**Звіт**

**До практичної роботи**

**з дисципліни «Основи інформаційної безпеки»**

Студента Київського Національного Університету ім. Тараса Шевченко

Переверзева Олексія Сергійовича

Факультет інформаційних технологій

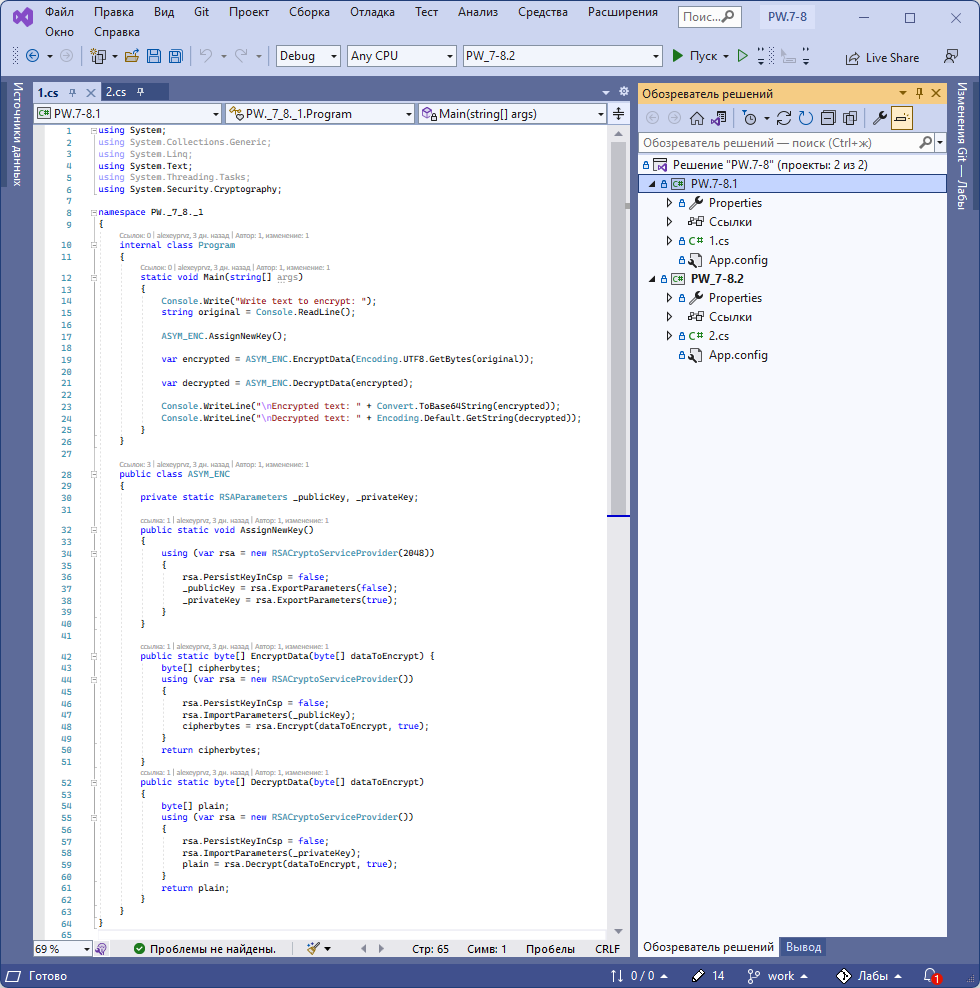
Група МІТ-21



**Практичне заняття №7-8**

**Тема:** Асиметричне шифрування як засіб забезпечення конфіденційності інформації

1. Написати програму, яка виконує зашифровування та розшифровування даних з використанням алгоритмів асиметричного шифрування RSA. Пара ключів зберігається у пам’яті.



Спочатку розглянемо класс public class ASYM\_ENC.

public class ASYM\_ENC

{

private static RSAParameters \_publicKey, \_privateKey;

public static void AssignNewKey()

{

using (var rsa = new RSACryptoServiceProvider(2048))

{

rsa.PersistKeyInCsp = false;

\_publicKey = rsa.ExportParameters(false);

\_privateKey = rsa.ExportParameters(true);

}

}

public static byte[] EncryptData(byte[] dataToEncrypt) {

byte[] cipherbytes;

using (var rsa = new RSACryptoServiceProvider())

{

rsa.PersistKeyInCsp = false;

rsa.ImportParameters(\_publicKey);

cipherbytes = rsa.Encrypt(dataToEncrypt, true);

}

return cipherbytes;

}

public static byte[] DecryptData(byte[] dataToEncrypt)

{

byte[] plain;

using (var rsa = new RSACryptoServiceProvider())

{

rsa.PersistKeyInCsp = false;

rsa.ImportParameters(\_privateKey);

plain = rsa.Decrypt(dataToEncrypt, true);

}

return plain;

}

}

Розглянемо метод AssignNewKey(), який створює ключі RSA у пам’яті:

private static RSAParameters \_publicKey, \_privateKey;

public static void AssignNewKey()

{

using (var rsa = new RSACryptoServiceProvider(2048))

{

rsa.PersistKeyInCsp = false;

\_publicKey = rsa.ExportParameters(false);

\_privateKey = rsa.ExportParameters(true);

}

}

Допустимі довжини для параметра key size, який ви передаєте конструктору, складають 1024 біта, 2048 біт і 4096 біт. У цьому методі ми створюємо відкритий та приватний ключі та зберігаємо їх в об'єктах у пам'яті. Властивості PersistKeyInCsp для цього прикладу йому присвоюємо значення false. Щоб експортувати відкритий ключ, ви передаєте значення false до параметрів експорту; щоб експортувати приватний ключ, ви передаєте значення true. Якщо ви використовуєте. Net Core, то цей метод створення ключів за допомогою RSACryptoServiceProvider працює міжплатформно в Windows, macOS та Linux, що є єдиним підтримуваним способом програмного створення ключів RSA на всіх трьох платформах.

