**Звіт ПЗ\_5**

Безпечне зберігання паролів.

Квартюк Олексій

1) В наш час безглуздо зберігати паролі в простому вигляду, адже будь-хто може скористатися ним та заволодіти нашою інформацією.

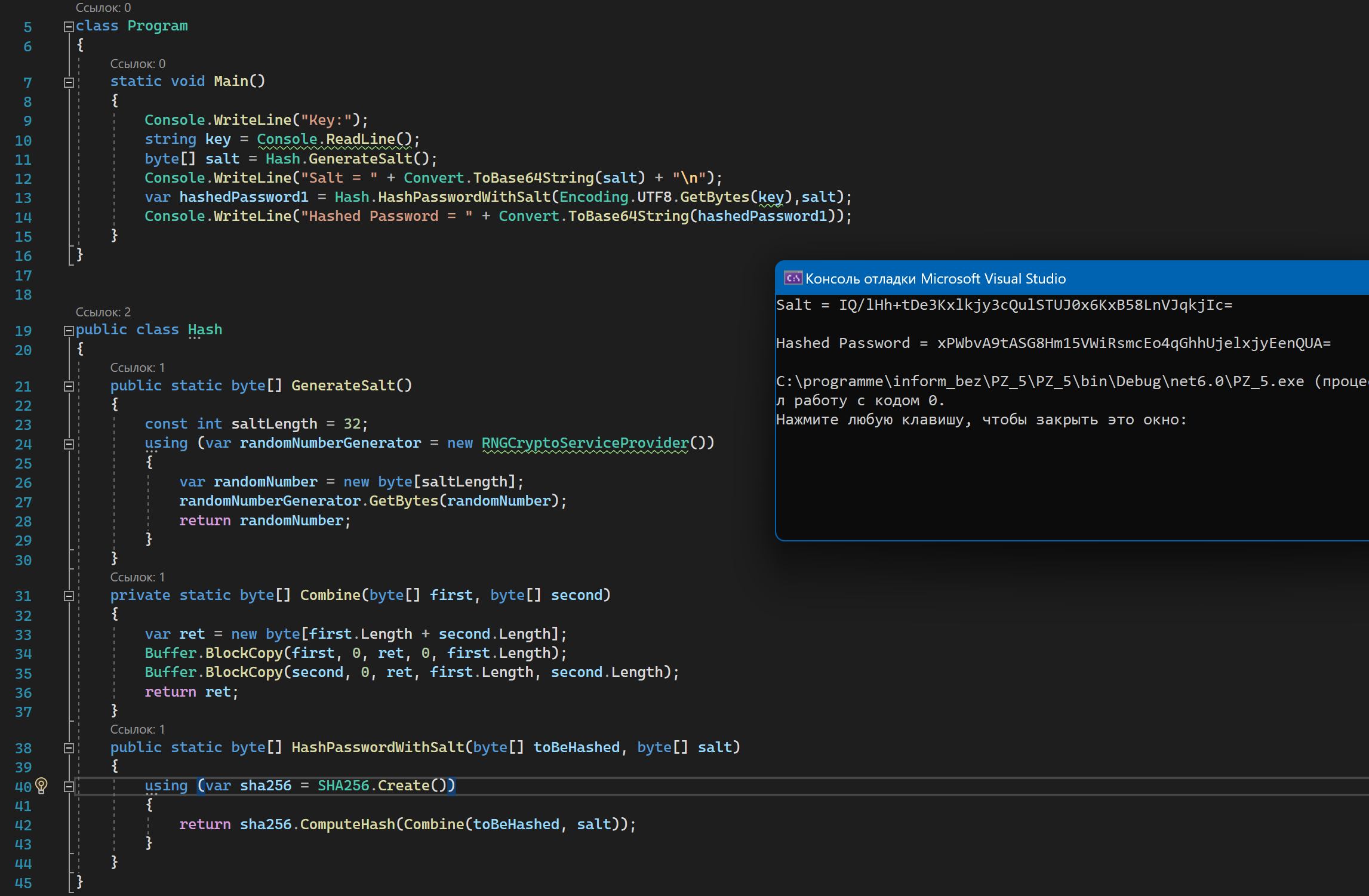
Першим етапом захищення даних є хешування. За допомогою чого наш пароль стане більш захищеним. Проте існування таких атак як брутфорс та радужні таблиці надають змогу злоумисникам з легкістю дізнатися наш “захищений пароль”

Наступним етапом нашої спроби захистити пароль буде додавання солі. Тобто генерування рандомних цифр з додаванням у кінець пароля перед хешуванням. Метод збільшую ентропію, що збільшує час на дешифрування нашого пароля зловмисника.

Останнім кроком до максимальної ентропії стане збільшення ітерації хешування. Тобто на вхід наступного хешування буде подаватися наш раніше створений хеш. Ця операція буде виконуватися більше ста тисяч разів.

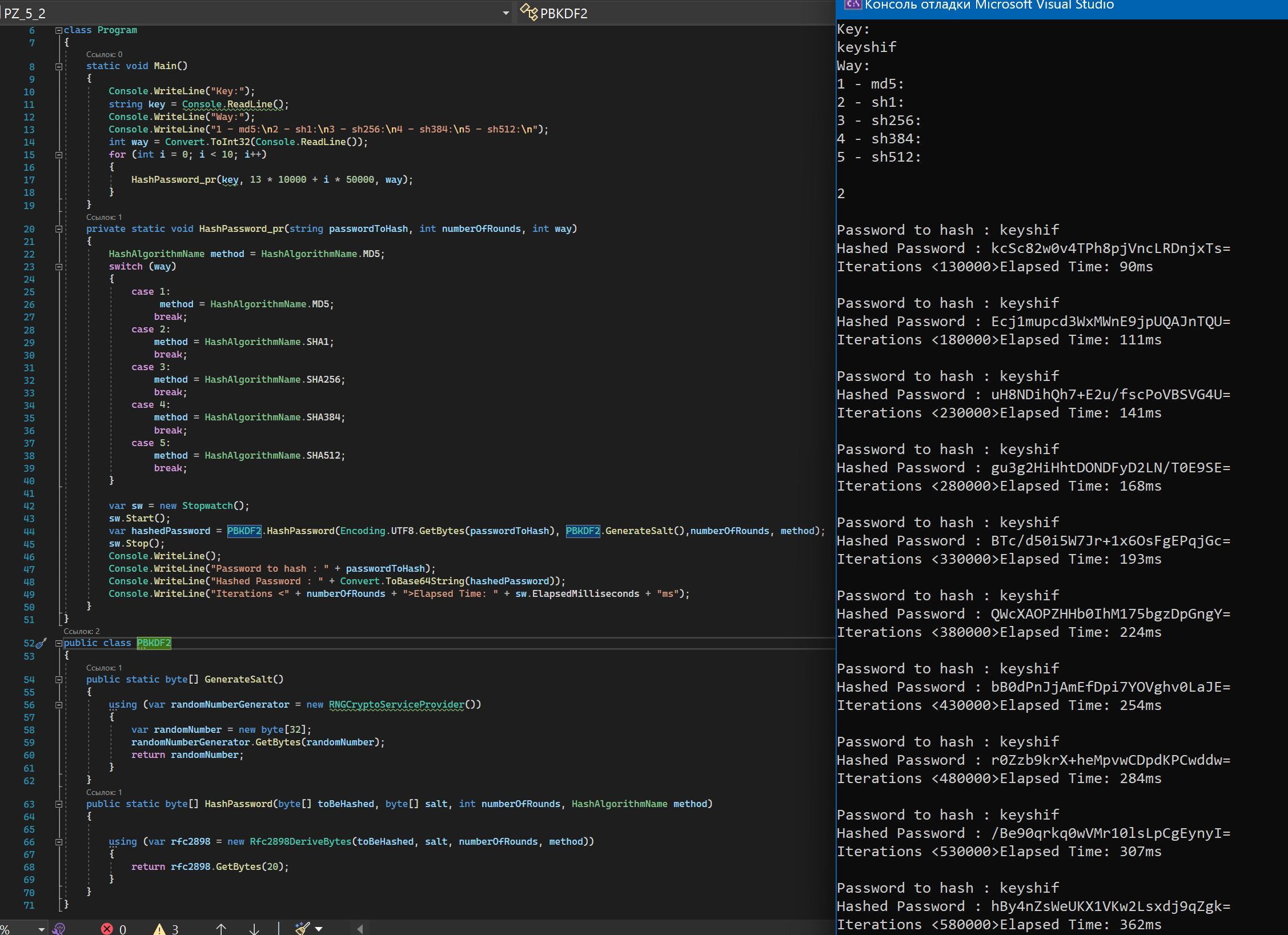
Після даних маніпуляцій наш пароль стане набагато складніше розшифрувати, що забезпечить для нас надійність цього пароля.

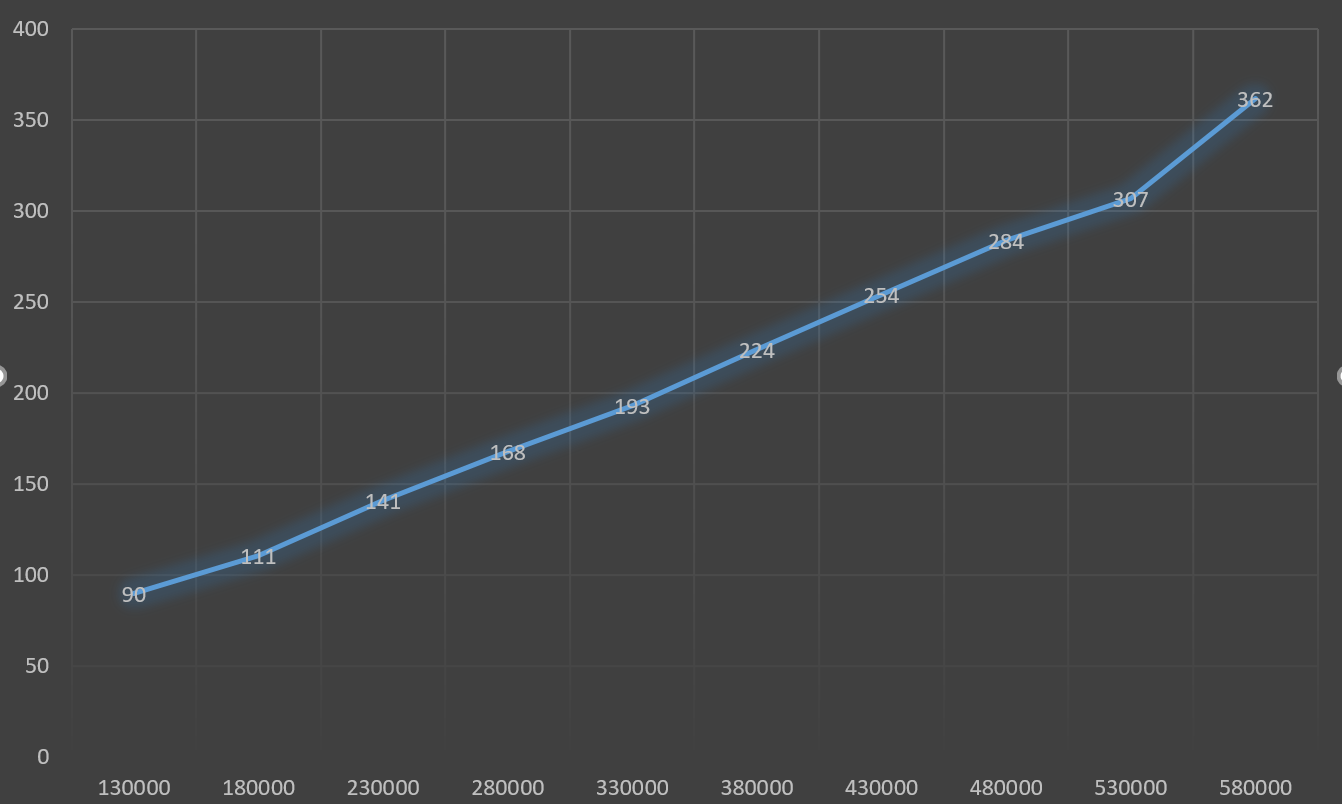
2) Розробити клас SaltedHash, що реалізує хешування паролів із додаванням додаткової ентропії. Продемонструвати роботу класу, обчислюючи хеш для заданого пароля та "солі".



У цьому коді ми генеруемо сіль за допомогою RNGCryptoServiceProvider, та використовуемо його для хешування. В самому хешуванні можемо побачити метод “Combine”, що використовуеться для об’єднання двух масивів байтів(солі та пароля). Після цих операцій в консолі можемо побачити нашу сіль та отриманий хеш.

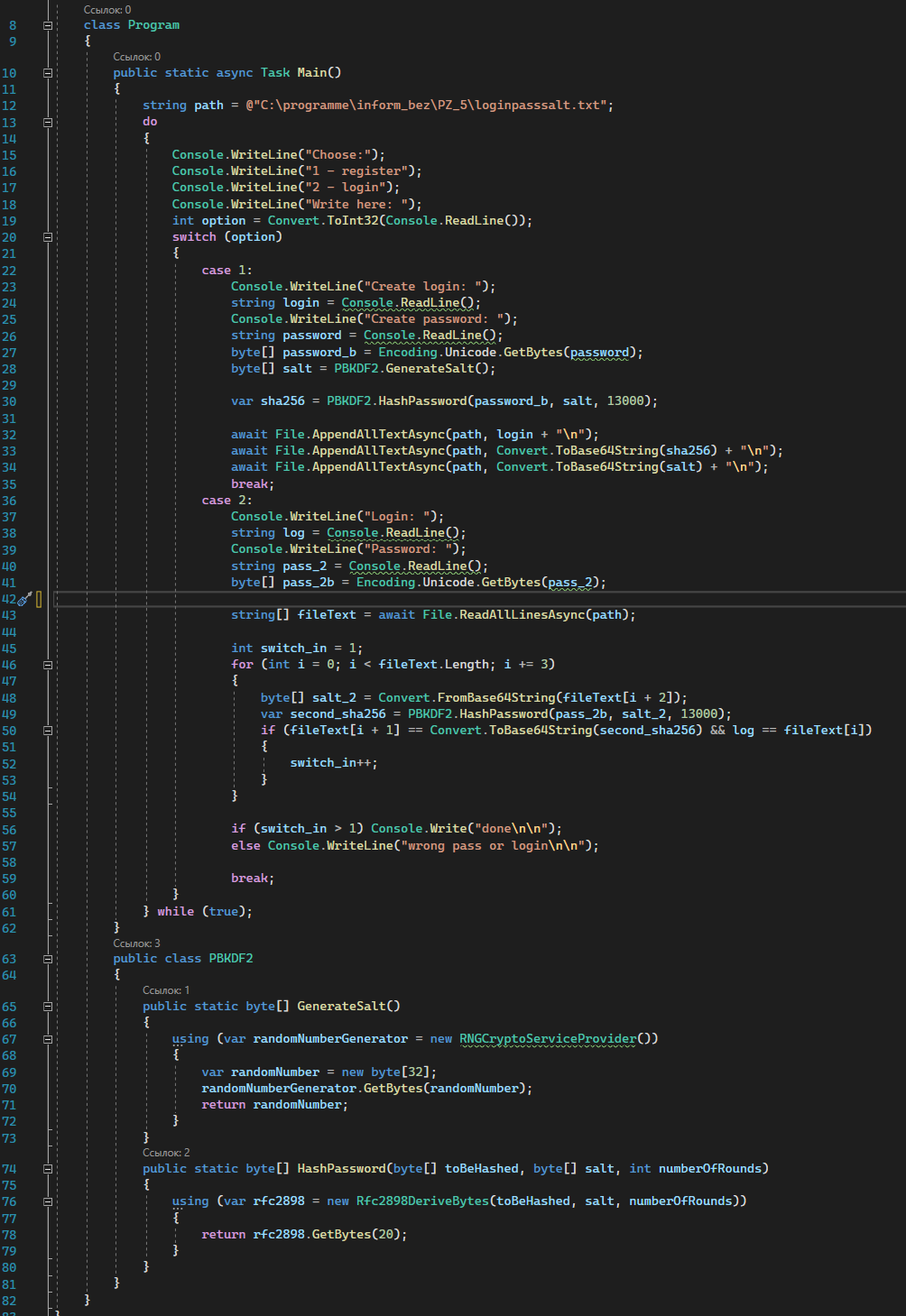
3) Розробити клас PBKDF2, що має наступну функціональність: генерує "сіль", задає алгоритм хешування (MD5, SHA1, SHA256, SHA384, SHA512) та обчислює хеш для заданого числа ітерацій. Створити програму, що обчислює час, витрачений на обчислення хешу для різного числа ітерацій (10 значень із кроком 50'000; перше значення = номер варіанта \* 10'000 ). Побудувати графік залежності витраченого часу від числа ітерацій.

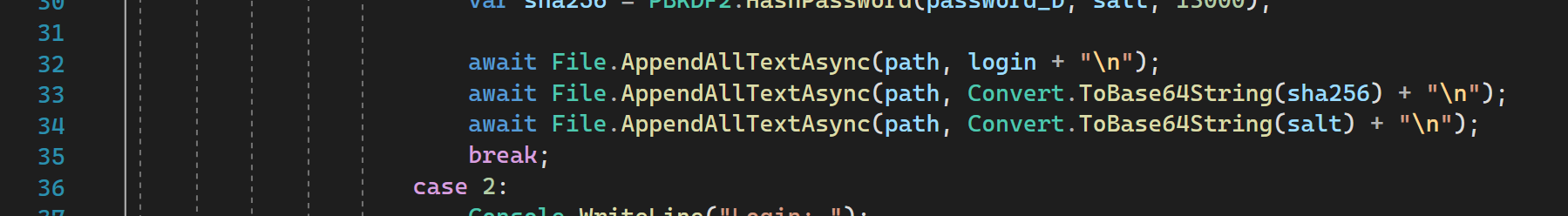


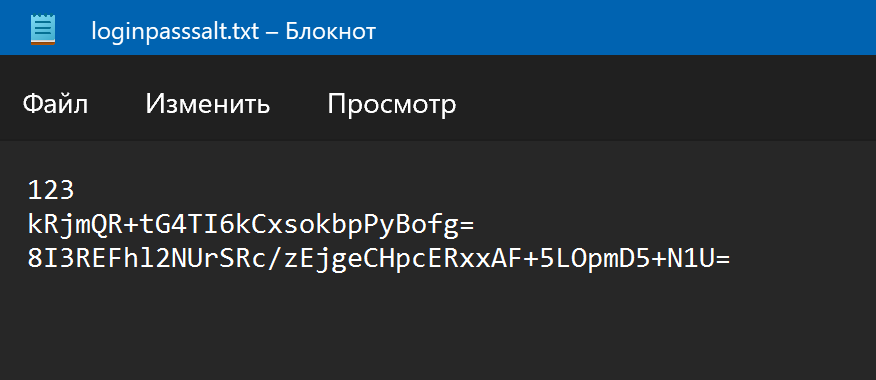


Ця реалізація солі, ітерацій та хешу схожа на минулий код, протре тут метод rfc2898 сам реалізує комбінування солі с паролем та додає можливість регулювати кількість ітерацій. Оскільки rfc2898 може змінювати У своїй програмі я реалізував конструкцію switch case для вибору назви алгоритму хешування і подальшої передачі як аргумента у rfc2898.

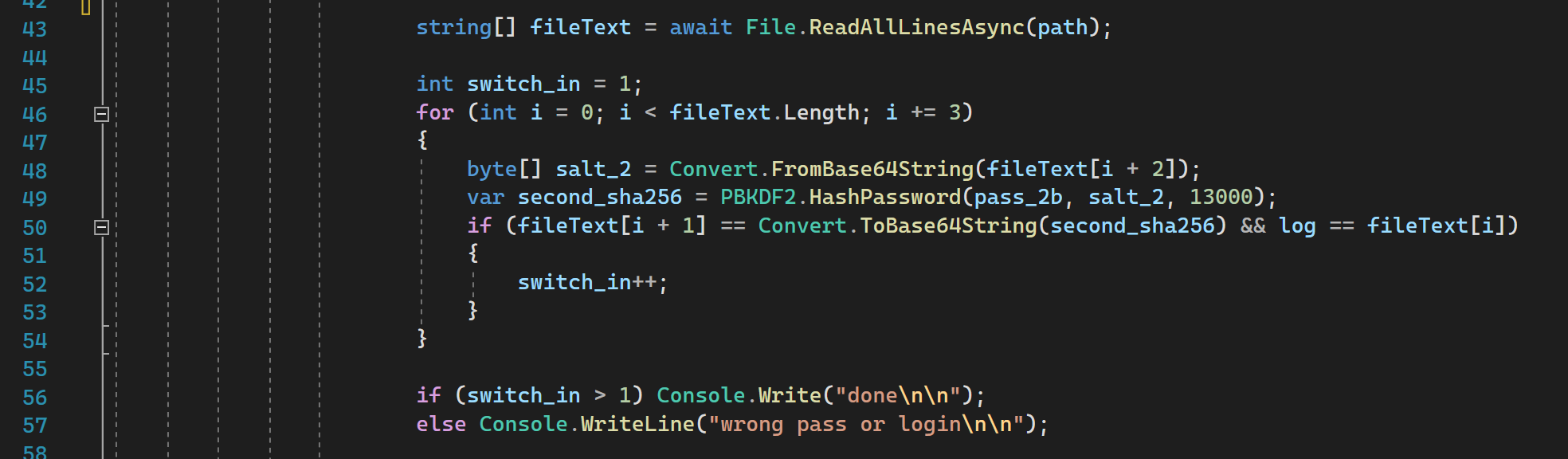
4) Написати програму, що реалізує хешування введеного пароля під час реєстрації користувача та зберігає логін, пароль та "сіль" у пам'яті. Реалізувати можливість автентифікації за логіном і паролем. Число ітерацій = номер варіанта \* 10'000.

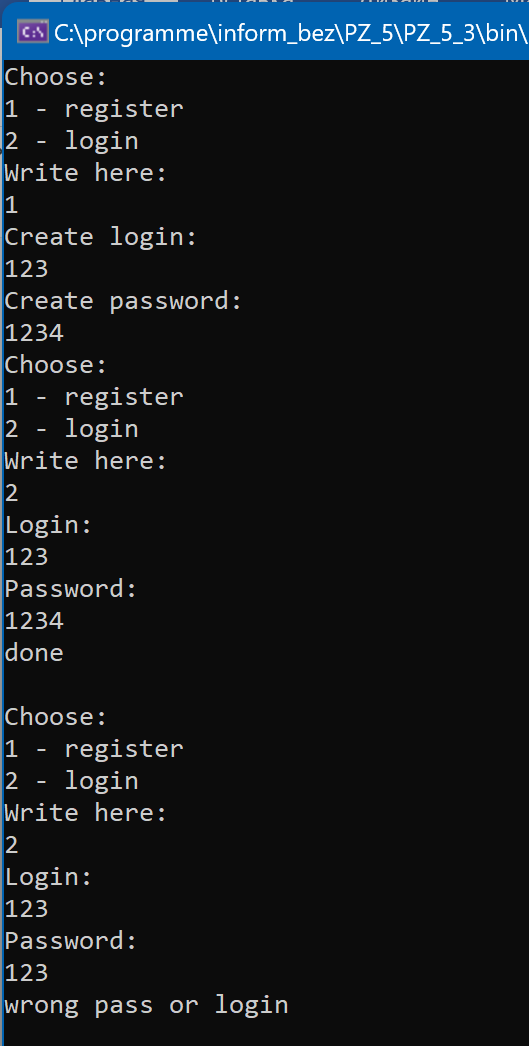


Загалом код дуже схожий на минулий, проте доповнений реалізацією входа та реєстрації за логіном та паролем. У цьому коді я вирішив використовувати роботу з текстовим файлом, бо так на мій погляд зручніше. У першому кейсі(switch case), відповідающим за логін, ми додаємо у кінець файлу три строки, а саме логин, соль, та хеш паролю. 



У другому кейсі, що стосується реєстрації ми вже зчитуємо файл по три строки та робимо кожен раз хешування з раніше вписаними у текстовий файл солями. Якщо ж хеш у текстовому файлі співпав з тільки зробленим і логіни зійшлись, то змінюємо switch\_in. По цій змінній далі будемо розуміти чи є такий логін та пароль.





**Висновок:** Отже, ми дізналися що таку солі та число ітерацій для хешування і для чого це потрібно. Навчились надійно зберігати паролі з непоганою ентропією та реалізували це у невеликій програмі, що співставляє логіни та паролі.