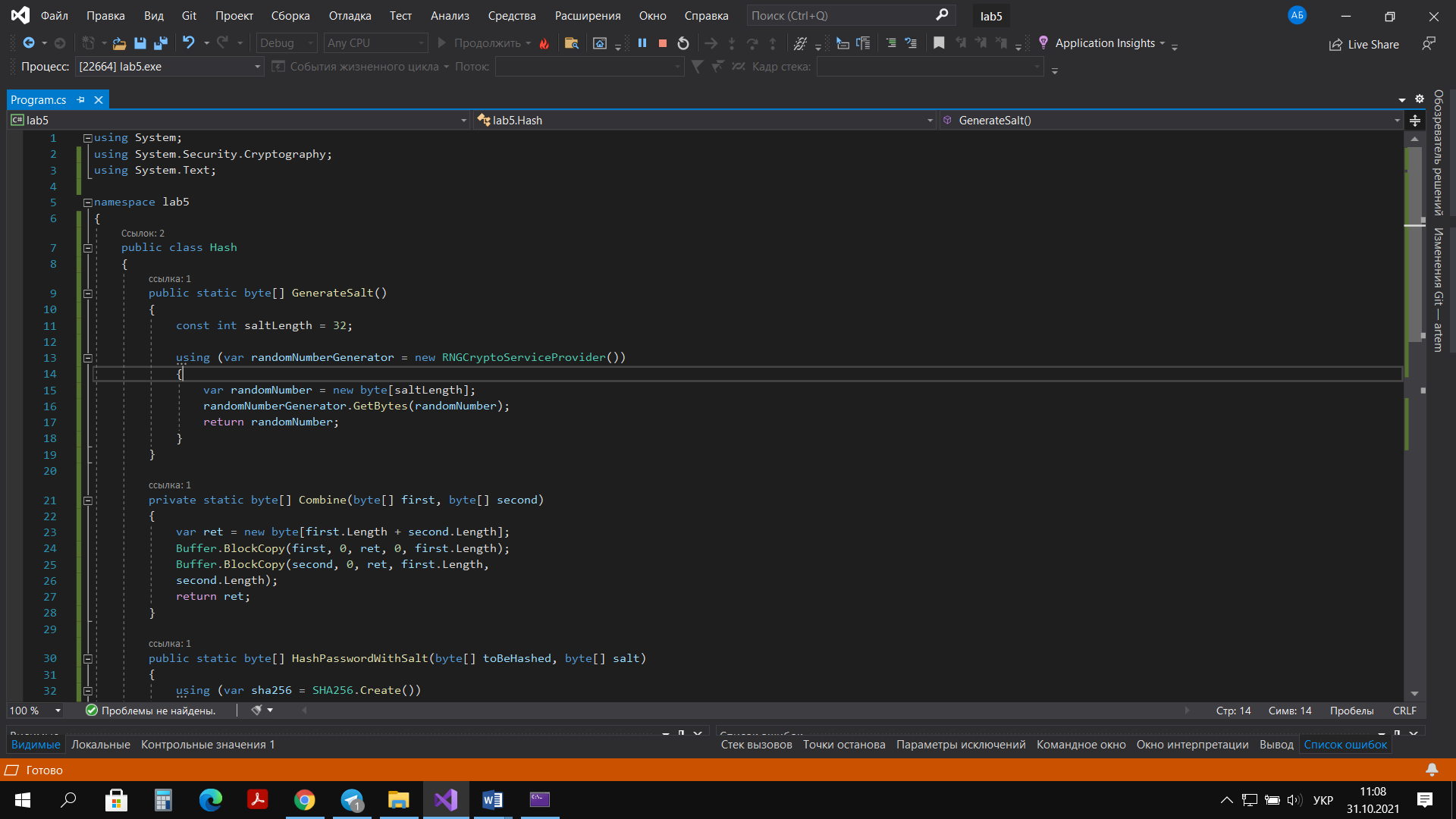
**5) Використання стандарту формування ключа на основі пароля**

Спосіб використовує пароль, «сіль» та «кількість ітерацій» значення. Значення кількості ітерацій повторює операцію хешування над паролем кілька разів, щоб створити похідний ключ для пароля, який можна зберегти в базі даних.

