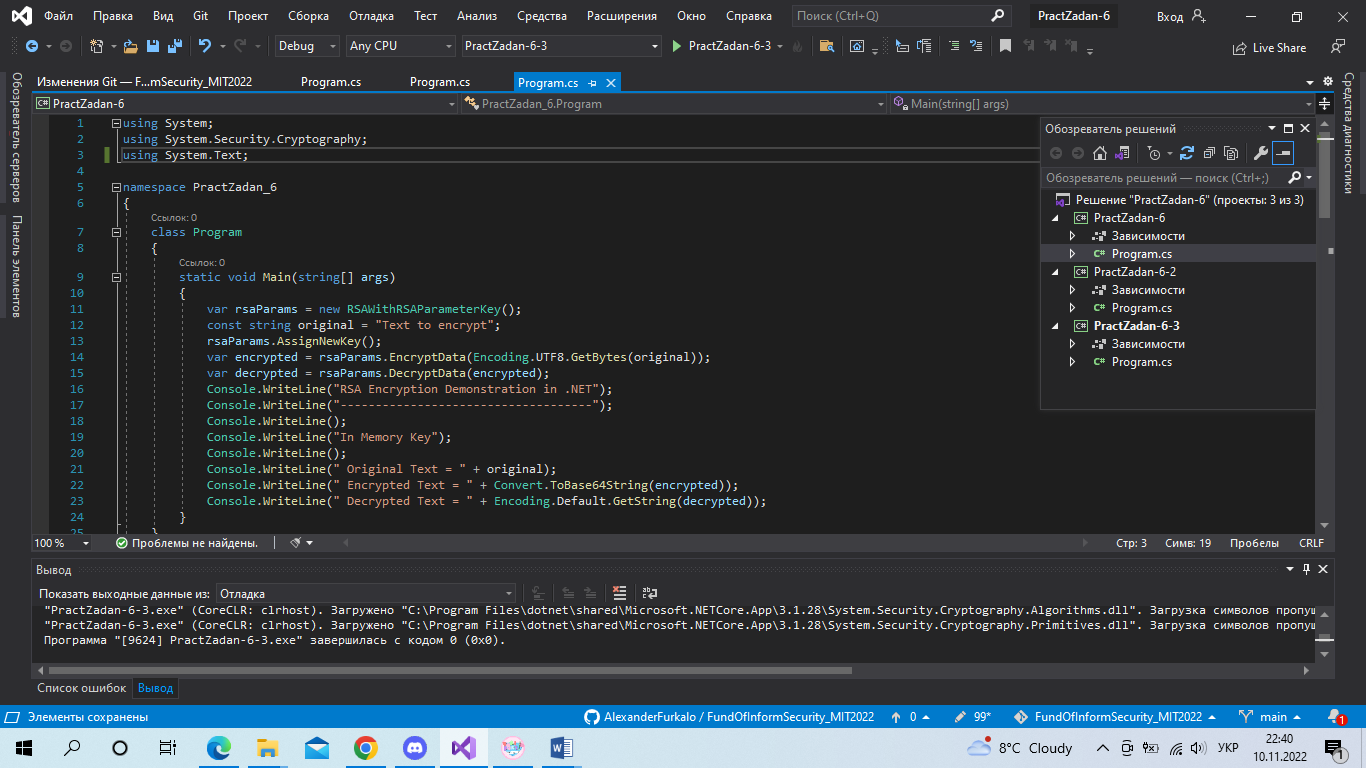
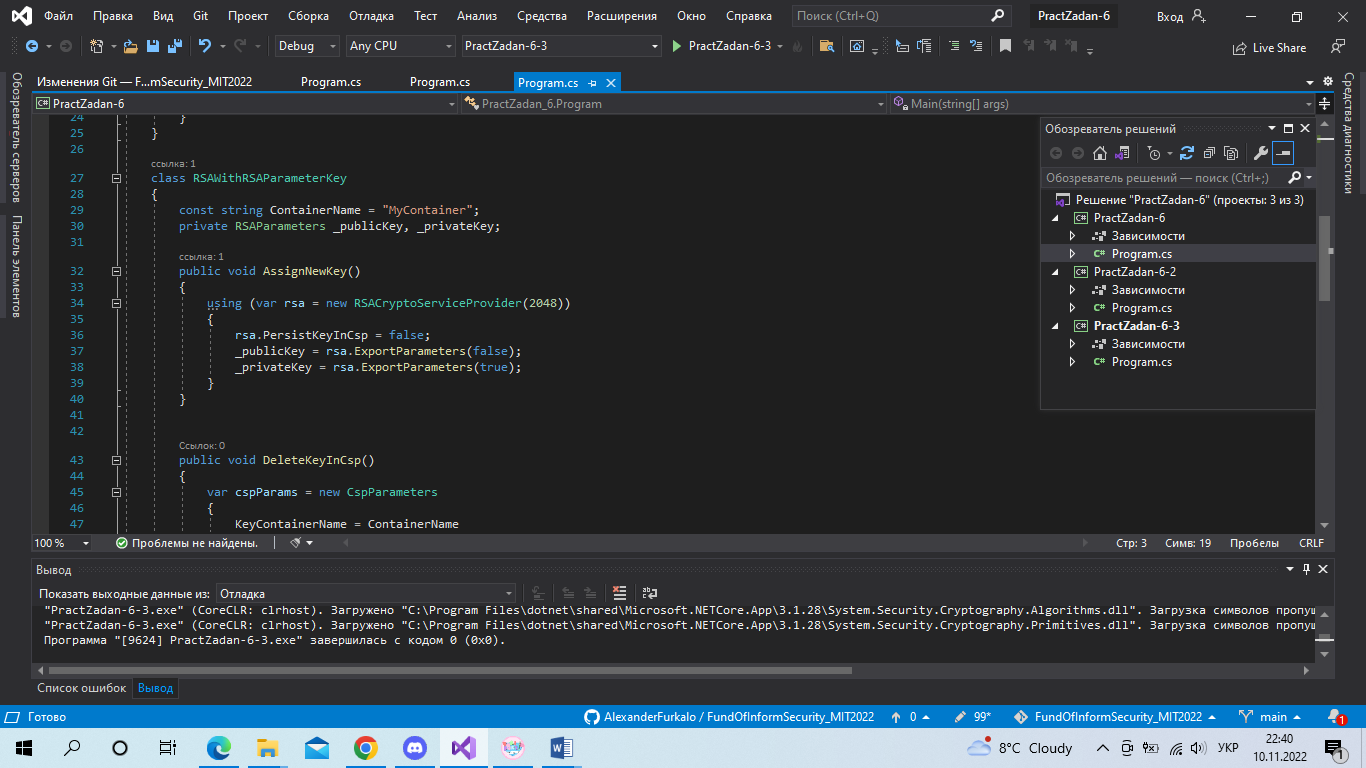
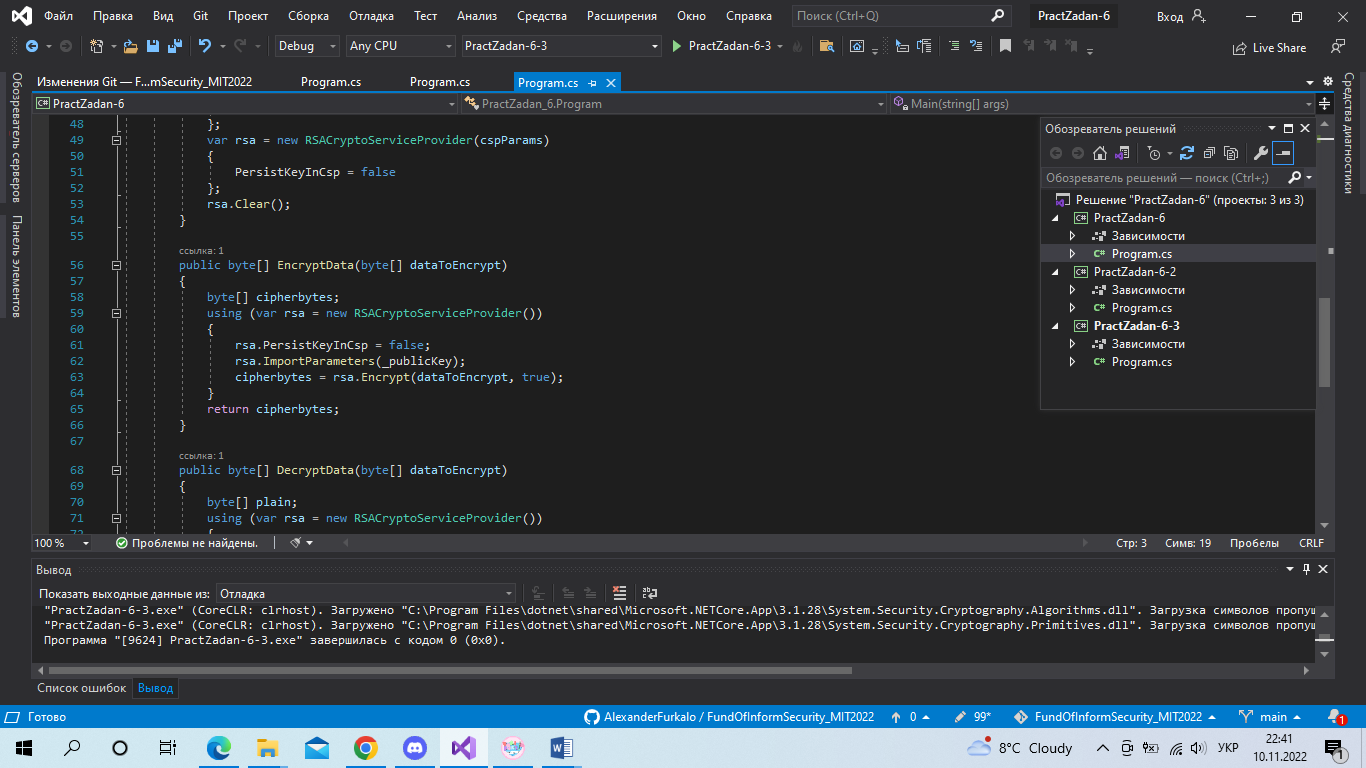
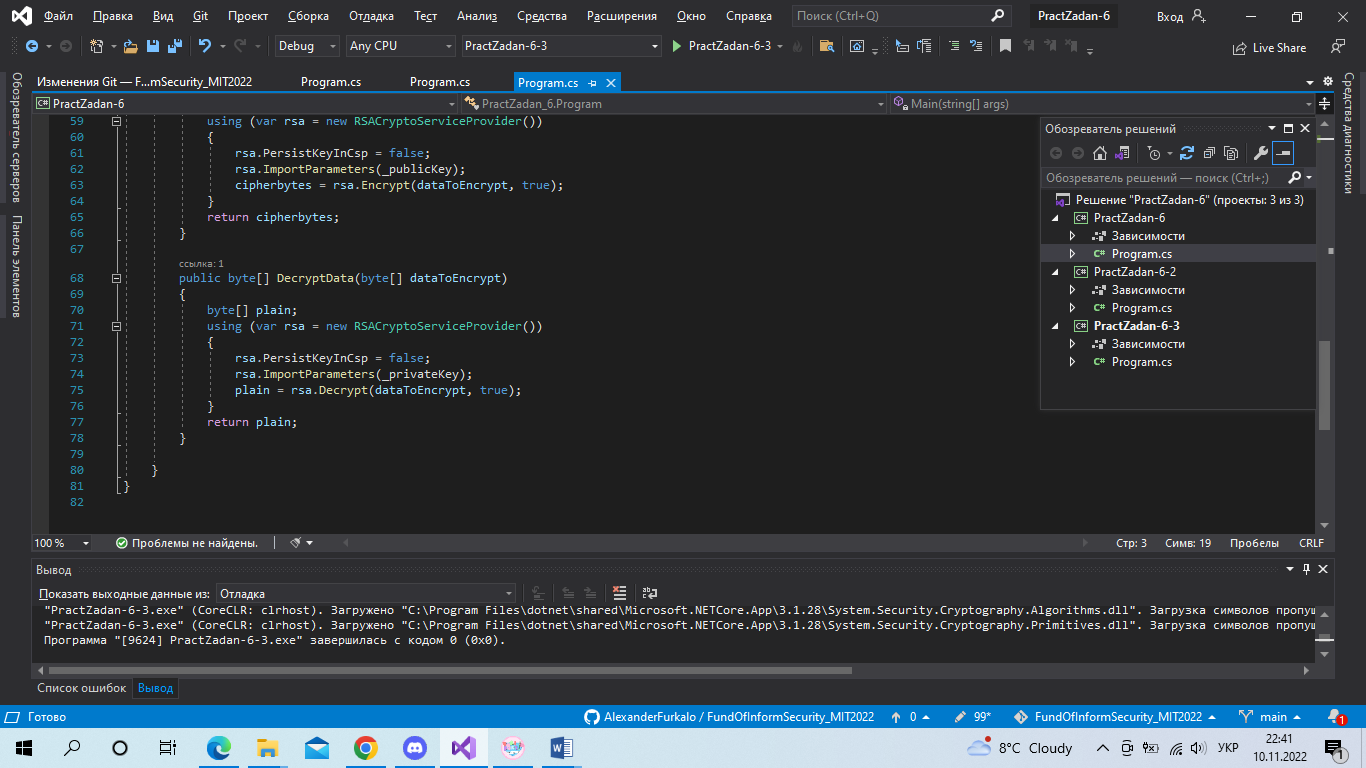
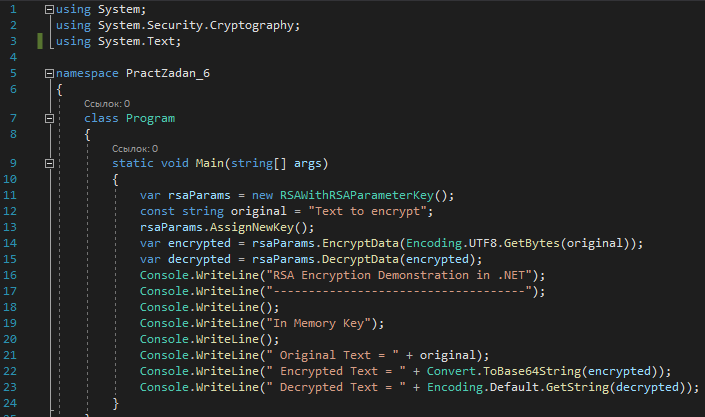
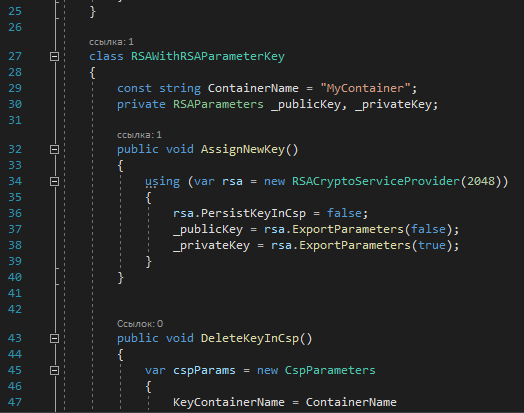
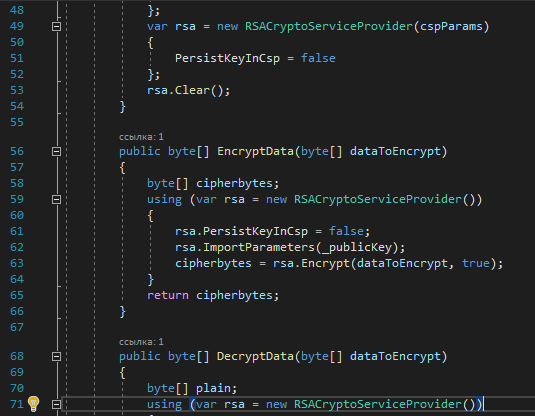
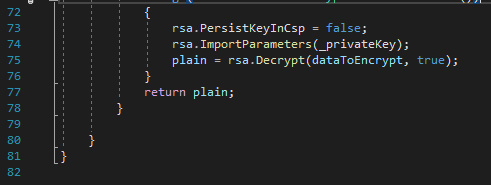
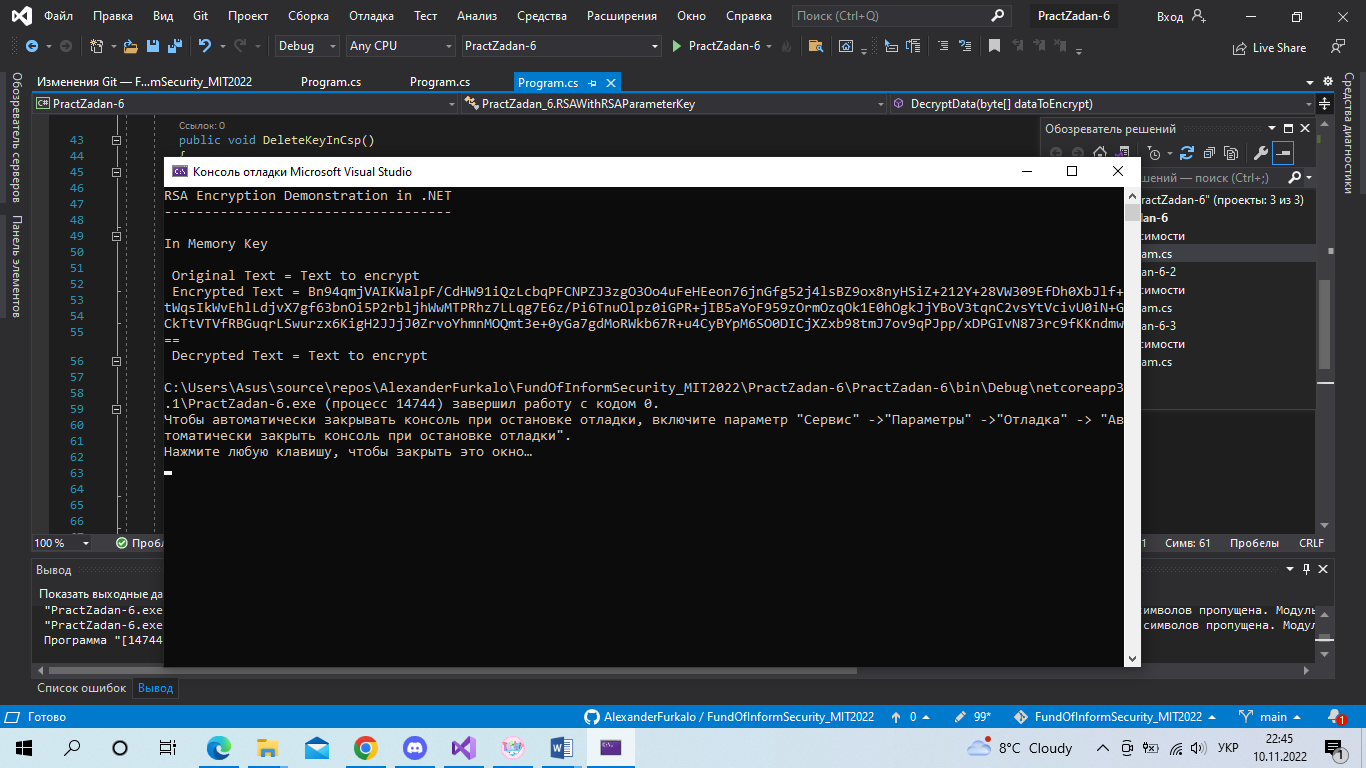
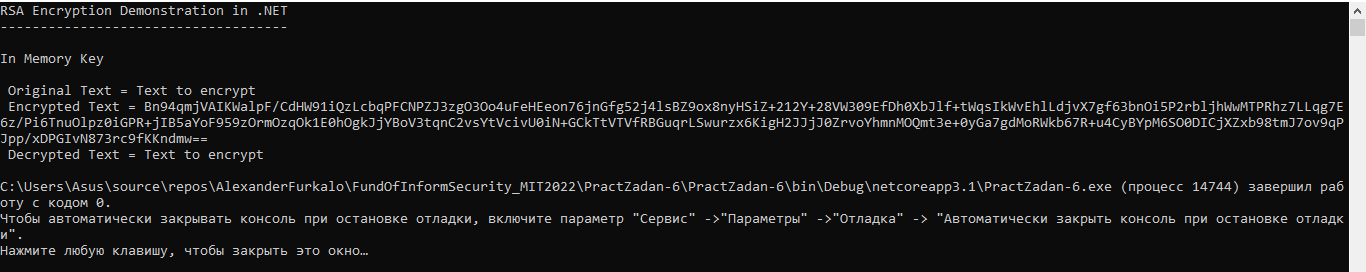
ПРАКТИЧНА РОБОТА №6  
З ОСНОВ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ  
СТУДЕНТА КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКО  
ФАКУЛЬТЕТУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  
КАФЕДРИ МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ  
ДРУГОГО КУРСУ, ДРУГОЇ ПІДГРУПИ  
ОЛЕКСАНДРА ОЛЕКСАНДРОВИЧА ФУРКАЛА  
ЗА ТЕМОЮ “АСИМИТРИЧНЕ ШИФРУВАННЯ ЯК ЗАСІБ ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ІНФОРМАЦІЇ”  
ЗВІТ  
10.11.2022