Далі розглянемо методи шифрування та дешифрування.

Метод шифрування EncryptData:

public static byte[] EncryptData(byte[] dataToEncrypt) {

byte[] cipherbytes;

using (var rsa = new RSACryptoServiceProvider())

{

rsa.PersistKeyInCsp = false;

rsa.ImportParameters(\_publicKey);

cipherbytes = rsa.Encrypt(dataToEncrypt, true);

}

return cipherbytes;

}

Передаємо свої дані для шифрування як байтовий масив, а потім створюєте новий екземпляр RSACryptoServiceProvider. Імпортуємо відкритий ключ для шифрування за допомогою методу ImportParameters, і ви повинні переконатися, що PersistKeyInCsp має значення false. Потім ви викликаєте метод Encrypt для Об'єкта RSACryptoServiceProvider. Другий параметр методу шифрування призначений для оптимального асиметричного шифрування заповнення (OAEP), що є схемою заповнення, що використовується в RSA. OEAP - це форма мережі Feistel, яка призначена для додавання елемента випадковості до процесу шифрування, що допомагає запобігти частковій розшифровці зашифрованого тексту, гарантуючи, що зловмисник не зможе відновити будь-яку частину відкритого тексту. Як тільки викликаємо Encrypt, отримаємо назад байтовий масив, що містить ваші зашифровані дані.

Метод шифрування DecryptData:

public static byte[] DecryptData(byte[] dataToEncrypt)

{

byte[] plain;

using (var rsa = new RSACryptoServiceProvider())

{

rsa.PersistKeyInCsp = false;

rsa.ImportParameters(\_privateKey);

plain = rsa.Decrypt(dataToEncrypt, true);

}

return plain;

}

Код виглядає майже ідентично методу шифрування, за винятком того, що ми завантажуємо закритий ключ, замість відкритого ключа, і викликаємо Decrypt замість Encrypt. За допомогою RSA ми шифруємо, використовуючи відкриті ключі наших отримувачів, яким вони можуть вільно поділитися з нами, і вони розшифровують повідомлення, використовуючи свій приватний ключ, який знають лише вони.

Розглянемо основний код програми:

Спочатку виконуємо запит на введення повідомлення для шифрування:

Console.Write("Write text to encrypt: ");

string original = Console.ReadLine();

Далі створюємо ключі:

ASYM\_ENC.AssignNewKey();

Після цього шифруємо:

var encrypted = ASYM\_ENC.EncryptData(Encoding.UTF8.GetBytes(original));

та дешифруємо повідомлення

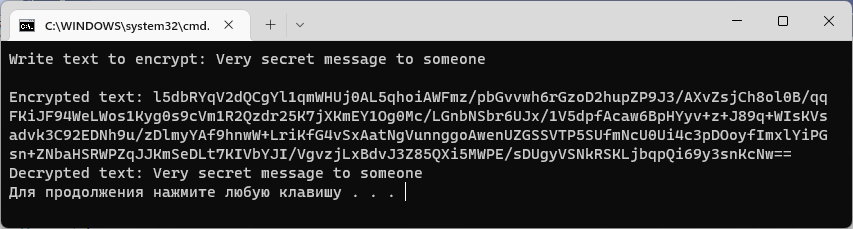
var decrypted = ASYM\_ENC.DecryptData(encrypted);

і виводимо все у консоль

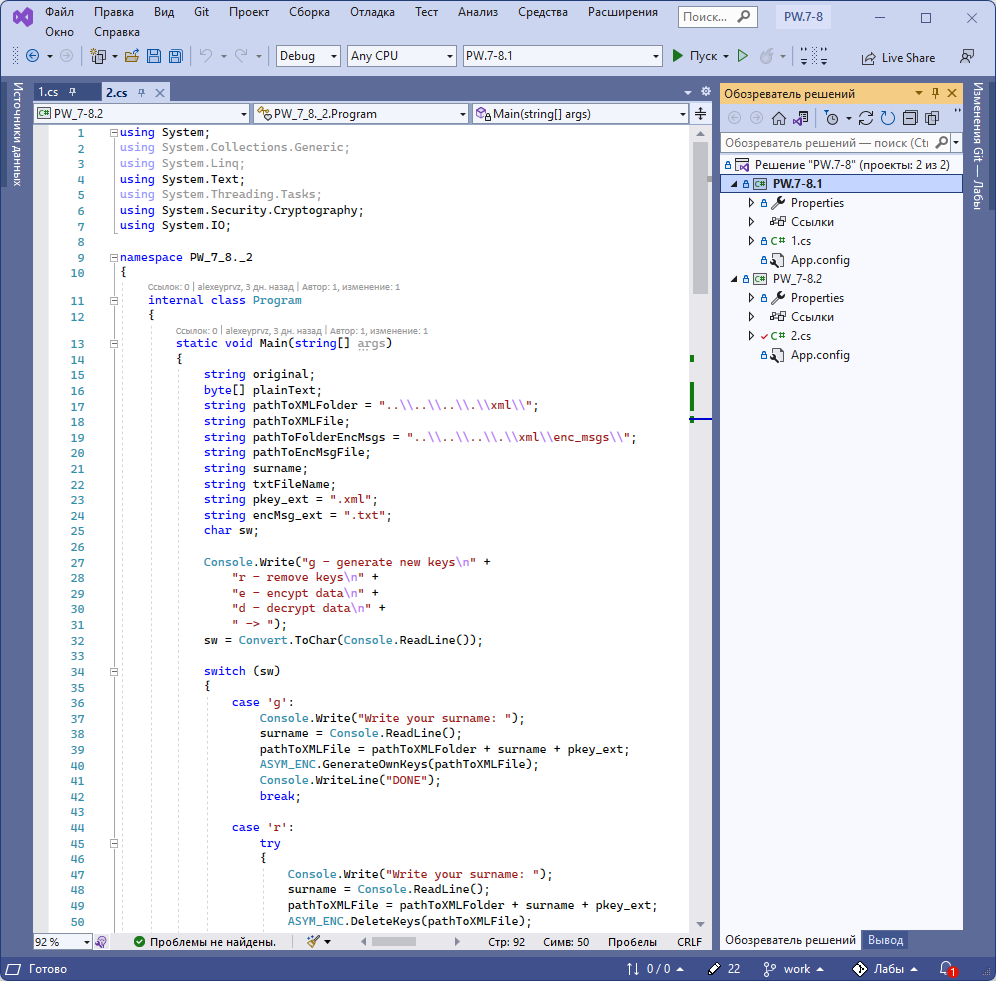
Console.WriteLine("\nEncrypted text: " + Convert.ToBase64String(encrypted));