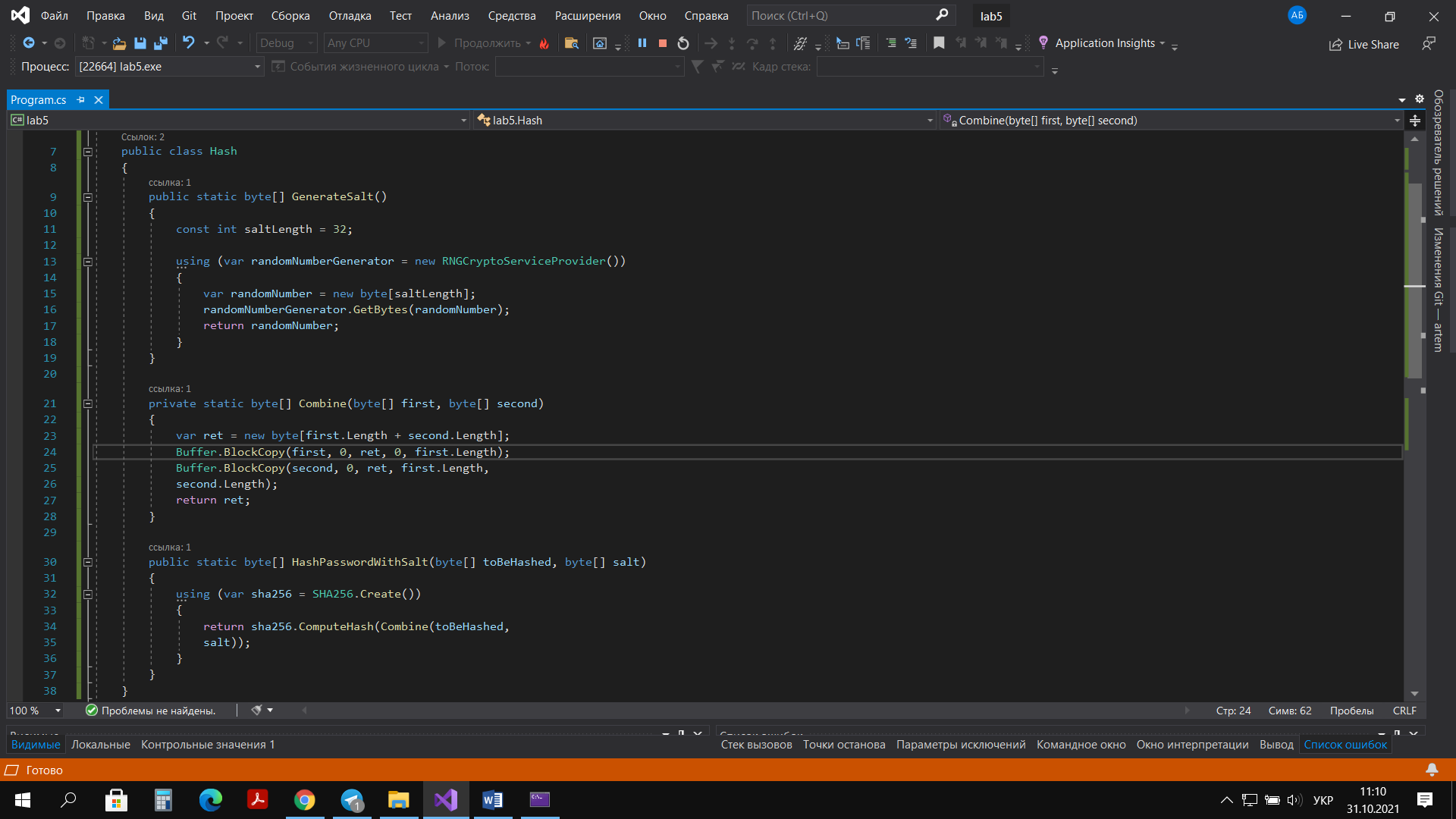
Завдання 3



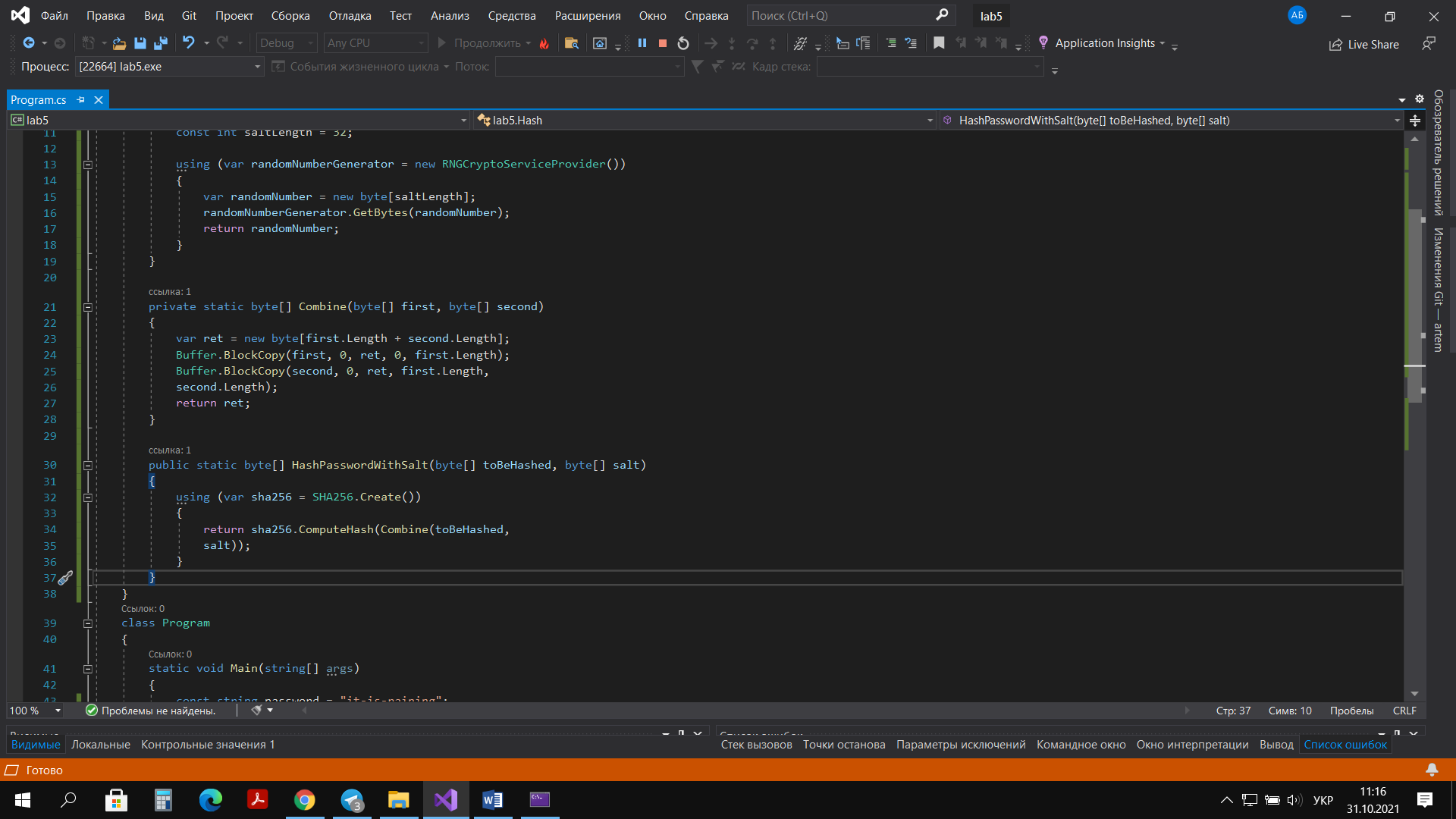
Використовуємо ці простори імен



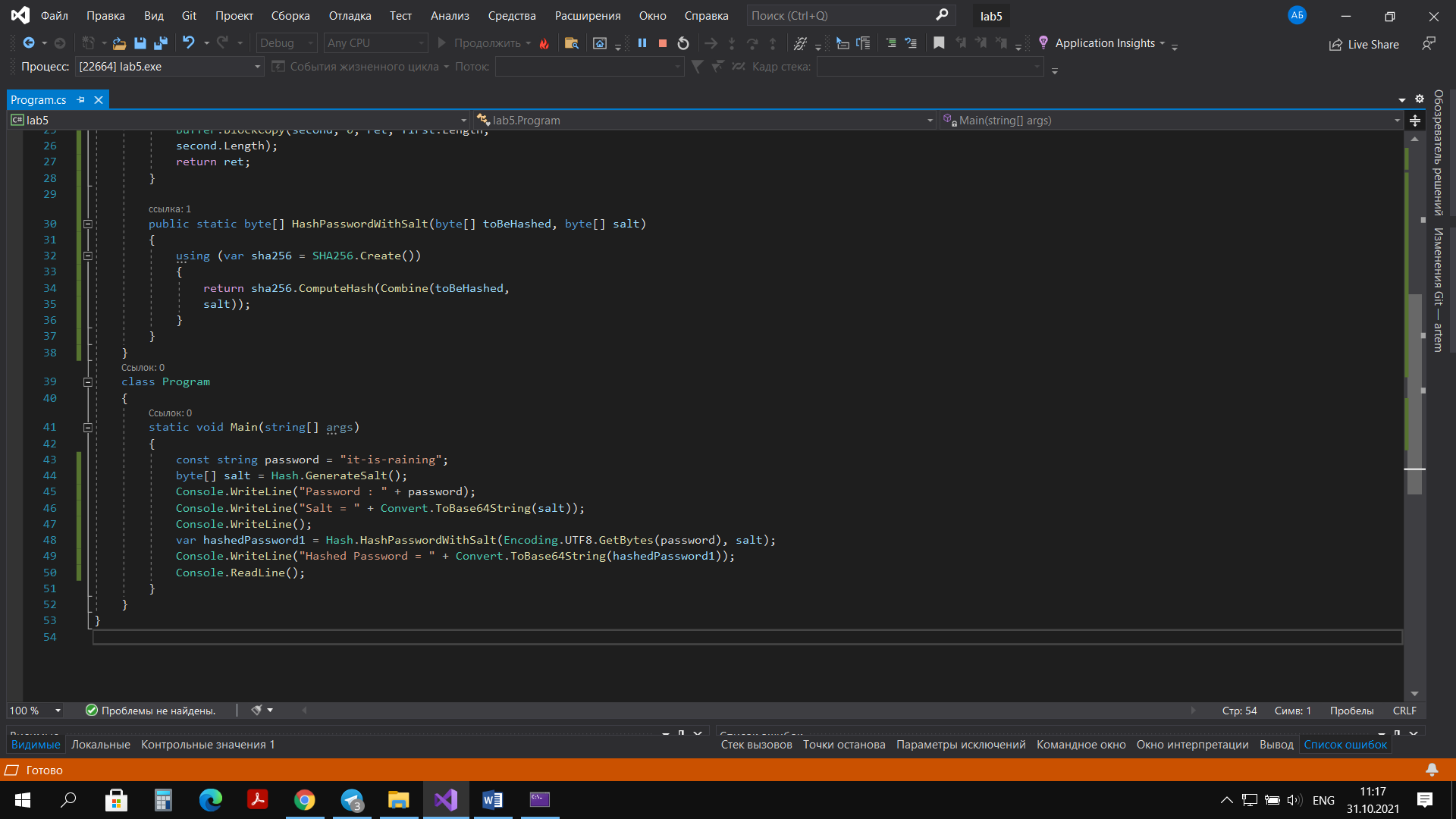
Згенеруємо «сіль» використовуючи метод RNGCryptoServiceProvider()



