Міністерство освіти і науки

Київський національний університет імені Тараса Григоровича Шевченка

Кафедра Мережевих та Інтернет технологій

**Звіт**

з практичної роботи **№5**

з дисципліни: “Основи інформаційної безпеки”

**«Безпечне зберігання паролів»**

Виконав:

Студент групи МІТ-21

Горбун О. М.

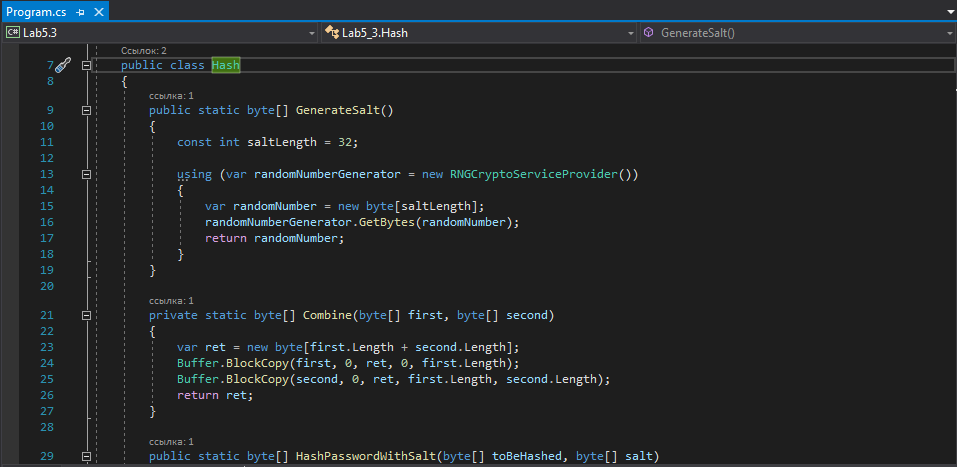
**Результати виконання завдань роботи:**

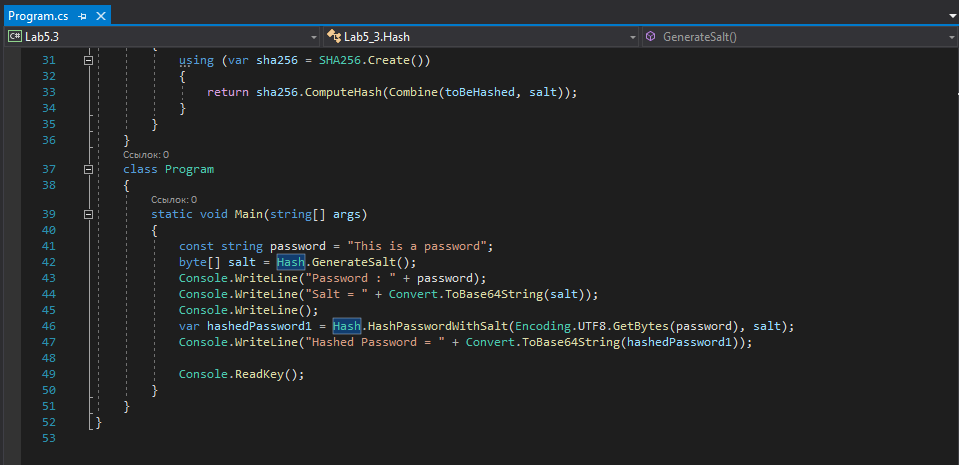
За завданнями практичної роботи було створено програми, які можна переглянути за посиланням: <https://github.com/OleksandrHorbun/Basics_Cybersecurity.git> (Посилання на відкритий репозиторій у веб-сервісі GitHub) завдання у вітці (Branch) Homework\_5

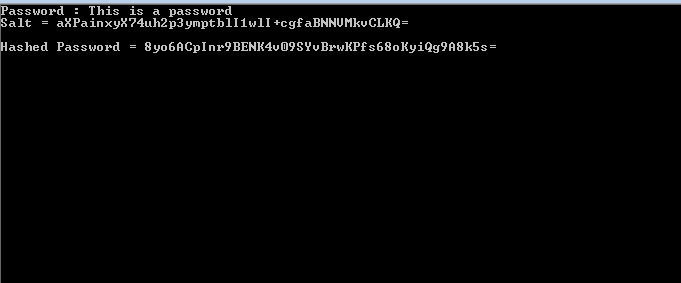
**I частина (третє завдання)**

Як можна побачити у програмі вже наявний деякий рядок, який виконує роль «пароля», для якого і буде розраховано хеш. Спочатку для цього «пароля» генерується «сіль» за допомогою функції *GenerateSalt* (генерація випадкової «солі», яка буде додана до пароля у процесі хешування) у публічному класі *Hash*. «Сіль» спочатку передається до основної програми й виводиться на екран, а після цього – разом з початковим «паролем» до функції *HashPasswordWithSalt* (знаходиться теж у вищезгаданому класі *Hash*). У цій ж функції відбувається запуск іншої функції *Combine* (додавання до «солі» до «пароля»), яка повертає скомбінований «пароль» з «сіллю» у функцію *HashPasswordWithSalt*, в якій вже і відбувається вирахування хешу й повертання його до основної програми, де відбувається вивід на екран.

Все описане вище можна побачити на знімках екрану:



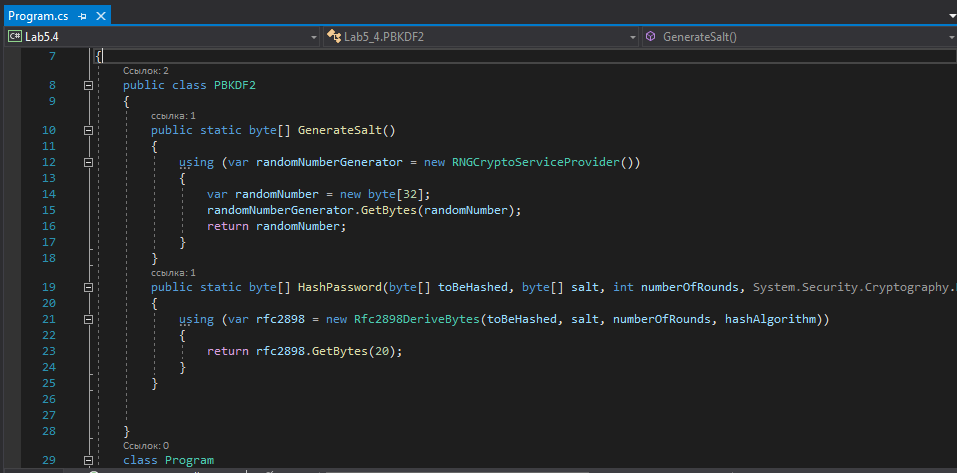


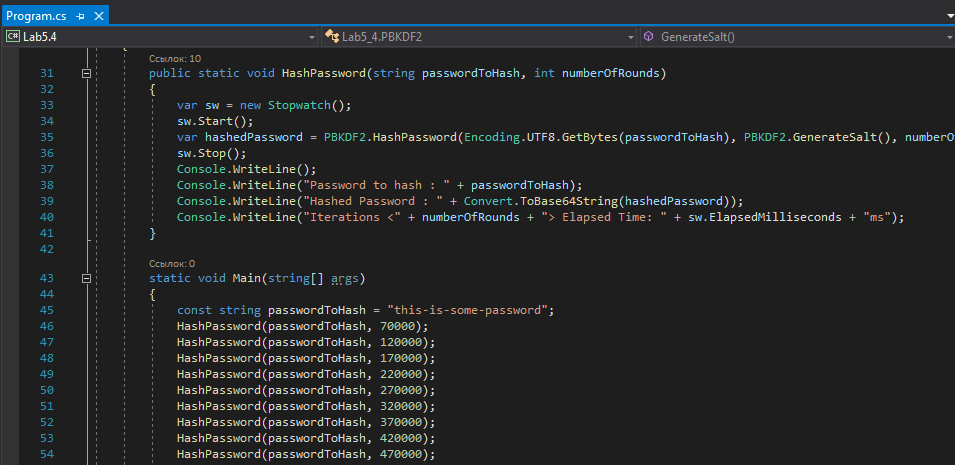


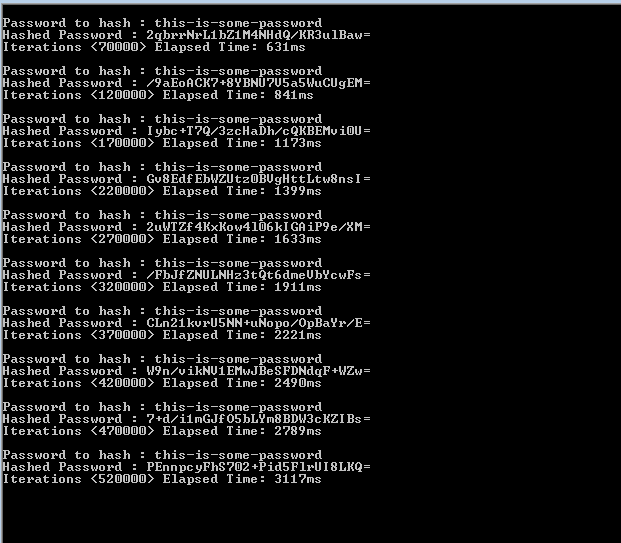
**IІ частина (четверте завдання)**

Як завжди: у програмі є деякий рядок, який треба захешувати. Спочатку «пароль» та кількість ітерацій (варіант 7 за загальним списком = 10000 \* 7 = 70000, другий крок: 70000 + 50000 = 120000 ітерацій, третій крок: 120000 + 50000 = 170000 ітерацій і так далі) передаються у функцію *HashPassword* (Передавання «пароля» і кількості ітерацій до функцій, які знаходяться в публічному класі *PBKDF2*, вивід часу виконання операцій та проміжних результатів роботи програми). Починається відлік часу й хешування за допомогою механізму, який майже ідентичний до того, що був описаний в першій частині (третьому завданні). Для хешування використовуються функції *HashPassword* і *GenerateSalt*, які знаходяться в класі *PBKDF2*. Дія функцій повторюється з певною кількістю разів. Алгоритм хешування: SHA512. Після відпрацювання певної кількості разів (наприклад: перших 70000), відлік часу завершується, отримані результати та витрачений на ітерації час виводяться на екран. Після цього процес повторюється знову ще 9 разів після чого виконання програми завершується.

Все описане вище можна побачити на знімках екрану:



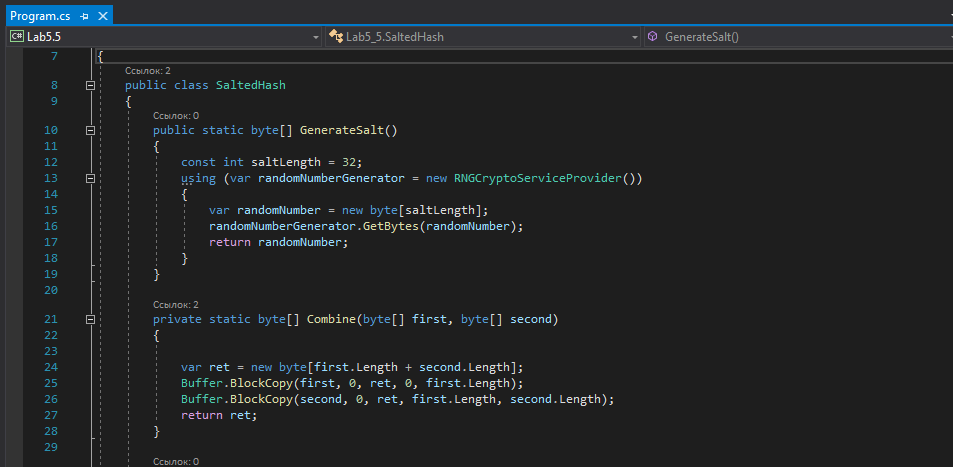


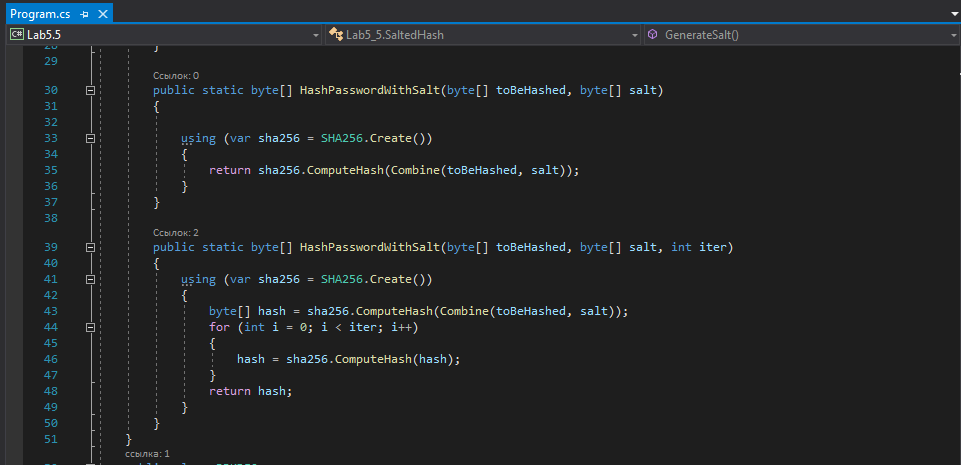


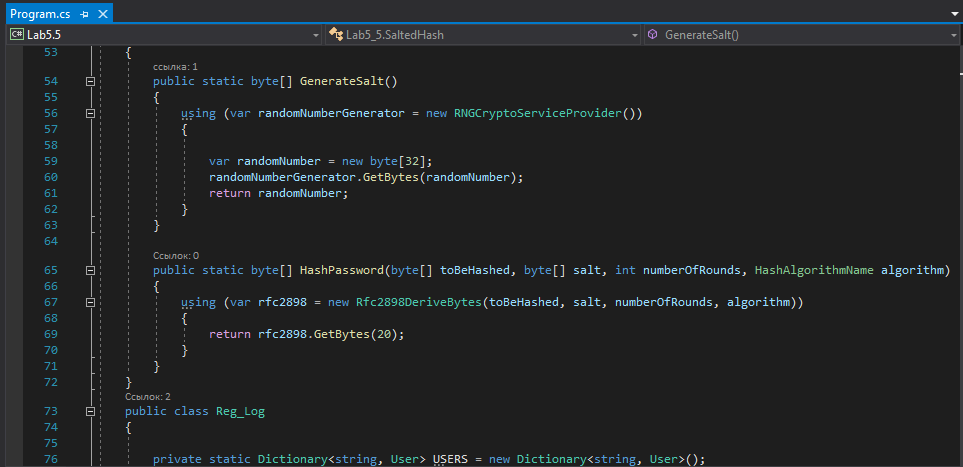
**IІІ частина (п’яте завдання)**

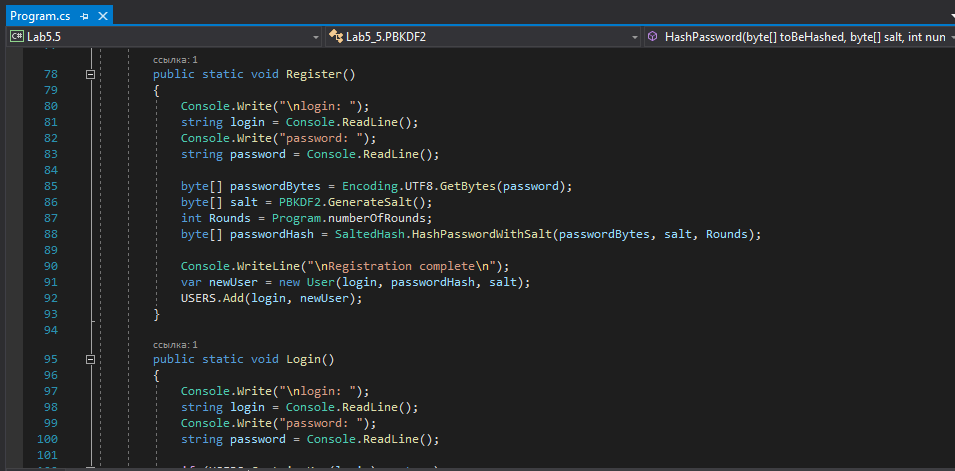
У цій частині все майже ідентично до перших двох, адже використано класи з тими ж самими функціями, але окрім того реалізовано інтерфейс логування та реєстрації, реалізовано клас *User* (клас для змінної, яка буде зберігати дані користувача: логін, пароль та «сіль»). Основні функції для логування, реєстрації та перевірку правильності хешу введеного пароля реалізовані у публічному класі *Reg\_Log*. Мова йде про функції *Register* та *Login*. Також з цього класу відбувається передача даних до класу *PBKDF2* (описаний вище у другій частині) та *SaltedHash* (Додаткова генерація «солі», комбінування з паролем та додаткове хешування певну кількість ітерацій).

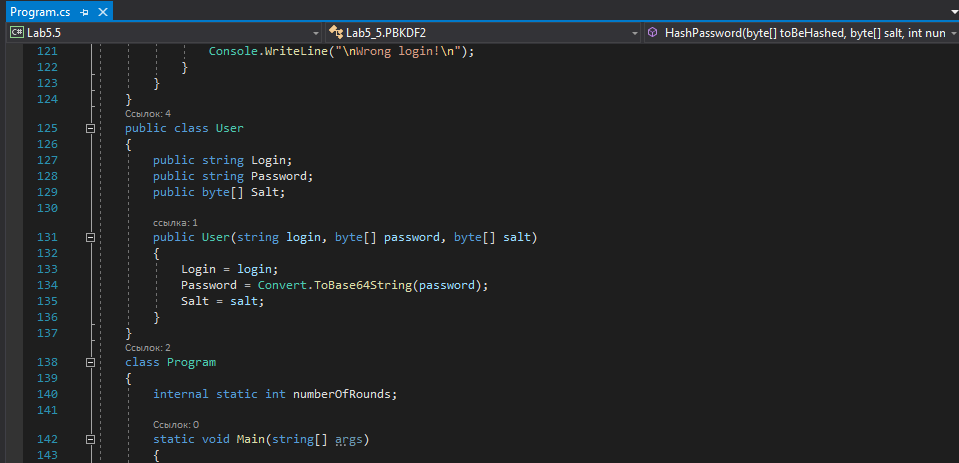
Все описане вище можна побачити на знімках екрану:

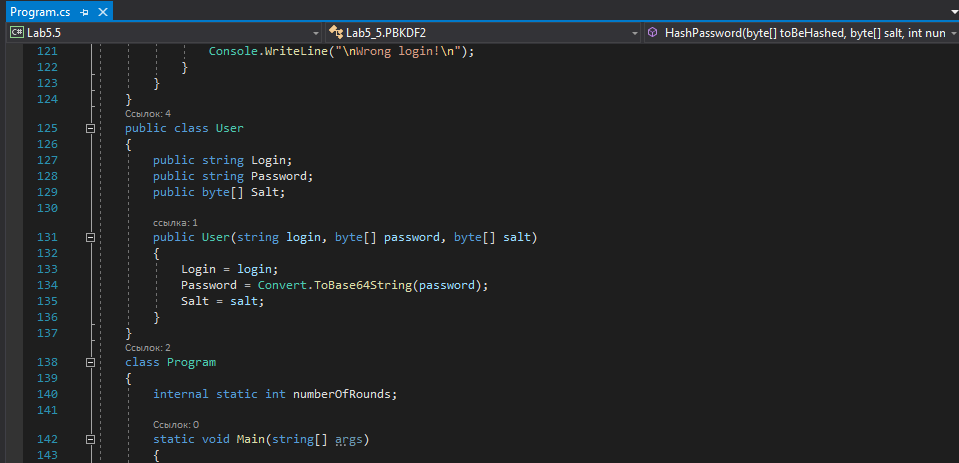


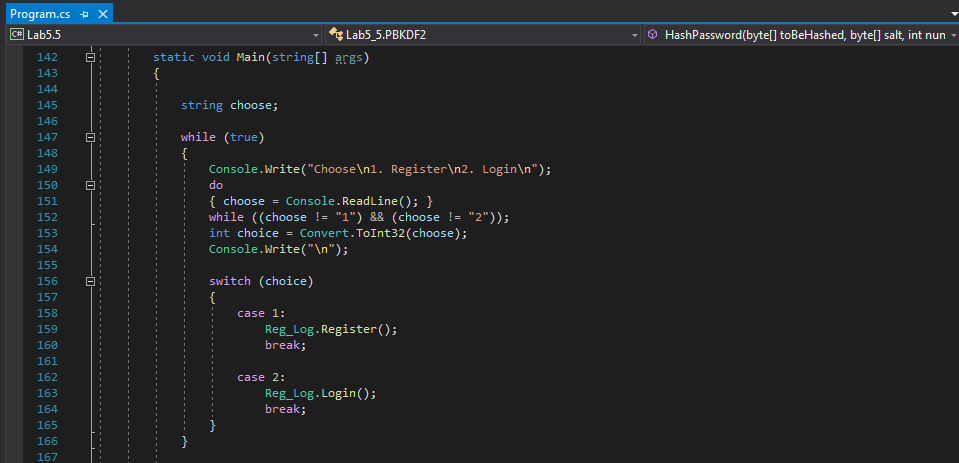


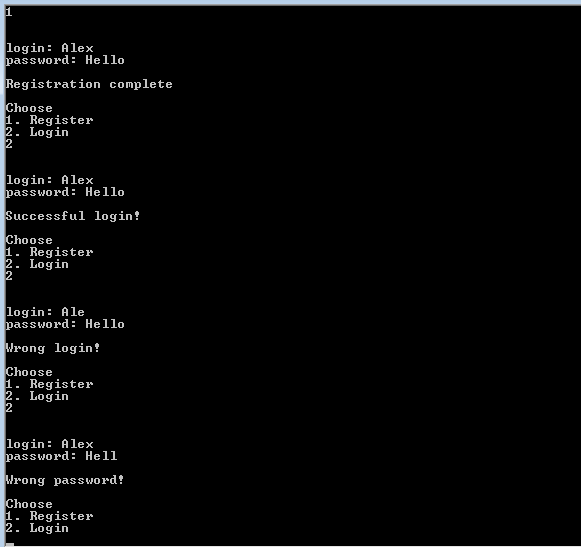












**Висновок:**

Виконуючи цю роботу я навчився безпечно зберігати паролі, розробив класи для хешування паролів з додаванням додаткової ентропії, генерування солі та обчислення хешу певну кількість ітерацій та просту програму для логування і авторизації користувачів.

.