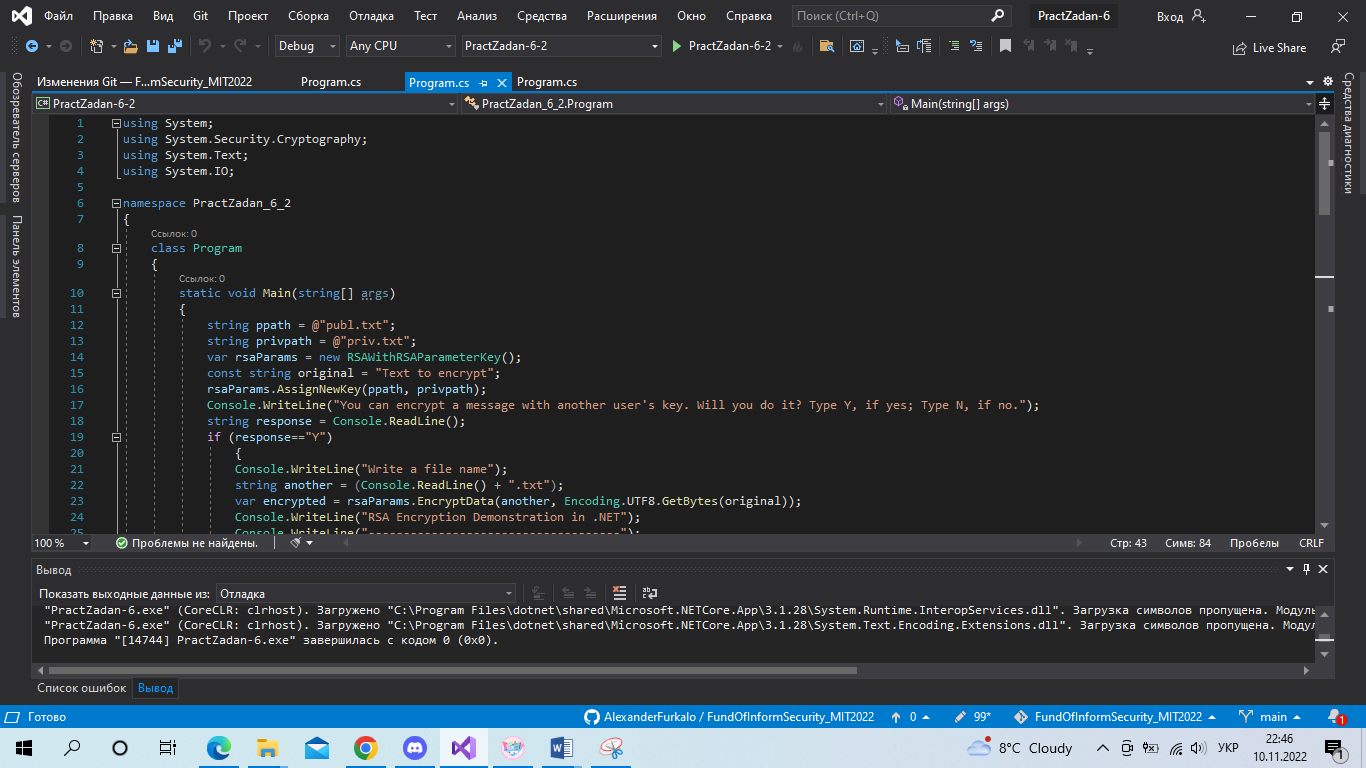
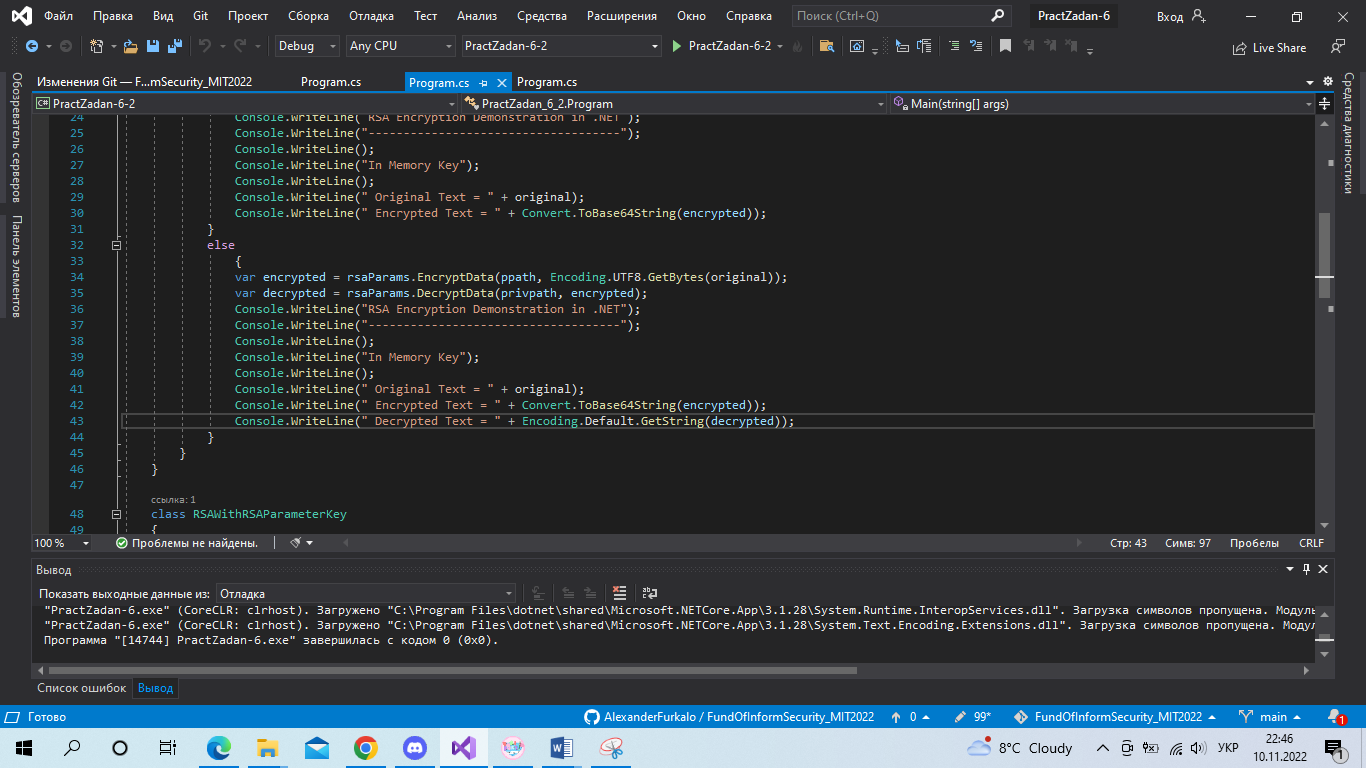
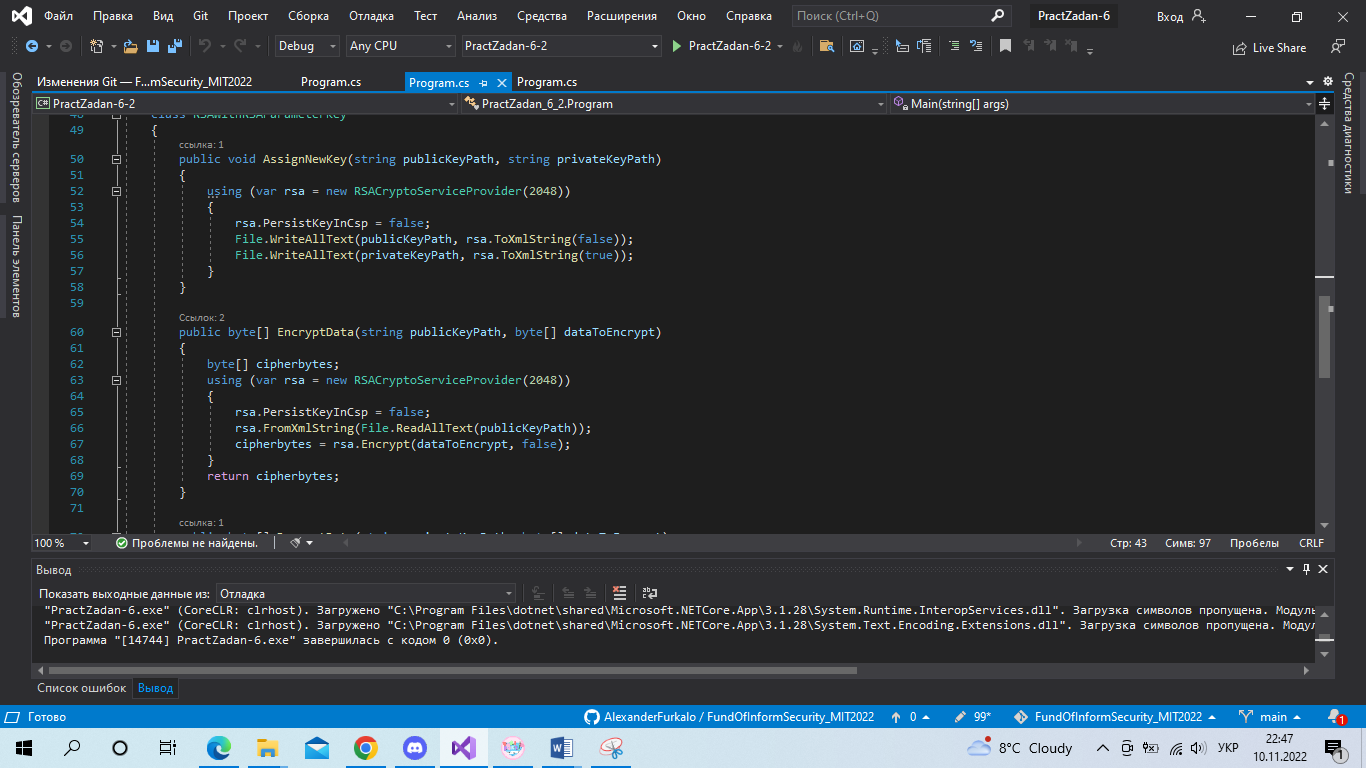
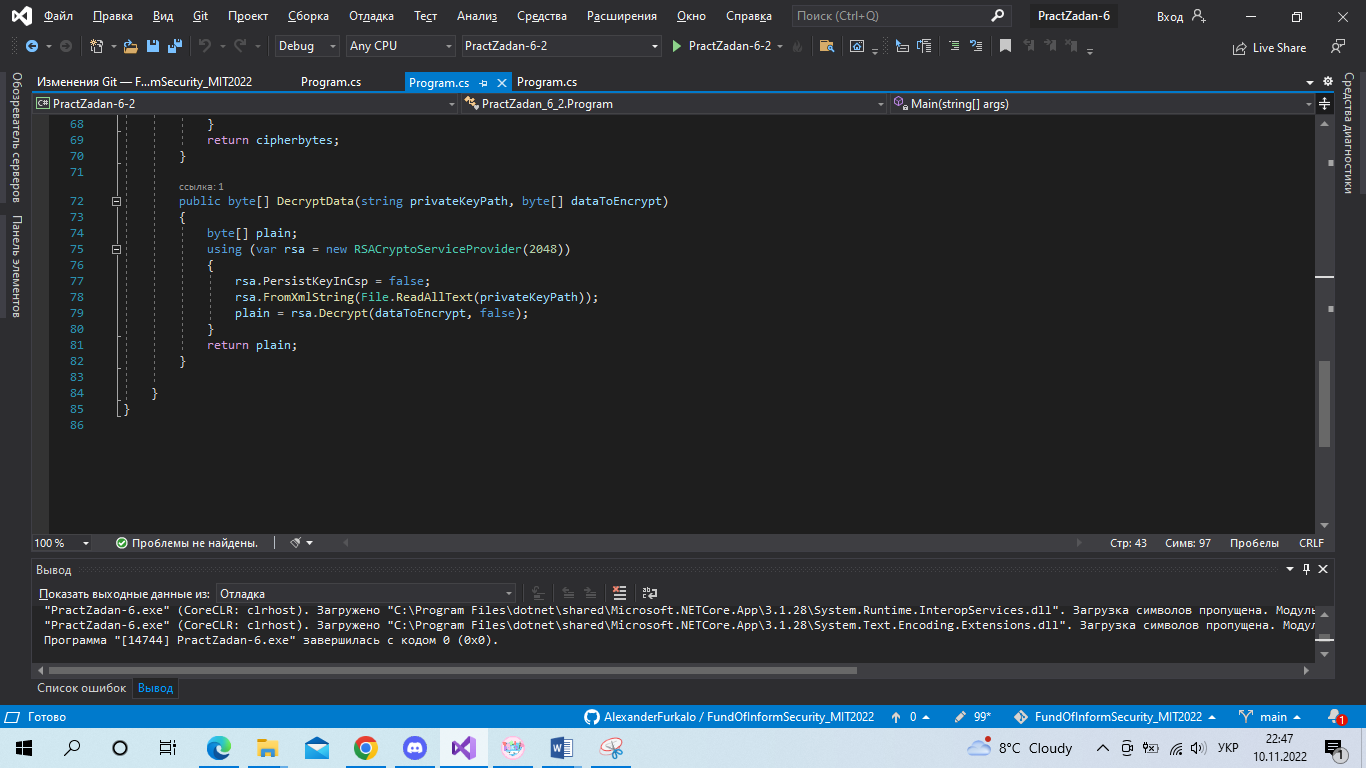
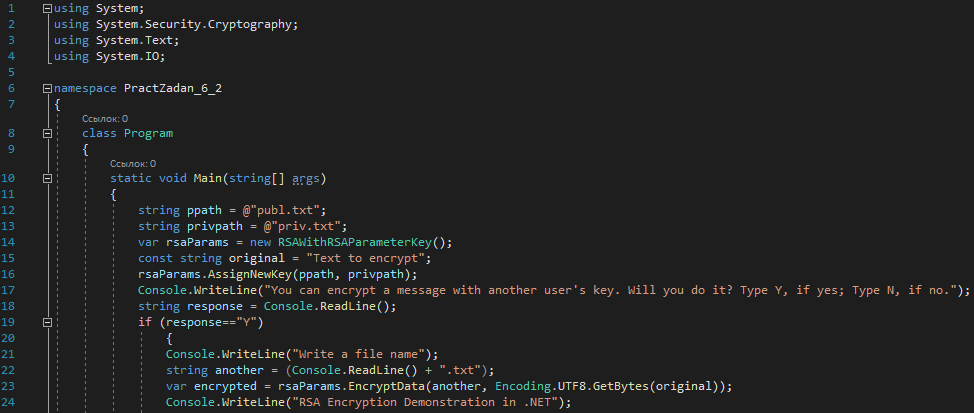
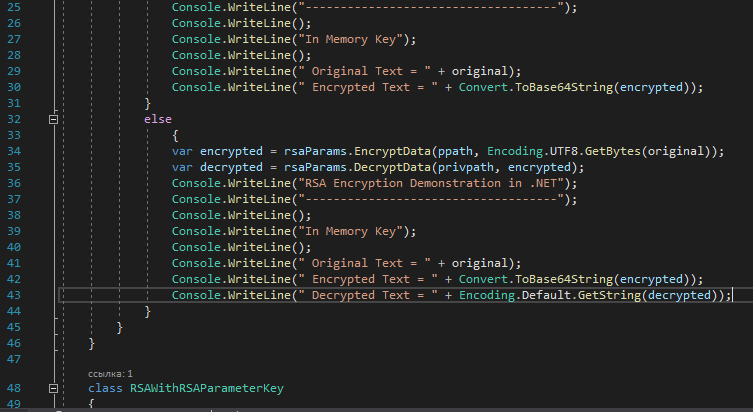
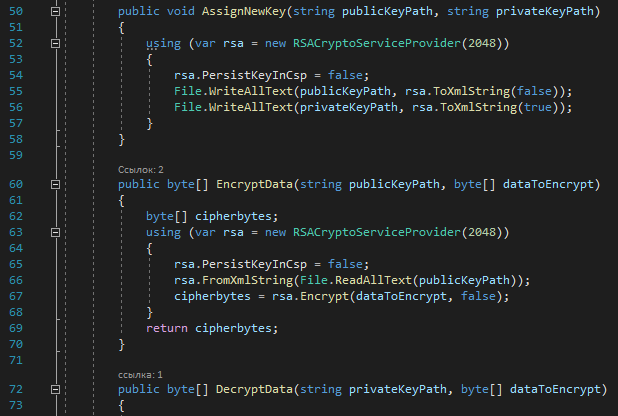
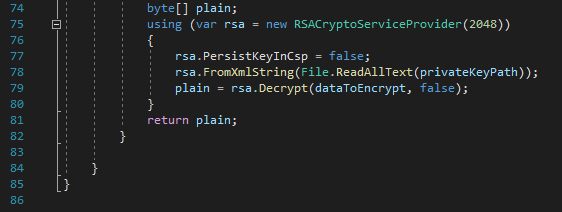
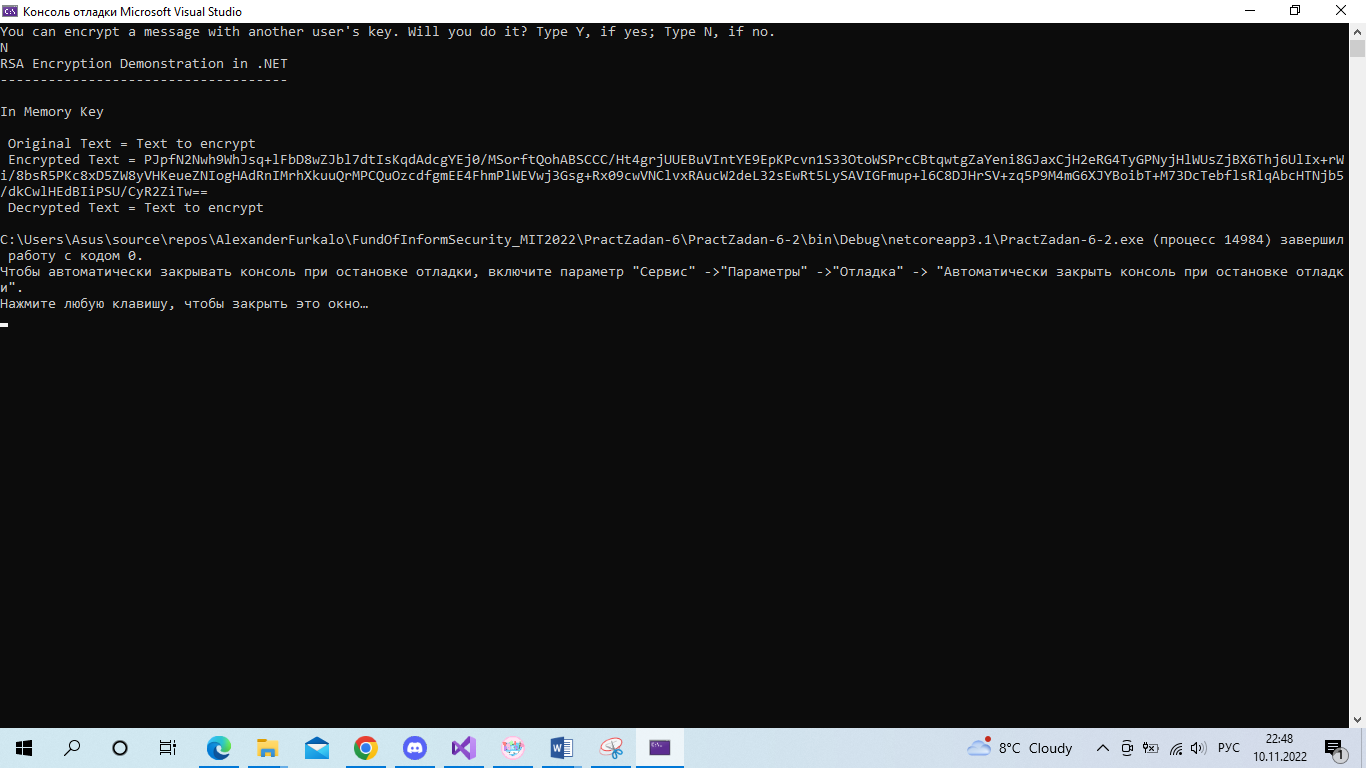
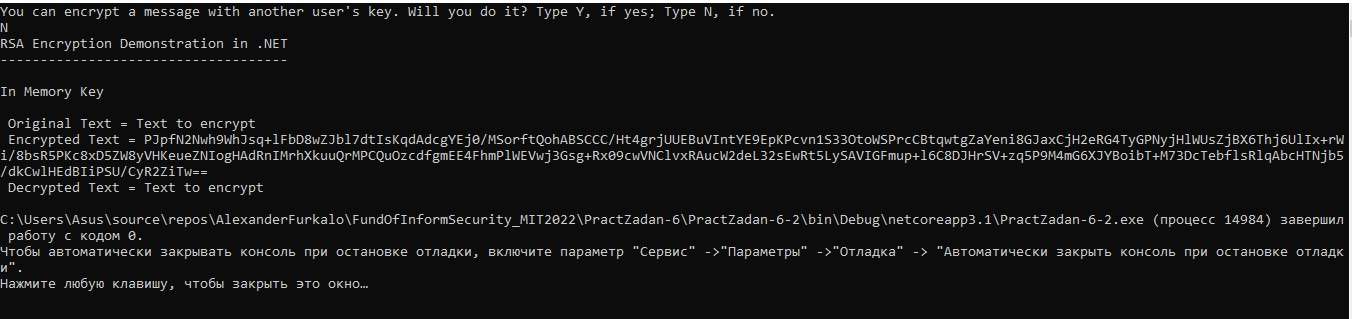
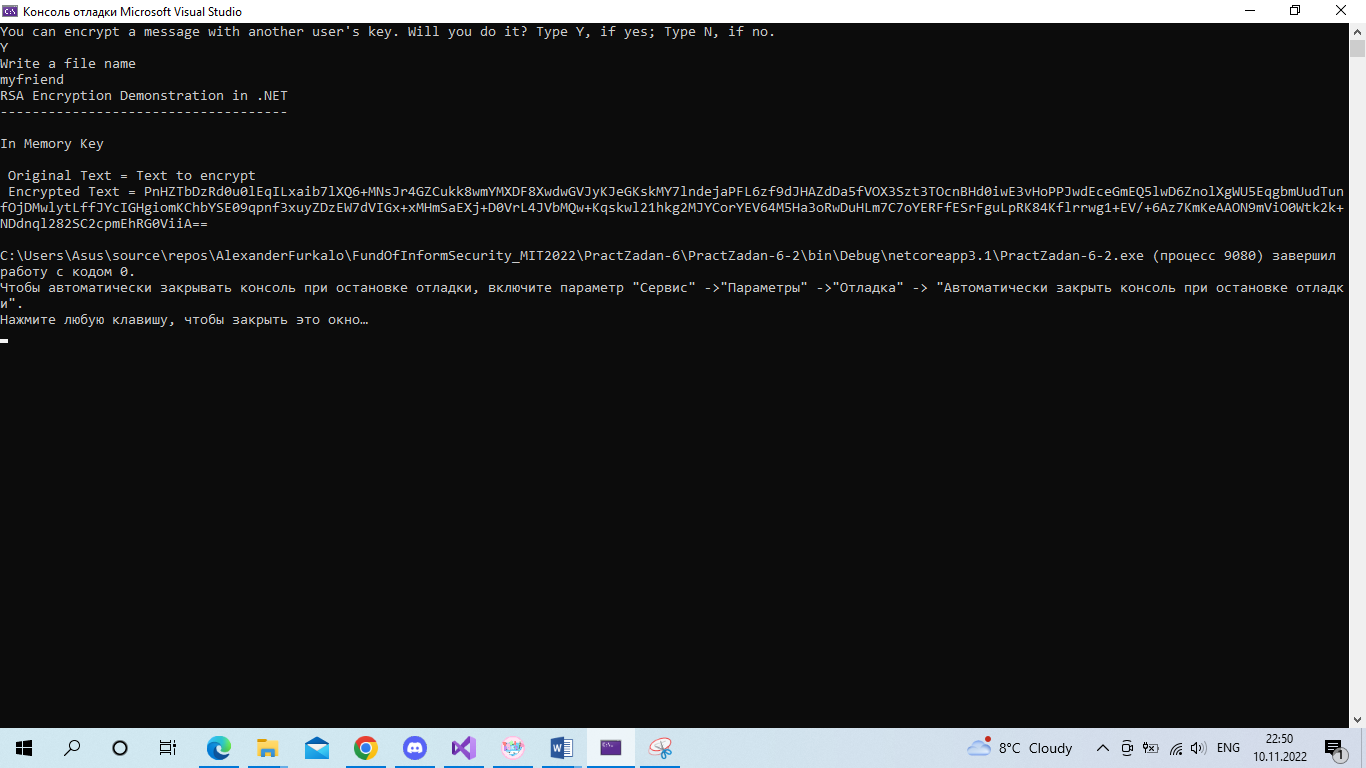
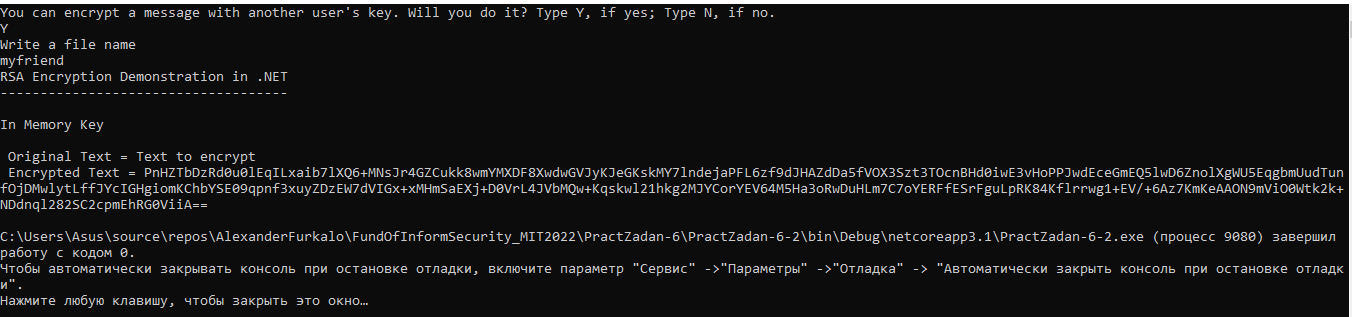
Завдання цієї практичної роботи були такі:  
1. Написати програму, яка виконує зашифровування та розшифровування даних з використанням алгоритмів асиметричного шифрування RSA. Пара ключів зберігається у пам’яті. 2. Для програми з п.1. реалізувати можливість збереження відкритого ключа у файлі. Реалізувати можливість зашифровувати повідомлення за допомогою файлів відкритих ключів інших користувачів.  
3. 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.  
4. Оформити звіт