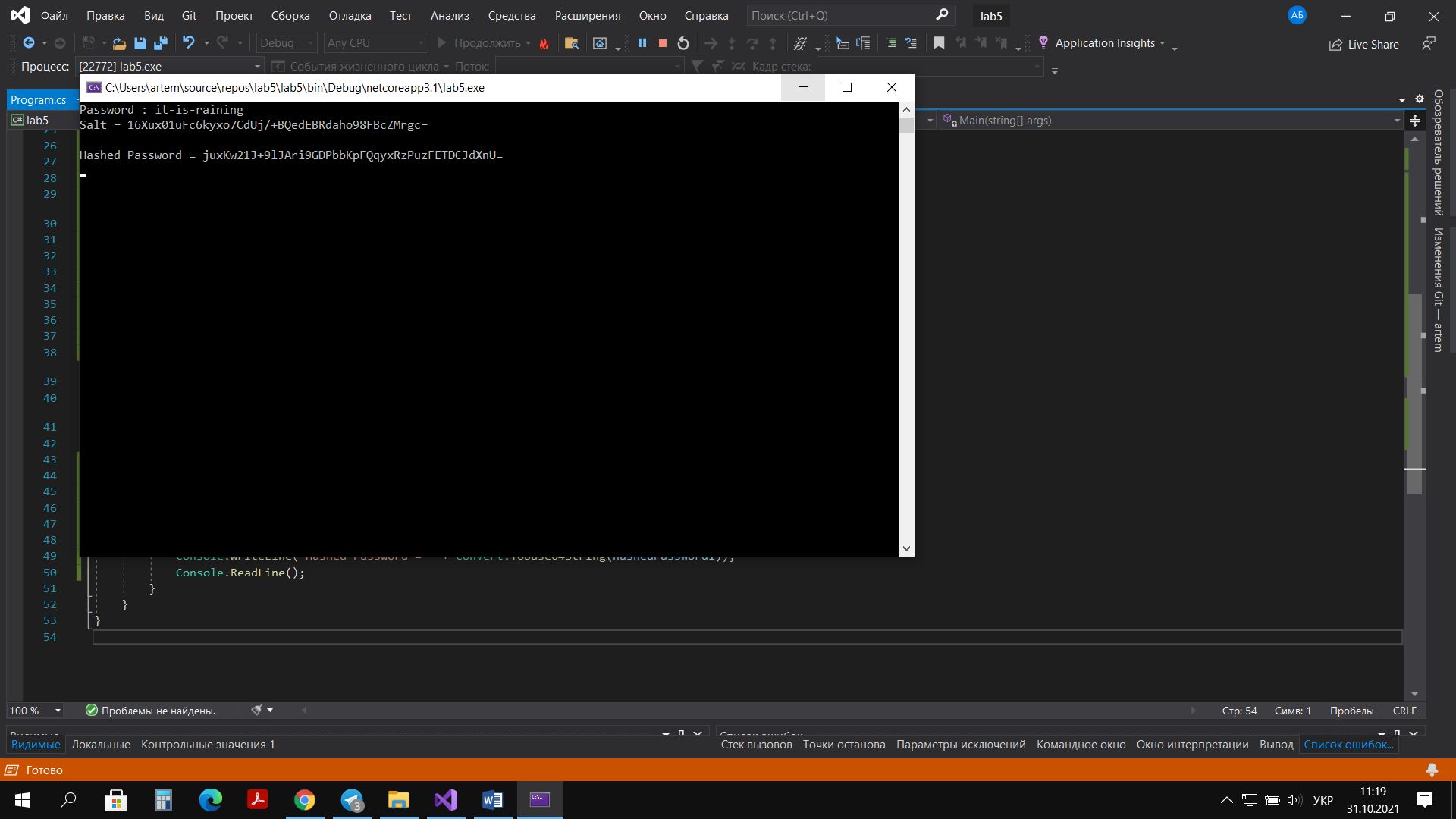
Наступним кроком потрібно скомбінувати нашу «сіль» та майбутній пароль в один масив. Тому використаємо метод Combine(). Всі операції потрібні для виконання видні на рисунку

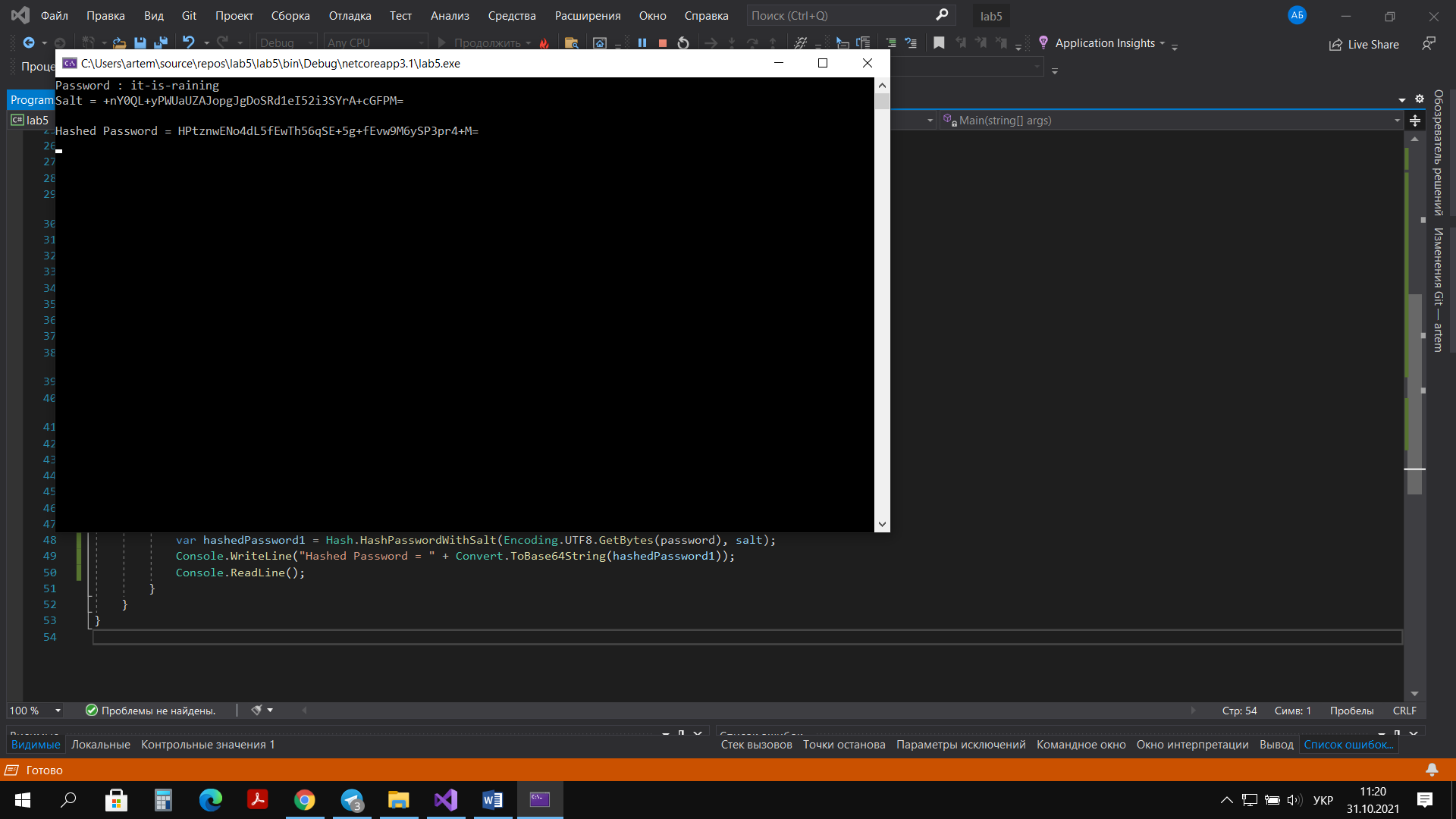


Хешуємо наш скомбінований масив з використанням алгоритму SHA-256



На фінальному кроці створюємо пароль. Виводимо на екран результати виконання хешування





