**Звіт**

**До практичної роботи**

**з дисципліни «Основи інформаційної безпеки»**

Студента Київського Національного Університету ім. Тараса Шевченко

Переверзева Олексія Сергійовича

Факультет інформаційних технологій

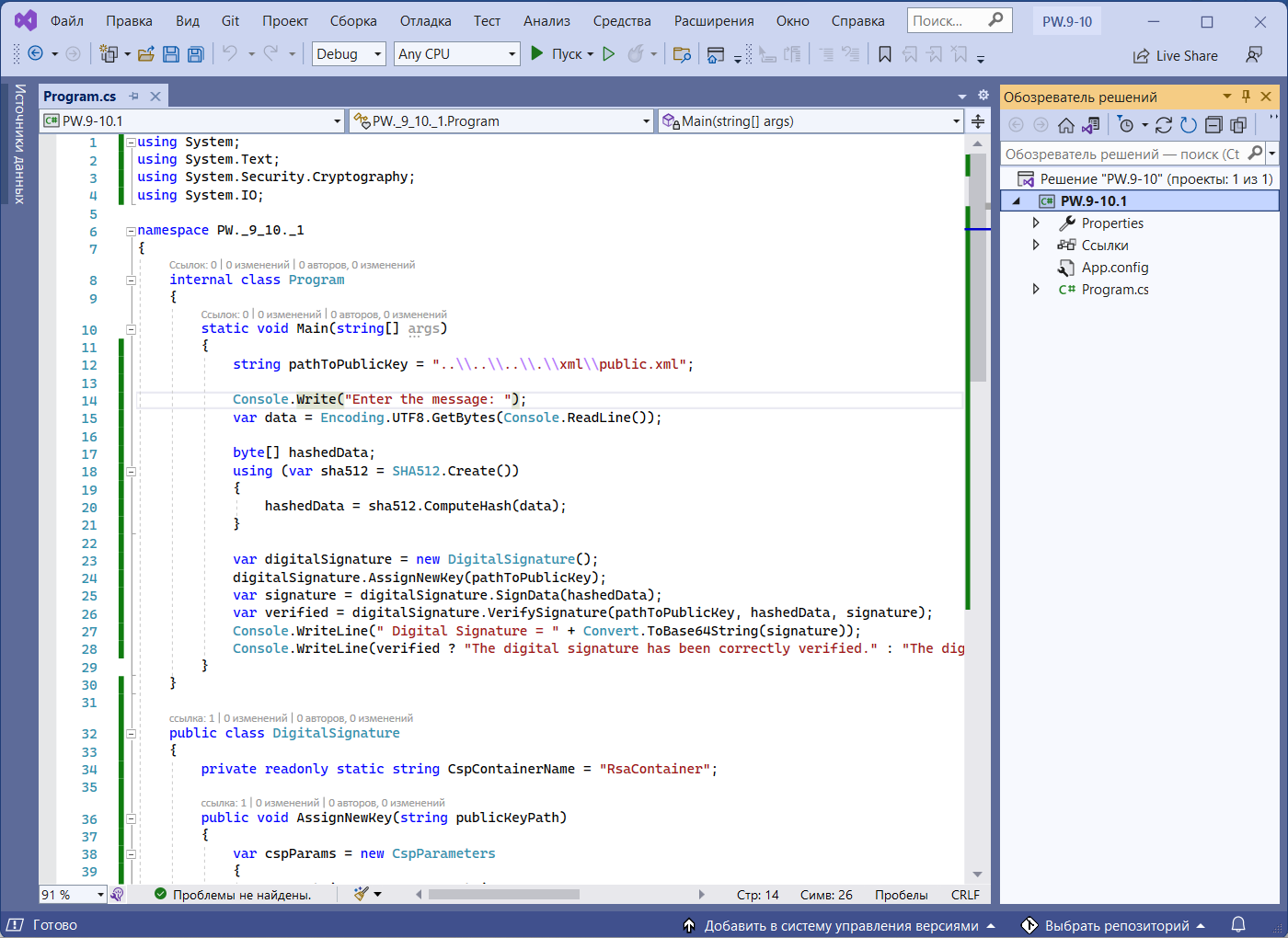
Група МІТ-21

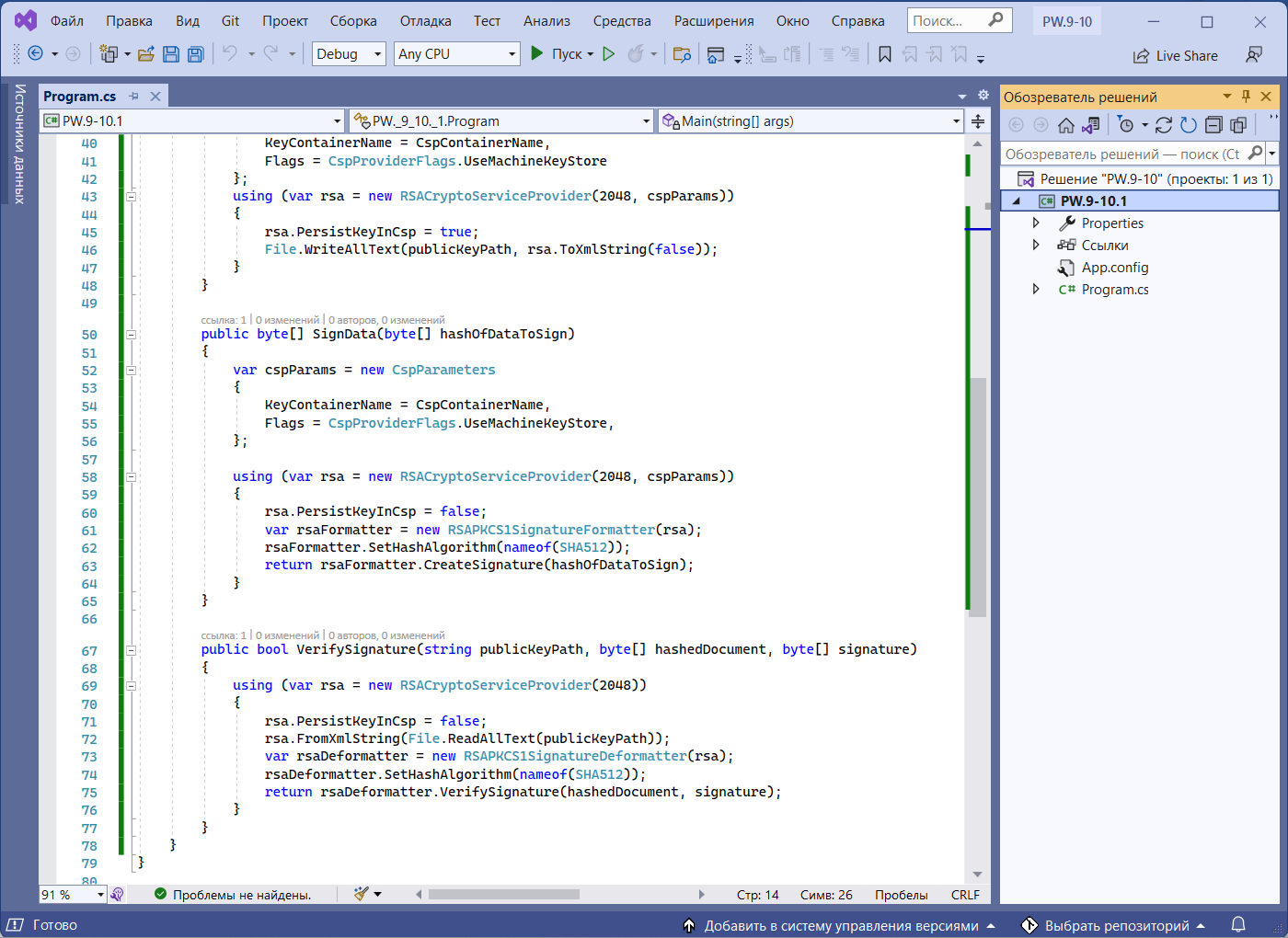


**Практичне заняття №9-10**

**Тема:** Електронний цифровий підпис

1. Написати програму, яка виконує підписування повідомлення та перевірку ЕЦП з використанням алгоритму асиметричного шифрування RSA. Відкритий ключ імпортується із XML-файлу, секретний – береться із контейнера.