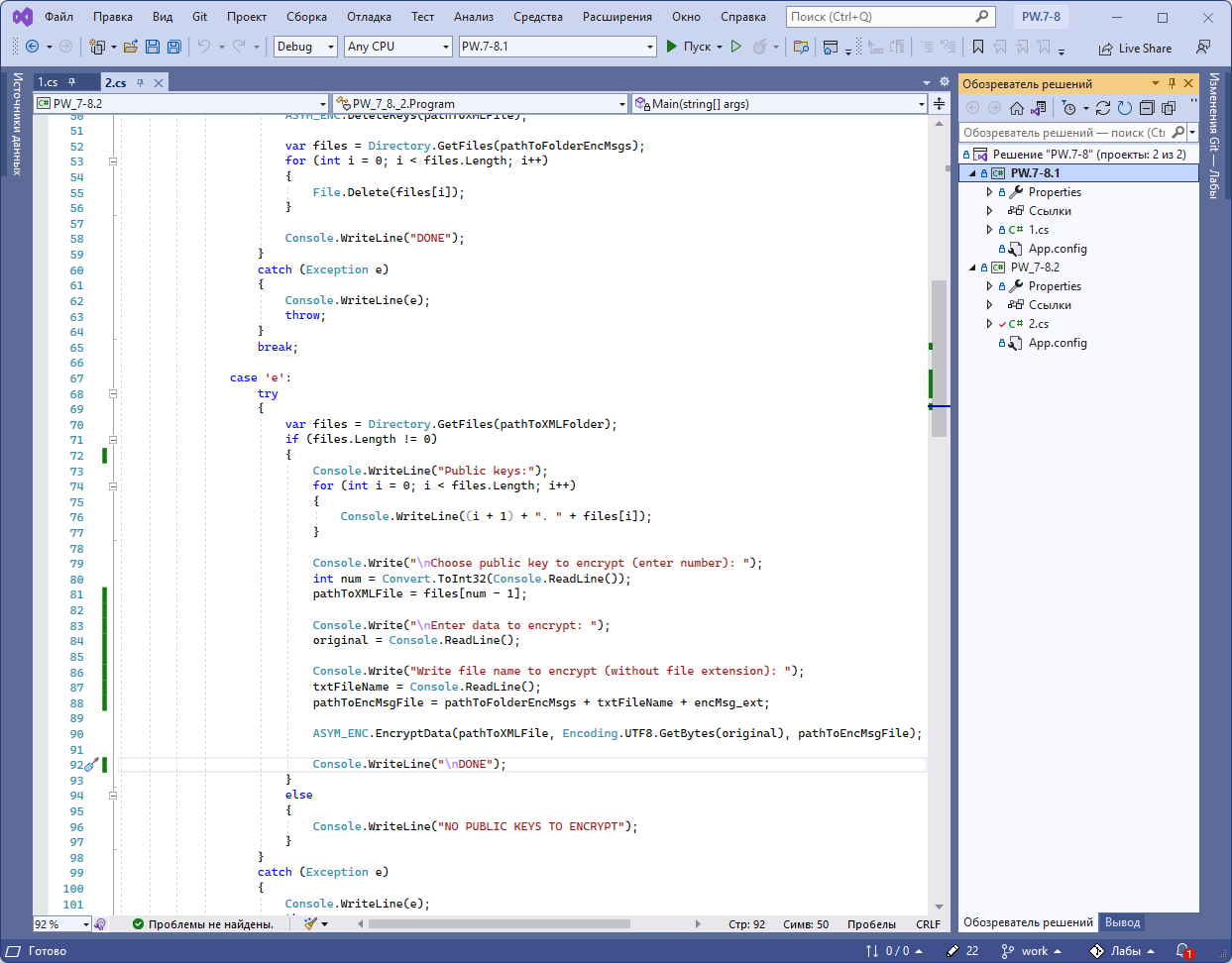
Console.WriteLine("\nDecrypted text: " + Encoding.Default.GetString(decrypted));