Результати виконання програми. Як можна бачити, що кожен раз маємо новий хеш пароль тому, що кожен раз генерується нова «сіль»

**Код:**

using System;

using System.Security.Cryptography;

using System.Text;

namespace lab5

{

public class Hash

{

public static byte[] GenerateSalt()

{

const int saltLength = 32;

using (var randomNumberGenerator = new RNGCryptoServiceProvider())

{

var randomNumber = new byte[saltLength];

randomNumberGenerator.GetBytes(randomNumber);

return randomNumber;

}

}

private static byte[] Combine(byte[] first, byte[] second)

{

var ret = new byte[first.Length + second.Length];

Buffer.BlockCopy(first, 0, ret, 0, first.Length);

Buffer.BlockCopy(second, 0, ret, first.Length, second.Length);

return ret;

}

public static byte[] HashPasswordWithSalt(byte[] toBeHashed, byte[] salt)

{

using (var sha256 = SHA256.Create())

{

return sha256.ComputeHash(Combine(toBeHashed, salt));

}

}

}

class Program

{

static void Main(string[] args)

{

const string password = "it-is-raining";

byte[] salt = Hash.GenerateSalt();

Console.WriteLine("Password : " + password);

Console.WriteLine("Salt = " + Convert.ToBase64String(salt));

Console.WriteLine();

var hashedPassword1 = Hash.HashPasswordWithSalt(Encoding.UTF8.GetBytes(password), salt);

Console.WriteLine("Hashed Password = " + Convert.ToBase64String(hashedPassword1));

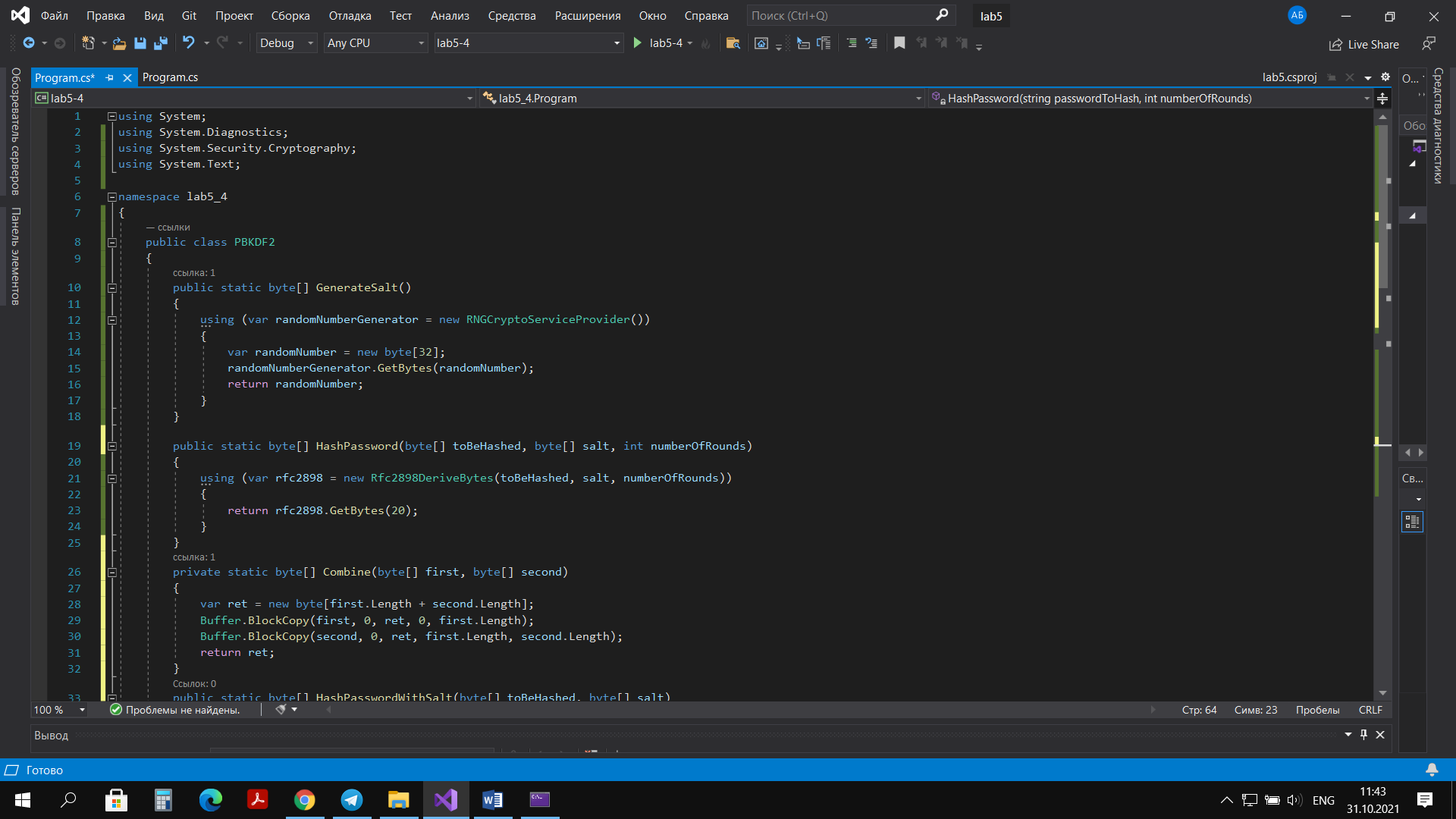
Console.ReadLine();

}

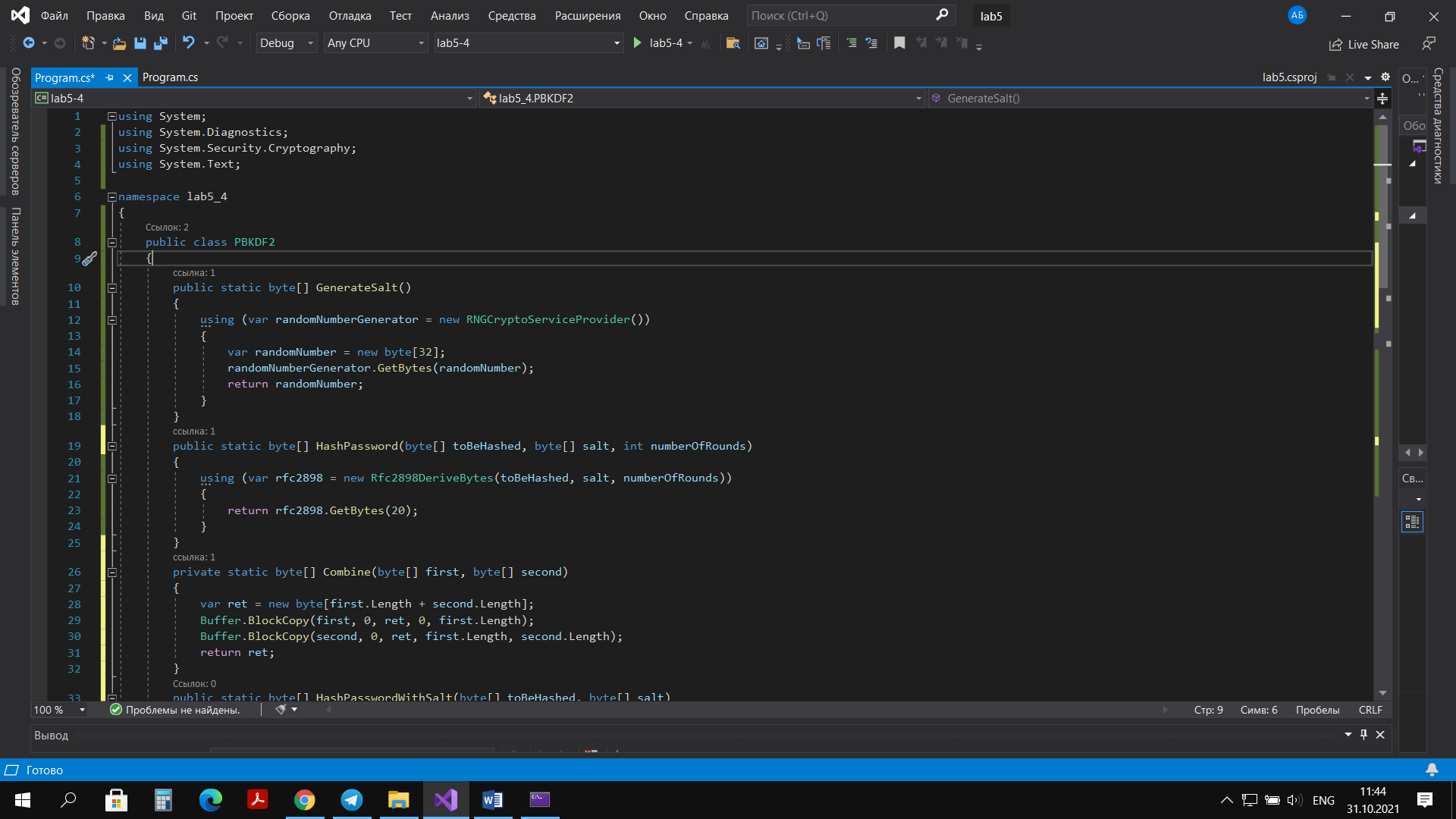
}

}

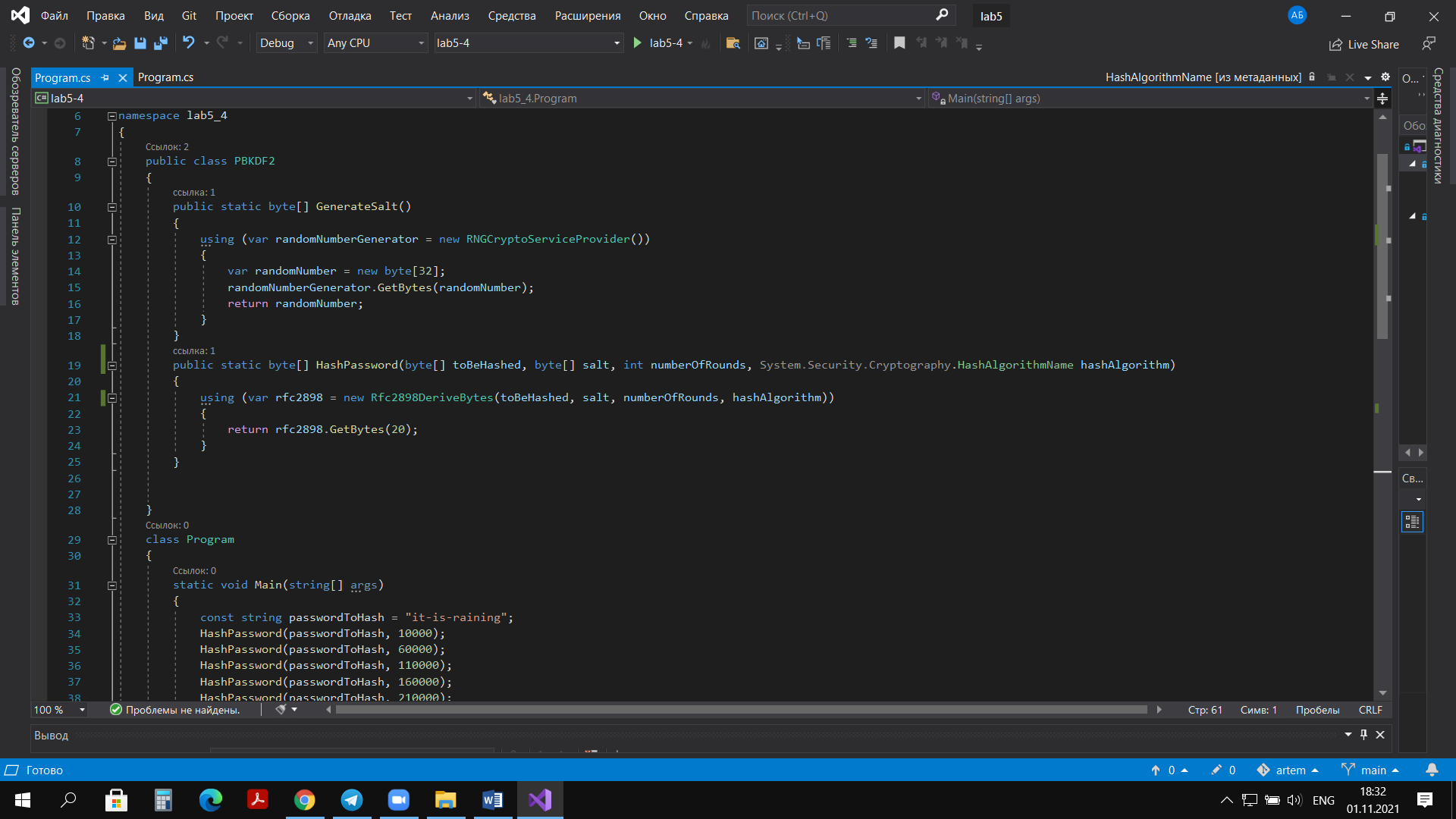
Завдання 4



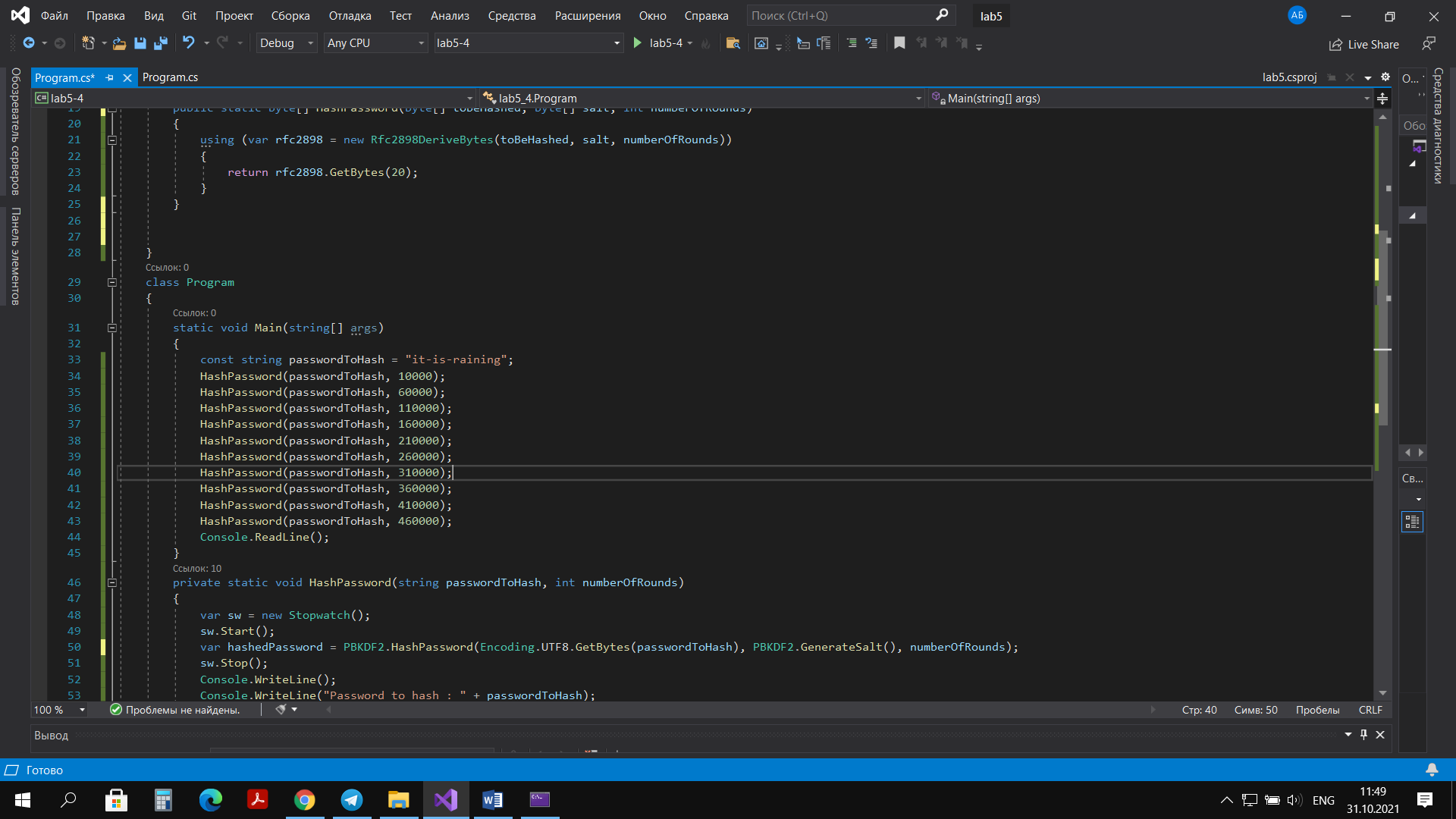
Використовуємо ці простори імен



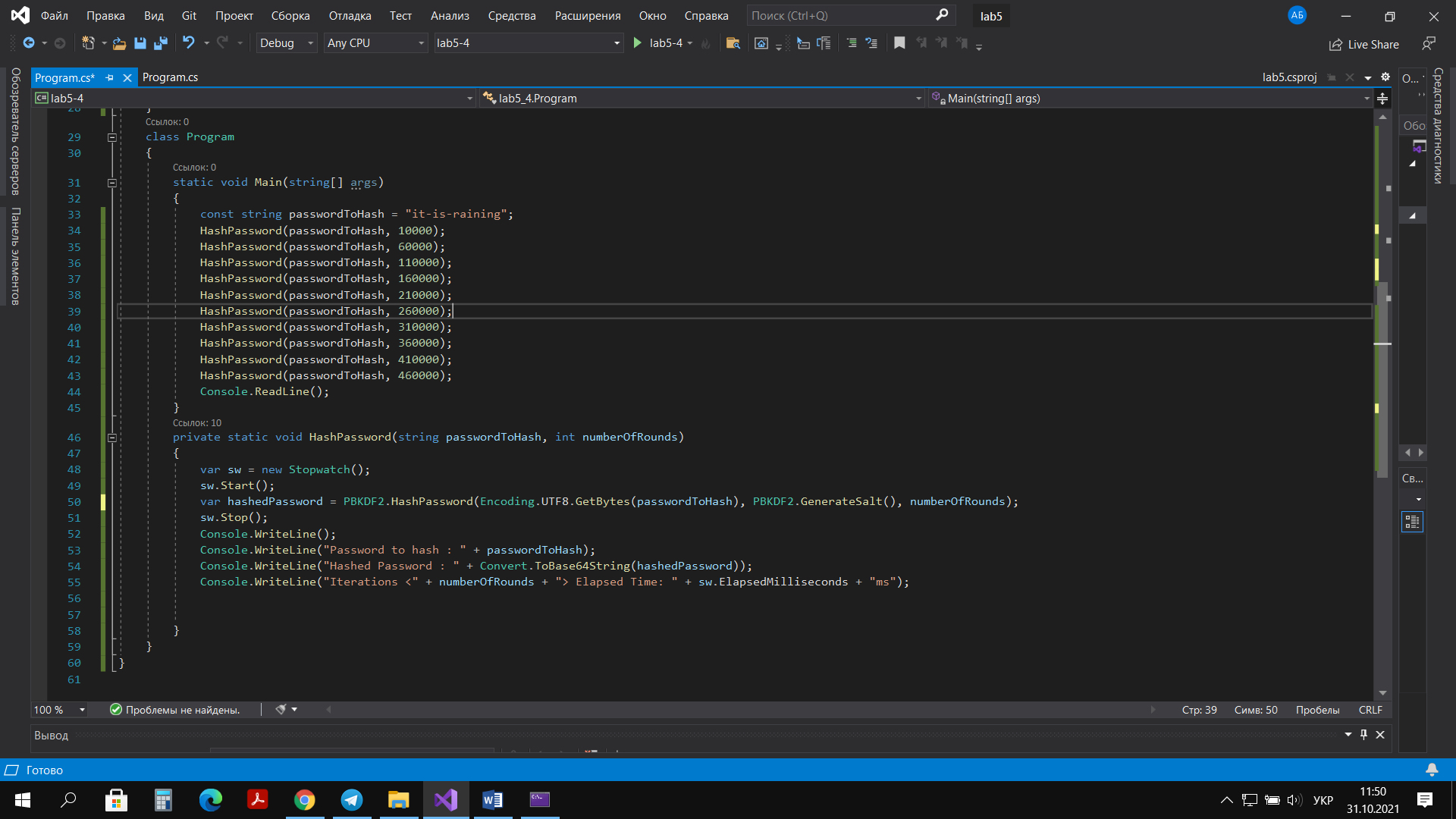
Генеруємо ключ довжиною 32 байти



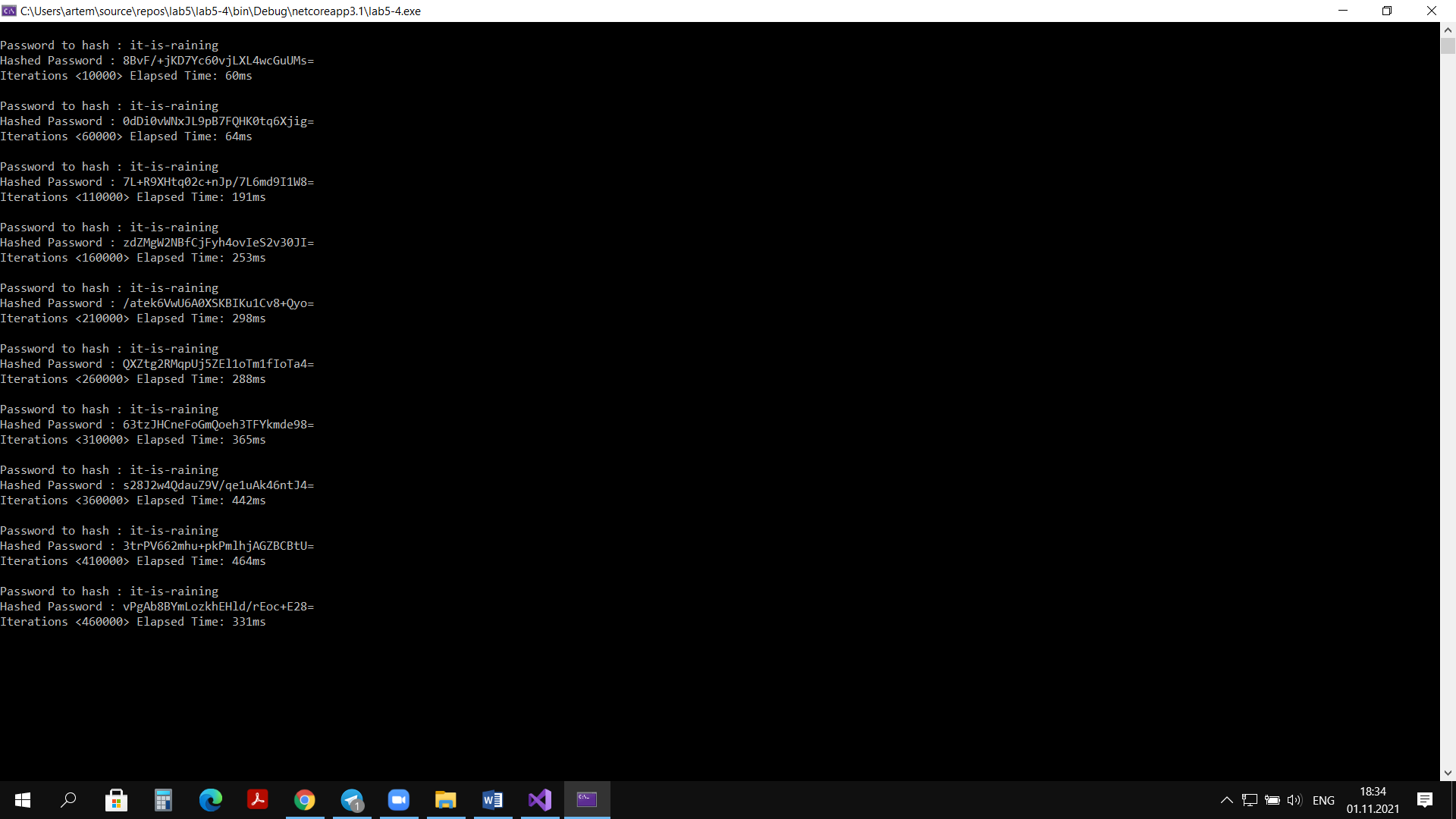
Для хешування будемо використовувати Rfc2898DeriveBytes(), який в свою чергу використовує алгоритм хешування.



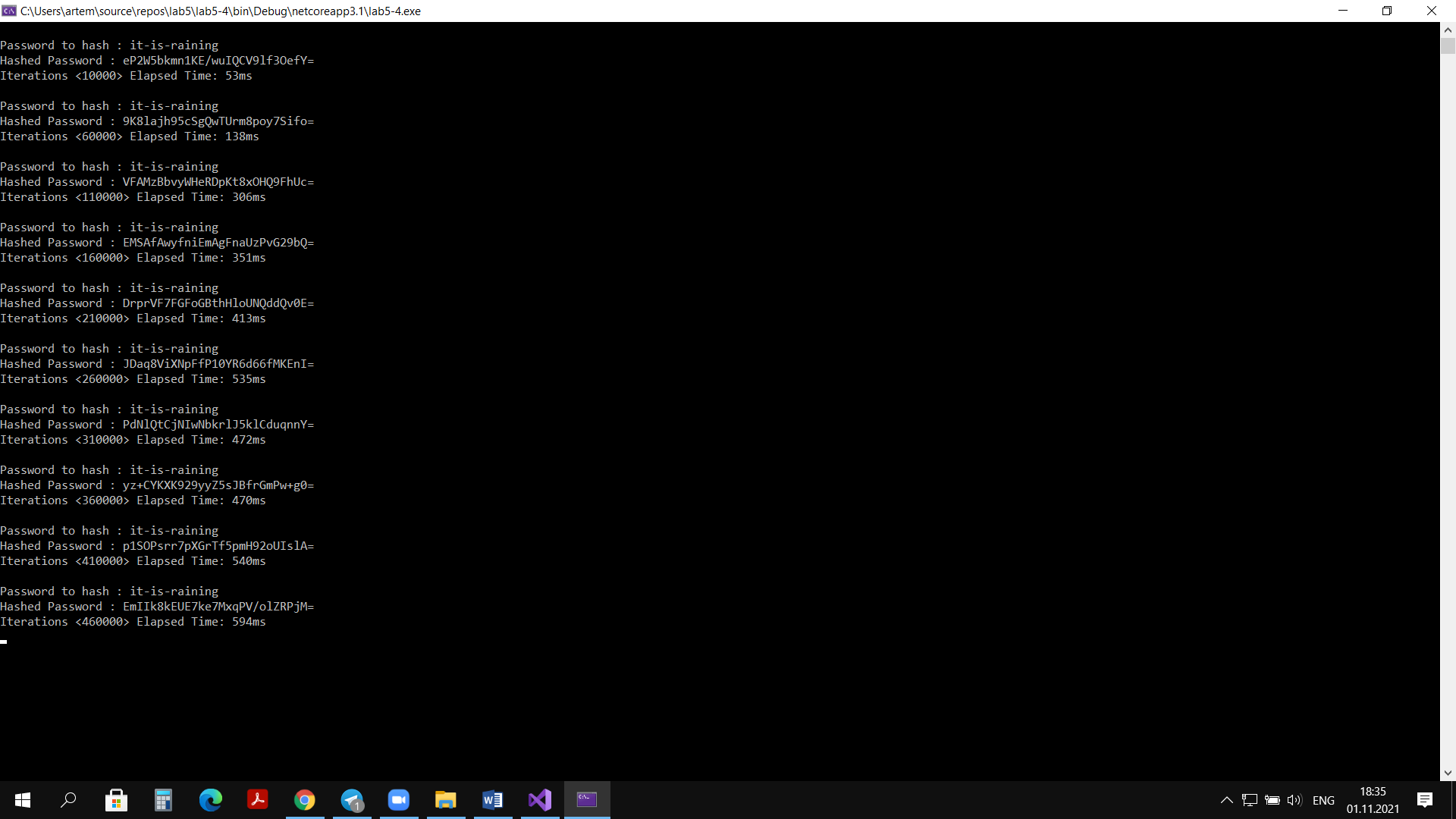
Створюємо пароль та задаємо кількість ітерацій згідно варіанту 1.



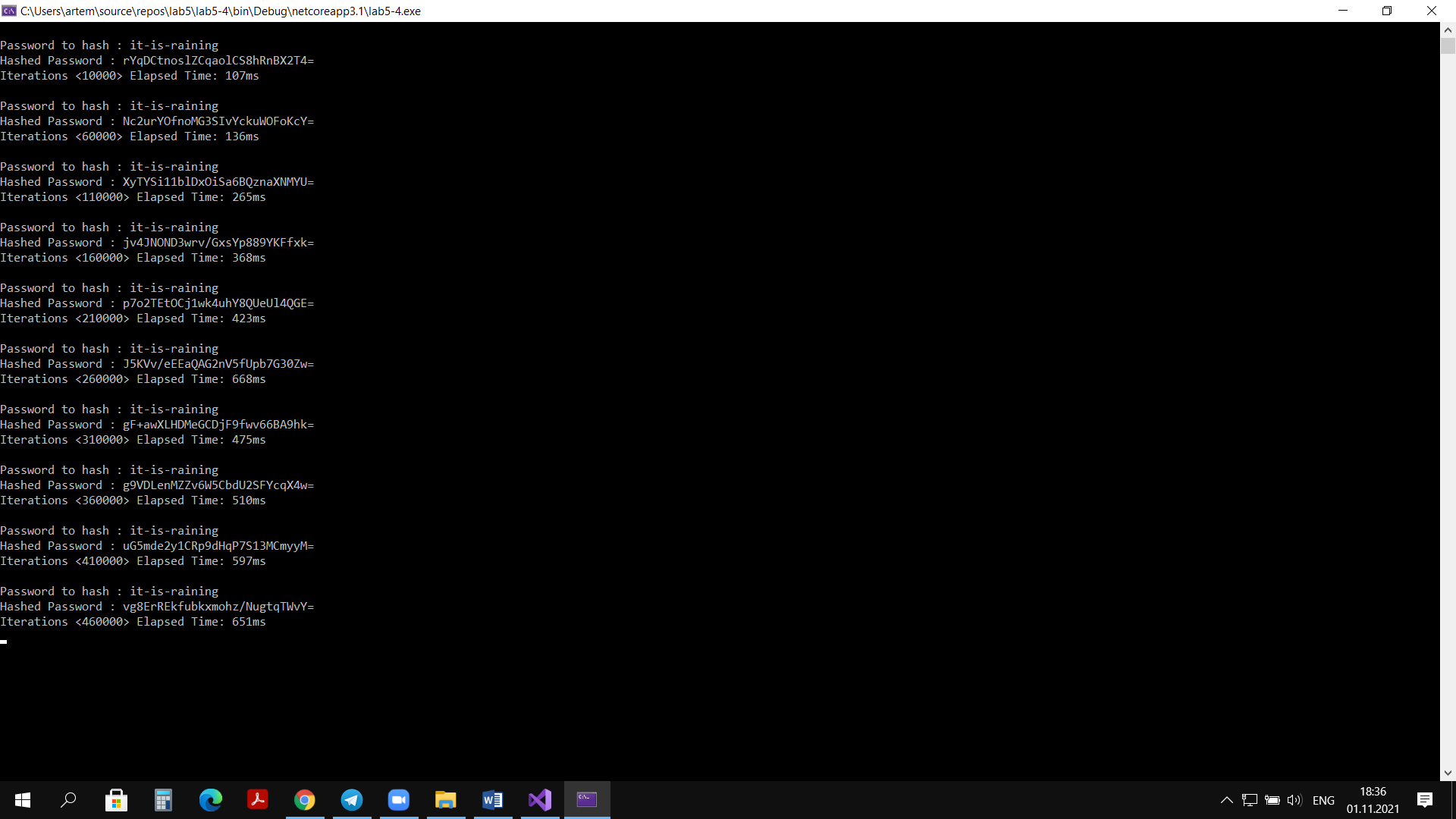
Для діагностики та аналізу часу затраченого на кількість ітерацій використовуємо команди var sw = new Stopwatch(), sw.Start(), sw.Stop();



