Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра прикладної математики

**КУРСОВА РОБОТА**

з дисципліни «Програмування»

на тему:

Шифрування та дешифрування тексту

Виконав:

студент І курсу групи КМ-03,

спеціальність 113

Прикладна математика

ПЕРЕДЕРЕЙ БОГДАН.

Керівник:

ЛЮБАШЕНКО Н.Д.

Оцінка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_

Київ - 2021

ЗМІСТ

[ПОСТАНОВКА ЗАДАЧІ 4](#_Toc72259584)

[2 Вибір методу розв’язання задачі 6](#_Toc72259585)

[2.1 Процес шифрування 6](#_Toc72259586)

[2.2 Процес дешифрування 13](#_Toc72259587)

[3 Алгоритм обраного методу 16](#_Toc72259588)

[5 Результати. Контрольні приклади 23](#_Toc72259589)

[5.1 Unit-testing 23](#_Toc72259590)

[5.2 Функціональне тестування 23](#_Toc72259591)

# ВСТУП

**Мета роботи:** засвоєння теоретичних та практичних навичок при проектуванні та написані програм на мові С.

**Об’єкт дослідження:** текстовий файл, інформація з якого представляється у вигляді байтів.

**Предмет дослідження:** шифрування та дешифрування тексту за допомогою симетричного алгоритму блочного шифрування AES-128.

# ПОСТАНОВКА ЗАДАЧІ

У даній курсовій роботі необхідно розробити програму для шифрування та дешифрування тексту за допомогою симетричного алгоритму блочного шифрування AES-128 (Advanced Encryption Standart).

Програма, повинна відповідати наступним критеріям:

* Програмний код написаний на мові програмування С із дотриманням стандарту форматування коду CS50.
* Ввід тексту для шифрування та дешифрування повинен відбуватися за допомогою текстовий файлів
* Програма повинна заносити послідовність своїх операцій у журнал (log-файл). Кожен запис у журналі повинен містити час, рівень логування (FATAL, ERROR, WARN, INFO, DEBUG) та довільний текст операції. Рівень глибини логування повинен задаватись із конфігураційного \*.ini файлу програми.
* Програма повинна вміти поводитись із нестандартними ситуаціями (відсутній файл із вхідними даними, некоректні команди користувача, тощо).
* Програмний код програми повинен бути зрозумілим. Необхідно забезпечити легкість підтримки програмного коду в майбутньому.
* Для забезпечення стабільності поведінки програми у майбутньому — необхідно реалізувати покриття програмного коду Unit-тестами мінімум на 60%.
* Для забезпечення стабільності програми під навантаженням — повинне бути проведене навантажувальне тестування основного алгоритму.

Додатково у програмі повинне бути реалізоване наступне:

* Програма повинна мати можливість бути запущеною із командного рядка з метою вводу ключа шифрування та вибору дії (шифрування чи дешифрування), а також повинна бути можливість перегляду документації про саму систему шифрування та про параметри, які може обирати користувач. Перегляд цієї інформації здійснюється за допомогою команди /help.
* Основний алгоритм програми повинен бути реалізований у проміжній динамічній бібліотеці, яку можна використовувати у мові програмування Python.

# 2 Вибір методу розв’язання задачі

**Для розв’язання задачі необхідно визначити 2 ключових моментів:**

1. Алгоритм шифрування та дешифрування інформації
2. Алгоритм приведення будь-якого паролю під розмір ключа для шифрування.

Перевага для алгоритму шифрування була віддана на користь алгоритму блочного шифрування AES-128 (American Encryption Standart), оскільки це один з найшвидших алгоритмів шифрування та дешифрування інформації, який є надійним від злому. Для приведення ключа до стандартного розміру був використана крипостійка хеш-функція SHA-256 завдяки її надійності та швидкодії, і яка трансформує будь-який набір бітів в 32 байти.

Теорія обраного методу

**AES-128** – блочний шифр з довжиною блоків рівній 128 бітам та з довжиною ключа також в 128 біт. Блок і ключ представляються у вигляді матриці 4x4, в якій записані числа у шістнадцятковій системі числення від 0 до 255 (тобто один байт), і над якими будуть відбуватися певні перетворення.

## 2.1 Процес шифрування

Сам алгоритм шифрування складається з 5 операція (Substitute Bytes, Add Round Key, Shift Rows, Mix Columns та Key Schedule), які потрібно виконувати в конкретній послідовності певну кількість раундів (разів). Спочатку потрібно розібрати кожну з цих операцій і потім представити загальний алгоритм шифрування одного блоку.

**Операція Substitute Byte**s представляє собою заміну кожного елемент блоку на певний елемент із таблиці замін (S-box). Така операція забезпечує не лінійність шифрування [1]. Сам S-box представляє собою таблицю із 255 заздалегідь занесених значень. Ці значення є константами для цього алгоритму шифрування, які в більшості випадків зберігаються у програмі у вигляді одновимірного масиву значень. Таким чином ця операція замінює кожне значення блоку на елемент з S-box з індексом, який дорівнює значенню, яке заміняється.

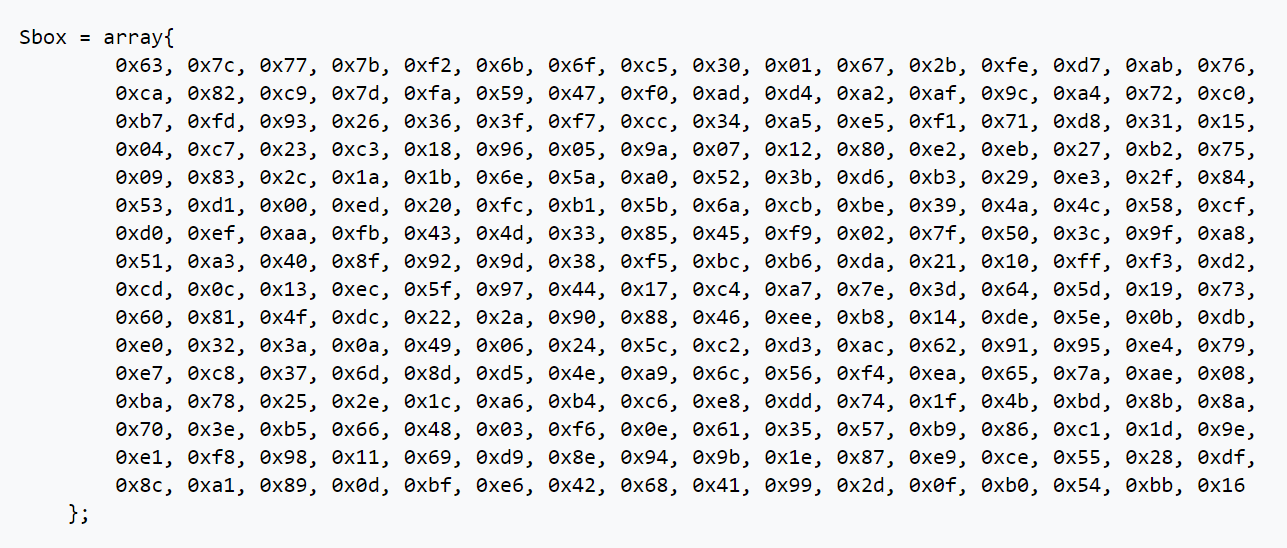
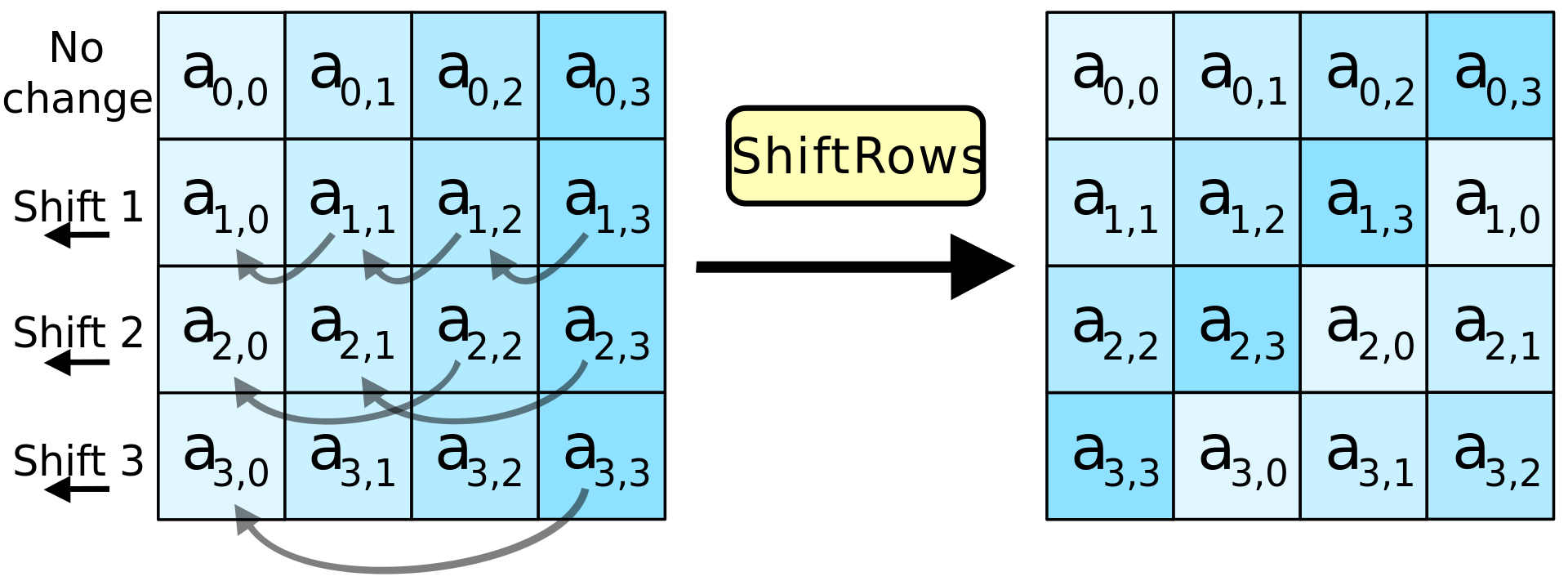


Рисунок 2.1 – S-box для операції Substitute Bytes

**Операція Add Round Key** представляє собою заміну кожного значення з блоку, що шифрується, на результат від побітового додавання цього значення і відповідного за індексами значення з ключа за модулем двійки. Тобто ця операція потребує взяти значення із блоку, який шифрується, взяти відповідне до значення із блоку за індексами значення з ключа, та побітово додати ці значення за mod(2), а результат записати на місце значення із блоку, що шифрується.