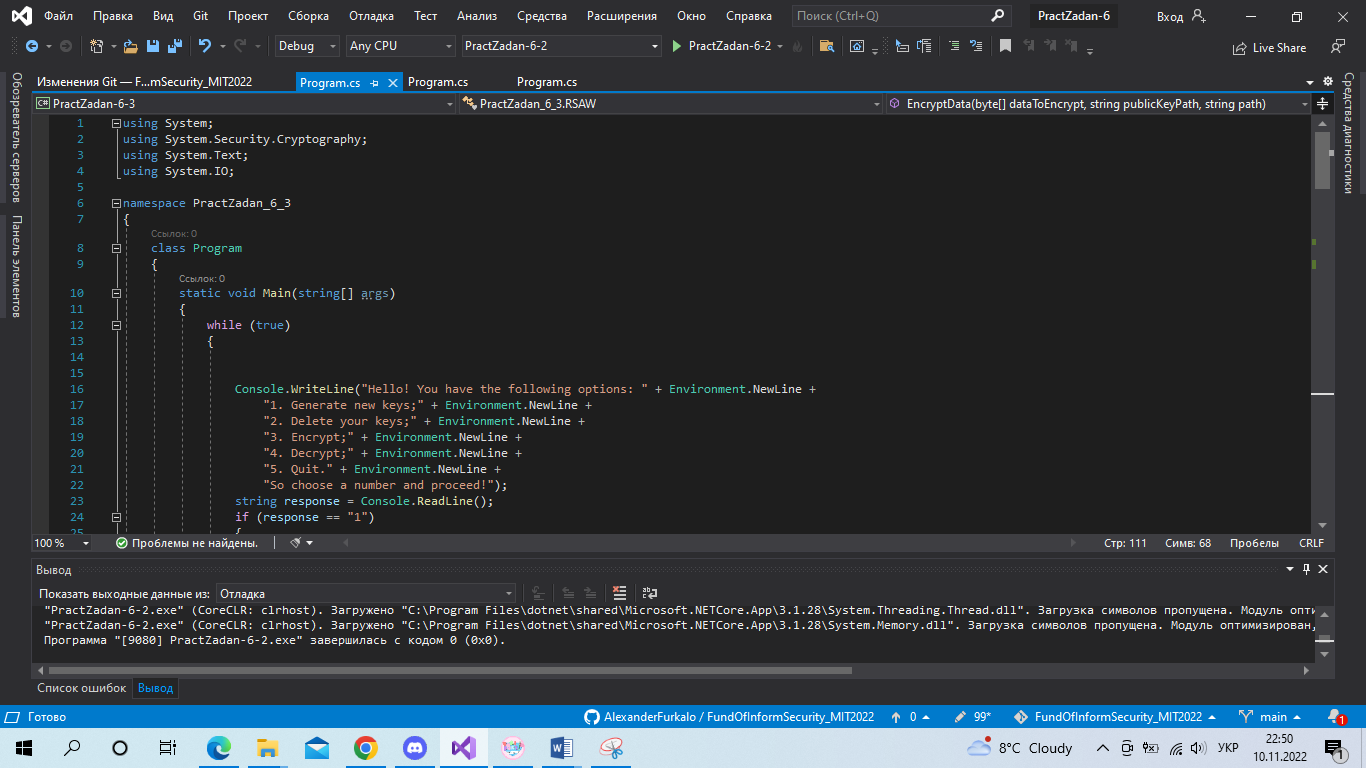
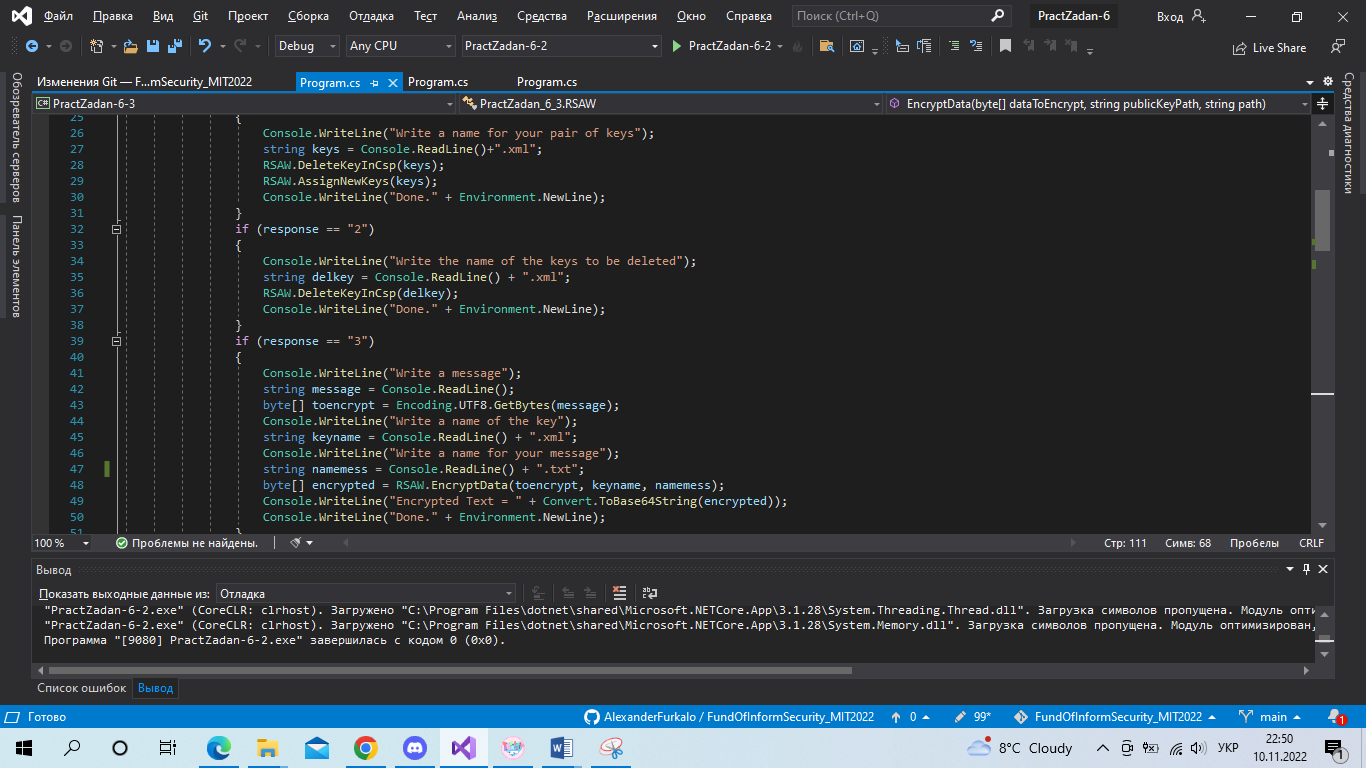
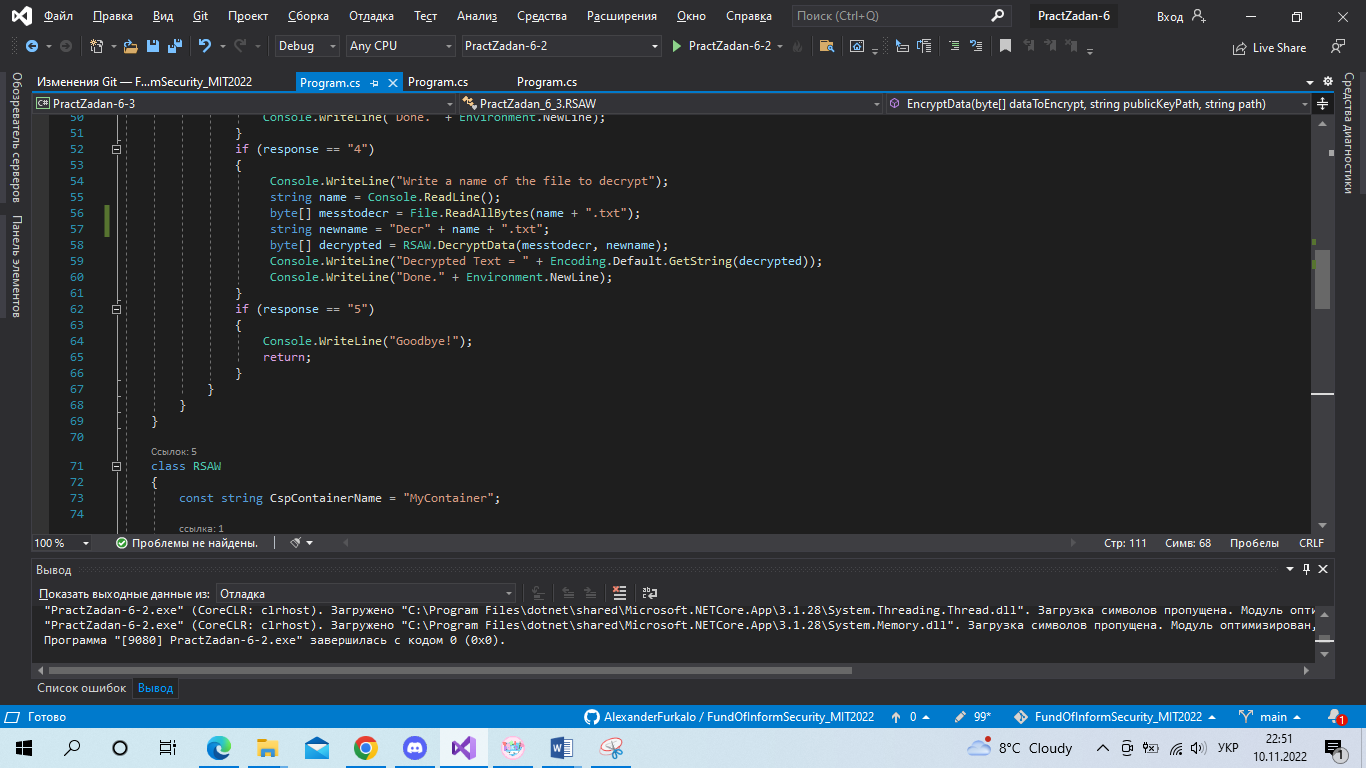
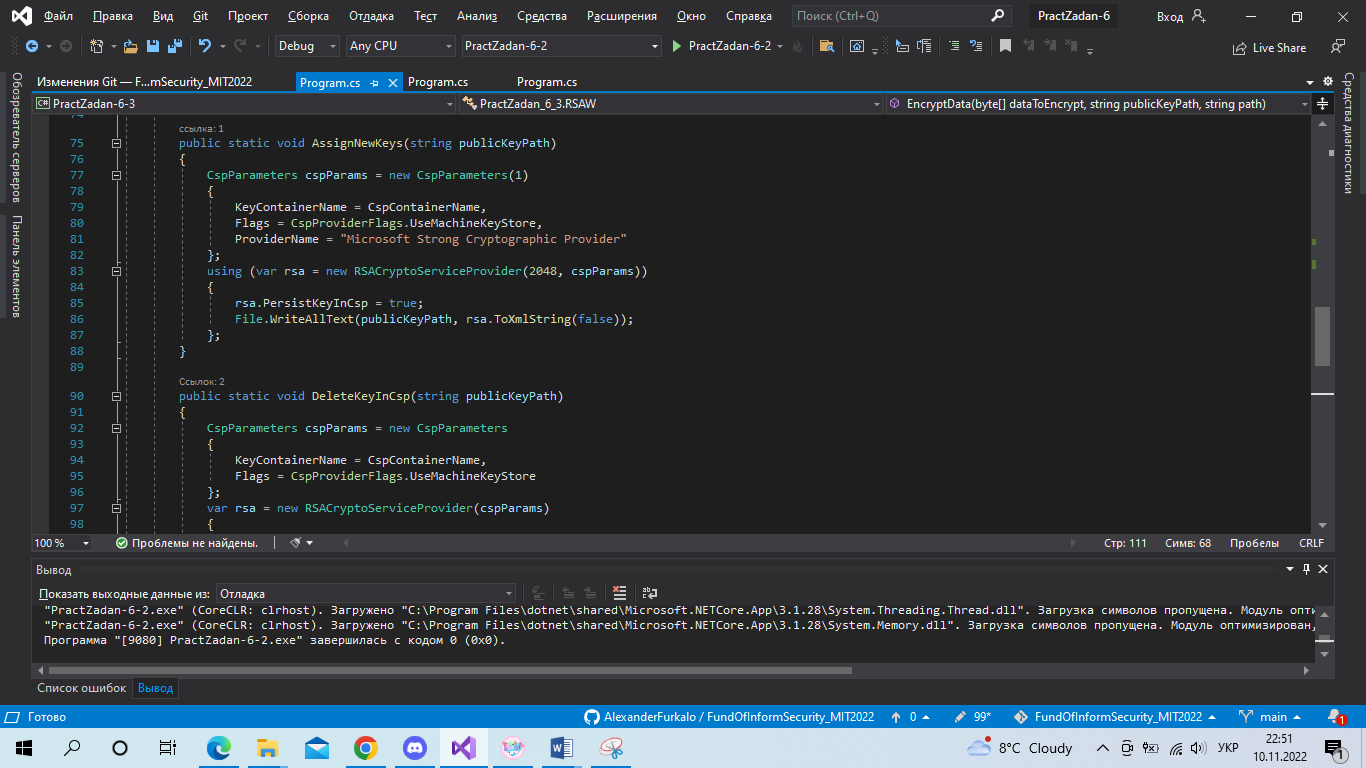
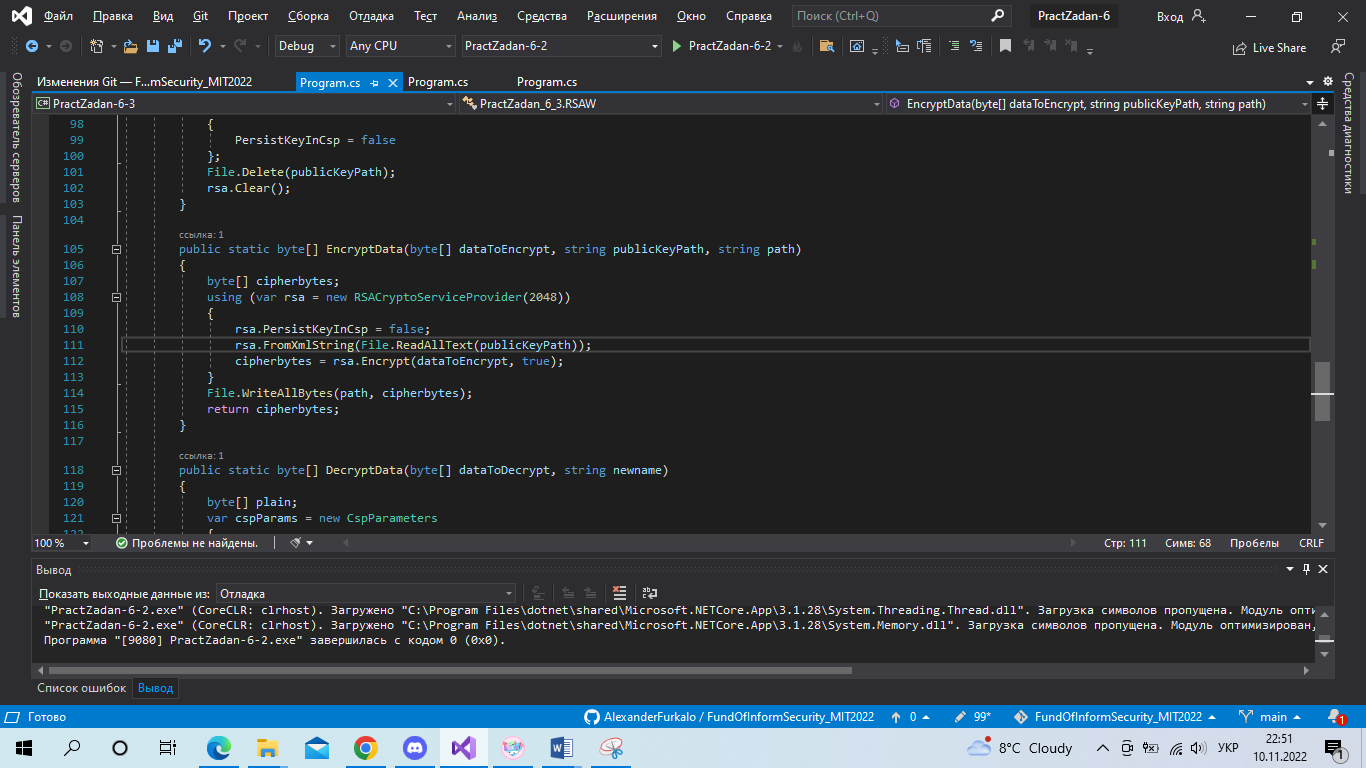
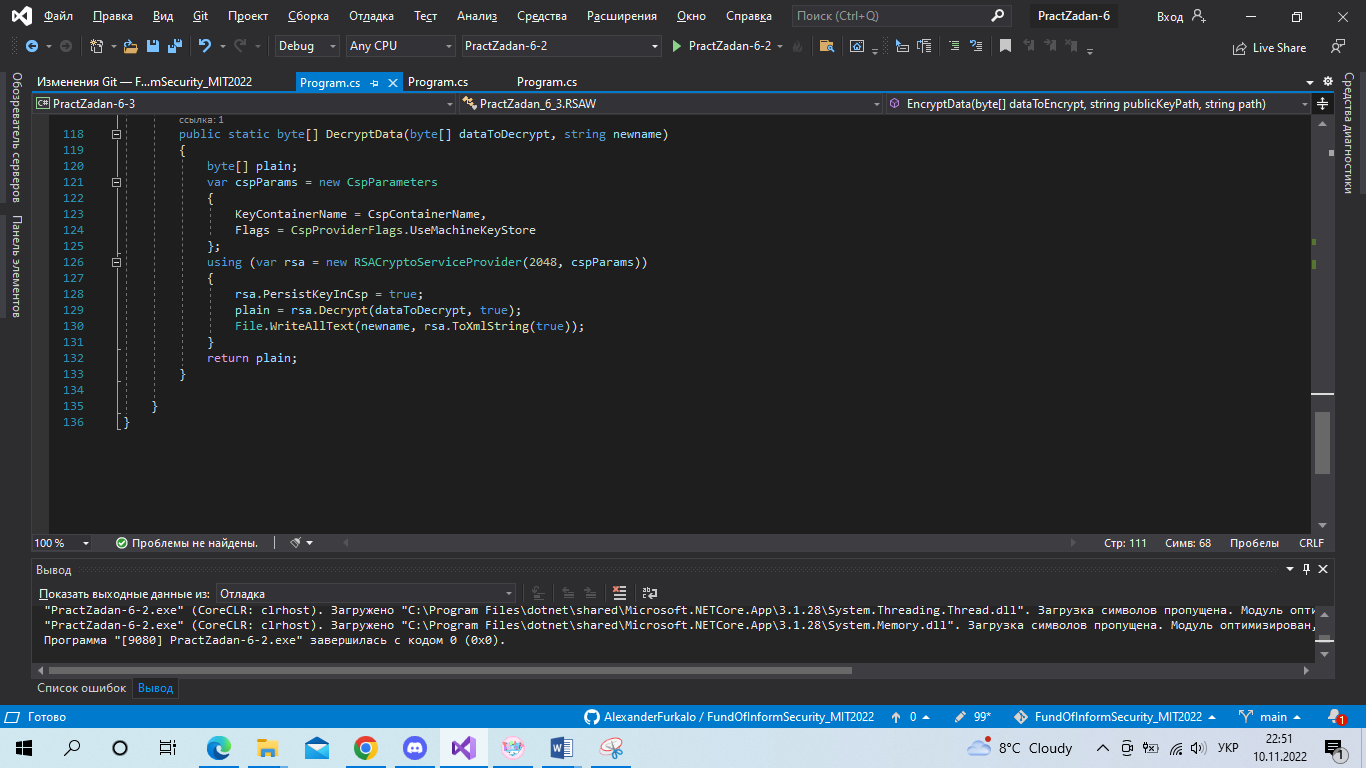
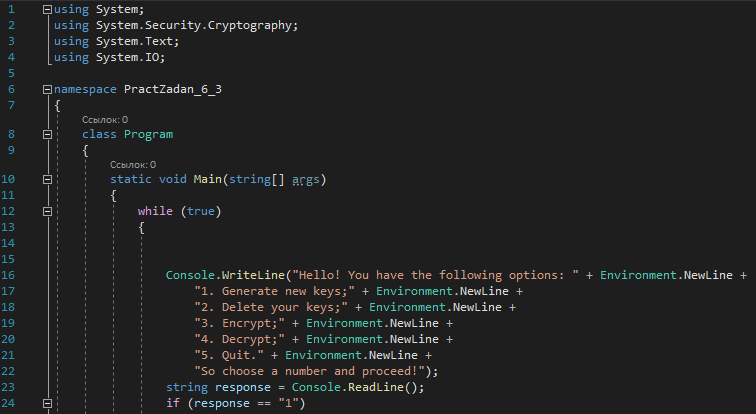
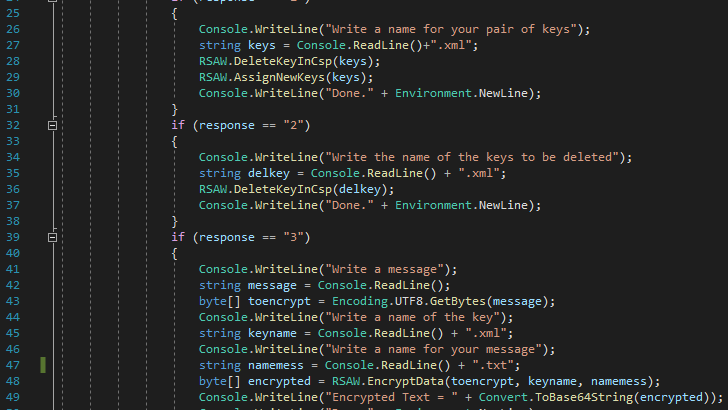
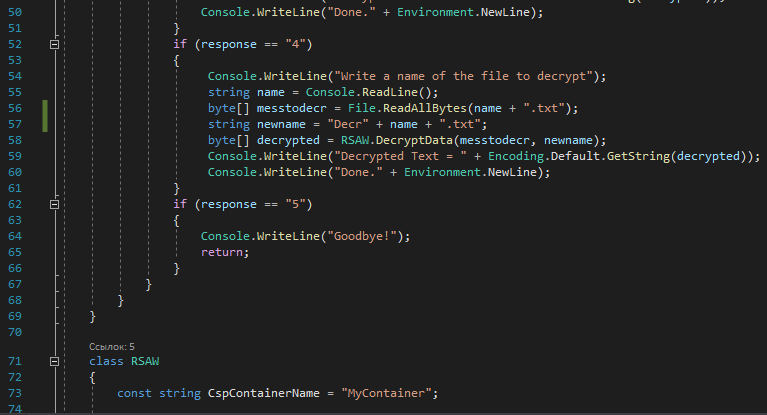
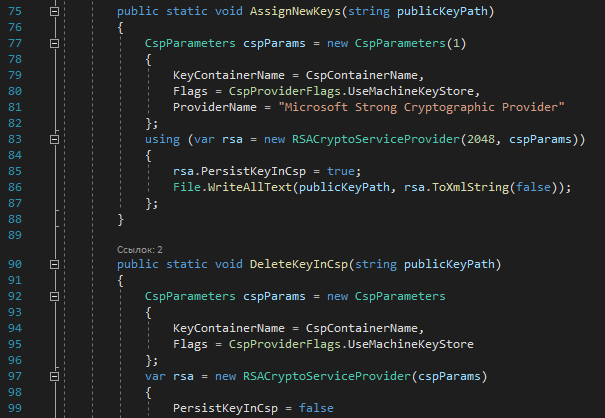
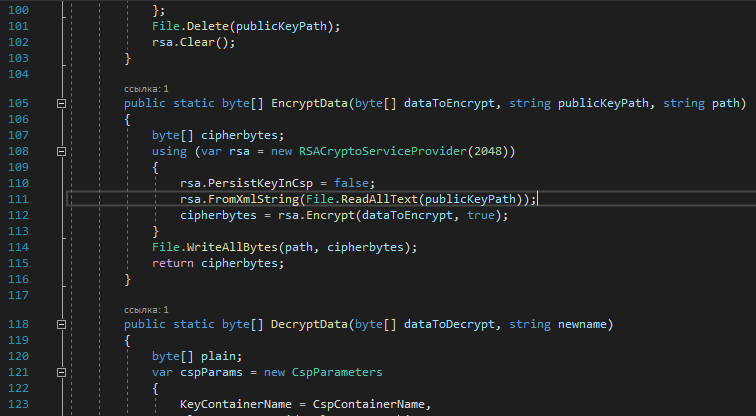
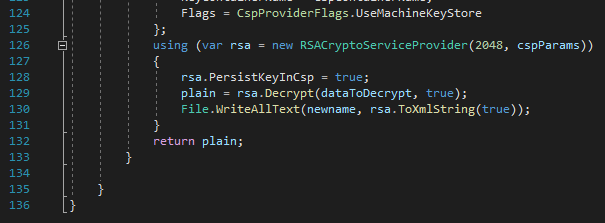
У минулій практичній роботі ми дослідили симетричне шифрування – головною проблемою для цього типу був розподіл ключів, бо отримувачу потрібно мати той самий секретний ключ, що й відправник, для того, щоб розшифровувати повідомлення.   
Асиметричне шифрування (або ж шифрування з відкритим ключем) вирішує цю проблему – є два ключа, відкритий та закритий: перший використовується для зашифровування повідомлень та їм можна ділитися, коли ж другий (секретний) потрібен для розшифровування та має бути лише у одержувача текстів. Пара ключів генеруються одночасно, але навіть маючи відкритий ключ, обчислити секретний ключ неможливо. Слово “асиметричний” використовується тому, що цей метод використовує два різні пов’язані ключі, які виконують зворотні операції, тоді як симетрична криптографія використовує той самий ключ для виконання обох операцій.   
Причина – в односторонньому математичному зв’язку, що існує між парою ключів, при якому інформація про відкритий ключ ніяк не допомагає відновити секретний ключ, проте володіння секретним ключем забезпечує можливість розшифровування повідомлень, що були зашифровані відкритим ключем.  
Основна перевага використання асиметричного шифрування полягає в тому, що двом сторонам не потрібно попередньо ділитися секретним ключем для спілкування за допомогою асиметричного шифрування. Особі, яка шифрує повідомлення, потрібно знати лише відкритий ключ одержувача, який доступний будь-кому за запитом. Тоді лише одержувач може розшифрувати повідомлення за допомогою свого закритого ключа.  
Основним недоліком є ​​те, що асиметричний алгоритм є порівняно складним порівняно з симетричним шифруванням, а це означає, що повідомлення потребують більше часу для шифрування та дешифрування.  
RSA — це технологія шифрування з відкритим ключем, розроблена компанією RSA Security LLC (раніше RSA Security, Inc.). Назва RSA утворює абревіатуру, яка розшифровується як імена її творців, Rivest, Shamir і Adelman. RSA отримує свою силу від ідеї, що не існує ефективного способу розкласти величезні прості числа.  
Одностороння функція, що покладена в основу алгоритму, базується на таких двох фактах з теорії чисел:  
- задача перевірки чисел на простоту є порівняно легкою;  
- задача розкладу чисел вигляду n = pq (p та q – прості числа) на множники є дуже складною, якщо ми знаємо лише n, а p та q – великі числа (так звана задача факторизації).  
Оскільки процес шифрування є математичним і базується на арифметиці за модулем, існують обмеження щодо кількості даних, які можна зашифрувати за один раз. Як правило, ви не можете шифрувати дані, розмір яких перевищує розмір ключа. Отже, якщо ви шифруєте за допомогою 2048-бітного ключа, який має довжину 256 байт, ви не можете зашифрувати більше 256 байт. Чим більше даних ви шифруєте, тим повільніше відбувається процес шифрування, хоча RSA полегшує проблему спільного використання ключів, але це досить неефективна система шифрування.  
У цьому завданні розглядається три методи створення та роботи з ключами, про них буде трохи пізніше.

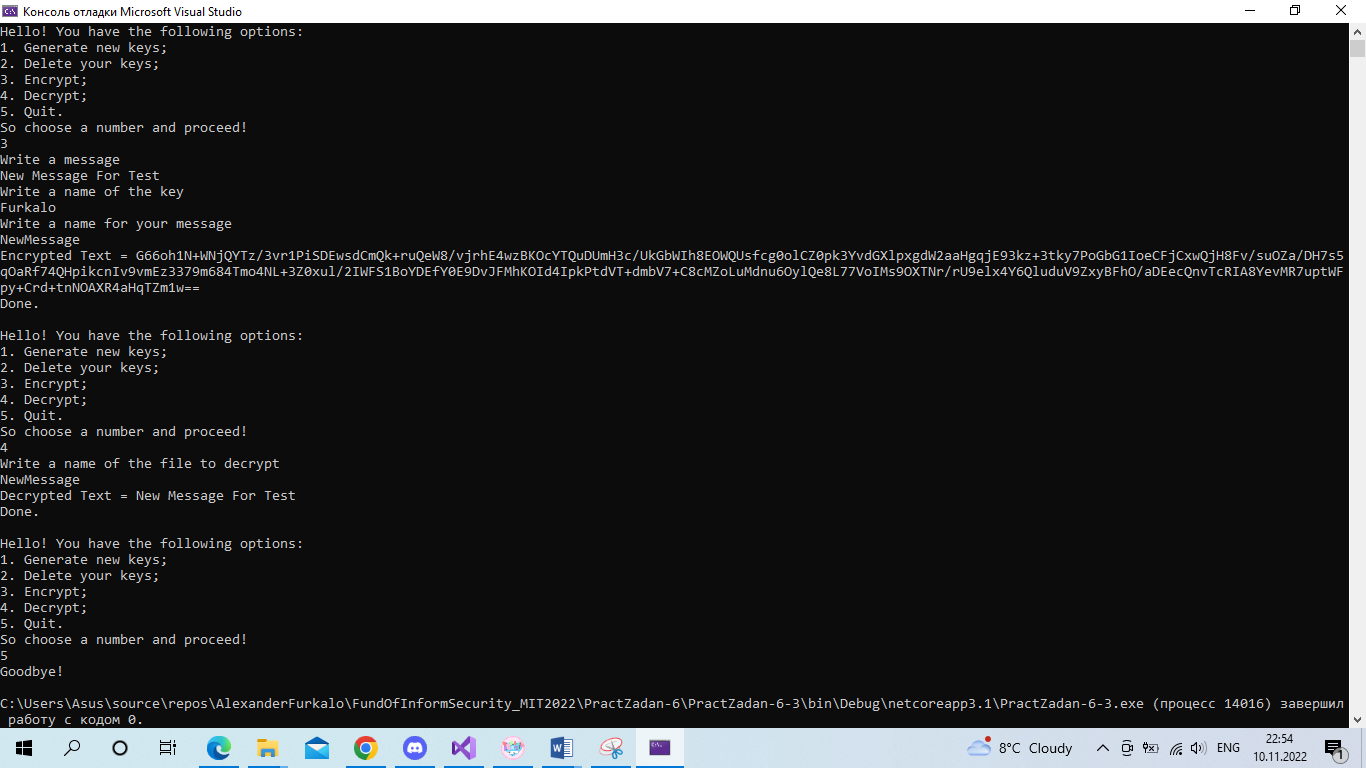
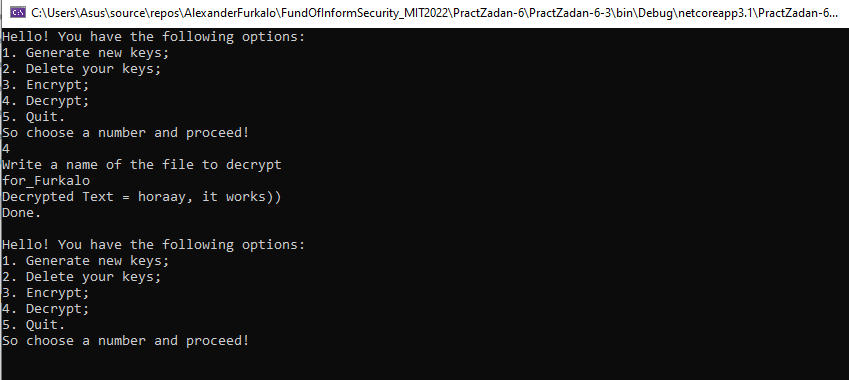
Перейдемо до першої з трьох програм:

  
  
  
  
Трохи наблизивши:  
  
  
  


Результати роботи:  
  
  
  
Основний клас, який вам потрібен, це RSACryptoServiceProvider, який використовується для генерації ключів, а також для шифрування та дешифрування. У цьому завданні, ми створюємо відкритий і закритий ключ і зберігаємо їх в об’єктах у пам’яті за допомогою змінних \_publicKey, \_privateKey типу RSAParameters. За замовчуванням довжина ключа становить 1024 біт, проте доступними є значення 2048 та 4096 біт.  
Якщо використовуємо .NET Core, цей метод створення ключів за допомогою RSACryptoServiceProvider працює між платформами в Windows, macOS і Linux, що є єдиним підтримуваним способом програмного створення ключів RSA на всіх трьох платформах.  
Для шифрування, ми передаємо дані як масив байтів, а потім створюємо новий екземпляр RSACryptoServiceProvider. Після цього імпортуємо відкритий ключ для шифрування за допомогою методу ImportParameters, параметр PersistKeyInCSP встановлен як false. Потім викликаємо метод Encrypt для об’єкта RSACryptoServiceProvider.  
Для розшифрування повідомлення, імпортуємо секретний ключ (замість відкритого) та викликаємо метод Decrypt (замість Encrypt).  
  
rsa.PersistKeyInCsp - повертає або вказує значення, чи слід зберегти ключ у постачальника служб шифрування (csp), ми вказуємо false (зберігати не потрібно).

Друге завдання:  
  
  
  
  
Приблизивши:  
  
  
  
  
Результати роботи:  
  
  
  
  
  
Це наступний метод, в якому ми зберігаємо ключі у файлі, скориставшись методом ToXmlString() класу RSACryptoServiceProvider (у цій програмі, обидва ключа зберігаються таким способом, що не є дуже безпечним та не рекомендується).   
rsa.ToXmlString - Створює та повертає рядок XML, який містить ключ поточного об'єкта RSA. Значення true, щоб включати закритий та відкритий ключ RSA (використовуємо для секретного ключа); значення false, щоб включати лише відкритий ключ (тому використовуємо для відкритого ключа).  
Ми використовуємо статичний метод WriteAllText у класі File, щоб зберегти на диску XML-текст відкритих і закритих ключів. Щоб експортувати фактичний матеріал ключа, ми використовуємо метод ToXmlString в екземплярі RSACryptoServiceParameter.  
Для шифрування та розшифрування, ми змінюємо рядок ImportParameters з першого коду на FromXmlString (Ініціалізує об'єкт RSA, використовуючи дані ключа з рядка XML). Крім того, ми вказуємо RSACryptoServiceProvider як (2048) в цьому прикладі. Під час операцій rsa.Encrypt та rsa.Decrypt, другий параметр в цьому випадку вказується як false - true використовується для виконання прямого розшифрування RSA за допомогою заповнення OAEP, та в іншому випадку false, для використання заповнення PKCS #1 v1.5.

Третє завдання:  
  
  
  
  
  
  
Приблизивши:  
  
  
  
  
  


Результати роботи:  
  
  
Для третього завдання ми використовуємо контейнер ключів (cryptographic service provider, CSP). Ключі можуть зберігатися на рівні користувача (має доступ лише певний користувач) та на рівні пристрою (мають доступ всі авторизовані користувачі).  
Щоб згенерувати асиметричний ключ RSA та зберегти його в контейнері ключів, потрібно спочатку створити екземпляр класу CspParameters і передати ім’я, яке буде використовувати для контейнера ключів, у поле CspParameters.KeyContainerName. Використовуючи властивість Flags, можете зберігати ключ або в контейнері ключів локального користувача, або в контейнері ключів на рівні машини (у цьому завданні, другий випадок). Поле ProviderName представляє конкретне ім’я, яке використовується для контейнера ключа в системі. Стандартним постачальником Windows є Microsoft Strong Cryptographic Provider.  
Щоб видалити ключ із контейнера ключа, спочатку потрібно створити інший екземпляр класу CspParameters і передати ім’я контейнера ключа у властивість CspParameters.KeyContainerName. Потім створюємо новий екземпляр класу RSACryptoServiceProvider і передаємо раніше створений об’єкт CspParameters його конструктору. Далі встановлюємо властивість PersistKeyInCSP класу RSACryptoServiceProvider на false. Нарешті, потрібно викликати метод Clear в екземплярі RSACryptoServiceProvider. Виклик цього методу звільняє всі ресурси примірника RSA та очищає контейнер ключа – так і працює видалення ключа.  
Для використання операцій Encrypt та Decrypt за допомогою контейнера, потрібно створити новий об’єкт CspParameters і завантажити ім’я контейнера у властивість KeyContainerName.   
У цьому завданні, ми використовуємо код Encrypt з другої програми. Це рішення було прийнято для того, щоб була можливість використовувати чужі відкриті ключі для зашифровування файлів.

Висновки

У цій практичній роботі, ми дослідили асиметричне шифрування як засіб забезпечення конфіденційності інформації. Дізналися про використання криптографічних бібліотек для здійснення асиметричного шифрування, про збереження відкритого ключа у файл та зчитування відкритого ключа з файлу. Написали програму, яка виконує зашифровування та розшифровування даних з використанням алгоритмів асиметричного шифрування RSA, та де пара ключів зберігається у пам’яті; реалізували можливість збереження відкритого ключа у файлі та можливість зашифровувати повідомлення за допомогою файлів відкритих ключів інших користувачів. Згенерували відкритий ключ RSA 2048bit та зберегли у файл з іменем, що відповідає транслітерації прізвища, та розширенням xml, поділилися збереженим відкритим ключем у Телеграм-каналі. Зашифрували та надіслали вітання двом студентам, отримані вітання розшифрували.