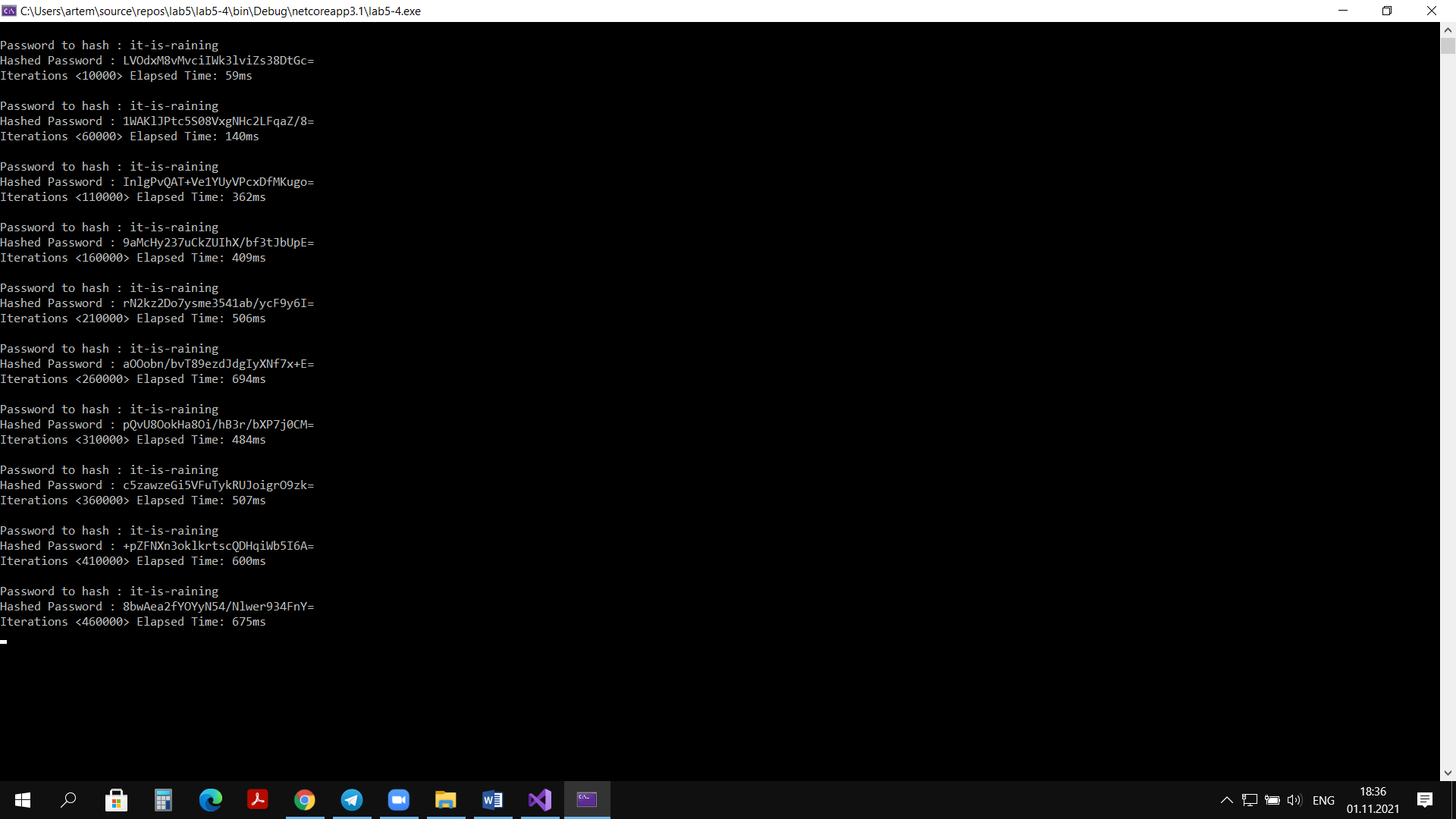
Результати виконання коду на SHA-1.



Результати виконання коду на SHA-256.



Результати виконання коду на SHA-384.



Результати виконання коду на SHA-512.

**Графік:**

**Код:**

using System;

using System.Diagnostics;

using System.Security.Cryptography;

using System.Text;

namespace lab5\_4

{

public class PBKDF2

{

public static byte[] GenerateSalt()

{

using (var randomNumberGenerator = new RNGCryptoServiceProvider())

{

var randomNumber = new byte[32];

randomNumberGenerator.GetBytes(randomNumber);

return randomNumber;

}

}

public static byte[] HashPassword(byte[] toBeHashed, byte[] salt, int numberOfRounds, System.Security.Cryptography.HashAlgorithmName hashAlgorithm)

{

using (var rfc2898 = new Rfc2898DeriveBytes(toBeHashed, salt, numberOfRounds, hashAlgorithm))

{

return rfc2898.GetBytes(20);

}

}

}

class Program

{

static void Main(string[] args)

{

const string passwordToHash = "it-is-raining";

HashPassword(passwordToHash, 10000);

HashPassword(passwordToHash, 60000);

HashPassword(passwordToHash, 110000);

HashPassword(passwordToHash, 160000);

HashPassword(passwordToHash, 210000);

HashPassword(passwordToHash, 260000);

HashPassword(passwordToHash, 310000);

HashPassword(passwordToHash, 360000);

HashPassword(passwordToHash, 410000);

HashPassword(passwordToHash, 460000);

Console.ReadLine();

}

private static void HashPassword(string passwordToHash, int numberOfRounds)

{

var sw = new Stopwatch();

sw.Start();

var hashedPassword = PBKDF2.HashPassword(Encoding.UTF8.GetBytes(passwordToHash), PBKDF2.GenerateSalt(), numberOfRounds, HashAlgorithmName.SHA512);

sw.Stop();

Console.WriteLine();

Console.WriteLine("Password to hash : " + passwordToHash);

Console.WriteLine("Hashed Password : " + Convert.ToBase64String(hashedPassword));

Console.WriteLine("Iterations <" + numberOfRounds + "> Elapsed Time: " + sw.ElapsedMilliseconds + "ms");

}

}

}