**Операція Shift Rows** – це циклічний здвиг вліво елементів блоку в рядках. Перший ряд залишається без змін, у другому ряду відбувається здвиг на одиницю, в третьому на двійку, а в четвертому на трійку. Приклад операції Shift Rows:

 Рисунок 2.2 – Операція Shift Rows

**Операція Mix Columns** представляє собою перемноження двох матриць – спеціальної циклової матриці, яка є константою для алгоритму шифрування AES-128 (рисунок 2.3), та блоку, який шифрується. Але операцію додавання потрібно замінити на побітове додавання значень за mod(2), а замість звичайного множення використовувати спеціальну операцію множення. Числа, які потрібно перемножити цією операцією, легше сприймати як многочлени. Наприклад, потрібно перемножити два числа в шістнадцятковій системі числення – 2 і 95. 95 у двійковій системі представляє собою число 10010101, а 2 – 00000010. Перепишемо ці числа у вигляді многочленів, розряди цифр який відповідають значенню в певній степені (для першого розряду – , а для останнього ). Таким чином отримуємо для 2 – , а для 95 – . Після цього ці два многочлени потрібно перемножити:

.

Було отримано новий многочлен і, так як немає іще одного біта, щоб записати , його потрібно розписати як (спеціальна константа). Після цього потрібно скоротити елементи які повторюються:

Таким чином було отримано многочлен , який при переведені в двійкову систему числення дорівнює 00110001, що представляє собою число 31 у шістнадцятковій системі числення. Після проведення множення матриць результат повинен використовуватися для майбутніх операцій, тобто блок потрібно переписати на результат множення і продовжити шифрування.

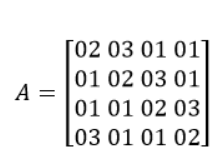


Рисунок 2.3 – Циклова матриця для операції Mix Columns

**Операція Key Schedule** – це спеціальна операція зміни ключа, яка трансформує ключ під час кожного нового раунди, щоб потім виконати операцію Add Round Key вже зі зміненим ключем. Для її виконання потрібно взяти старий ключ та взяти з нього останній стовпець. Останній стовпець потрібно циклічно здвинути вгору один раз. Наприклад:

Після цього необхідно виконати над цим стовпцем операцію Substitute Bytes. Після цього потрібно взяти перший стовпець із ключа, який трансформується, та побітово додати за mod(2) відповідні елементи із стовпця, який отримався після операції Subtitute Bytes. Але для першого елемента в стовпчику після цього потрібно іще виконати побітове додавання за mod(2) з елементом із матриці Rcon, яку можна представляти як вектор, оскільки всі значення, крім першого рядка, є нульовими. Цей вектор також є сталим для процесу шифрування і складається з 10 констант, які додаються згідно того, який раунд відбувається, тобто якщо зараз відбувається 5 раунд шифрування, то 5 значення із вектора Rcon буде взято для додавання. Після цих операція було отримано перший стовпчик для нового ключа. Другий стовпчик нового ключа формується з побітового додавання першого стовпчика з нового ключа до другого стовпчика з старого ключа. Третій стовпчик нового ключа формується з додавання другого стовпчика нового ключа до третього стовпчика третього ключа, а четвертий стовпчик нового ключа – додаванням третього стовпчика нового до четвертого стовпчика старого ключа. Таким чином після цієї процедури повинно бути отримано новий ключ для майбутніх операцій шифрування.

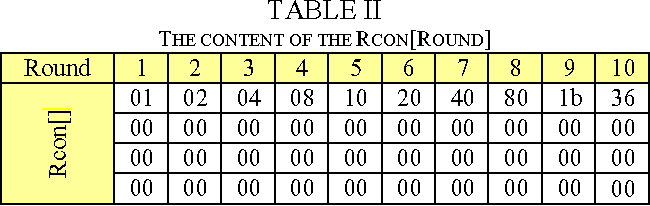


Рисунок 2.4 – Матриця Rcon

Таким чином весь процес шифрування полягає у виконанні п’яток вищезазначених операцій в правильному порядку, а саме: Add Round Key, повтор дев’ять раз набору наступний дій: Substitute Bytes, Shift Rows, Mix Columns, Key Shedule з відповідним значенням з Rcon залежно від номеру повтору, Add Round Key, - і виконання фінального раунди: Substitute Bytes, Shift Rows, Key Shedule з останнім елементом з Rcon, Add Round Key. Порядок виконання операція зазначено у вигляді блок-схеми (рисунок 2.5).

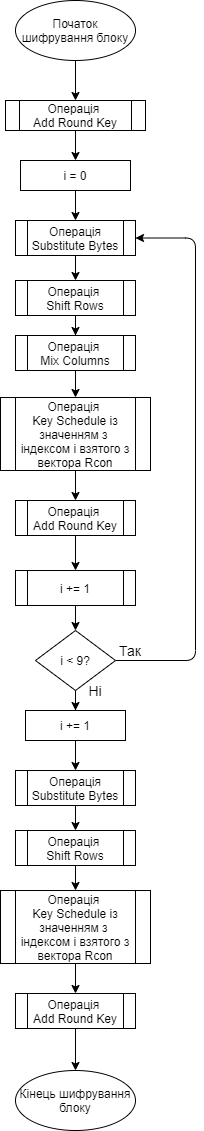


Рисунок 2.5 – Процес шифрування блоку

## 2.2 Процес дешифрування

Режим дешифрування також складається з 5 операцій – 3 операцій (Inverse Substitute Bytes, Inverse Shift Rows, Inverse Mix Columns), які є оберненими до операцій шифрування, та 2 операцій, який залишаються без змін (Add Round Key, Key Schedule). Тобто, для дешифрування інформації потрібно виконати процес шифрування у зворотному порядку із зворотними функціями. Спочатку потрібно розібрати зворотні функції, щоб потім записати загальний процес дешифрування.

**Операція Inverse Substitute Byte**s – ідентична до операції Substitute Bytes, але замість S-box в ній використовується Inverse S-box (рисунок 2.6), який також можна представляти як масив значень, який складається з констант, які використовуються для дешифрування будь-якого блоку.

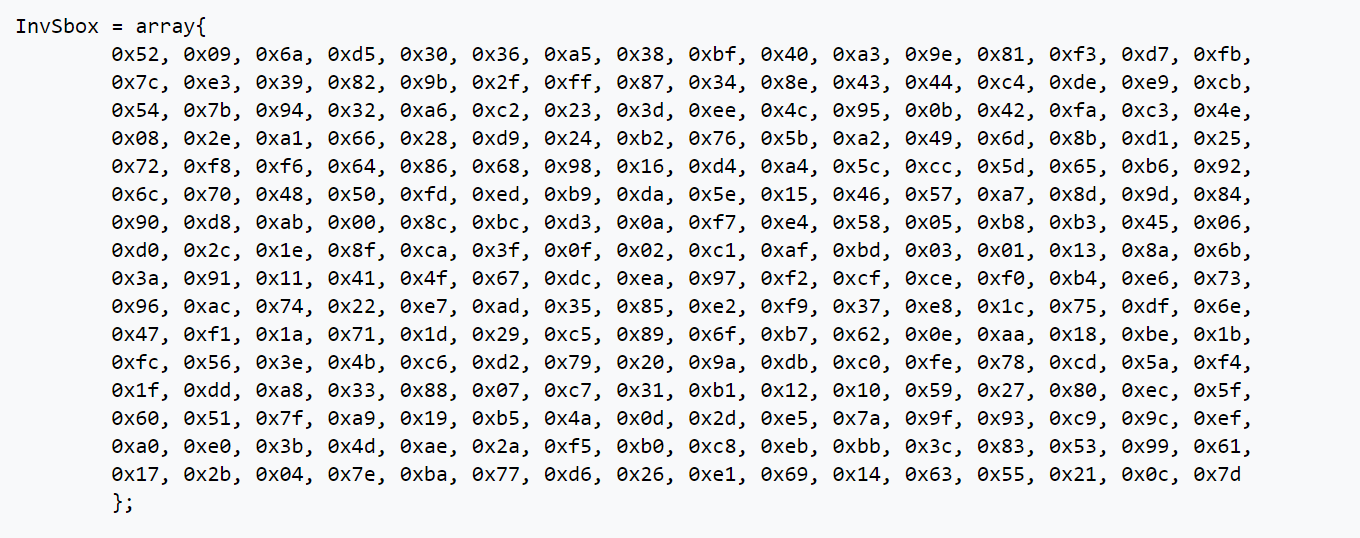


Рисунок 2.6 – Inverse S-box для операції Inverse Substitute Bytes

**Операція Inverse Shift Rows** – обернена до операції Shift Rows. Це означає, що кількість циклічних здвигів для рядків залишається тією самою, але здвиги відбувається не вліво, а в право. Приклад:

**Операція Inverse Mix Columns** – обернена до операції Mix Columns, різниця полягає лише у тому, що множення відбувається не просто на циклічну матрицю, а на спеціально обернену до неї матрицю (рисунок 2.7).

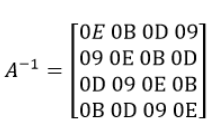


Рисунок 2.7 – Обернена циклічна матриця для операції Inverse Mix Columns

Ще потрібно зазначити, що при процесі дешифрування Key Schedule ключ не можна переписувати, ми використовуємо ключі у зворотному порядку, тому при дешифруванні потрібно зробити копію ключа, виконати над ним певну кількість операцій Key Schedule, і потім використати у одному із раундів. У наступному раунді дешифрування для Key Schedule потрібно буде зробити те саме, але вже на одну операцію Key Schedule менше.

Після цього можна записати загальний процес дешифрування блоку, який собою представляє наступні кроки: виконання Key Schedule 10 разів, щоб отримати ключ для 10 ітерації, Add Round Key з ключем після Key Schedule, Inverse Shift Rows, Inverse Substitute Bytes, потім повтор 9 раз наступного порядку дій: Key Schedule кількість раз, яка дорівнює номеру ітерації, Add Round Key з ключем після Key Schedule, Inverse Mix Columns, Inverse Shift Rows, Inverse Substitute Bytes, - і в кінці останньою операцією є Add Round Key блоку, який дешифрується, з початковим ключем. Загальний процес представлений у блок-схемі (рисунок 2.8).

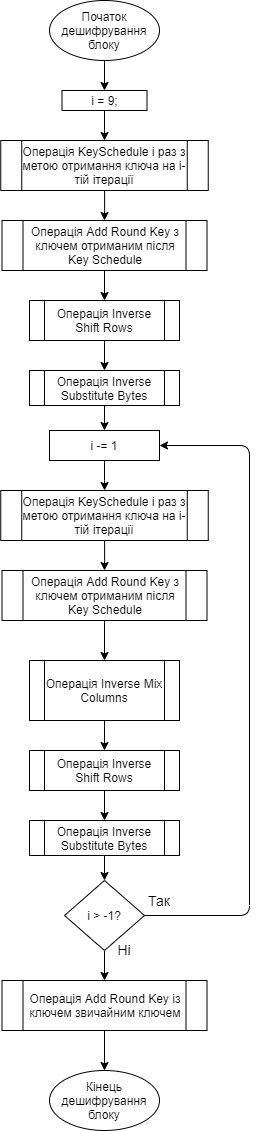


Рисунок 2.8 – Процес дешифрування блоку

# 3 Алгоритм обраного методу

У цьому розділі будуть наведені блок-схеми для функцій, які потрібні для роботи програми.

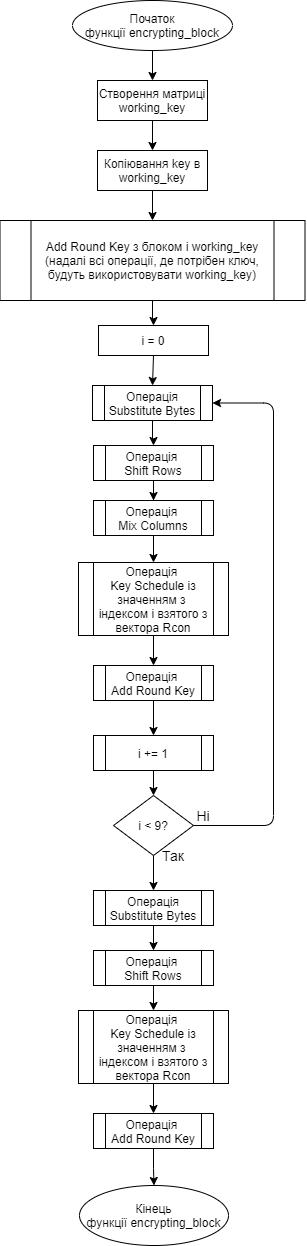


Рисунок 3.1 – Блок-схема функції encrypting\_block

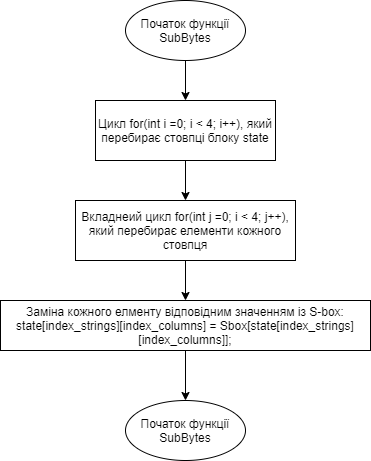


Рисунок 3.2 – Блок-схема функції SubBytes

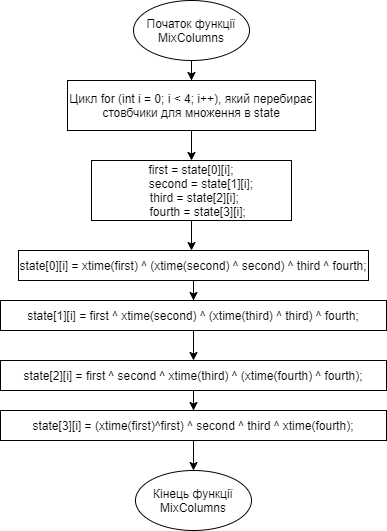


Рисунок 3.3 – Блок-схема функції MixColumns



Рисунок 3.4 – Блок-схема функції ShiftRows

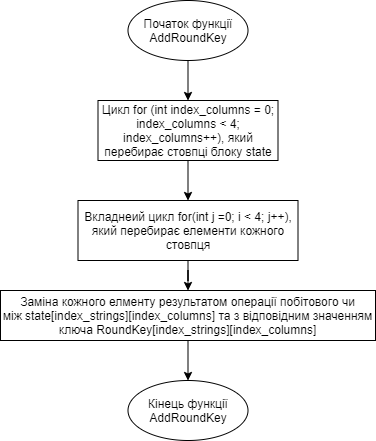


Рисунок 3.5 – Блок-схема функції AddRoundKey

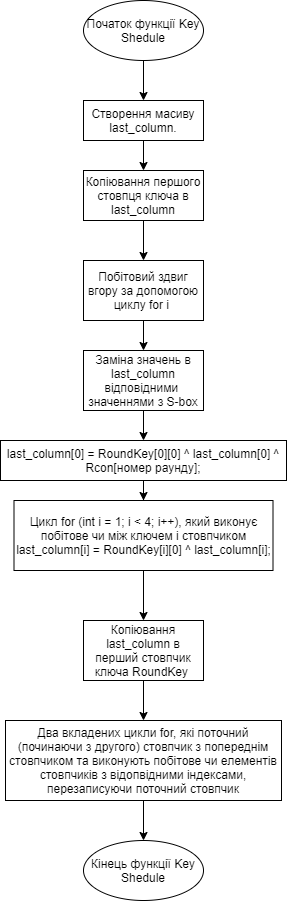


Рисунок 3.6 – Блок-схема функції KeySchedule

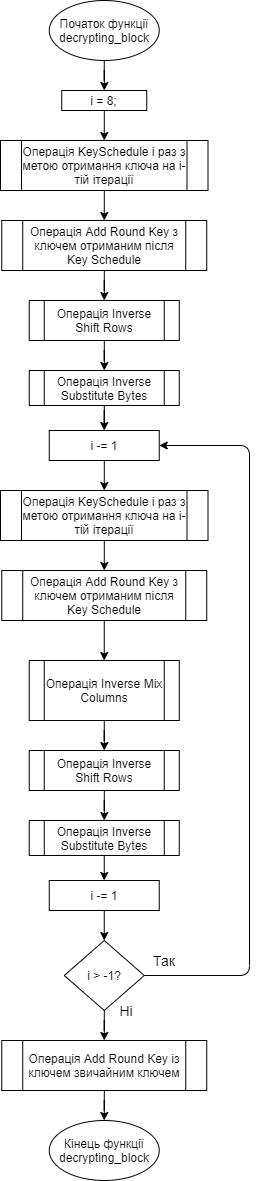


Рисунок 3.6 – Блок-схема функції decrypting\_block

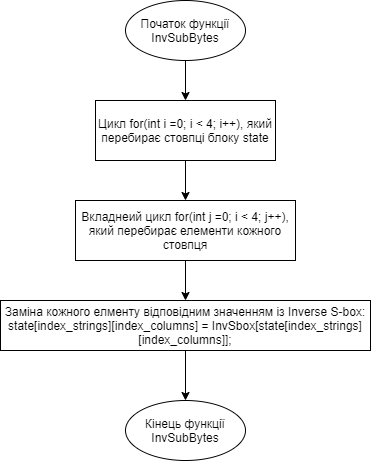


Рисунок 3.7 – Блок-схема функції InvSubBytes

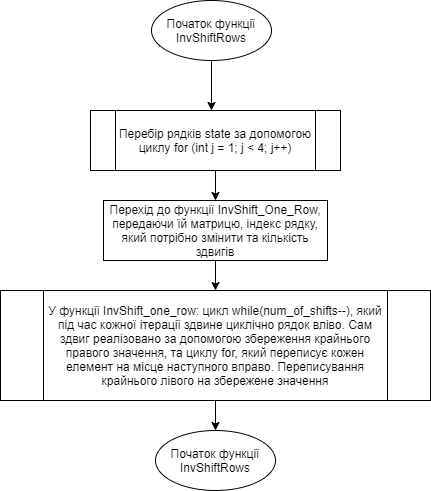


Рисунок 3.8 – Блок-схема функції InvShiftRows



Рисунок 3.9– Блок-схема функції InvMixColumns

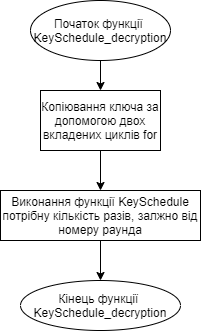


Рисунок 3.10 – Блок-схема функції KeySchedule\_decryption

# 5 Результати. Контрольні приклади

## 5.1 Unit-testing

В якості контрольних прикладів було використано unit-testing, під час якого тестується один із прикладів знайдених в мережі. Крім цього, відбувається перевірка правильності роботи алгоритму SHA-256. Результати unit-testing знаходяться на рисунку 5.1

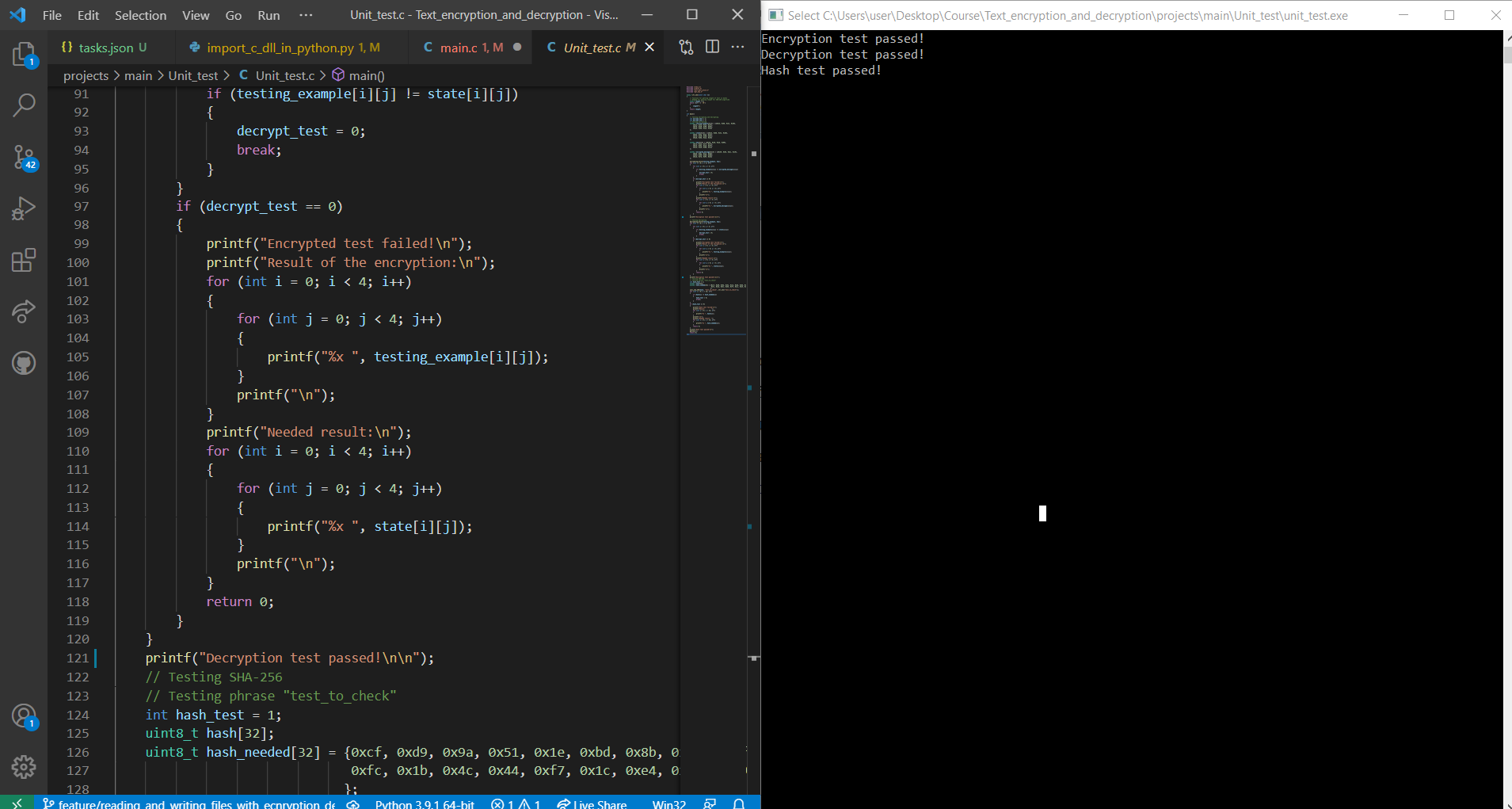


Рисунок 5.1 – Результати unit-testing

## 5.2 Функціональне тестування

Після успішного проходження unit-testing, потрібно виконати тестування функціональності. Воно буде виконано на файлі functionality\_testing розміром з 10 мегабайт, для перевірки того, чи програма працює справно. Результати шифрування знаходяться на рисунках 5.2 і 5.3. Результат дешифрування знаходиться в файлі result\_of\_encr.txt.

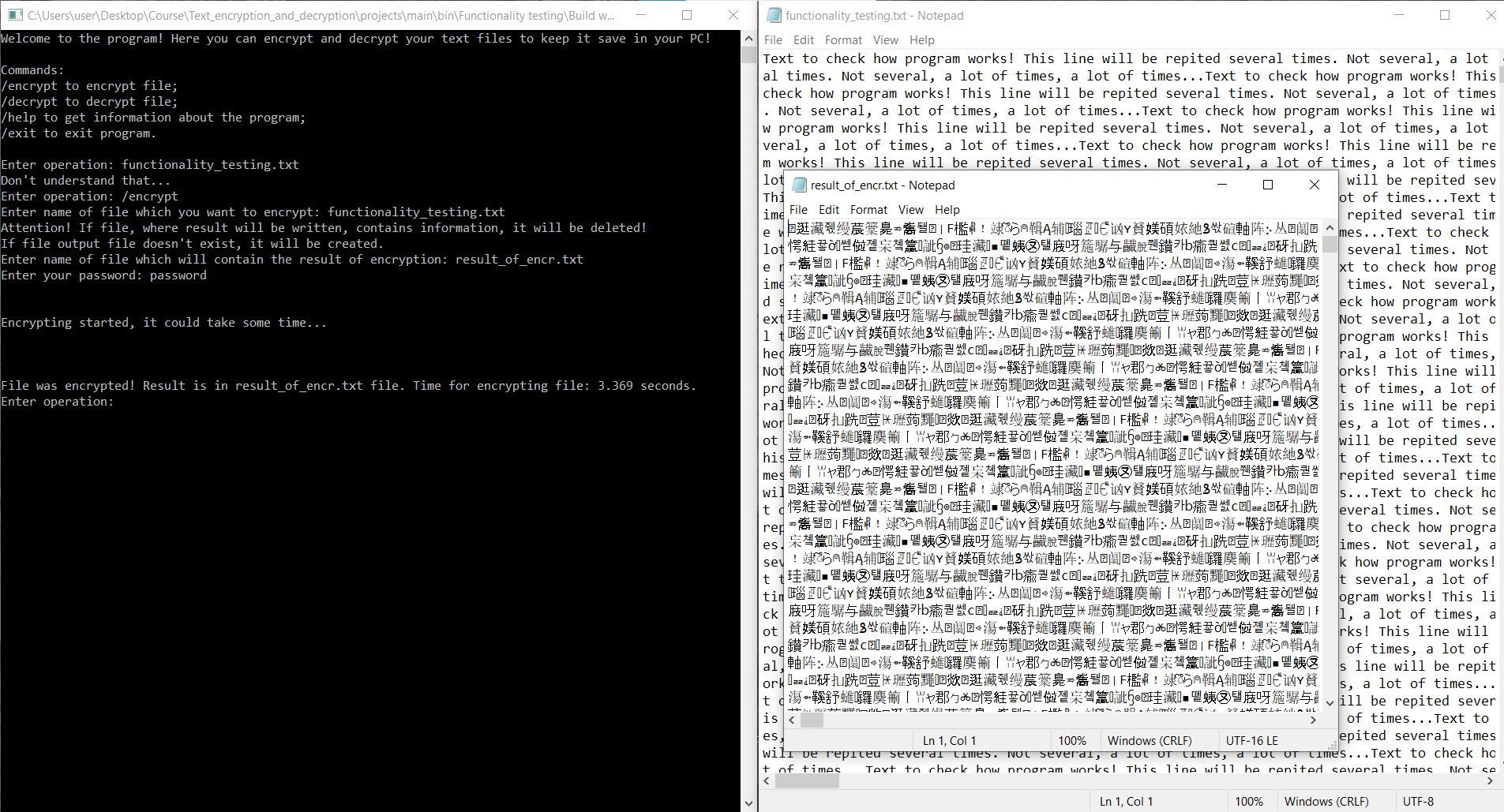


Рисунок 5.2 - Результат шифрування файлу functionality\_text.txt

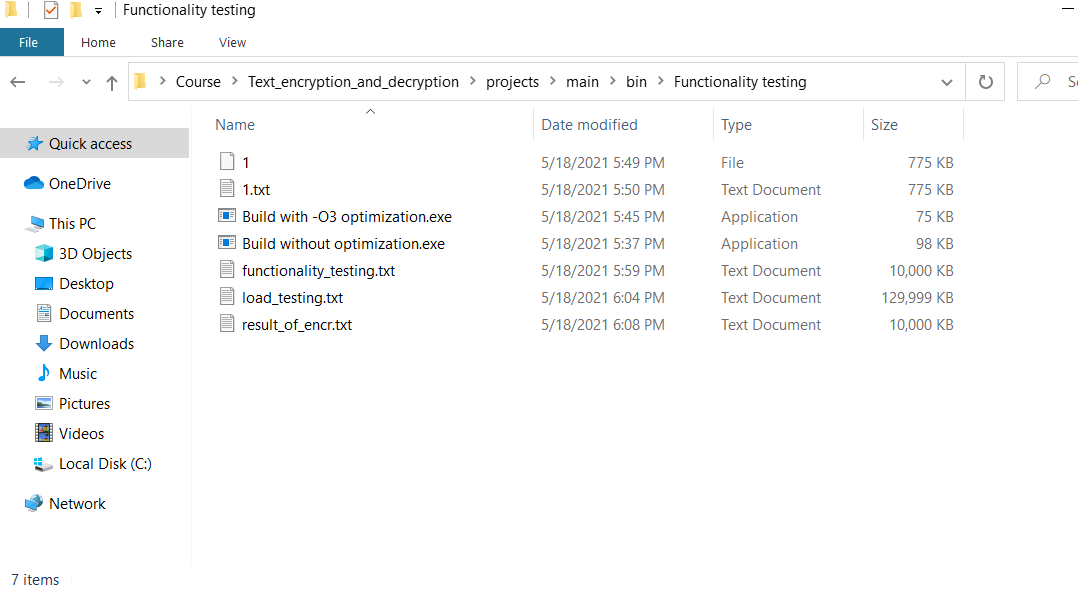


Рисунок 5.3 – Вміст папки після шифрування

Для перевірки дешифрування потрібно дешифрувати файл result\_of\_encr.txt і подивитися на результат. Результат дешифрування знаходиться в файлі result\_of\_decr.txt (рисунок 5.4 і 5.5).

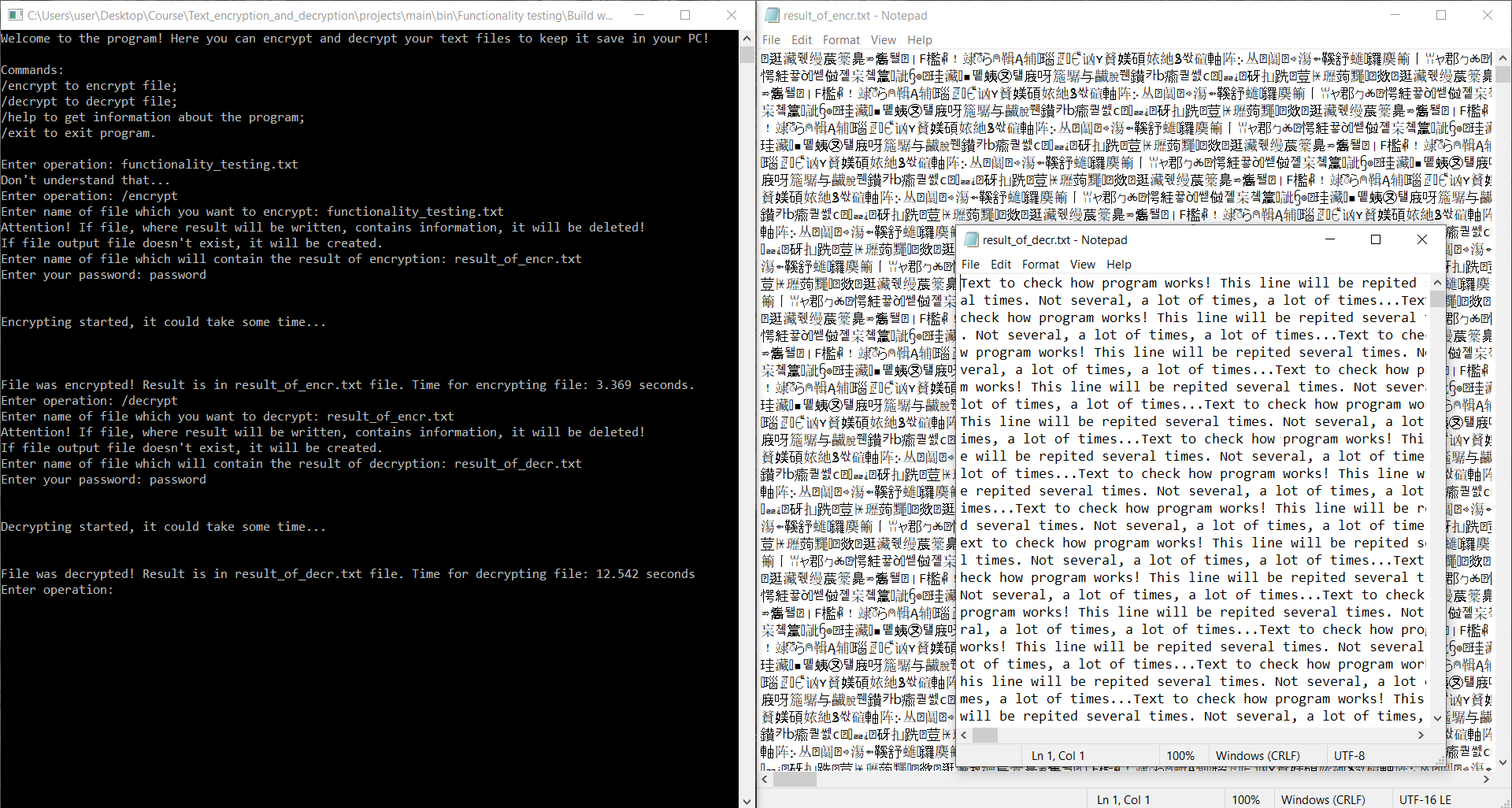


Рисунок 5.3 - Результат дешифрування файлу result\_of\_encr.txt

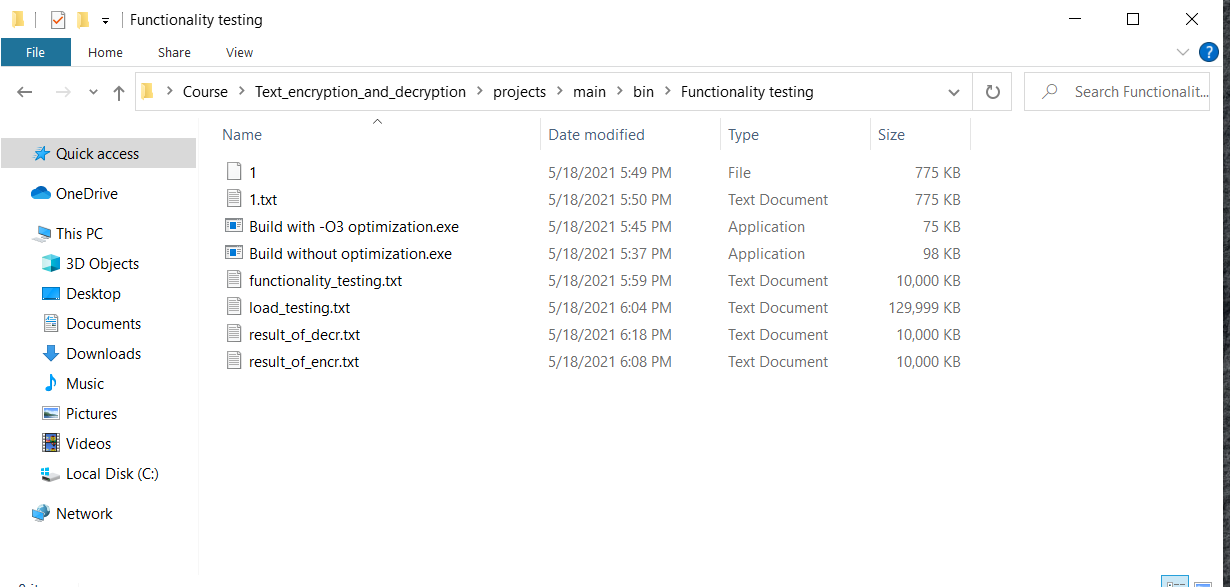


Рисунок 5.4 – Вміст папки після дешифрування

Як видно на рисунках 5.1-5.4, програма працює справно і виконує поставлені задачі шифрування та дешифрування тексту.

## 5.3 Навантажувальне тестування. Витоки пам’яті. Оптимізація

Спочатку було перевірено роботу програми на витоки пам’яті за допомогою спеціальної програми «Valgrind», під час тесту було декілька раз зашифровано та розшифровано різні файли невеликого розміру з метою перевірити програму на витоки. Вірогідність витоків є невеликою, оскільки динамічна пам’ять використовується лише для вводу імен файлів і паролів. Результати знаходяться на рисунках 5.5 і 5.6.

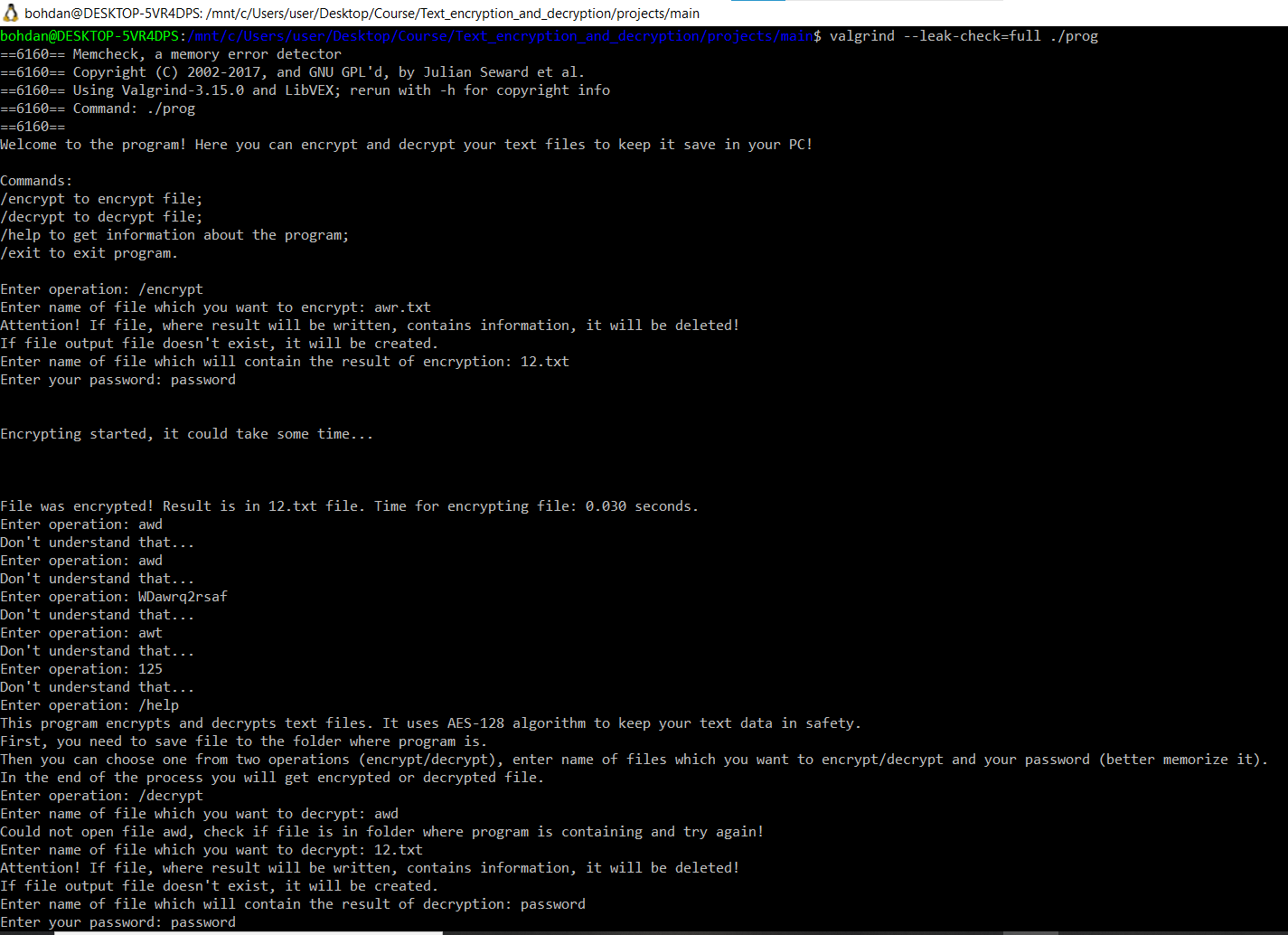


Рисунок 5.5 – Перевірка на витоки пам’яті

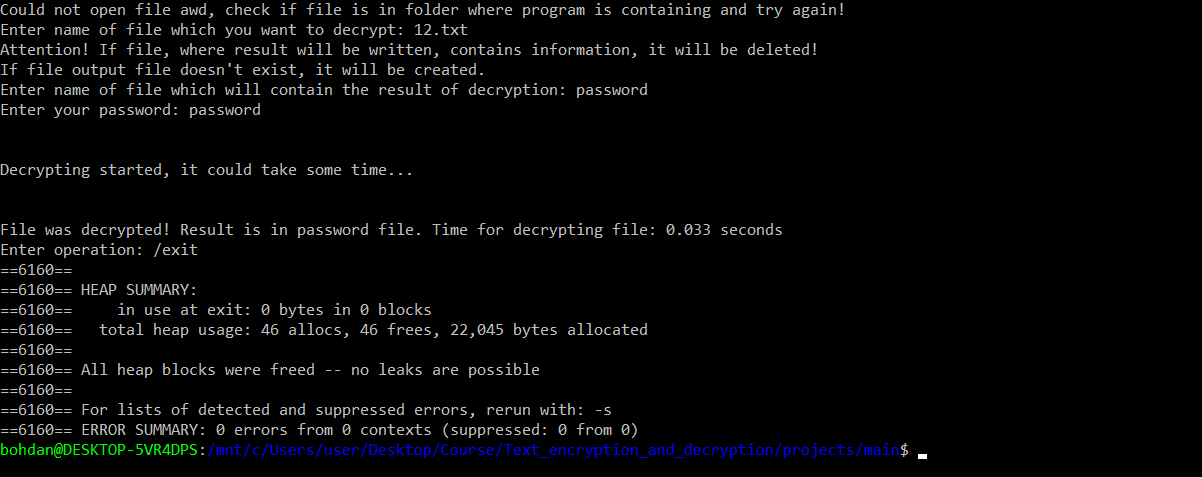


Рисунок 5.5 – Перевірка на витоки пам’яті

Після потрібно перевірити, чи програма здатна зашифрувати та розшифрувати файл великого об’єму. Для тесту було обрано файл великого об’єму (100 мегабайт) під назвою load\_testing.txt.

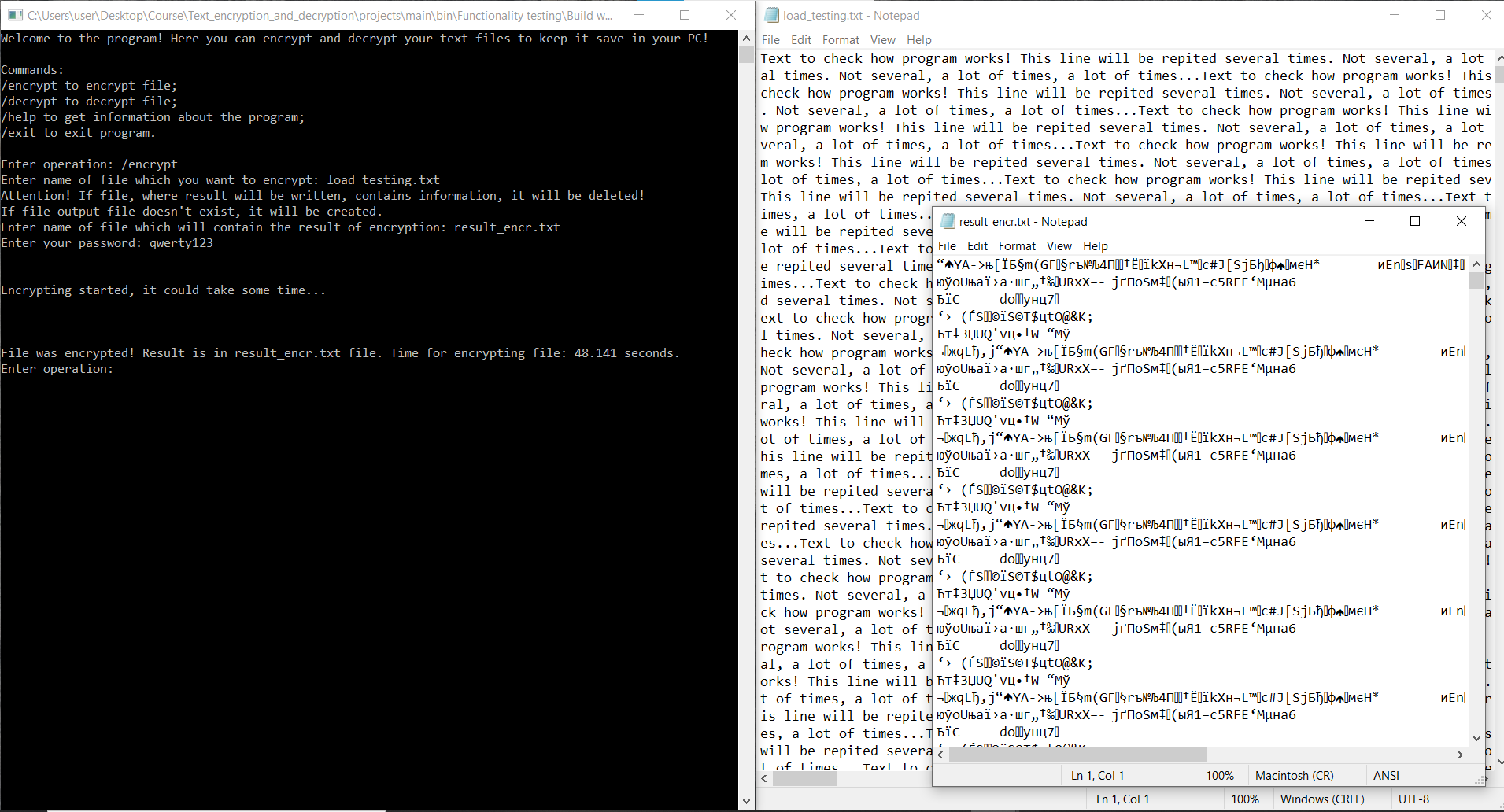


Рисунок 5.6 – Навантажувальне тестування, шифрування, вміст файлів

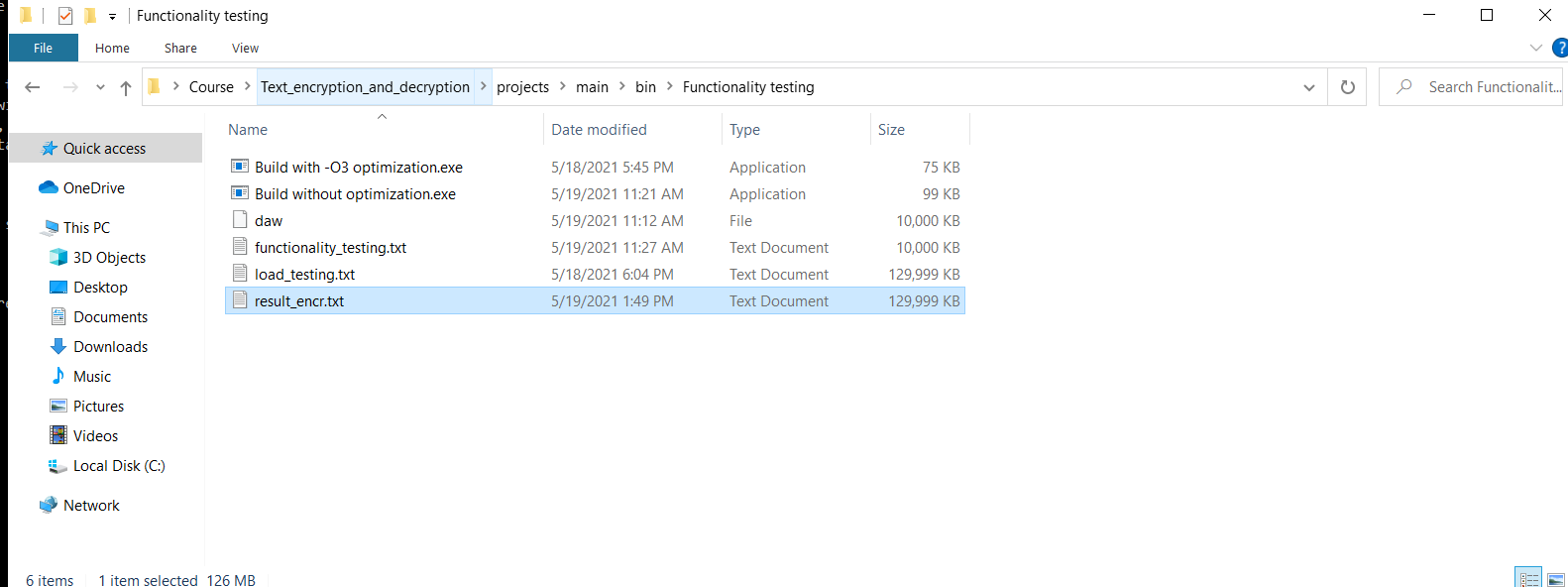


Рисунок 5.7 – Навантажувальне тестування, шифрування, вміст папки

Можна помітити, що швидкість шифрування не надто висока (приблизно 2 мегабайти в секунду), але програма справилася з шифруванням. Тепер потрібно перевірити дешифрування

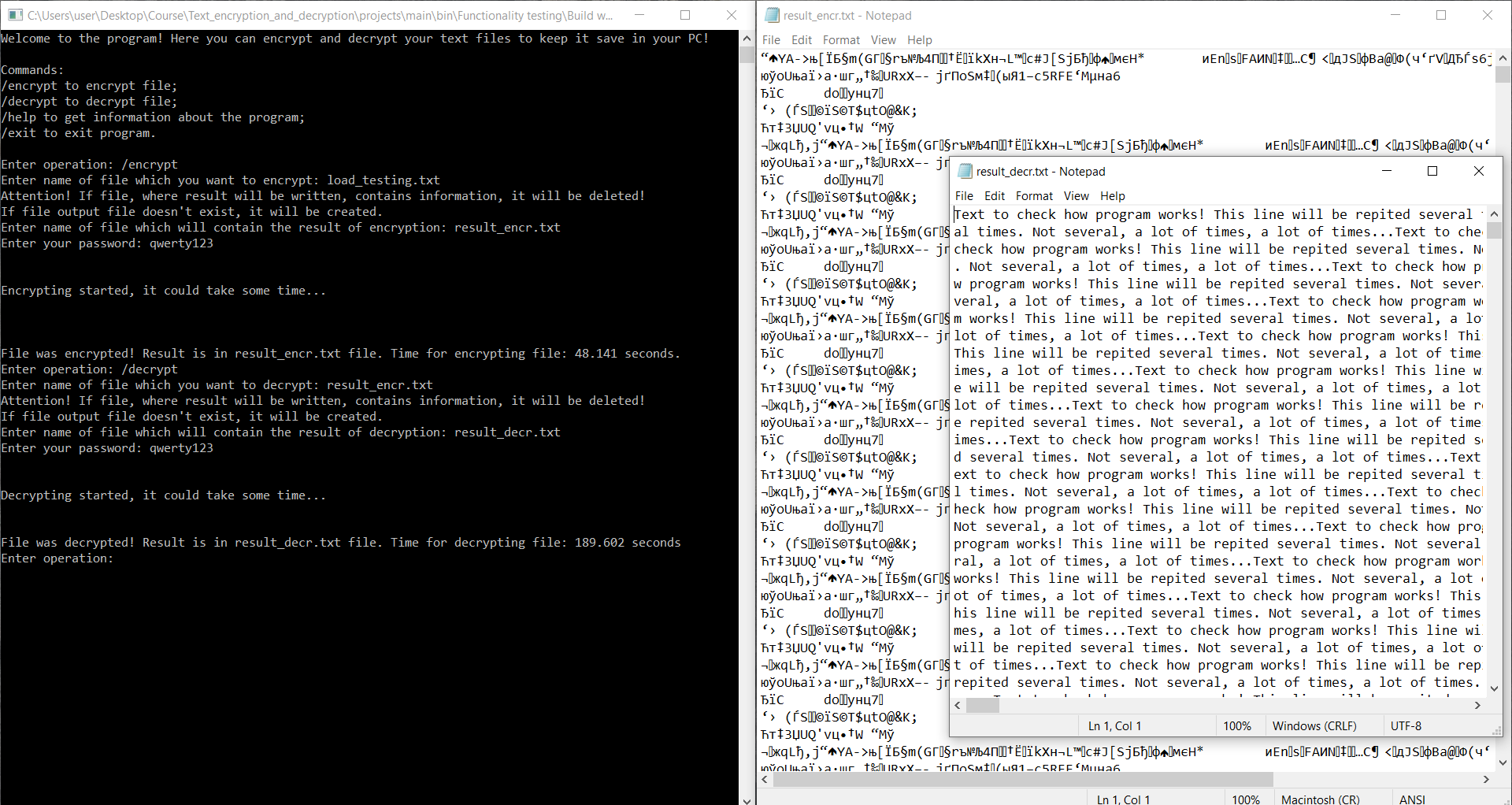


Рисунок 5.8 – Навантажувальне тестування, дешифрування, вміст файлів

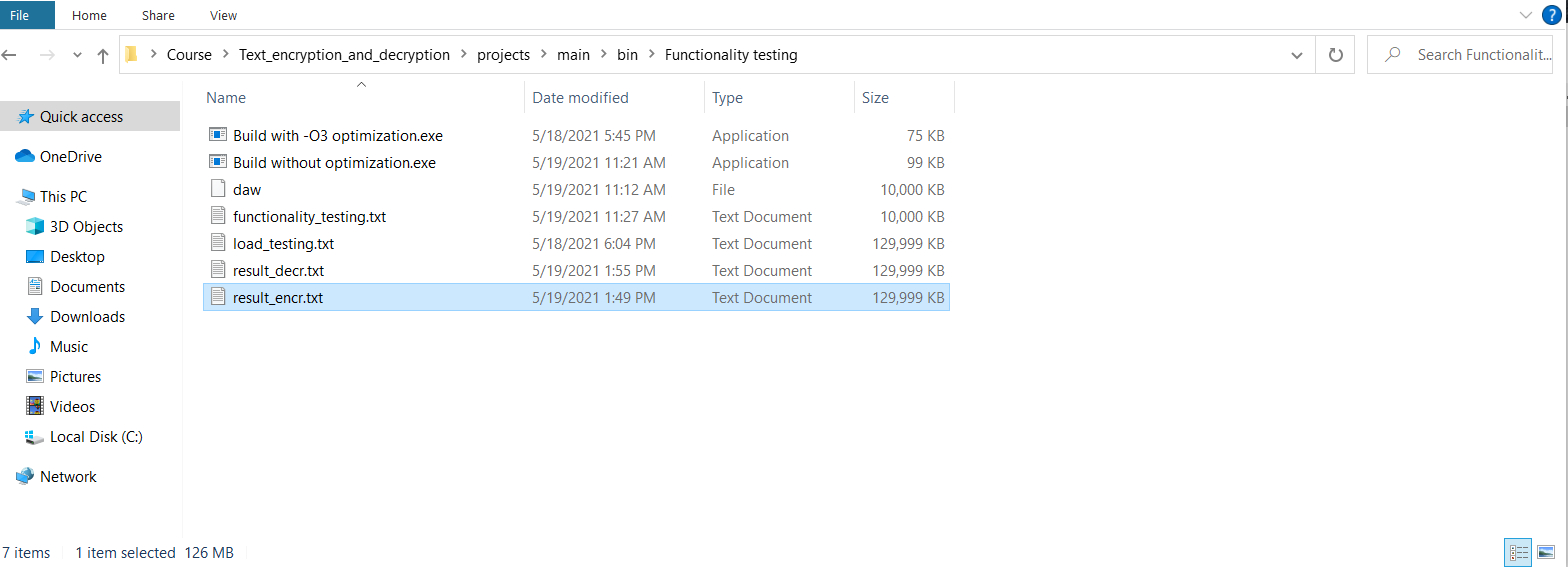


Рисунок 5.9 – Навантажувальне тестування, дешифрування, вміст папки

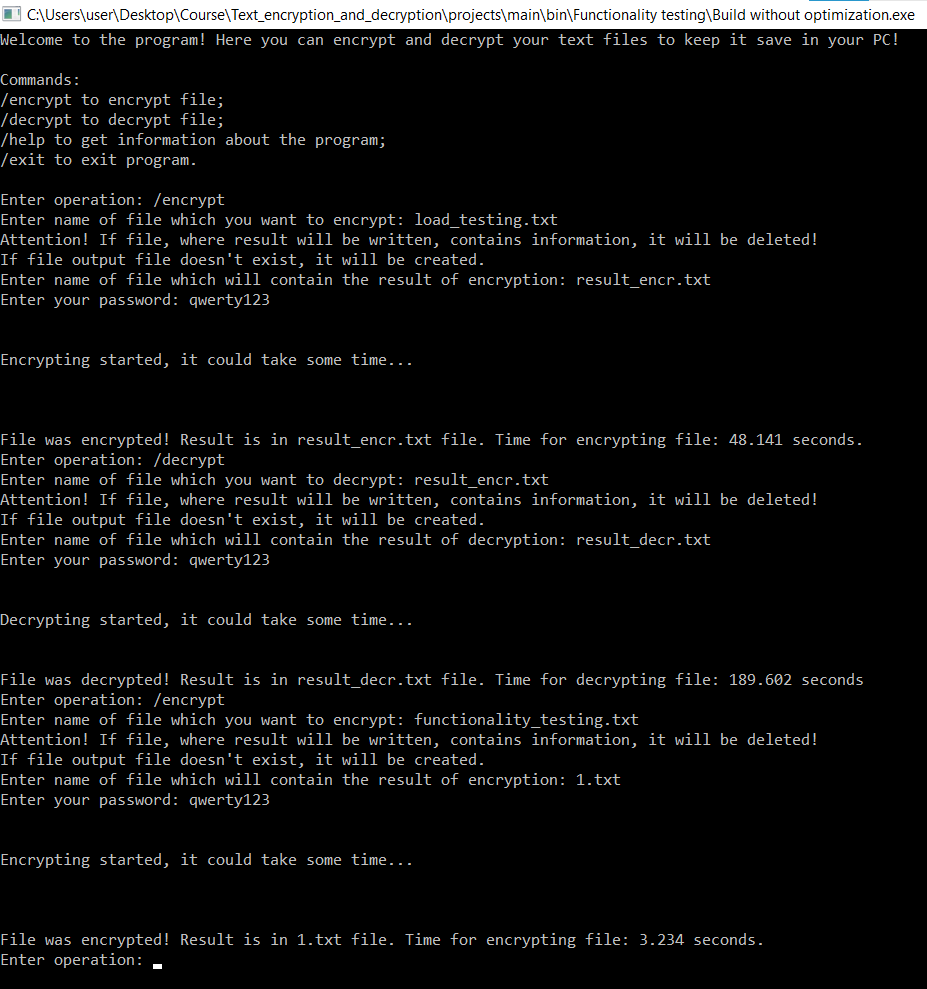


Рисунок 5.10 – Навантажувальне тестування, перевірка роботи програми після шифрування та дешифрування файлу великого об’єму

Програма змогла справитися з файлами великого об’єму і пройшла навантажувальне тестування, але швидкість шифрування склала 2 мегабайти/секунда, а дешифрування – 0.5 мегабайт/секунда, що є доволі низьким показником для AES-128. Скоріше всього це зв’язано з компілюванням програми без режиму оптимізації. Це можна виправити, скомпілювавши програми в режимі оптимізації O3. Результати роботи програми з оптимізацією знаходяться на рисунку 5.11.

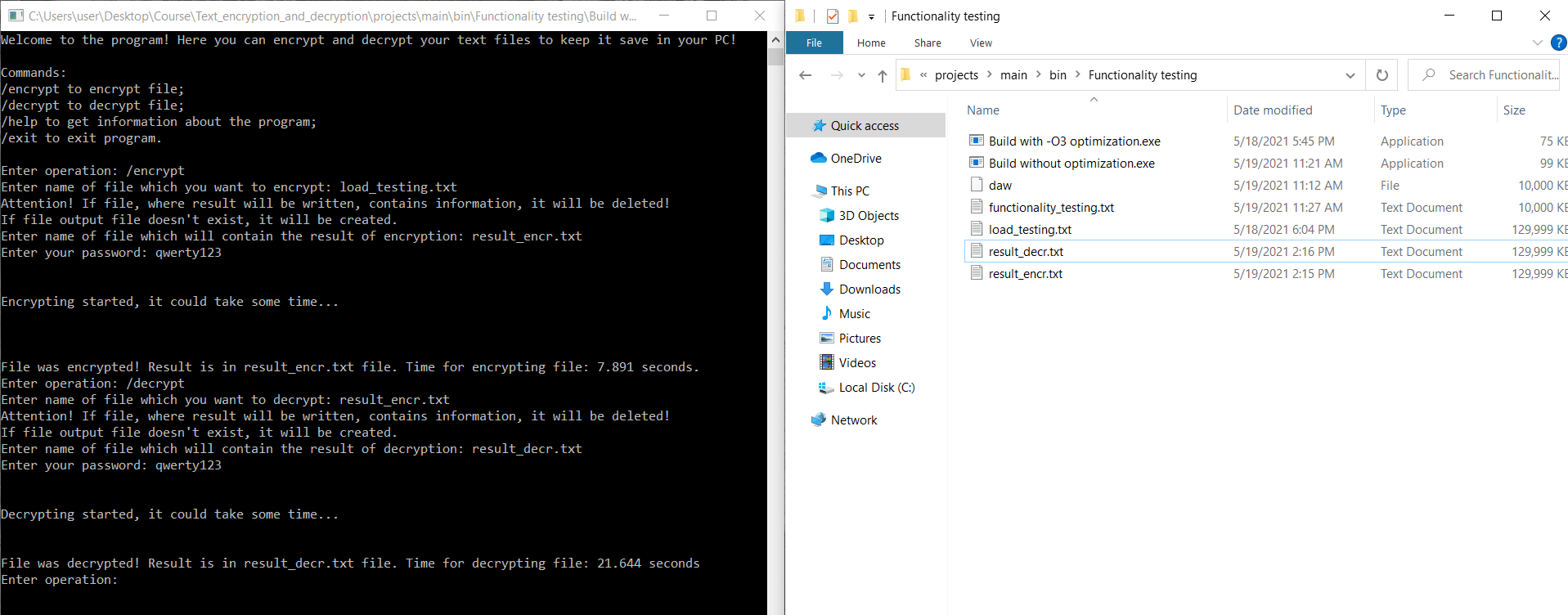


Рисунок 5.11 – шифрування файлу на 100 мегабайт з оптимізацією програми O3

Після оптимізації швидкість шифрування склала 12 мегабайт/секунда, дешифрування – 4.6 мегабайт/секунда. Таким чином після оптимізації шифрування стало в 6 разів швидше, а дешифрування – в 9 разів.