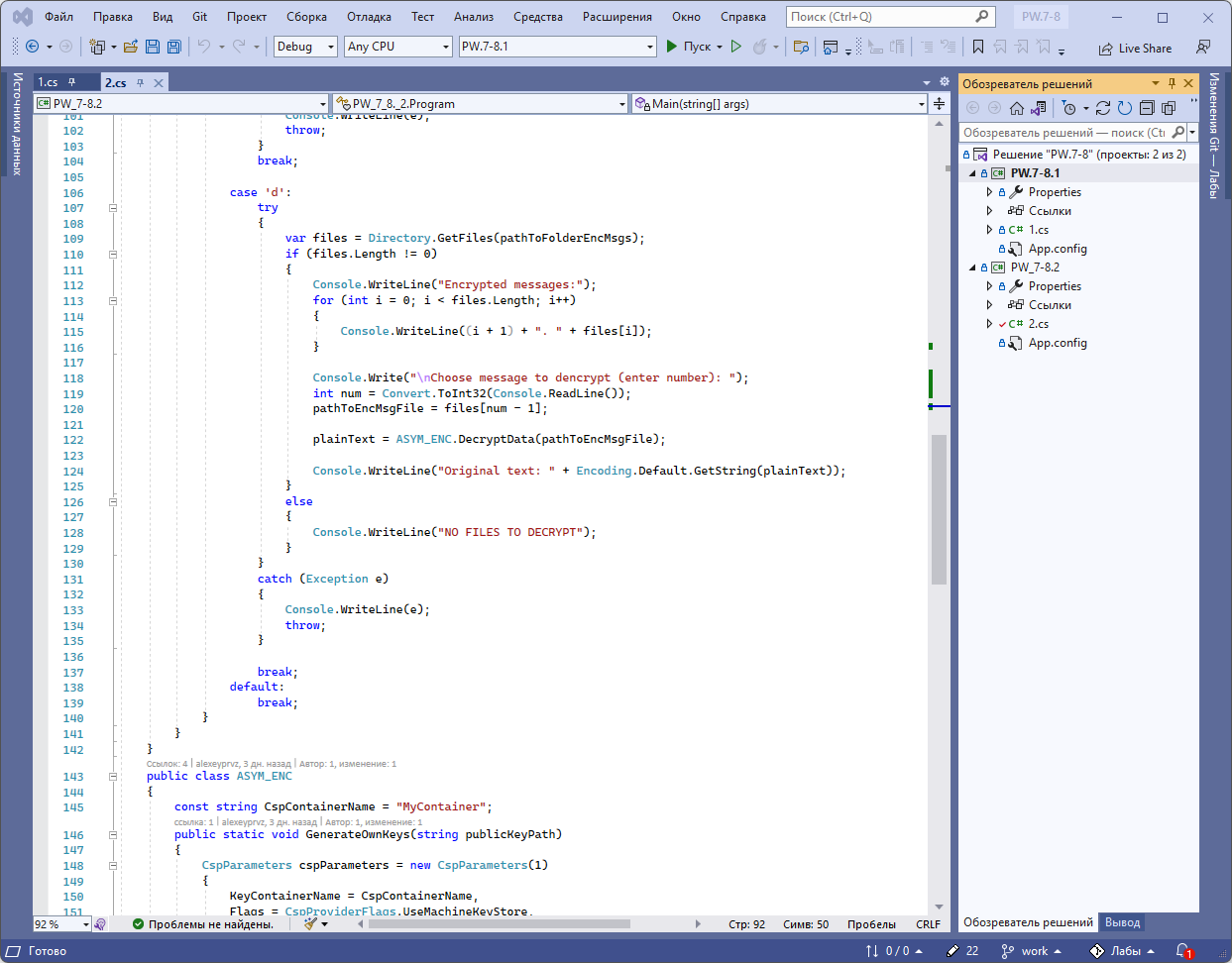
Виконання програми:

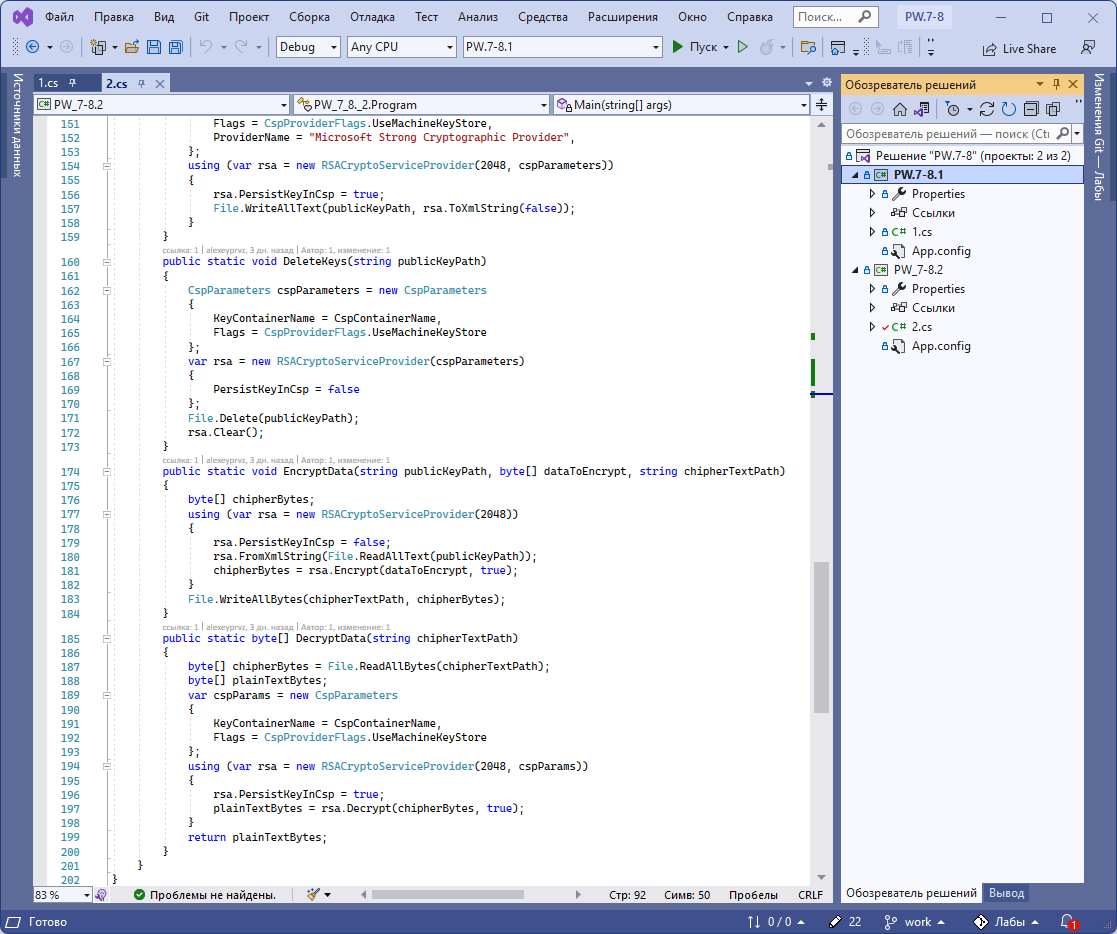


1. Для програми з п.1. реалізувати можливість збереження відкритого ключа у файлі. Реалізувати можливість зашифровувати повідомлення за допомогою файлів відкритих ключів інших користувачів.









Також розглянемо різницю класу ASYM\_ENC від попереднього завдання.

public class ASYM\_ENC

{

const string CspContainerName = "MyContainer";

public static void GenerateOwnKeys(string publicKeyPath)

{

CspParameters cspParameters = new CspParameters(1)

{

KeyContainerName = CspContainerName,

Flags = CspProviderFlags.UseMachineKeyStore,

ProviderName = "Microsoft Strong Cryptographic Provider",

};

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

{

rsa.PersistKeyInCsp = true;

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

}

}

public static void DeleteKeys(string publicKeyPath)

{

CspParameters cspParameters = new CspParameters

{

KeyContainerName = CspContainerName,

Flags = CspProviderFlags.UseMachineKeyStore

};

var rsa = new RSACryptoServiceProvider(cspParameters)

{

PersistKeyInCsp = false

};

File.Delete(publicKeyPath);

rsa.Clear();

}

public static void EncryptData(string publicKeyPath, byte[] dataToEncrypt, string chipherTextPath)

{

byte[] chipherBytes;

using (var rsa = new RSACryptoServiceProvider(2048))

{

rsa.PersistKeyInCsp = false;

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

chipherBytes = rsa.Encrypt(dataToEncrypt, true);

}

File.WriteAllBytes(chipherTextPath, chipherBytes);

}

public static byte[] DecryptData(string chipherTextPath)

{

byte[] chipherBytes = File.ReadAllBytes(chipherTextPath);

byte[] plainTextBytes;

var cspParams = new CspParameters

{

KeyContainerName = CspContainerName,

Flags = CspProviderFlags.UseMachineKeyStore

};

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

{

rsa.PersistKeyInCsp = true;

plainTextBytes = rsa.Decrypt(chipherBytes, true);

}

return plainTextBytes;

}

}

Спочатку розглянемо метод GenerateOwnKeys генерування та збергігання ключів у контейнері:

const string CspContainerName = "MyContainer";

public static void GenerateOwnKeys(string publicKeyPath)

{

CspParameters cspParameters = new CspParameters(1)

{

KeyContainerName = CspContainerName,

Flags = CspProviderFlags.UseMachineKeyStore,

ProviderName = "Microsoft Strong Cryptographic Provider",

};

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

{

rsa.PersistKeyInCsp = true;

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

}

}

Зберігання приватних ключів у файловій системі не найкраща ідея, але можна використовувати вбудовану в Windows систему контейнерів ключів, яка називається постачальником криптографічних послуг (CSP). Контейнер ключів у Windows це місце, де операційна система може зберігати ключі, і вони можуть зберігатися на рівні користувача або комп'ютера. Контейнери ключів машинного рівня доступні всім користувачам, а контейнери рівня користувача доступні лише користувачеві, який створив або імпортував цей контейнер ключів.

У даному коді використовуємо сховище ключів машинного рівня. Поле ProviderName представляє конкретну назву, що використовується для контейнера ключів у системі. Постачальником Windows за замовчуванням є Microsoft Strong Cryptographic Provider. Потім контейнер ключів можна використовувати для шифрування та дешифрування даних для будь-яких програм, які запускаються за допомогою цього ідентифікатора користувача.

Створюючи новий екземпляр RSACryptoServiceProvider, передаємо раніше створений об'єкт CspParameters його конструктору. Також записуємо відкритий ключ в .xml файл для поширення іншим користувачам, які хочуть надіслати нам повідомлення.

Також розглянемо метод видалення ключів DeleteKeys:

public static void DeleteKeys(string publicKeyPath)

{

CspParameters cspParameters = new CspParameters

{

KeyContainerName = CspContainerName,

Flags = CspProviderFlags.UseMachineKeyStore

};

var rsa = new RSACryptoServiceProvider(cspParameters)

{

PersistKeyInCsp = false

};

File.Delete(publicKeyPath);

rsa.Clear();

}

Щоб видалити ключ із контейнера ключів, спочатку потрібно створити екземпляр іншого екземпляр класу CspParameters і передати ім'я ключового контейнера властивість CspParameters.keyContainerName. Потім створюємо новий екземпляр класу RSACryptoServiceProvider і передаємо раніше створений об'єкт CspParameters його конструктору. Потім встановимо для властивості PersistKeyInCsp класу RSACryptoServiceProvider значення false. В кінці потрібно викликати метод Clear для екземпляра RSACryptoServiceProvider. Виклик цього методу звільняє всі ресурси екземпляра RSA та очищає контейнер ключів. Також видаляємо файл, у якому збережений відкритий відповідний ключ, бо, без відповідного приватного ключа, який ми вже видалили, в ньому немає сенсу.

Далі відповідно методи шифрування та дешифрування.

Шифрування EncryptData:

public static void EncryptData(string publicKeyPath, byte[] dataToEncrypt, string chipherTextPath)

{

byte[] chipherBytes;

using (var rsa = new RSACryptoServiceProvider(2048))

{

rsa.PersistKeyInCsp = false;

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

chipherBytes = rsa.Encrypt(dataToEncrypt, true);

}

File.WriteAllBytes(chipherTextPath, chipherBytes);

}

Публічний ключ зчитуємо з файлу, це дає можливість шифрувати повідомлення ключами інших користувачів.

Ми використовуємо статичний метод WriteAllText у класі File для збереження XML-тексту відкритого і закритого ключів на диск. Щоб експортувати фактичний ключовий матеріал, ми використовуємо метод ToXmlString в екземплярі параметра криптосервісу RSA. В кінці записуємо зашифроване повідомлення у файл.

Дешифрування DecryptData:

public static byte[] DecryptData(string chipherTextPath)

{

byte[] chipherBytes = File.ReadAllBytes(chipherTextPath);

byte[] plainTextBytes;

var cspParams = new CspParameters

{

KeyContainerName = CspContainerName,

Flags = CspProviderFlags.UseMachineKeyStore

};

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

{

rsa.PersistKeyInCsp = true;

plainTextBytes = rsa.Decrypt(chipherBytes, true);

}

return plainTextBytes;

}

Так як дешифруємо ми повідомлення приватним ключем, збережений у контейнері, ми створюємо новий об'єкт CspParameters і завантажуємо ім'я контейнера в властивість keyContainerName. Потім, коли ми створюємо клас RSSCryptoServiceProvider, ми передаємо екземпляр CspParameters як параметр конструктора секунд, який завантажує відповідний ключ.

Розберемо основний код програми:

Спочатку ініціалізуємо потрібні змінні:

string original;

byte[] plainText;

string pathToXMLFolder = "..\\..\\..\\.\\xml\\";

string pathToXMLFile;

string pathToFolderEncMsgs = "..\\..\\..\\.\\xml\\enc\_msgs\\";

string pathToEncMsgFile;

string surname;

string txtFileName;

string pkey\_ext = ".xml";

string encMsg\_ext = ".txt";

char sw;

Змінні pathToXMLFolder та pathToFolderEncMsgs зберігають шлях до папки з публічними ключами та зашифрованими повідомленнями відповідно.

Далі запитуємо користувача яку операцію потрібно виконати для подальшої роботи конструктора switch:

Console.Write("g - generate new keys\n" +

"r - remove keys\n" +

"e - encypt data\n" +

"d - decrypt data\n" +

" -> ");

sw = Convert.ToChar(Console.ReadLine());

Відповідно розглянемо фрагменти коду у switch.

Генерування ключів:

case 'g':

Console.Write("Write your surname: ");

surname = Console.ReadLine();

pathToXMLFile = pathToXMLFolder + surname + pkey\_ext;

ASYM\_ENC.GenerateOwnKeys(pathToXMLFile);

Console.WriteLine("DONE");

break;

Спочатку запитуємо у користувача його прізвище, саме таку назву буде мати файл .xml, який буде зберігати публічний ключ. Далі відповідно, використовуючи конкатенацію, створюємо шлях до цього файлу, після чого викликаємо метод генерування ключів.

Видалення ключів:

case 'r':

try

{

Console.Write("Write your surname: ");

surname = Console.ReadLine();

pathToXMLFile = pathToXMLFolder + surname + pkey\_ext;

ASYM\_ENC.DeleteKeys(pathToXMLFile);

var files = Directory.GetFiles(pathToFolderEncMsgs);

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

{

File.Delete(files[i]);

}

Console.WriteLine("DONE");

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

break;

Виконуємо запит на введення прізвища, для того щоб видалити потрібний .xml файл, після чого викликаємо метод видання ключів. Також автоматично видаляємо усі зашифровані файли у папці enc\_msgs, бо ми все одно більше не зможемо їх рощифрувати.

Шифрування:

case 'e':

try

{

var files = Directory.GetFiles(pathToXMLFolder);

if (files.Length != 0)

{

Console.WriteLine("Public keys:");

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

{

Console.WriteLine((i + 1) + ". " + files[i]);

}

Console.Write("\nChoose public key to encrypt (enter number): ");

int num = Convert.ToInt32(Console.ReadLine());

pathToXMLFile = files[num - 1];

Console.Write("\nEnter data to encrypt: ");

original = Console.ReadLine();

Console.Write("Write file name to encrypt (without file extension): ");

txtFileName = Console.ReadLine();

pathToEncMsgFile = pathToFolderEncMsgs + txtFileName + encMsg\_ext;

ASYM\_ENC.EncryptData(pathToXMLFile, Encoding.UTF8.GetBytes(original), pathToEncMsgFile);

Console.WriteLine("\nDONE");

}

else

{

Console.WriteLine("NO PUBLIC KEYS TO ENCRYPT");

}

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

}

break;

Спочатку ініціюємо масив рядків, у який записуємо шляхи до усіх публічних ключів, які знаходяться у папці xml. Далі, якщо масив не пустий, виводимо список доступних ключів для шифрування, інакше виводимо повідомлення про відсутність ключів і програма завершує свою роботу. Відповідно користувач обирає потрібний ключ, тобто потрібний елемент з масиву, пише повідомлення та назву файлу, у який збережемо це повідомлення, створюючи далі назву для цього файлу. В кінці викликаємо метод шифрування.