Розглянемо спочатку public class DigitalSignature:

private readonly static string CspContainerName = "RsaContainer";

public void AssignNewKey(string publicKeyPath)

{

var cspParams = new CspParameters

{

KeyContainerName = CspContainerName,

Flags = CspProviderFlags.UseMachineKeyStore

};

using (var rsa = new RSACryptoServiceProvider(2048, cspParams))

{

rsa.PersistKeyInCsp = true;

File.WriteAllText(publicKeyPath, rsa.ToXmlString(false));

}

}

В цій частині коду ми створюємо два асиметричні ключі і зберігаємо їх у контейнері і публічний записуємо у .xml файл.

Крім генерації ключів, для роботи цифрового підпису в .NET/.NET Core крім RSACryptoServiceProvider потрібні ще два класи. Ці класи — RSAPKCS1SignatureFormatter і RSAPKCS1SignatureDeformatter, які підписують і перевіряють наші дані відповідно.

Цей метод відповідно обчислює ЕЦП, передаючи отриманий хеш з інформації, для якої ми хочемо отримати ЕЦП:

public byte[] SignData(byte[] hashOfDataToSign)

{

var cspParams = new CspParameters

{

KeyContainerName = CspContainerName,

Flags = CspProviderFlags.UseMachineKeyStore,

};

using (var rsa = new RSACryptoServiceProvider(2048, cspParams))

{

rsa.PersistKeyInCsp = false;

var rsaFormatter = new RSAPKCS1SignatureFormatter(rsa);

rsaFormatter.SetHashAlgorithm(nameof(SHA512));

return rsaFormatter.CreateSignature(hashOfDataToSign);

}

}

А цей перевіряє ЕЦП, передаючи шлях до файлу відкритого ключа, повідомлення, та ЕЦП, що перевіряється:

public bool VerifySignature(string publicKeyPath, byte[] hashedDocument, byte[] signature)

{

using (var rsa = new RSACryptoServiceProvider(2048))

{

rsa.PersistKeyInCsp = false;

rsa.FromXmlString(File.ReadAllText(publicKeyPath));

var rsaDeformatter = new RSAPKCS1SignatureDeformatter(rsa);

rsaDeformatter.SetHashAlgorithm(nameof(SHA512));

return rsaDeformatter.VerifySignature(hashedDocument, signature);

}

}

Основний код:

Вказуємо шлях до файлу з відкритим ключем:

string pathToPublicKey = "..\\..\\..\\.\\xml\\public.xml";

Далі виконуємо запит на введення інформації та записуємо у змінну, перетворивши дані на масив байтів:

Console.Write("Enter the message: ");

var data = Encoding.UTF8.GetBytes(Console.ReadLine());

та отримуємо хеш-код:

byte[] hashedData;

using (var sha512 = SHA512.Create())

{

hashedData = sha512.ComputeHash(data);

}

Далі ми створюємо екземпляр нашого класу DigitalSignature, який містить наші методи AssignNewKey, SignData та VerifySignature.

var digitalSignature = new DigitalSignature();

та створюємо нові ключі:

digitalSignature.AssignNewKey(pathToPublicKey);

Далі ми викликаємо SignData і передаємо хеш документа, який повертає наш цифровий підпис у вигляді масиву байтів.

var signature = digitalSignature.SignData(hashedData);

Одразу після цього ми викликаємо VerifySignature, щоб перевірити, чи він дійсний, що й повинно бути, оскільки ми не змінили оригінальний документ або хеш.

var verified = digitalSignature.VerifySignature(pathToPublicKey, hashedData, signature);

Виводимо у консоль ЕЦП:

Console.WriteLine(" Digital Signature = " + Convert.ToBase64String(signature));

Та за допомогою тернарної умовної операції виводимо результат перевірки у консоль:

Console.WriteLine(verified ? "The digital signature has been correctly verified." : "The digital signature has NOT been correctly verified.");

Виконання програми:

