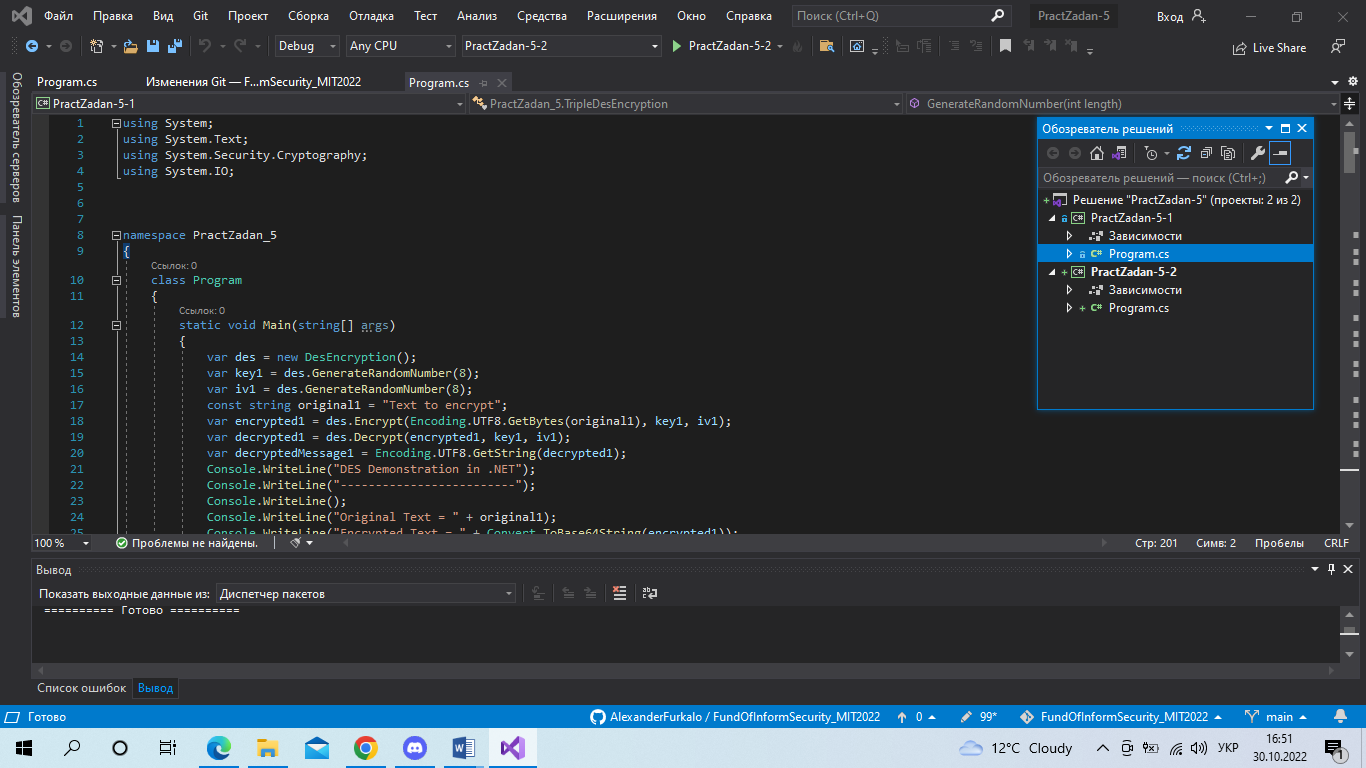
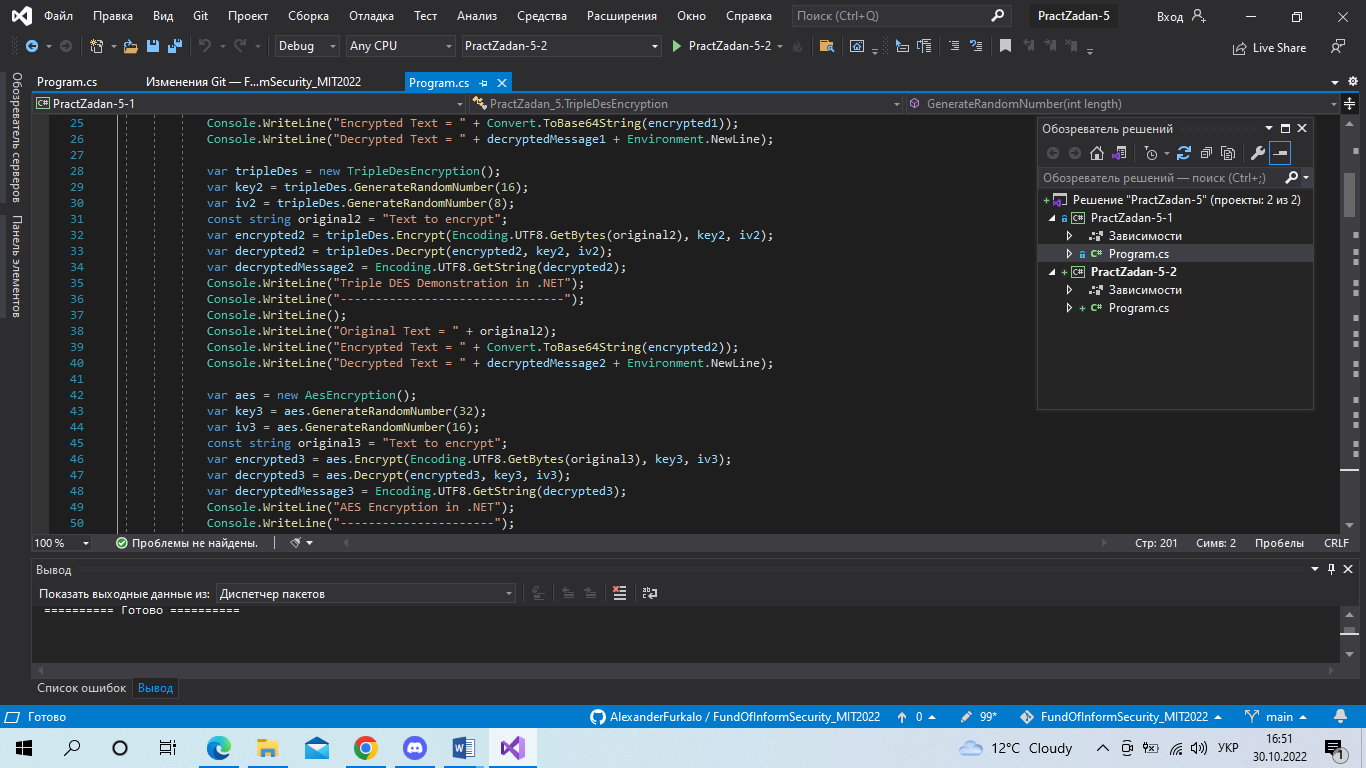
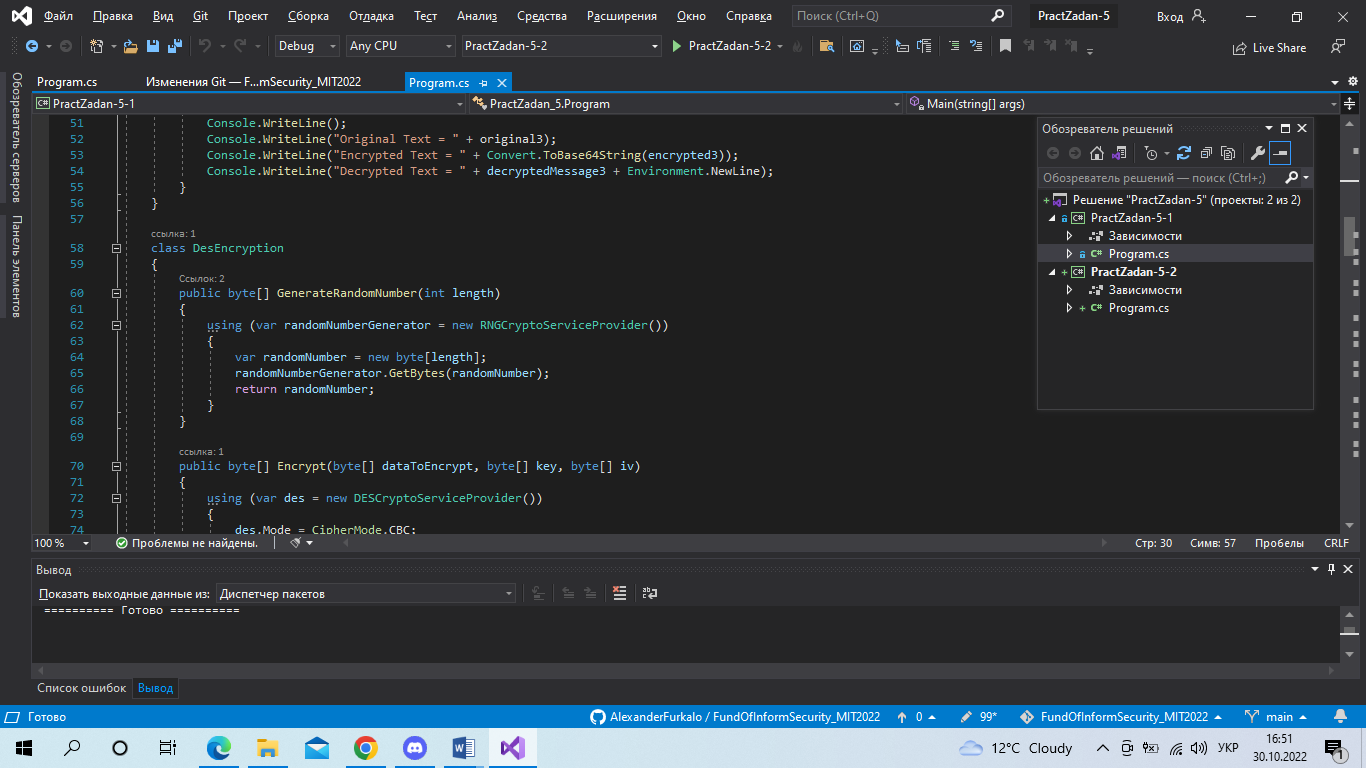
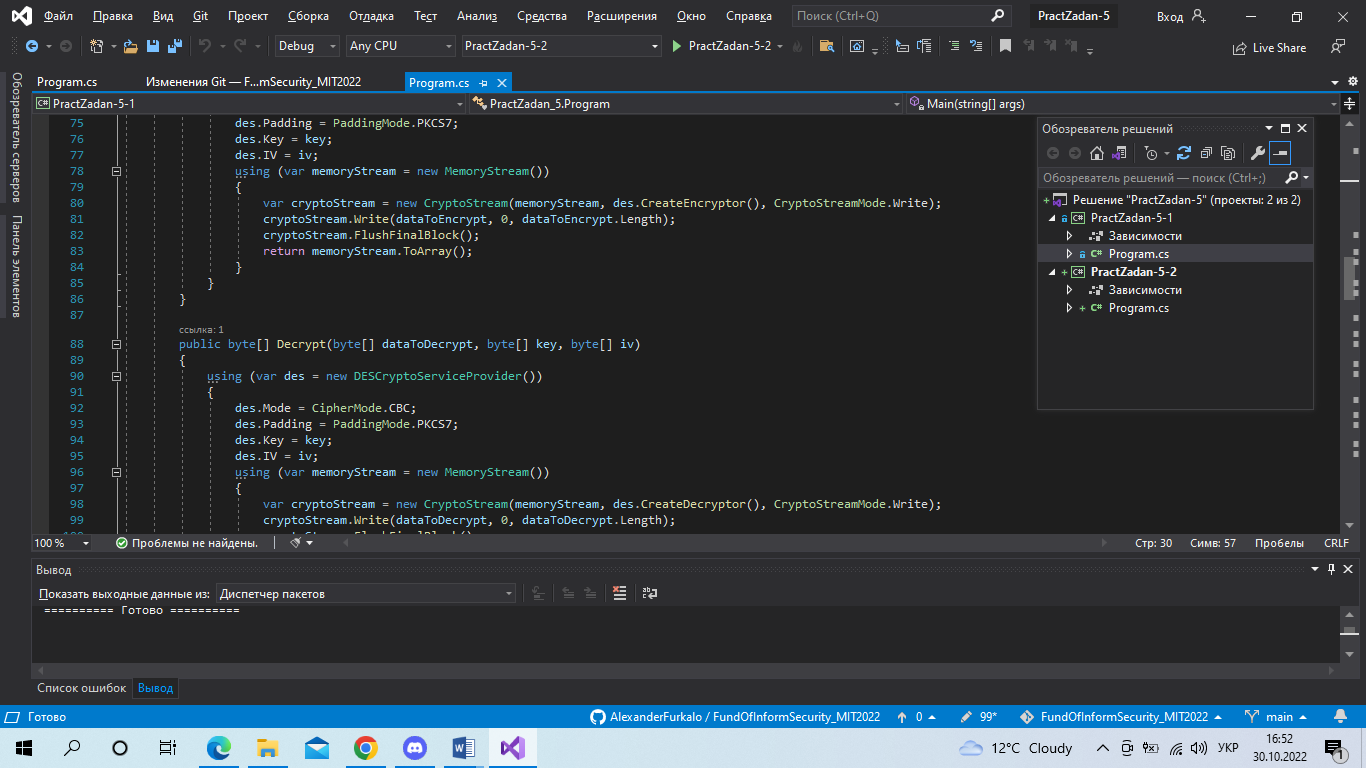
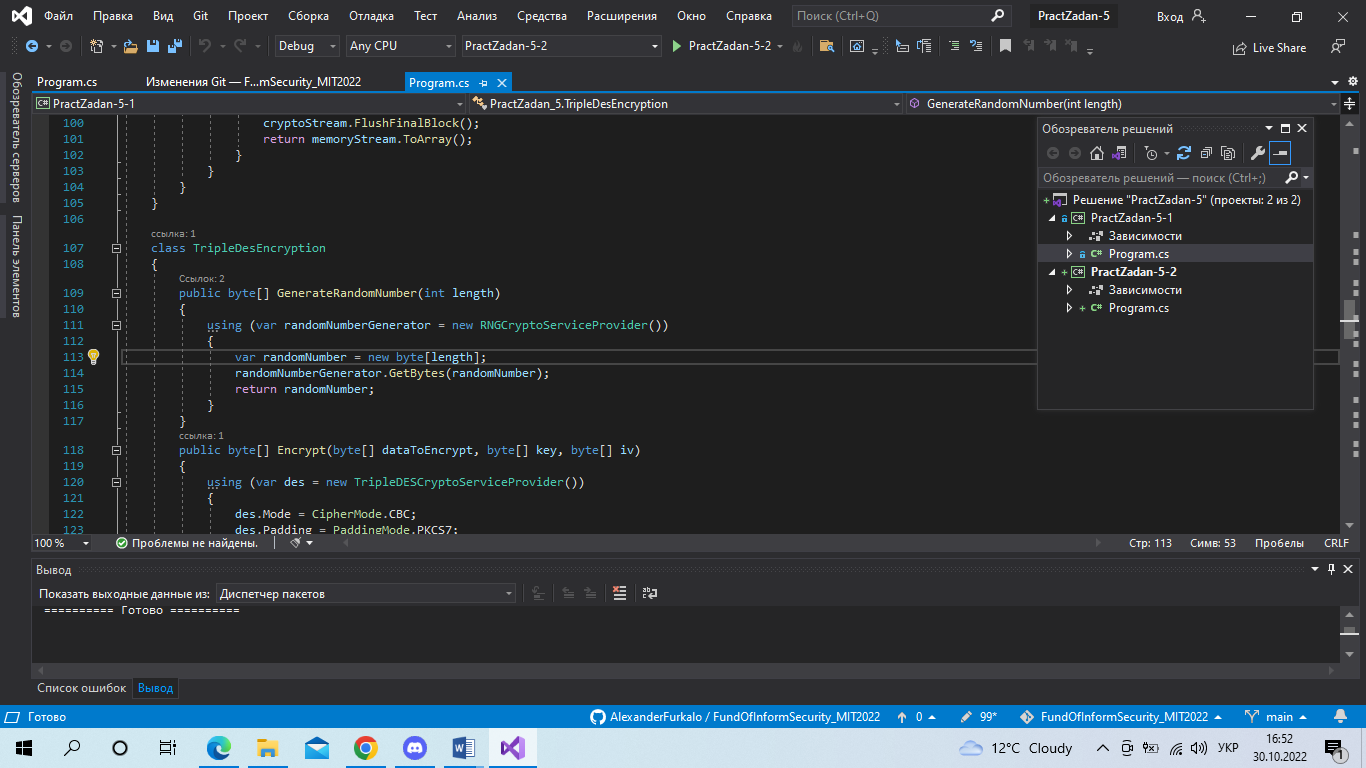
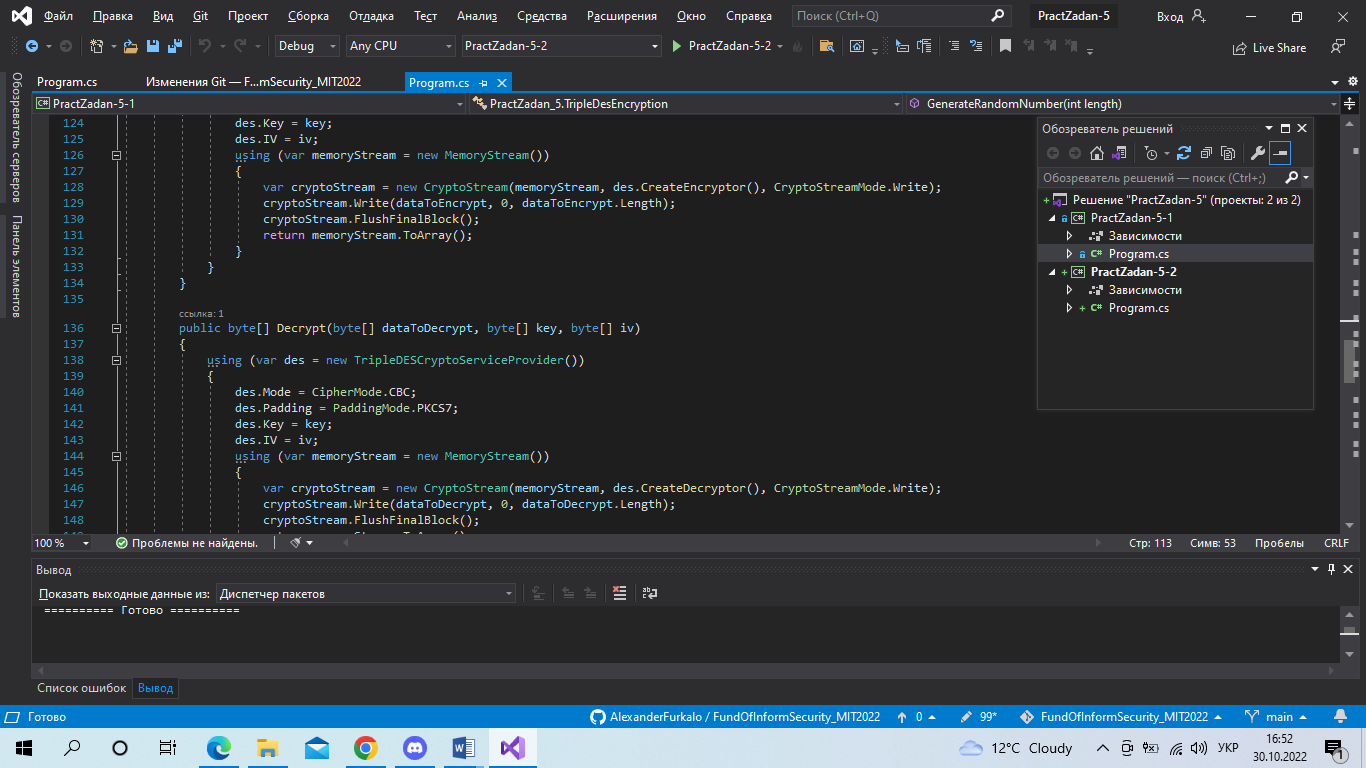
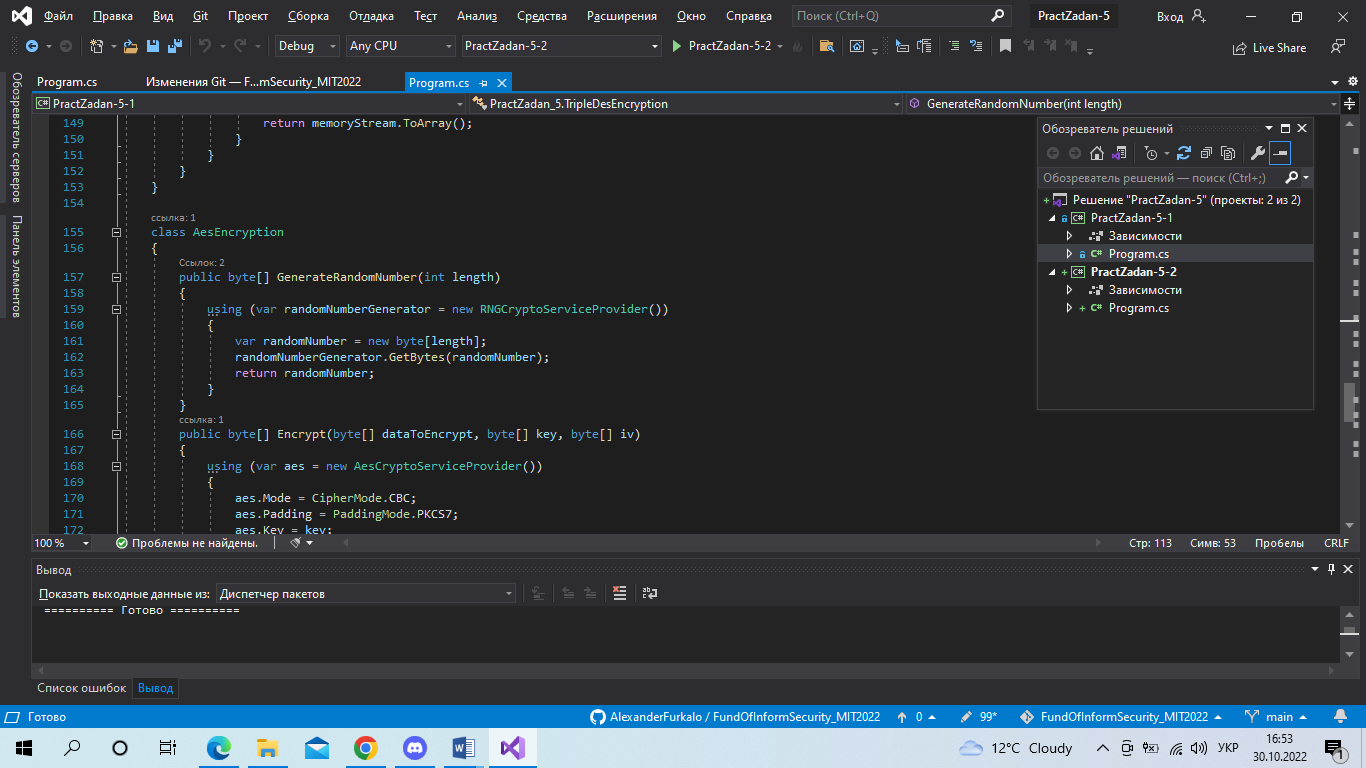
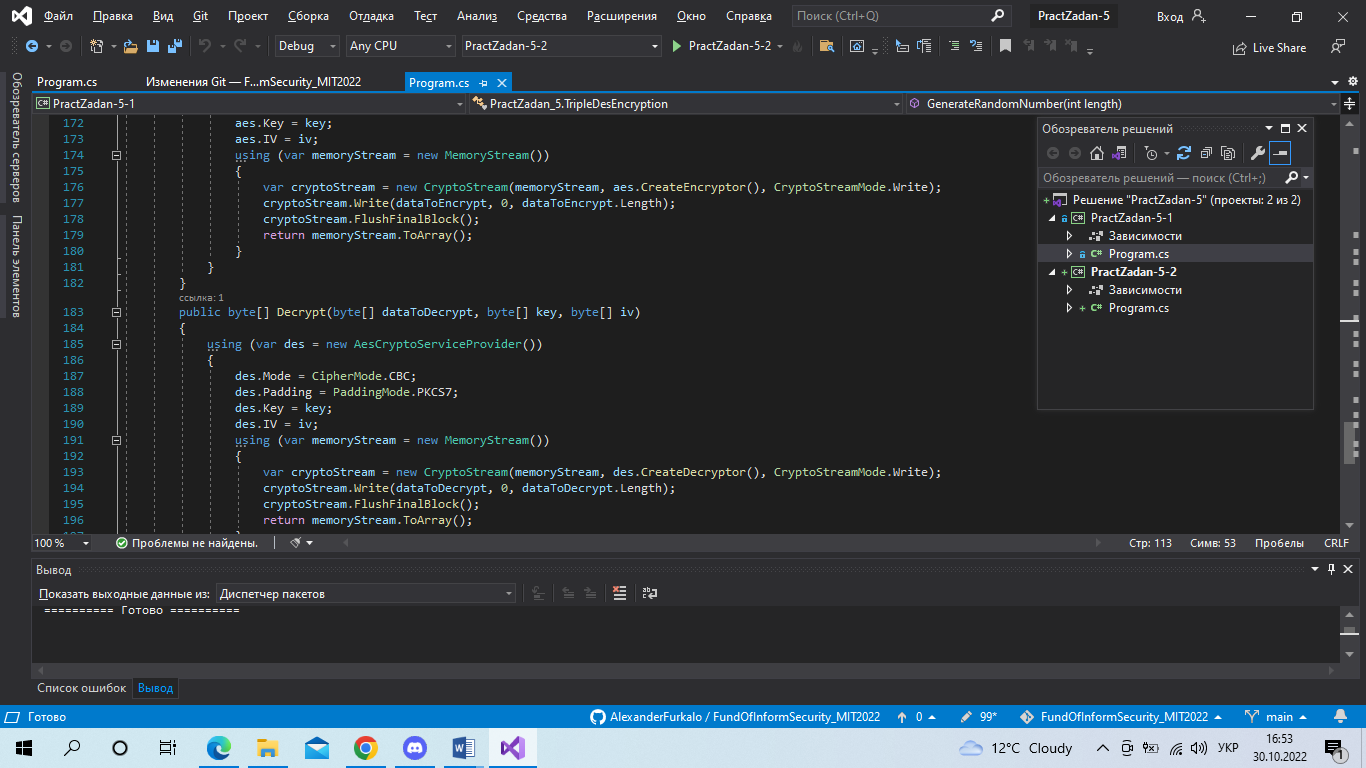
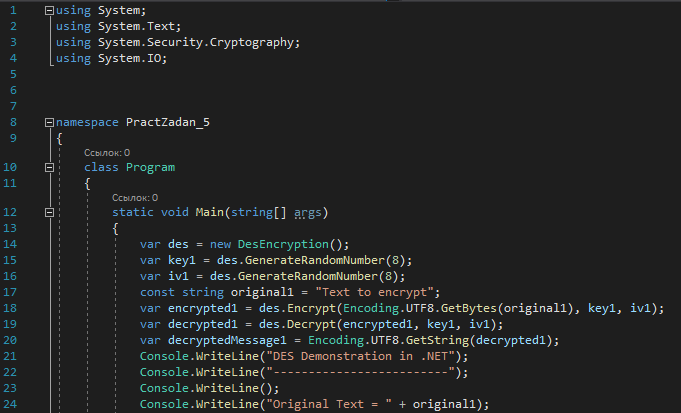
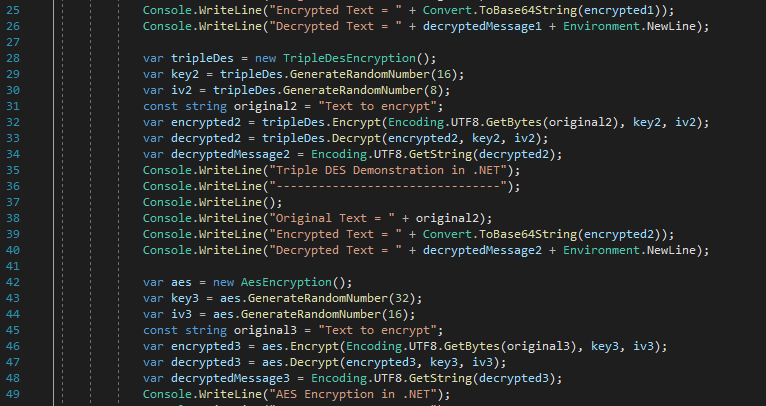
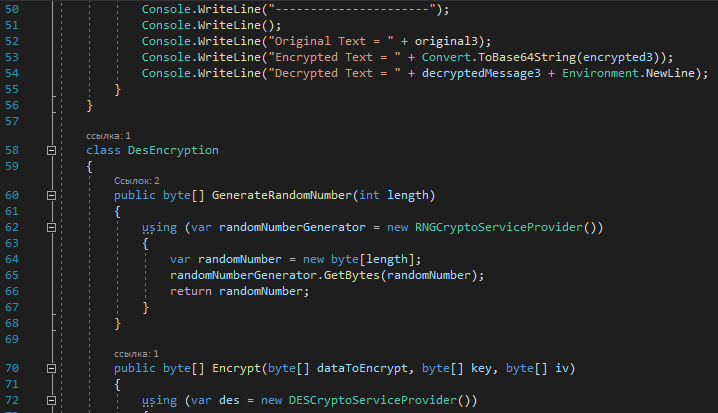
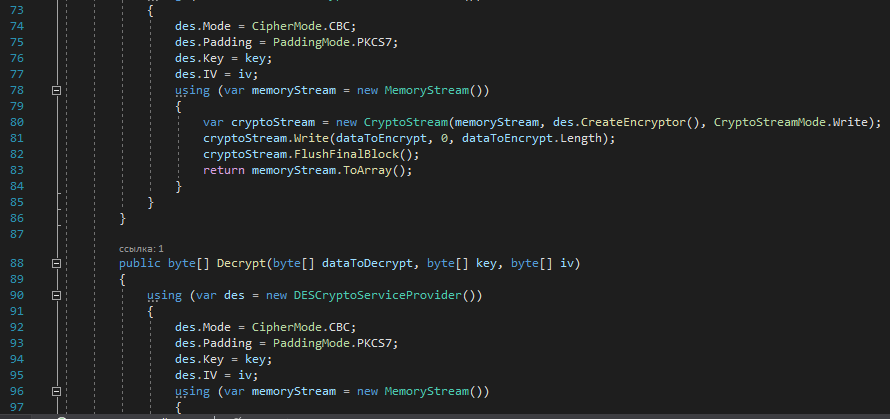
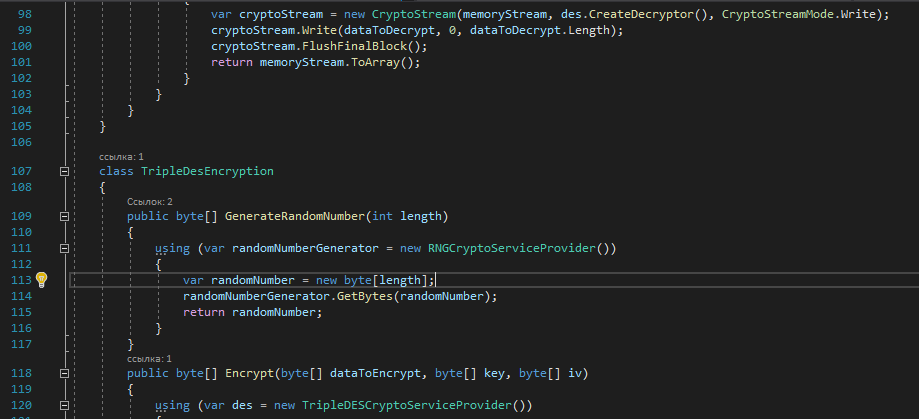
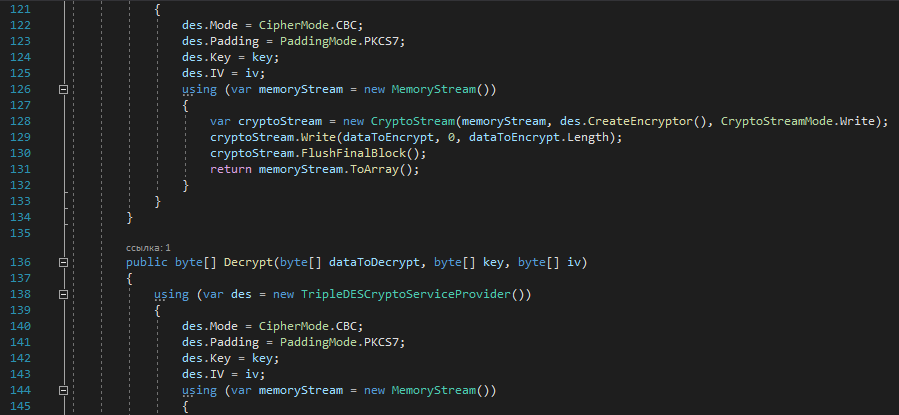
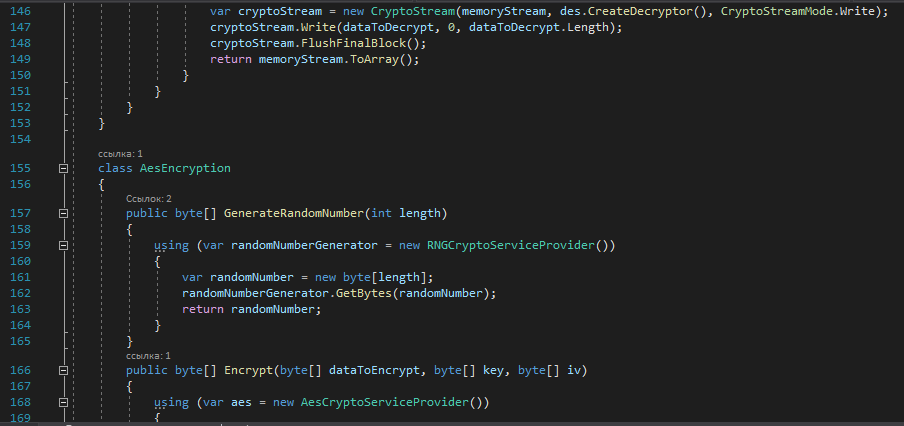
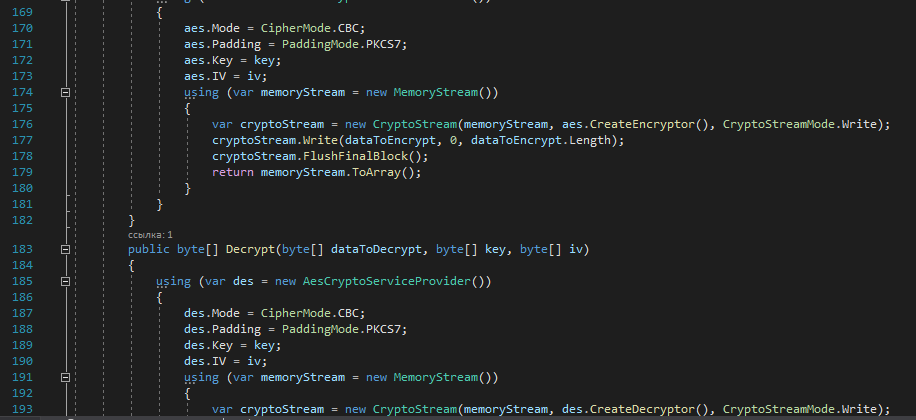
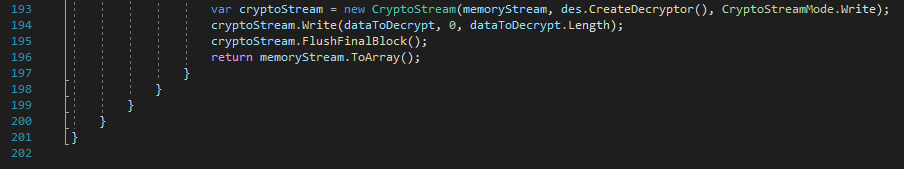
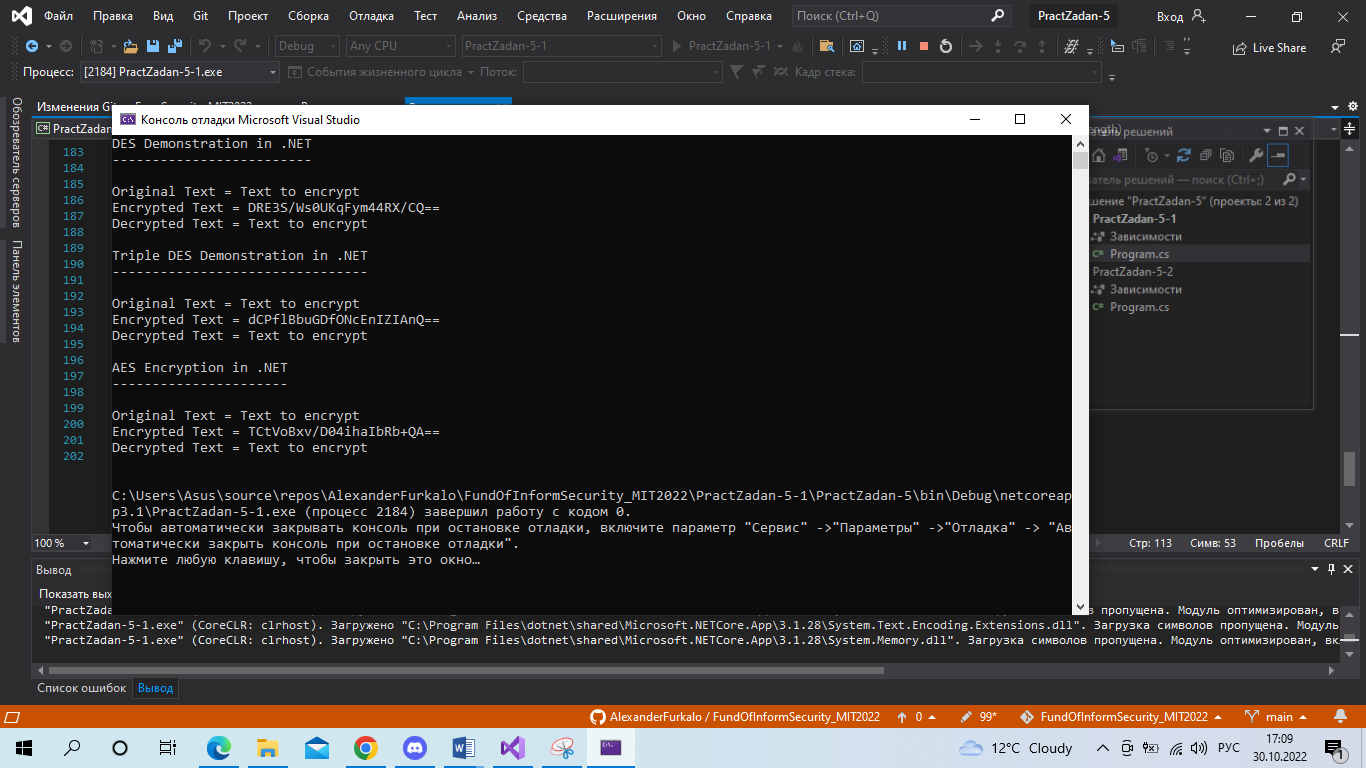
ПРАКТИЧНА РОБОТА №5  
З ОСНОВ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ  
СТУДЕНТА КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКО  
ФАКУЛЬТЕТУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  
КАФЕДРИ МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ  
ДРУГОГО КУРСУ, ДРУГОЇ ПІДГРУПИ  
ОЛЕКСАНДРА ОЛЕКСАНДРОВИЧА ФУРКАЛА  
ЗА ТЕМОЮ “ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ІНФОРМАЦІЇ З ВИКОРИСТАННЯМ СИМЕТРИЧНОГО ШИФРУВАННЯ”  
ЗВІТ  
30.10.2022

Завдання цієї практичної роботи були такі:  
1. Написати програму, яка виконує зашифровування та розшифровування даних з використанням алгоритмів симетричного шифрування DES, Triple-DES, AES. Секретний ключ та вектор ініціалізації генерується випадковим чином.   
2. Для програми з пункту 1 реалізувати можливість задання секретного ключа та вектору ініціалізації за допомогою псевдовипадкової послідовності із використанням пароля. «Сіль» генерувати як випадкову послідовність байтів. Число ітерацій = номер варіанта (тобто, 23) \* 10'000. Потрібно використовувати клас Rfc2898DeriveBytes  
3. Оформити звіт

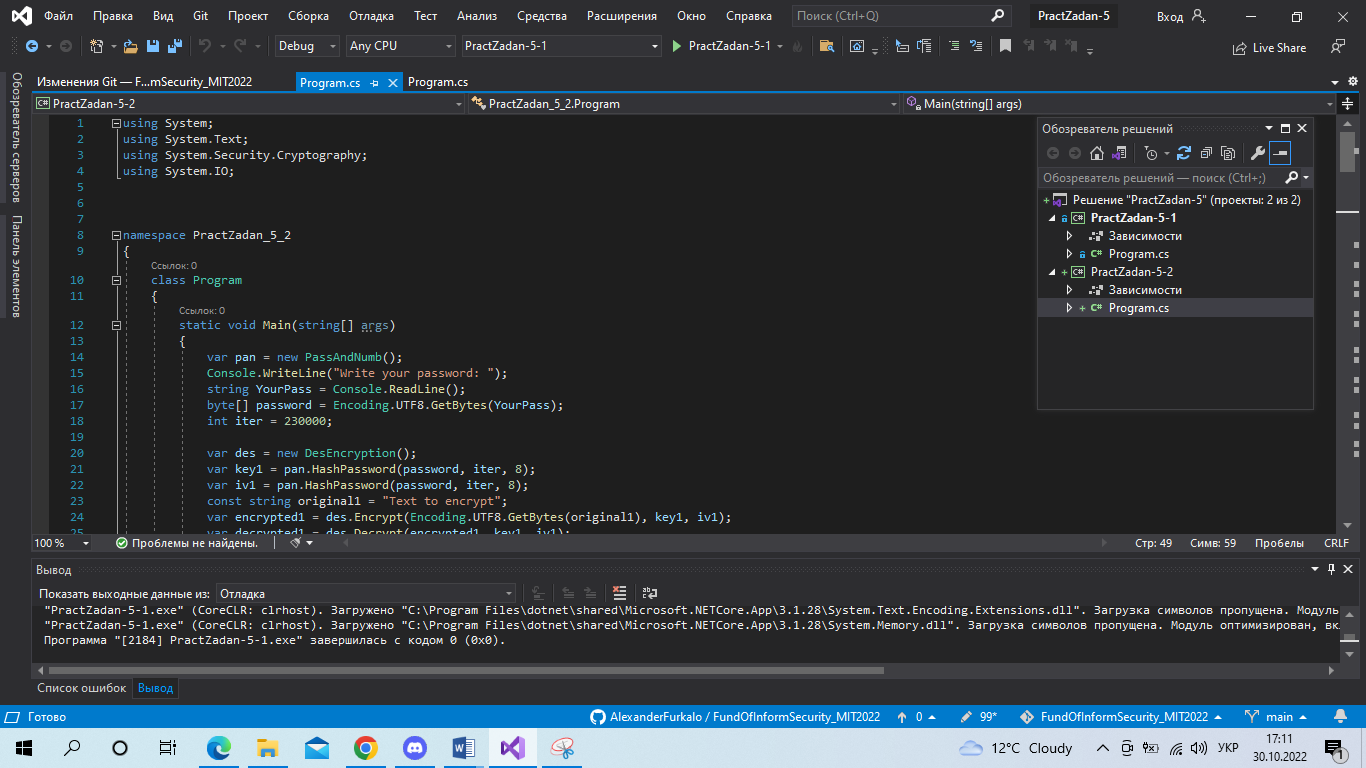
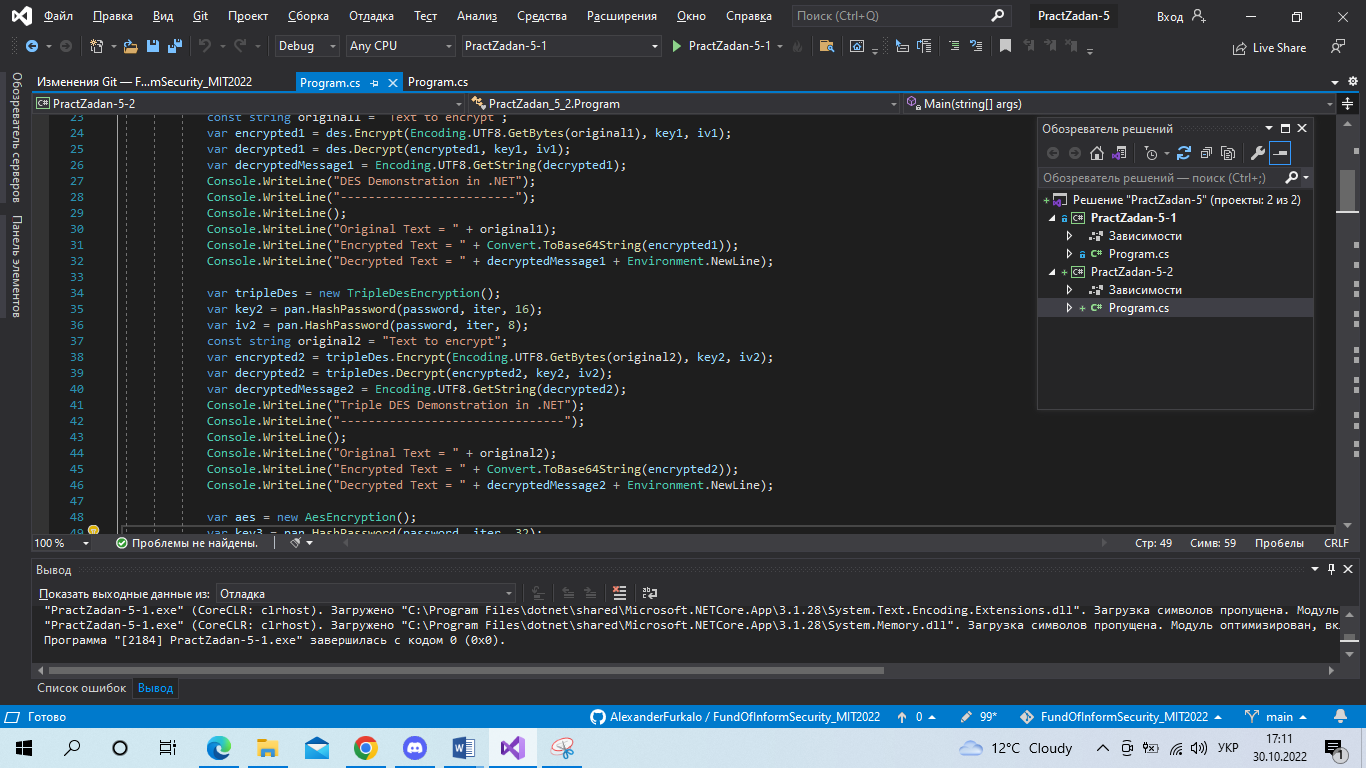
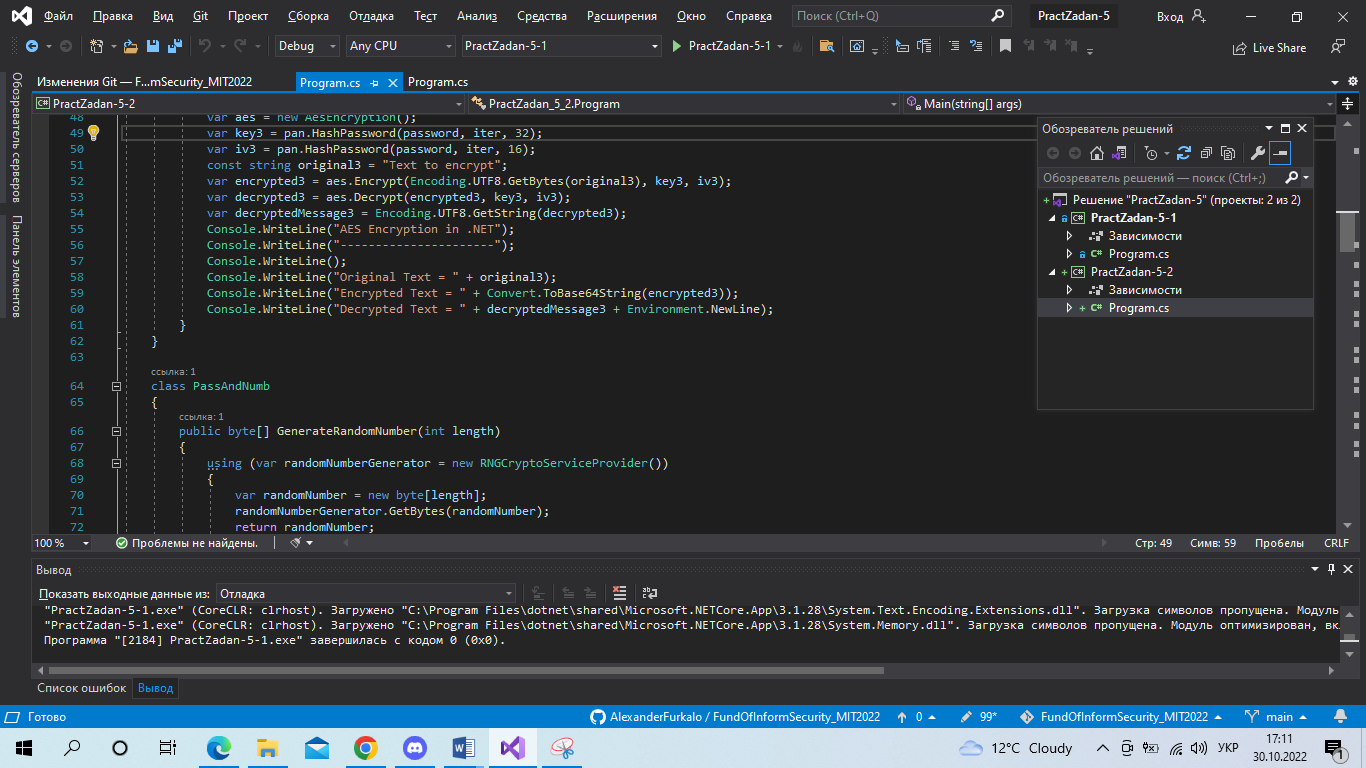
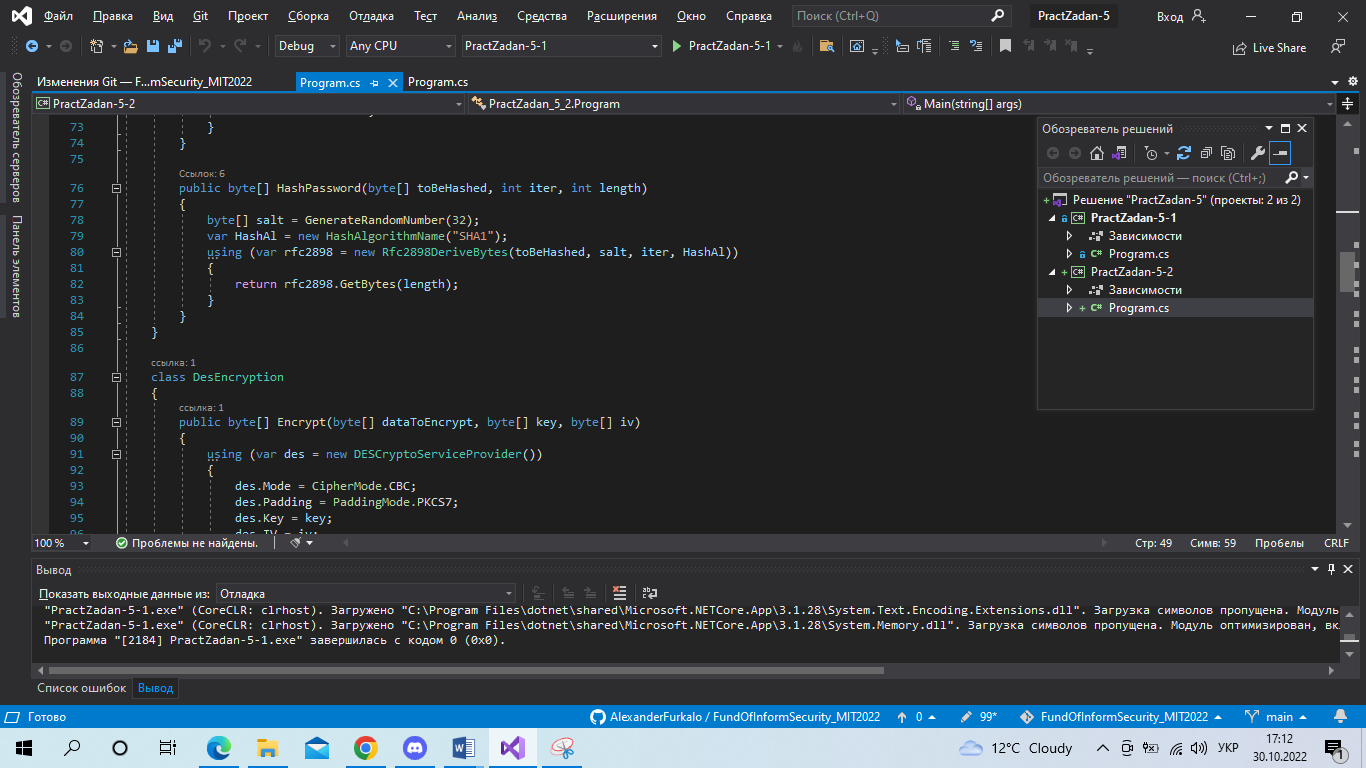
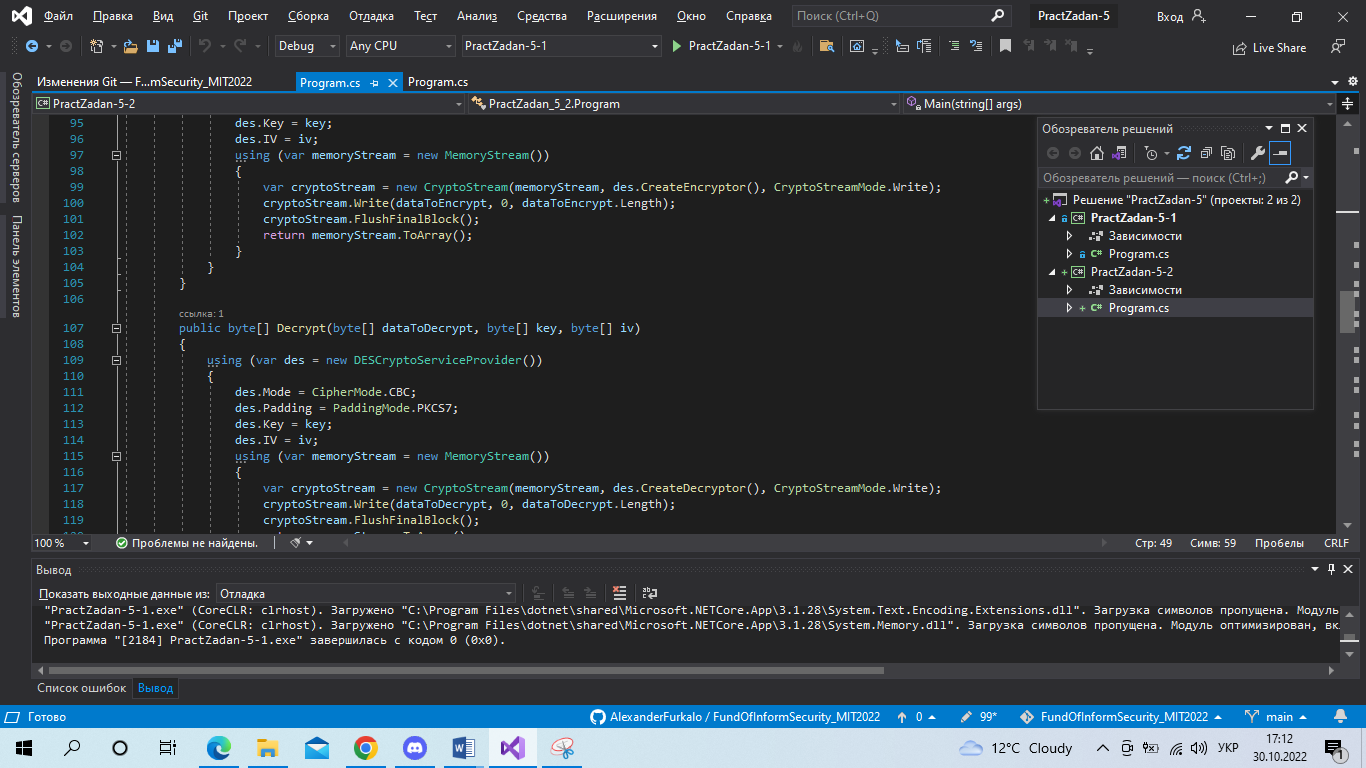
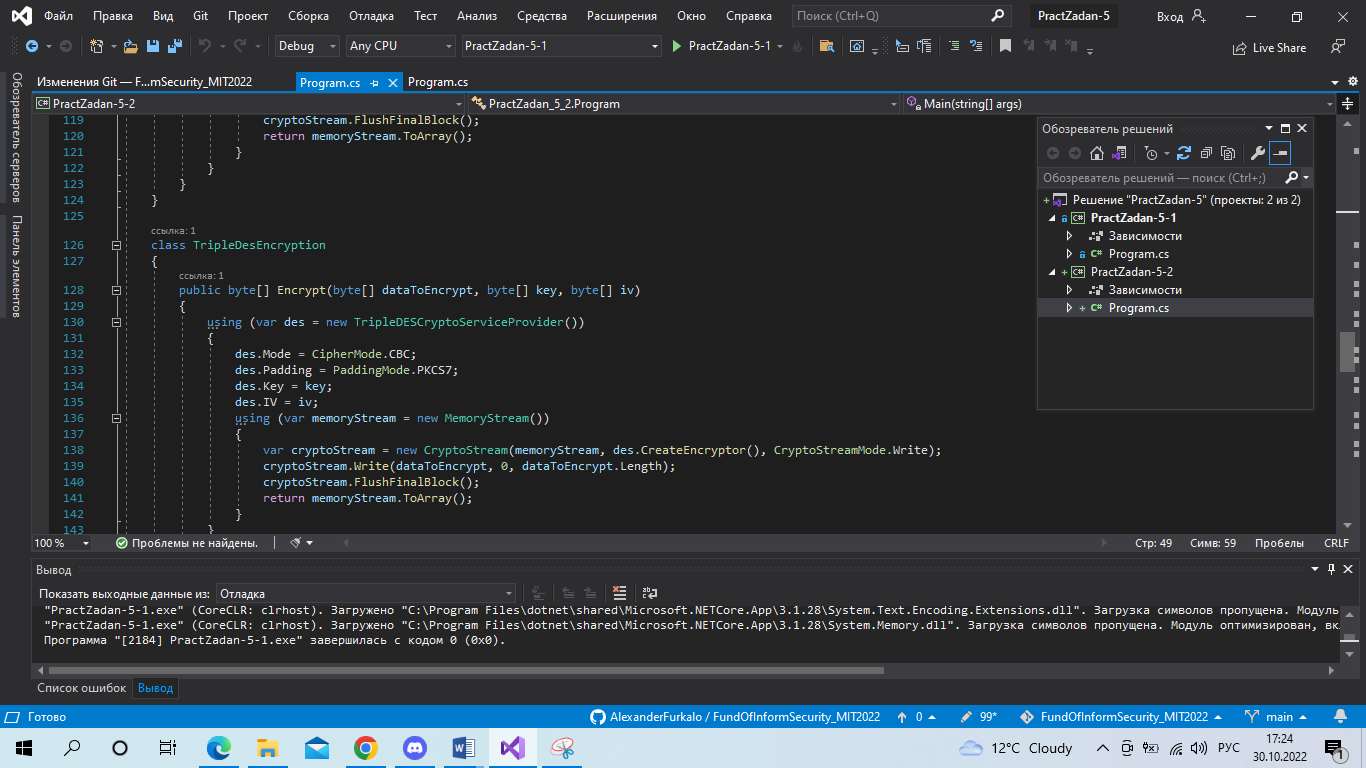
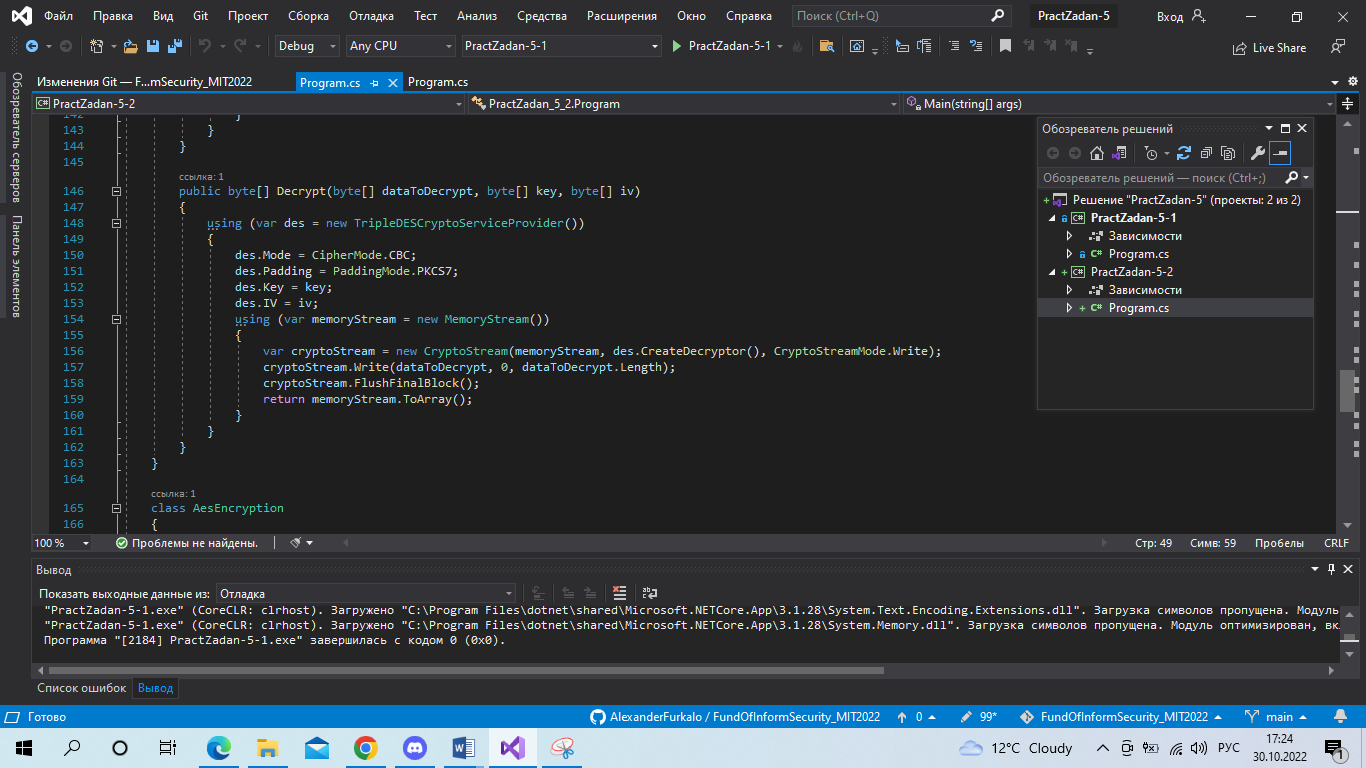
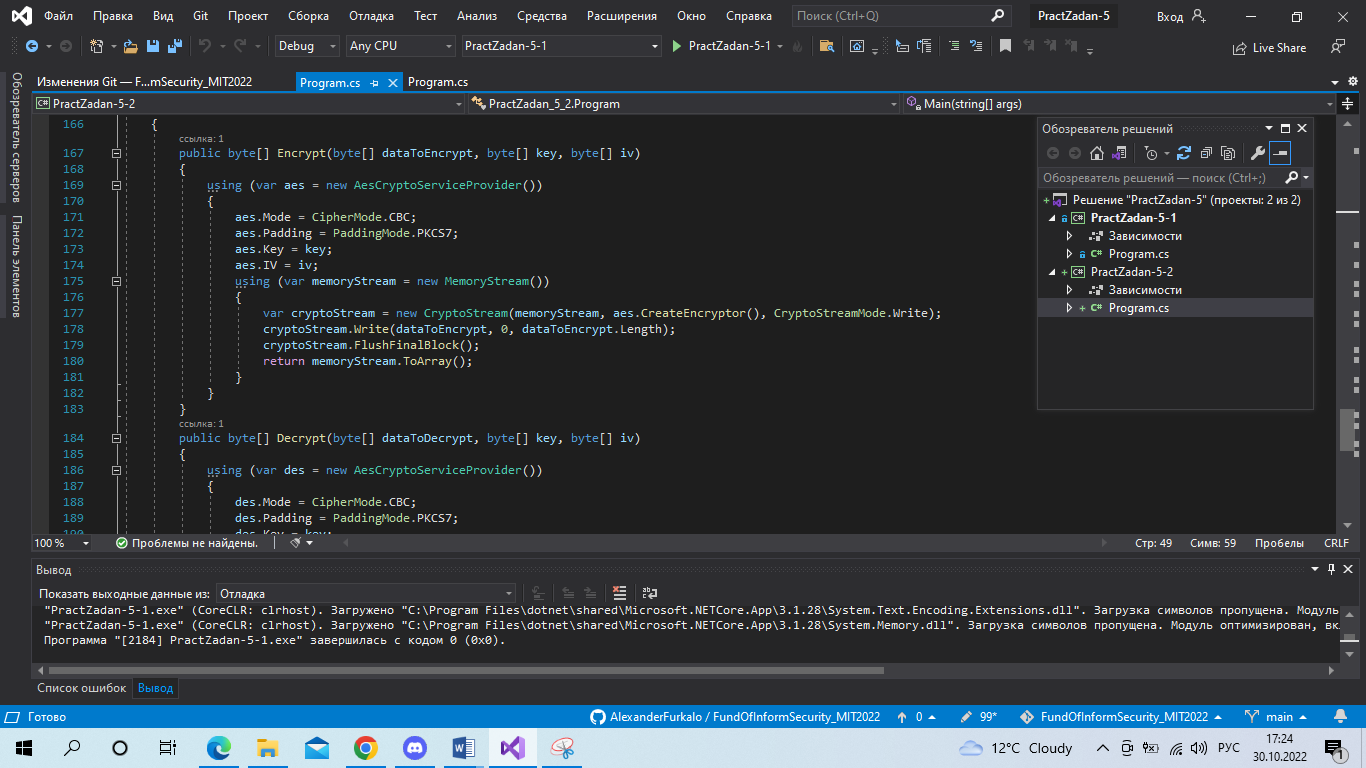
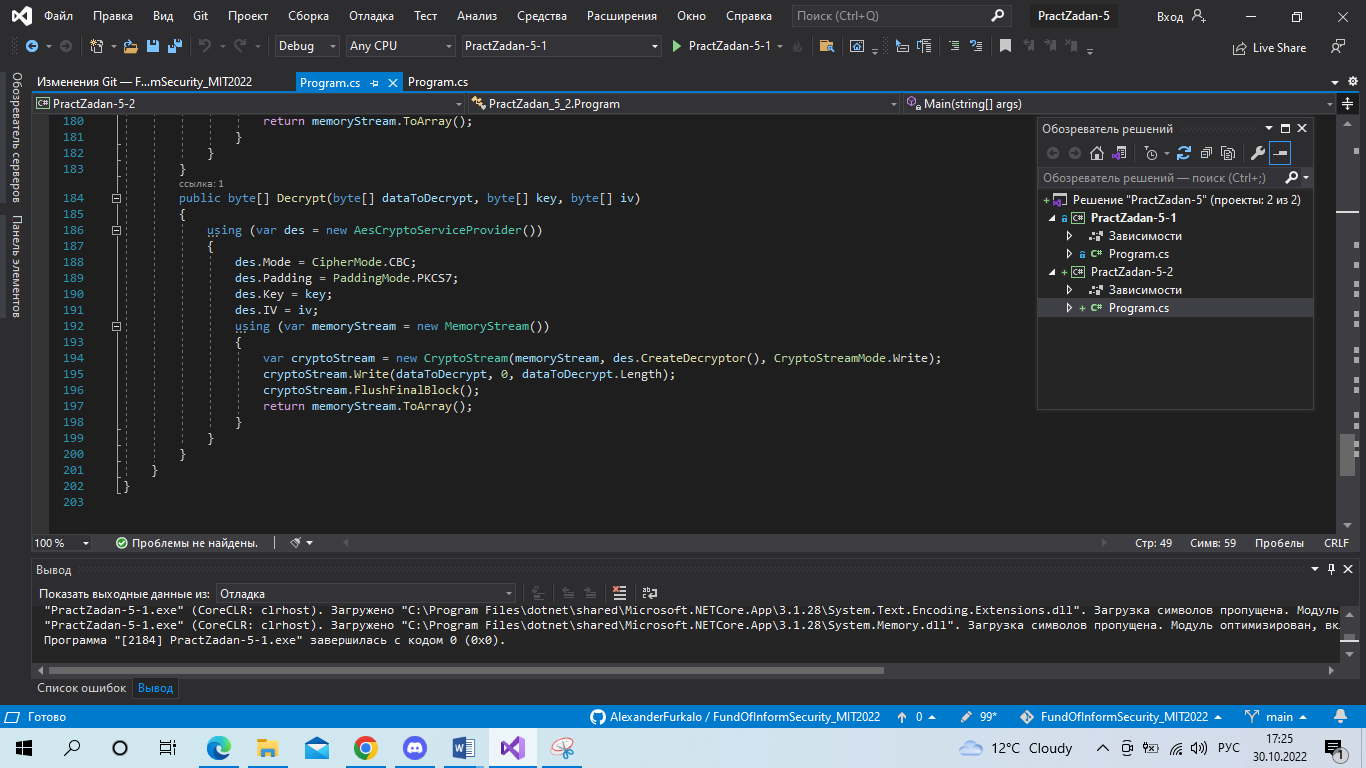
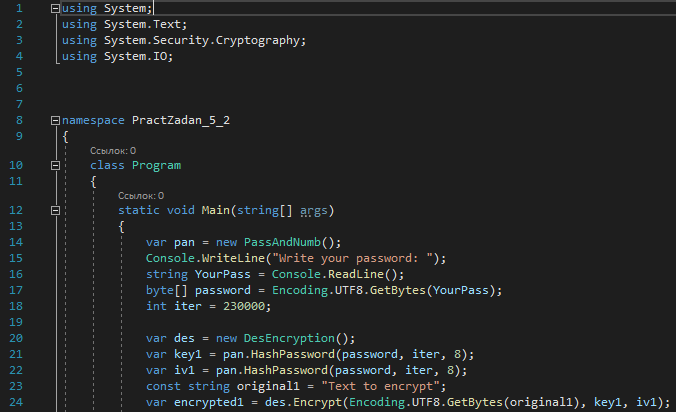
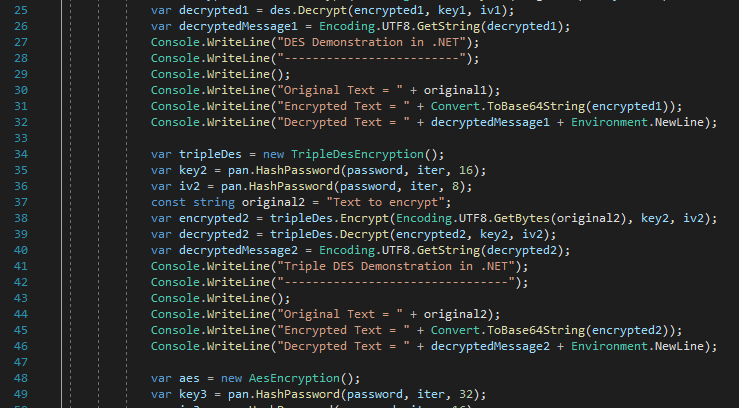
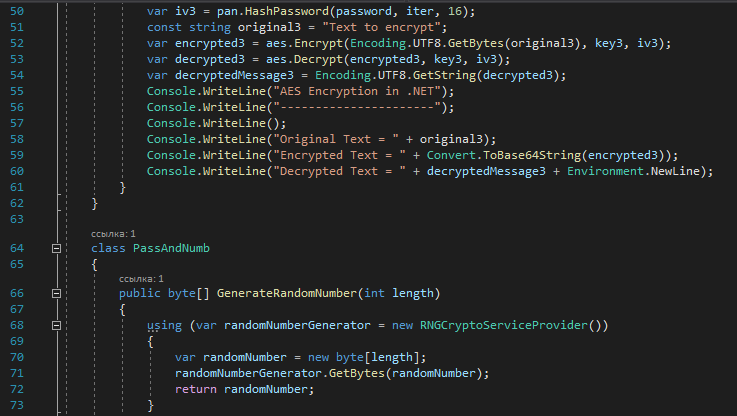
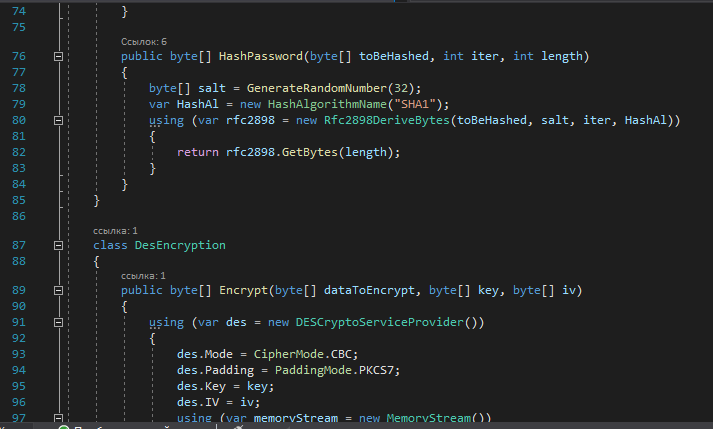
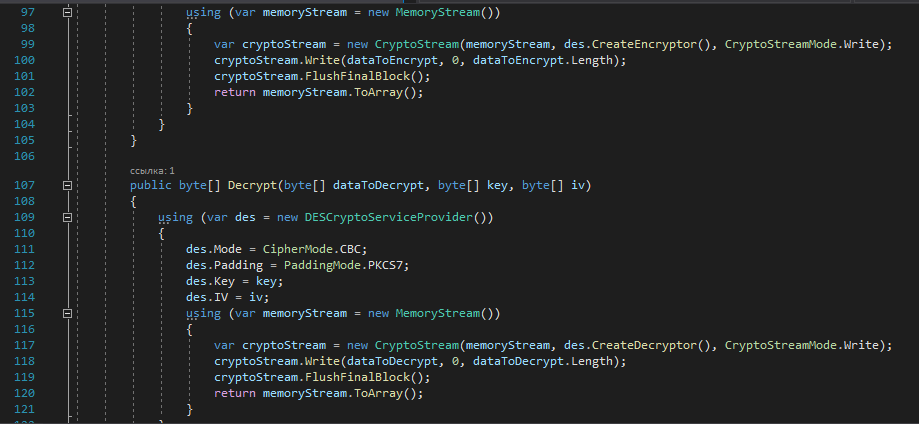
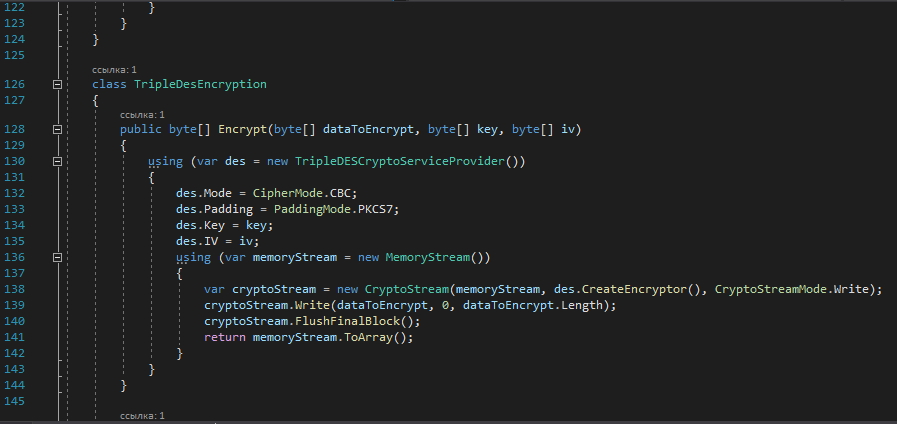
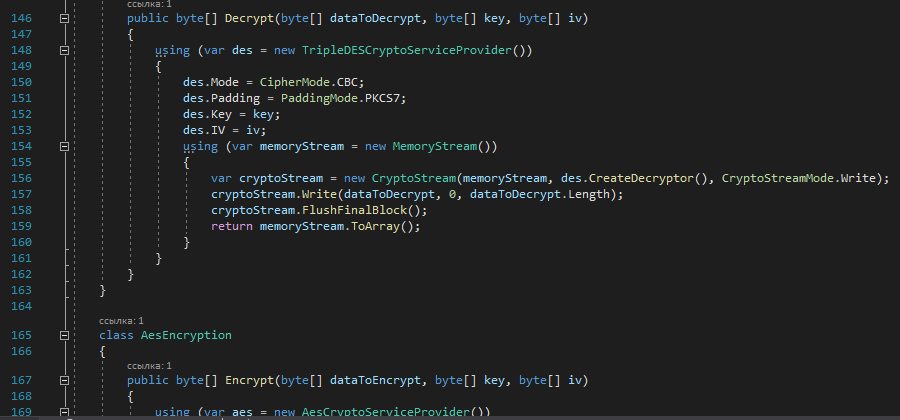
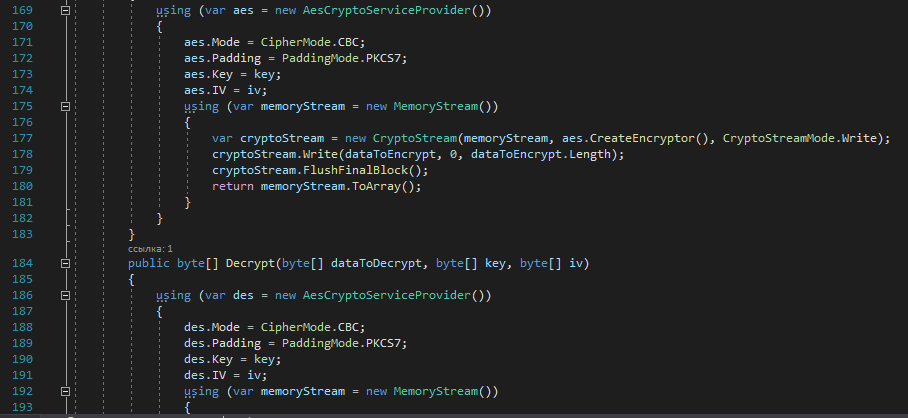
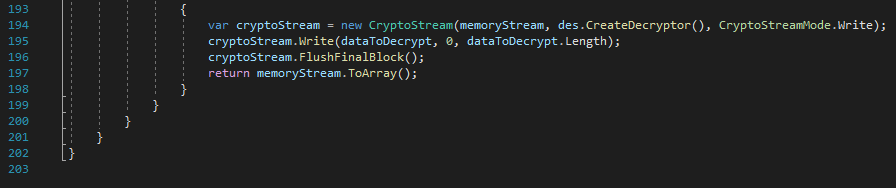
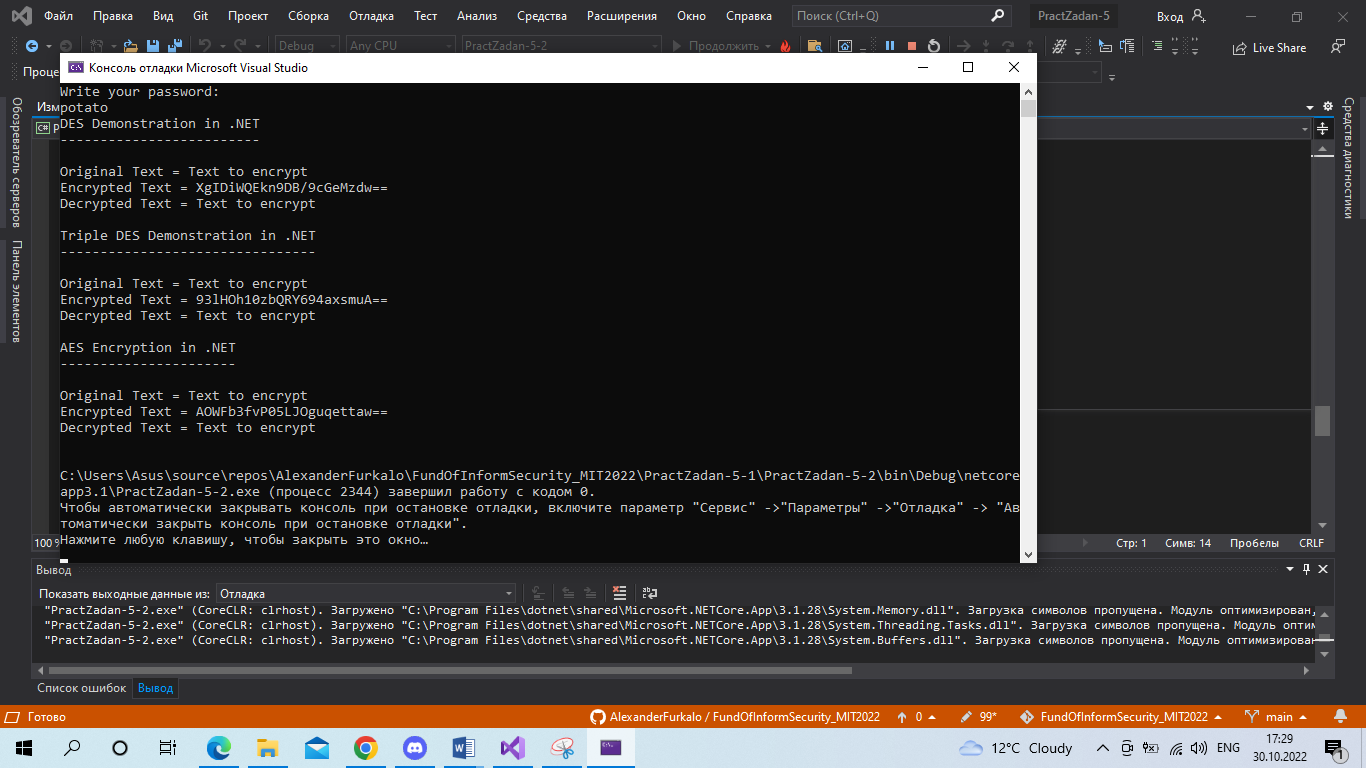
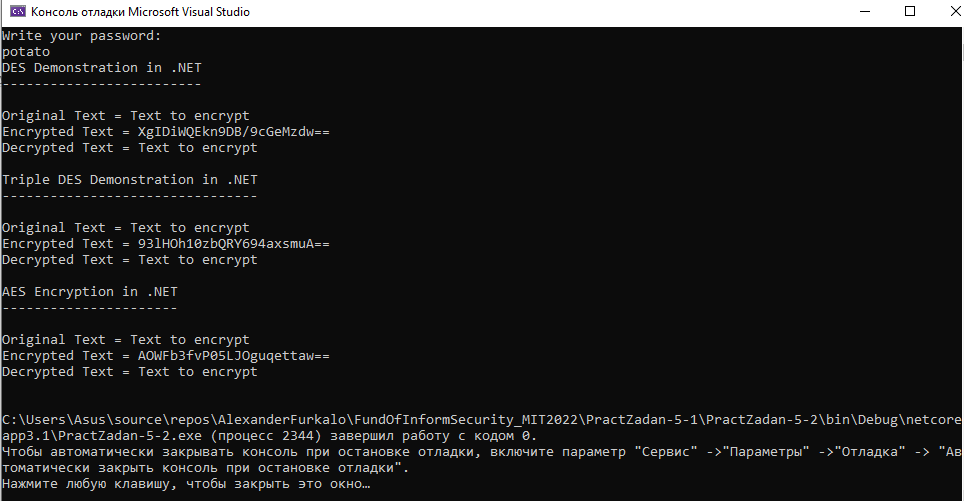
Хешування, яке ми розглядали на минулих заняттях – це одностороння операція, тобто ми не можемо повернутися до вхідних даних після того, як їх зашифруємо. Симетричні алгоритми шифрування, однак, мають можливість дешифрувати повідомлення за допомогою ключа, яким було спочатку воно зашифроване (саме тому вони називаються “симетричними”).  
З переваг: надзвичайна секретність (зашифроване повідомлення буде винятково секретним) та швидкість (зашифровування та розшифровування легкі для виконання)  
З недоліків: потрібно мати надійний спосіб передати секретний ключ та небезпека у випадку компрометації ключа (той, хто має ключ, має й можливість розшифрувати всі зашифровані цим ключем дані).  
Ми розглядаємо три алгоритми: DES, Triple DES та Aes.  
Шифрування у DES відбувається 64-бітними блоками (Рис. 2), де вхідні розділяються на 8-ми байтові (64-бітні) ланки, які зашифровуються з використанням 56-бітного секретного симетричного ключа для забезпечення конфіденційності. За часів розробки, такий ключ був достатнім для забезпечення стійкості алгоритму до brute-force атак, але із зростанням обчислювальної потужності довелося перейти до наступного алгоритму, triple DES.  
Потрійне застосування DES алгоритму дозволило збільшити розмір ключа для захисту від brute-force атак без необхідності розробляти новий алгоритм. Працює він так:  
- Подається відкритий текст – він зашифровується ключем 1  
- Після цього, отриманий текст зашифровується ключем 2  
- У кінці, вже цій текст зашифровується ключем 3 (або ж знову ключем 1).  
AES — це варіант шифру Rijndael, який використовує фіксований розмір блоку 128 біт і ключ змінного розміру 128, 192 або 256 біт. Одна з головних цілей розробки AES полягала в тому, щоб зробити його більш простим для реалізації як апаратного, так і програмного забезпечення. AES працює, повторюючи ті самі визначені кроки кілька разів, які називаються раундами. Кожен раунд шифрування складається з кількох етапів обробки, у тому числі з використанням підключа шифрування/дешифрування, створеного зі спільного ключа. Зазвичай, кількість цих раундів – 10, 12 або 14, в залежності від довжини ключа.

Перейдемо до першої з двох програм:

  
  
  
  
  
  
  
  
Трохи наблизивши:  
  
  
  
  
  
  
  
  


Результати роботи:  
  


Для того, щоб використати будь-який алгоритм симетричного шифрування, нам потрібні секретний ключ та вектор ініціалізації.  
- Секретний ключ: масив байтів, у якому зберігається ключ шифрування перед виконанням операцій шифрування та дешифрування.  
- Вектор ініціалізації: масив байтів, у якому зберігається випадкове число, яке використовується разом із ключем для шифрування. Це число також називається nonce (одноразове число, NumberOnce) і використовується лише один раз у будь-якому сеансі шифрування. Використання IV запобігає повторенню шифрування даних, що ускладнює пошук шаблонів для зламу шифру хакером, який використовує атаку за словником. Наприклад, послідовність може з’являтися двічі або більше в тілі повідомлення.  
Обидва вони в коді отримуються за допомогою GenerateRandomNumber – передаємо якесь ціле число (наприклад, 8), яка буде представляти собою довжину байтового масиву, який буде заповнений випадковими даними – вони створюються за допомогою RNGCryptoServiceProvider.  
Ключ, вектор та повідомлення, яке буде зашифроване, передаються в Encrypt, який першим рядком викликає DESCryptoServiceProvider. Після цього, потрібно задати Mode та Padding: перше задає режим блочного шифру для використання під час шифрування, друге задає тип заповнення, що використовується, коли блок даних повідомлення коротше за повне число байтів, необхідного для криптографічної операції. Ми використовуємо CBC та PKCS7:  
- Mode CBC: реалізує зворотний зв'язок. Перед шифруванням кожного блоку відкритого тексту він поєднується із зашифрованим текстом попереднього блоку за допомогою побітової операції "що виключає АБО". При цьому навіть якщо відкритий текст містить велику кількість ідентичних блоків, при їх шифруванні будуть отримані різні блоки зашифрованого тексту. Перед шифруванням першого блоку відкритого тексту з ним за допомогою побітової операції "що виключає" об'єднується вектор ініціалізації.  
- Pudding PKCS7: Рядок заповнення складається з послідовності байтів, кожен з яких записується загальна кількість доданих байтів заповнення.  
Тільки вже потім передаються значення секретного ключа та вектору ініціалізації.   
Тоді код також викликає клас MemoryStream та CryptoStream. У CryptoStream, ми створюємо об’єкт-шифратор алгоритму симетричного шифрування за допомогою CreateEncryptor та з обраними нами параметрами. Крім того, ми задаємо режим криптографічного потоку як Write, тобто “для запису”.  
FlushFinalBlock оновлює базове джерело даних або сховище поточного вмісту буфера, а потім очищає буфер.   
Write приймає дані, які ми хочемо зашифрувати, і їх довжину в байтах і виконує операцію шифрування. Потім ми викликаємо FlushFinalBlock, який оновлює джерело даних поточним станом буфера. Потім буфер очищається. Наші зашифровані дані повертаються до методу виклику.  
Для розшифрування зашифрованого ряду даних, ми використовуємо CreateDecryptor замість CreateEncryptor – в іншому, процес буде ідентичний.  
Така ж ситуація буде й з TripleDes та Aes – в коді для них ми змінюємо лише перший рядок в Encrypt та Decrypt (замість DESCryptoServiceProvider – TripleDESCryptoServiceProvider або AesCryptoServiceProvider).

Друге завдання:  
  
  
  
  
  
  
  
  
  
Приблизивши:  
  
  
  
  
  
  
  
  
  
Результати:  
  


Трохи перероблена версія з минулого завдання. В цьому випадку, для генерації ключа та вектору використовується не GenerateRandomNumber, а метод з минулої лабораторної роботи, Rfc2898DeriveBytes (але треба зазначити, що RngCryptoServiceProvider залишився – він потрібен для генерації солі).  
В цій версії перед основним кодом у користувача просять ввести пароль. Після цього, він використовується у розрахунках ключа та вектору (key, iv), сіль у коді не зберігається та генерується окремо на кожному виклику.

Висновки

У цій практичній роботі, ми дізналися про алгоритми симетричного шифрування DES, TRIPLE-DES та AES; написали програму, яка виконує зашифровування та розшифровування даних з використанням алгоритмів симетричного шифрування DES, Triple-DES, AES, де секретний ключ та вектор ініціалізації генерується випадковим чином. Для цієї програми також реалізували можливість задання секретного ключа та вектору ініціалізації за допомогою псевдовипадкової послідовності із використанням пароля - «Сіль» генерували як випадкову послідовність байтів, а число ітерацій – 230.000