Дешифрування:

case 'd':

try

{

var files = Directory.GetFiles(pathToFolderEncMsgs);

if (files.Length != 0)

{

Console.WriteLine("Encrypted messages:");

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

{

Console.WriteLine((i + 1) + ". " + files[i]);

}

Console.Write("\nChoose message to dencrypt (enter number): ");

int num = Convert.ToInt32(Console.ReadLine());

pathToEncMsgFile = files[num - 1];

plainText = ASYM\_ENC.DecryptData(pathToEncMsgFile);

Console.WriteLine("Original text: " + Encoding.Default.GetString(plainText));

}

else

{

Console.WriteLine("NO FILES TO DECRYPT");

}

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

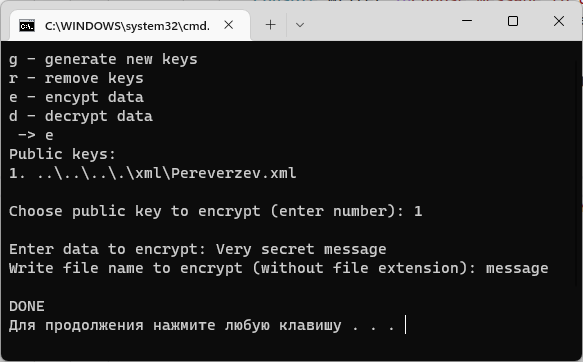
}

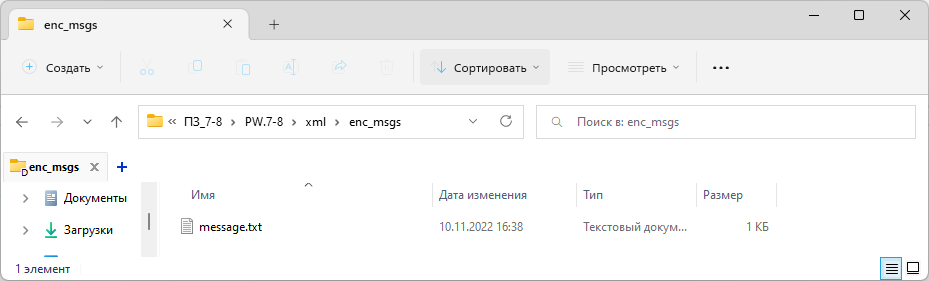
break;

Аналогічно попередньому коду, записуємо у масив рядків шляхи до файлів, але вже зашифрованих повідомлень. Так само перевіряємо чи пустий масив, якщо так, то виводимо відповідне повідомлення. Виводимо в консоль доступні для дешифрування повідомлення, після чого користувач обирає одне з них. Викликаємо метод дешифрування і виводимо у консоль оригінальне повідомлення.

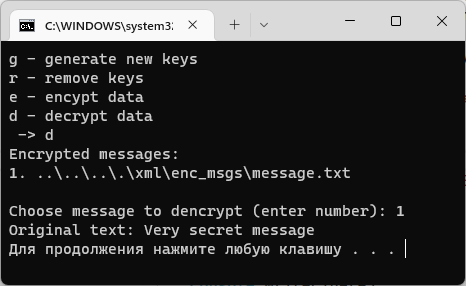
Виконання програми:

Шифрування:



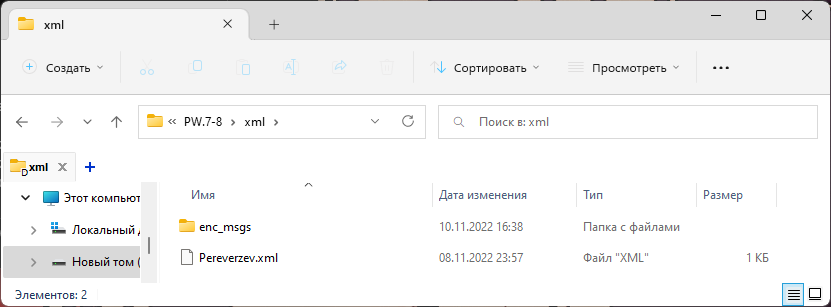


Дешифрування:

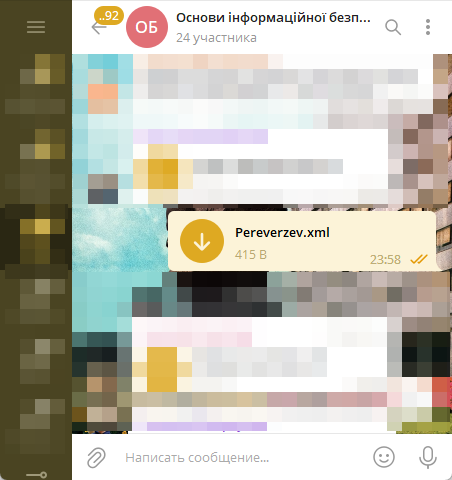


1. Згенерувати відкритий (!!!) ключ RSA 2048bit та зберегти у файл з іменем, що відповідає транслітерації прізвища, та розширенням xml. Поділитися збереженим відкритим ключем у Телеграм-каналі. Зашифрувати та надіслати вітання двом (наступним за списком) студентам. Отримані вітання розшифрувати та вивести на екран.

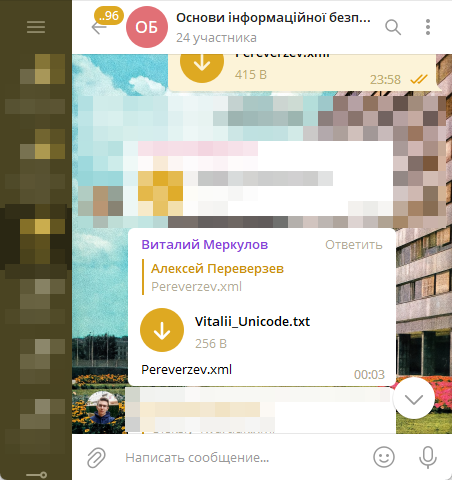
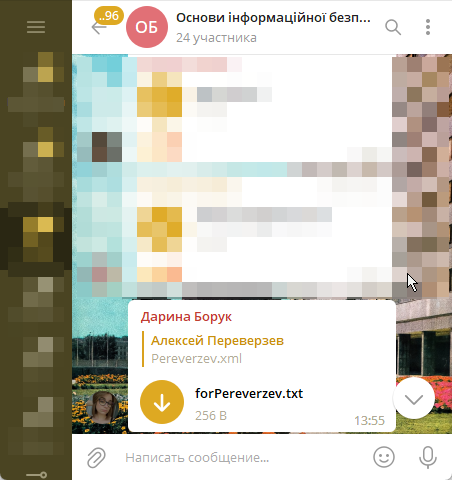
Генерую ключ з відповідним прізвищем



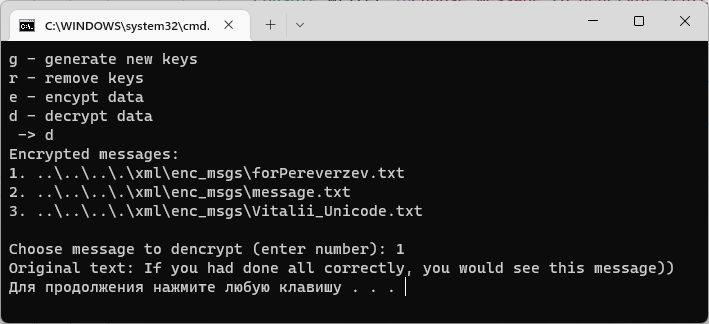
та поширюю його у нашій телеграм групі «Основи інформаційної безпеки»

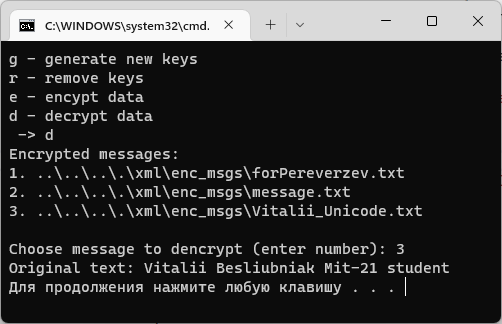


Після чого отримую повідомлення від двох одногрупників, які зашифровані моїм ключем:

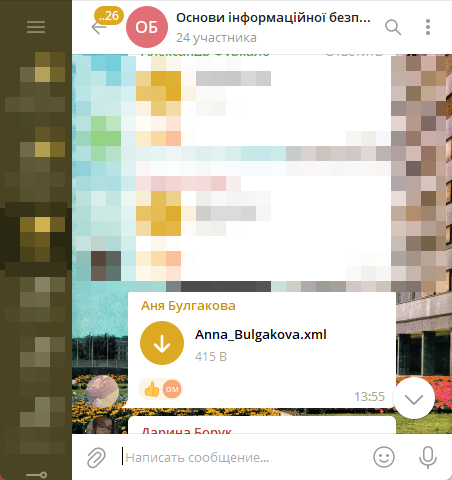
Спробуємо їх дешифрувати, завантаживши у відповідну папку:



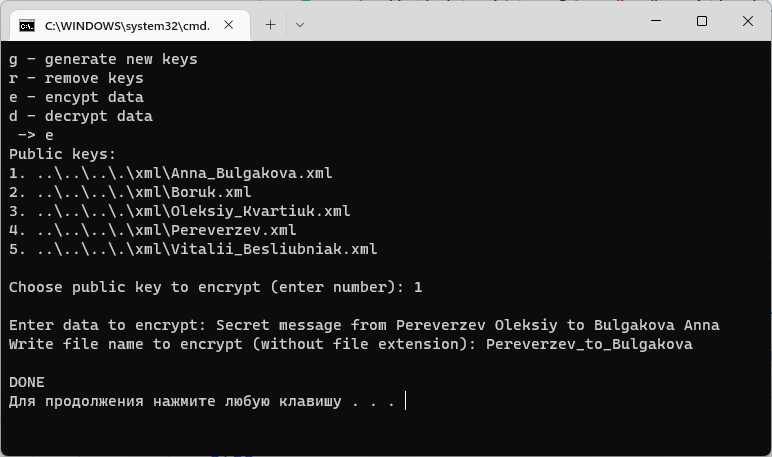


Отже все коректно працює і ми змогли отримати зашифроване повідомлення від інших користувачів.

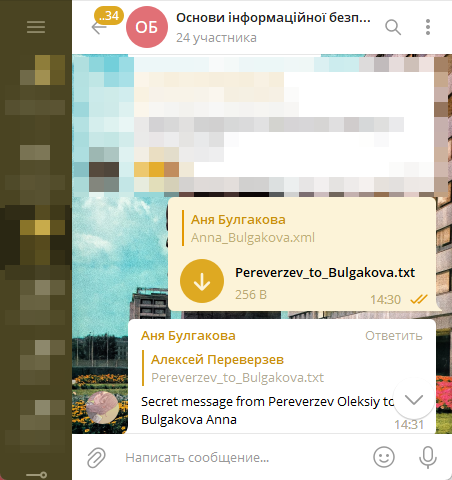
Спробуємо відправити повідомлення інших учасникам групи:



Завантажую цей файл з ключем у відповідну папку та шифрую свою повідомлення:



Та надсилаю це повідомлення у групу, отримуючи оригінальний текст:



Аналогічно надсилаю зашифровані повідомлення ще декільком студентам:

