МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
червня 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система криптозахисту даних на вбудованих носіях»
здобувача групи ІН-02 Мягких Романа Сергійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Роман МЯГКИХ
(підпис)

Керівник,
старший викладач,
кандидат технічних наук
Артем КОРОБОВ

_____
(підпис)

Суми – 2024

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-02 Мягких Романа Сергійовича

1. Тема роботи: «Інформаційна система криптозахисту даних на вбудованих носіях» затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для криптозахисту даних. 3) Розробка інтелектуальної системи криптозахисту даних на вбудованих носіях. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження	06.05-08.05	
2	Огляд технологій, що використовуються для криптозахисту даних на вбудованих носіях	09.05-13.05	
3	Розробка інформаційної системи криптозахисту даних на вбудованих носіях	14.05-28.05	
4	Аналіз отриманих результатів	29.05	
5	Оформлення пояснювальної записки до кваліфікаційної роботи	30.05-01.06	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 63 стр., 40 рис., 4 додатка, 15 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі криптозахисту даних на вбудованих носіях та інтеграція додаткових методів захисту додатку, його пришвидшення за допомогою відповідних інструментів.

Об'єкт дослідження — криптозахист даних на вбудованих носіях.

Мета роботи — розробка інформаційної системи криптозахисту даних на вбудованих носіях.

Методи дослідження — алгоритми криптозахисту даних, алгоритми обфускації створеного коду і алгоритми побудови інформаційної системи.

Результати — розглянуто та проаналізовано основні принципи криптографічного захисту даних, включаючи симетричні та асиметричні шифри, хеш-функції, алгоритми електронного цифрового підпису. Розроблено інформаційну систему криптозахисту даних на вбудованих носіях, яка використовує сучасні методи шифрування та підпису даних для забезпечення безпеки і конфіденційності інформації. Проведено тестові дослідження ефективності розробленої системи на різних типах вбудованих носіїв даних та різних обсягах інформації. Проведено manual тестування додатку для виявлення можливих вразливостей і недоліків в роботі програмного забезпечення та подальшого удосконалення.

ІНФОРМАЦІЙНА СИСТЕМА, КРИПТОЗАХИСТ ДАНИХ, ВБУДОВАНІ НОСІЇ, OBFUSCATION, JAVA, JAVA CRYPTOGRAPHY EXTENSION.

ЗМІСТ

ВСТУП

Робота присвячена темі захисту даних на вбудованих носіях (флешка, зовнішні накопичувачі і т.п.) за допомогою криптозахисту, оскільки вони забезпечують збереження, обробку та передачу важливої інформації, а тому важливо, щоб ці дані були належним чином захищені.

Криптозахист даних - це комплекс заходів та технологій, спрямованих на забезпечення конфіденційності, цілісності та доступності інформації шляхом застосування криптографічних методів. Він використовує шифрування, хеш-функції, цифрові підписи, аутентифікацію та інші техніки для захисту даних від несанкціонованого доступу, модифікації та втрати.

Незастосування криптозахисту даних може призвести до серйозних наслідків, таких як несанкціонований доступ до конфіденційної інформації, виток важливих даних, порушення приватності користувачів, можливість викрадення та зловживання інтелектуальною власністю, а також пошкодження репутації організації або особистій безпеці, тому захист є дуже важливим і нехтувати їм не варто.

У даній роботі буде розглянуто основні принципи криптографії, які лягають в основу інформаційної системи криптозахисту даних на вбудованому носії. Детально будуть розглянуті методи та алгоритми шифрування, що застосовуються для захисту інформації, а також механізми контролю доступу.

1. Інформаційний огляд

1.1. Криптозахист даних

Криптозахист даних є важливою складовою сфери інформаційних технологій і використовується для забезпечення конфіденційності, цілісності та доступності цінної інформації. Він використовує криптографічні методи для захисту даних від несанкціонованого доступу та змін. [1]

Основні види криптозахисту даних включають:

- **Симетричне шифрування:** Використовує один ключ для як шифрування, так і розшифрування даних. Він є швидким та ефективним, але потребує безпечного обміну ключем між відправником і одержувачем.
- **Асиметричне шифрування:** Використовує пару ключів - публічний і приватний. Публічний ключ використовується для шифрування даних, а приватний ключ - для їх розшифрування. Цей підхід забезпечує більшу безпеку, але є менш ефективним у порівнянні з симетричним шифруванням.
- **Хеш-функції:** Використовуються для перетворення даних будь-якого розміру в фіксований вихідний хеш-код. Це дозволяє перевірити цілісність даних, порівнюючи отриманий хеш-код зі збереженим значенням.
- **Цифровий підпис:** Використовується для підтвердження автентичності та цілісності повідомлення. Повідомлення підписується за допомогою приватного ключа, а отримувач може перевірити підпис, використовуючи публічний ключ.

Криптозахист даних використовується у багатьох галузях, включаючи:

- Банківські та фінансові установи для захисту фінансових транзакцій та особистих даних клієнтів.
- Комерційні підприємства для забезпечення конфіденційності комерційної інформації та інтелектуальної власності.
- Урядові організації для захисту конфіденційної інформації, включаючи державні секрети та особисті дані громадян.
- Корпоративні мережі для захисту даних, що передаються через мережу.
- Онлайн-платформи та сервіси для захисту особистих даних користувачів.

Криптозахист даних є невід'ємною частиною сучасного світу інформаційних технологій, сприяючи захисту важливої інформації від потенційних загроз та несанкціонованого доступу. [2][3]

1.2. Аналіз аналогічних проектів

У даному розділі проводиться аналіз програмних засобів, що забезпечують шифрування та захист конфіденційної інформації, схожих на обрану для дослідження тему. Зокрема, досліджуються такі програмні рішення, як VeraCrypt, PGP Desktop та Folder Lock.

VeraCrypt є продовженням і розвитком проекту TrueCrypt, який був припинений у 2014 році. Він надає можливість створювати зашифровані контейнери для зберігання даних або шифрувати цілі диски. Однією з головних переваг VeraCrypt є відкритий вихідний код, що дозволяє перевірити його безпеку та незалежно аудитувати. Програма підтримує різні алгоритми шифрування, включаючи AES, Serpent та Twofish, що забезпечує високий рівень безпеки.

PGP Desktop (Pretty Good Privacy) є іншим популярним засобом для шифрування файлів та комунікацій. Він пропонує широкий спектр функцій, включаючи зашифровану електронну пошту, створення цифрових підписів та шифрування дискових об'ємів. Однією з його особливостей є використання

публічних та приватних ключів для шифрування та розшифрування даних, що забезпечує високий рівень безпеки.

Folder Lock є програмою, спрямованою на захист конфіденційних даних на рівні окремих папок або файлів. Вона дозволяє створювати пароль-захищені "скриньки" для зберігання файлів, які будуть шифруватися та захищатися від несанкціонованого доступу. Крім того, Folder Lock може шифрувати файли на зовнішніх пристроях, таких як USB-накопичувачі або зовнішні жорсткі диски.

Порівняльний аналіз

У порівняльному аналізі цих програмних засобів важливо враховувати їхні можливості, швидкодію та рівень безпеки. VeraCrypt відрізняється відкритим вихідним кодом та широким спектром алгоритмів шифрування. PGP Desktop славиться своїми засобами електронного підпису та використанням ключів PGP для шифрування. Folder Lock спрощує захист окремих файлів та папок, забезпечуючи високий рівень доступу.

1.3. Постановка задачі

Метою роботи є розробка та реалізація додатку криптозахисту даних на вбудованих носіях та її вдосконалення шляхом інтеграції додаткових захистів для програмного коду та пришвидшення роботи за допомогою відповідних інструментів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Створення прототипу інтерфейсу програми;
2. Вибір алгоритму шифрування, мови програмування для реалізації додатку та відповідних бібліотек;
3. Програмна реалізація додатку криптозахисту даних;
4. Інтеграція додаткових методів захисту додатку та його пришвидшення за допомогою відповідних інструментів.

2. Вибір методів рішення задачі

2.1. Алгоритм шифрування методу AES

AES (Advanced Encryption Standard) є одним з найбільш поширених алгоритмів шифрування, використовуваних для забезпечення конфіденційності та захисту даних. Він забезпечує надійний рівень шифрування та широко використовується в різних сферах, таких як інформаційна безпека, комунікації та зберігання даних.

Принцип роботи AES полягає в тому, що вхідний текст (який потрібно зашифрувати) розбивається на блоки фіксованої довжини (наприклад, 128 біт), і кожен блок обробляється незалежно. Шифрування використовує ключ, який також має фіксовану довжину (наприклад, 128, 192 або 256 біт).

Процес шифрування AES включає кілька раундів шифрування, кількість яких залежить від довжини ключа. Кожен раунд включає чотири основних етапи: заміна байтів (SubBytes), зсуви рядків (ShiftRows), змішування стовпців (MixColumns) та додавання ключа (AddRoundKey).

- Заміна байтів (SubBytes): Кожен байт вхідного блоку замінюється на відповідний байт з S-блока (S-box), що забезпечує необоротність шифрування та додає нелінійність.

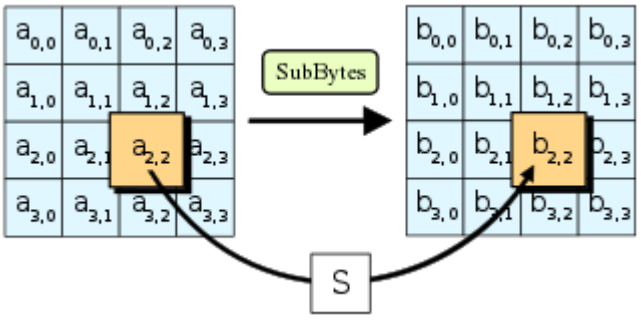

Рисунок 1. Принцип роботи етапу SubTypes

- Зсуви рядків (ShiftRows): Байти в кожному рядку блоку зсуваються циклічно на певну кількість позицій вліво. Це перетворення змішує байти у різних рядках і розподіляє їх по всьому блоку.

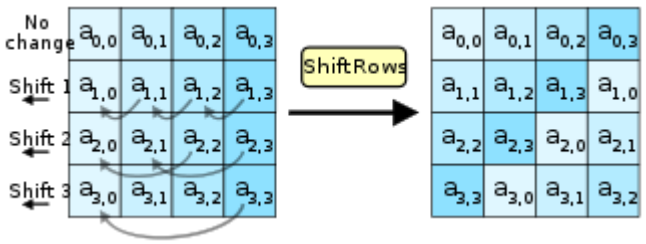

Рисунок 2. Принцип роботи етапу ShiftRows

- Змішування стовпців (MixColumns): Кожен стовпець блоку множиться на певну матрицю, що перемішує байти в стовпцях і забезпечує додаткову безпеку.

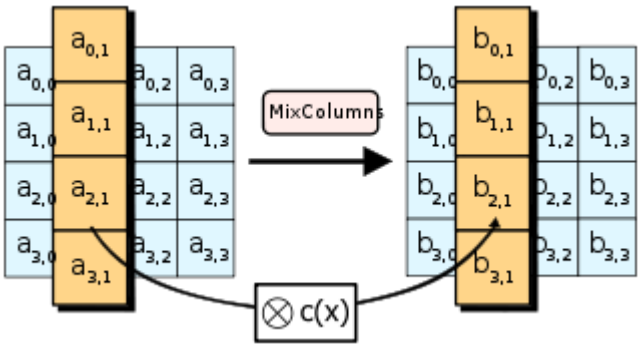

Рисунок 3. Принцип роботи етапу MixColumns

- Додавання ключа (AddRoundKey): Кожен байт в блоку XOR-ується з відповідним байтом з ключа. Ключом для кожного раунду є підключ, який отримується з основного ключа шляхом розширення ключа (key expansion).

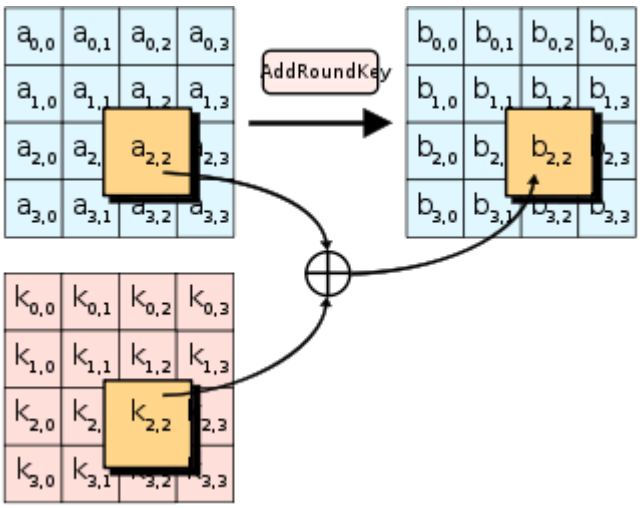

Рисунок 4. Принцип роботи етапу AddRoundKey

В цьому етапі використовується операція XOR (ексклюзивне або). XOR - це бінарна операція, яка працює з двійковими числами або бітами.

У контексті AES, операція XOR використовується для комбінування даних блоку зі спеціальним ключем раунду. Кожен байт в блоку XOR-ується з відповідним байтом з ключа. Це означає, що відповідні біти в обох числах (байтах) перевертаються, якщо вони однакові, і залишаються без змін, якщо вони відрізняються.

Операція XOR має такі особливості:

- Якщо два біти мають однакове значення (обидва 0 або обидва 1), результат буде 0.
- Якщо два біти мають різне значення (один 0, а інший 1), результат буде 1.

Застосування операції XOR в AES дозволяє комбінувати дані блоку з ключем раунду, що додає додатковий рівень безпеки і комплексності до шифрування. Це забезпечує нелінійність шифрування та робить AES більш стійким до атак.

Отже, в кожному раунді AES, операція XOR використовується для комбінування даних блоку з відповідним підключем раунду, що додає ще один шар шифрування та забезпечує безпеку та конфіденційність даних.

Кількість раундів залежить від довжини ключа: 10 раундів для AES-128, 12 раундів для AES-192 та 14 раундів для AES-256. Кожен раунд включає всі перетворення, описані вище. [4]

2.2. Застосування AES шифрування на мові Java за допомогою бібліотеки Java Cryptography Extension (JCE)

Java Cryptography Extension (JCE) є набором інструментів, який надає розширені функції криптографії для мови програмування Java. JCE включає в себе реалізацію різних алгоритмів шифрування, хешування та підписування, які можуть використовуватись для захисту даних в Java-програмах.[8]

JCE надає методи шифрування, такі як шифрування блоками (**Block Cipher**), шифрування потоку даних (**Stream Cipher**) і шифрування з відкритим ключем (**Public Key Cryptography**). API JCE підтримує різні алгоритми, такі як DES (**Data Encryption Standard**), AES (**Advanced Encryption Standard**), RSA (**Rivest-Shamir-Adleman**) і інші. Крім того, JCE включає в себе можливості для генерації випадкових чисел, обробки цифрових підписів і хеш-функцій.

Переваги використання JCE полягають у наступному:

- **Безпека:** JCE надає високий рівень безпеки шляхом підтримки сильних алгоритмів шифрування та криптографічних протоколів.
- **Гнучкість:** JCE дозволяє розширювати функціональність шифрування шляхом додавання власних алгоритмів і реалізацій.
- **Інтеграція:** JCE добре інтегрується з існуючими Java-бібліотеками та інфраструктурою, що дозволяє легко використовувати його в Java-проектах.
- **Переносимість:** JCE підтримується на різних платформах, що дозволяє розробникам використовувати однаковий код на різних середовищах. [9]

2.3. Створення інтерфейсу на основі платформи JavaFX

Для створення програми шифрування вбудованих носіїв даних, було прийнято рішення розробити її інтерфейс на основі дуже гнучкої та функціональної платформи – JavaFX. [5]

JavaFX - це фреймворк для створення графічних інтерфейсів користувача (GUI) в програмах, написаних на мові програмування Java. Основні переваги JavaFX включають:

- **Модульність:** JavaFX побудований на основі модульної архітектури, що дозволяє використовувати тільки необхідні компоненти і зменшує розмір виконуваних файлів.
- **Багатофункціональність:** JavaFX надає багатий набір готових компонентів і контролів для створення різноманітних інтерфейсів, включаючи кнопки, таблиці, меню, вкладки тощо.
- **Висока продуктивність:** JavaFX використовує апаратне прискорення для досягнення високої продуктивності графічних операцій, що забезпечує плавну анімацію і швидку відгуку інтерфейсу.
- **Широкі можливості стилізації:** JavaFX має гнучкий механізм стилізації, що дозволяє змінювати зовнішній вигляд елементів інтерфейсу за допомогою CSS.
- **Підтримка мультимедіа:** JavaFX надає можливості для відтворення звуку і відео, а також роботи з графікою, включаючи векторну і 3D-графіку.
- **Можливість інтеграції:** JavaFX можна легко інтегрувати з іншими технологіями Java, такими як Java Swing і JavaFX Webview.
- **Підтримка мультиплатформеності:** JavaFX підтримує різні операційні системи, включаючи Windows, macOS і Linux, що дозволяє розробляти кросплатформенні програми. [6][7]

3. Обфускація коду за допомогою відповідних інструментів для додаткового захисту додатку

3.1. Інформаційний огляд додаткового захисту за допомогою обфускації коду

У сучасному світі безпека програмного забезпечення стає все важливішою з прогресом технологій. Одним із ефективних методів захисту додатків від зловмисників є обфускація коду. Обфускація - це процес перетворення зрозумілого коду в складночитабельний, але функціонально еквівалентний код, що ускладнює аналіз та розуміння коду.

Навіщо це потрібно?

Як відомо, одним з основних методів взлому програмного забезпечення є аналіз коду, отриманого в результаті роботи дизасемблера на предмет вразливостей. На основі такого аналізу нескладно, наприклад, скласти програму для генерації ключів активації комерційного програмного забезпечення або, навпаки, внести зміни до виконуваного файлу - патч, який дозволить зловмисникам відключити "небажані" модулі вихідної програми.

Усьому вищеописаному може протидіяти спеціальна програма - обфускатор.

Також, алгоритми обфускації активно використовуються не лише для ускладнення аналізу коду, але й для зменшення розміру програмного коду, що, у свою чергу, активно використовується при розробці різних веб-сервісів та баз даних.

Як це повинно працювати?

Як зрозуміло з вищезазначеного, методи обфускації повинні ускладнити код, перетворивши його таким чином, щоб приховати логіку його роботи від третіх осіб.

В ідеалі хотілося б, щоб програма, яка пройшла обфускацію, надавала не більше інформації, ніж чорний ящик, що імітує поведінку вихідної

програми. Гіпотетичний алгоритм, що реалізує таке перетворення, називається **"Обфускація чорного ящика"**. Декомпіляція зашифрованої таким чином програми дала б зловмисникам не більше інформації, ніж декомпіляція клієнта месенджера, який представляє собою лише обгортку над API "справжньої" програми, що б повністю вирішило поставлену в попередньому блоку проблему. Однак показано, що реалізація такого алгоритму для довільної програми неможлива.

Як це працює?

Більшість методів обфускації перетворюють наступні аспекти коду:

- **Дані:** роблять елементи коду схожими на те, чим вони насправді не є.
- **Потік коду:** встановлюють виконувану логіку програми абсурдною або навіть недетермінованою.
- **Структура формату:** застосовують різне форматування даних, перейменування ідентифікаторів, видалення коментарів коду та інше.

Інструменти обфускації можуть працювати як з вихідним або байтовим кодом, так і з бінарним, але обфускація двійкових файлів складніша і повинна змінюватися залежно від архітектури системи.

При обфускації коду важливо правильно оцінити, які частини можна ефективно заплутати. Слід уникати обфускації коду, який критичний з точки зору продуктивності.[10]

3.2. Методи обфускації коду

Трансформація даних.

Одним з найважливіших елементів обфускації є перетворення даних, що використовуються програмою, в іншу форму, яка має мінімальний вплив на продуктивність коду, але значно ускладнює можливість зворотного інжинірингу для хакерів.

Обфускація потоку управління кодом.

Обфускація потоку управління може бути виконана шляхом зміни порядку виконання операторів програми. Зміна графа управління за

допомогою вставки довільних інструкцій переходу та трансформації деревоподібних умовних конструкцій у плоскі оператори перемикання, як показано на наступній діаграмі.

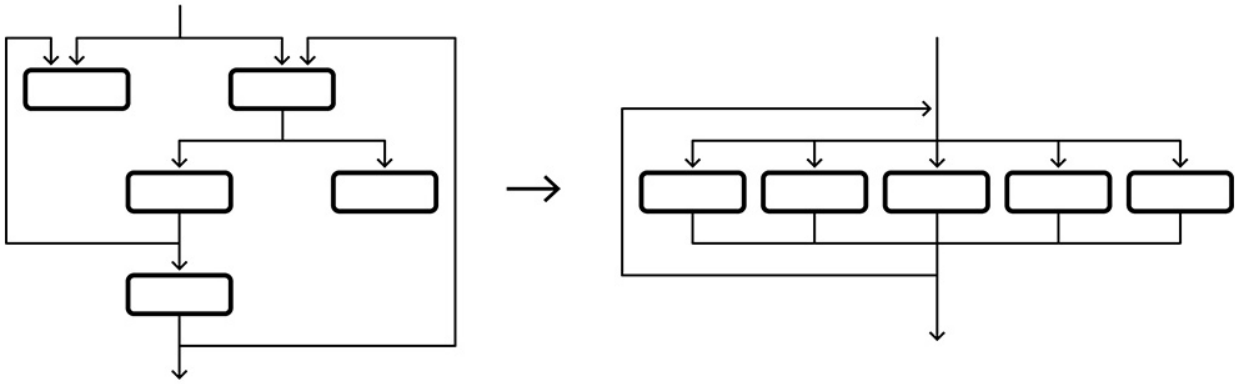

Рисунок 5. Обфускація потоку управління кодом

Обфускація адрес.

Цей метод змінює структуру зберігання даних, щоб ускладнити їх використання. Наприклад, алгоритм може обирати випадкові адреси даних у пам'яті, а також відносні відстані між різними елементами даних. Цей підхід варто відзначити тим, що навіть якщо зловмисник і зможе "декодувати" дані, що використовуються додатком на певному конкретному пристрої, то на інших пристроях він все одно не зможе відтворити свій успіх.

Регулярне оновлення коду.

Цей метод запобігає атакам, регулярно випускаючи оновлення обфускованого програмного забезпечення. Своєчасна заміна частин існуючого програмного забезпечення новими обфускованими екземплярами може змусити зловмисника відмовитися від існуючого результату зворотного аналізу, оскільки зусилля щодо взлому коду в такому випадку можуть перевищити отриману від цього цінність.

Обфускація інструкцій асемблера.

Трансформація та зміна асемблерного коду також може ускладнити процес зворотного інжинірингу. Одним із таких методів є використання перекриваючихся інструкцій (jump-in-a-middle), внаслідок чого дизасемблер

може зробити неправильний висновок. Асемблерний код також може бути посилено проти проникнення за рахунок включення непотрібних управляючих операторів та іншого сміттєвого коду.

Обфускація відлагоджувальної інформації.

Відлагоджувальну інформацію можна використовувати для зворотного проектування програми, тому важливо блокувати несанкціонований доступ до даних відлагодження. Інструменти обфускації досягають цього, змінюючи номери рядків та імена файлів у відлагоджувальних даних або повністю видаляючи з програми відлагоджувальну інформацію.[11][12]

3.3. Обфускація коду за допомогою інструментів ProGuard

Одним з інструментів, який забезпечує можливість обфускації коду, є ProGuard. ProGuard - це відкрите програмне забезпечення для зменшення, оптимізації та обфускації байт-коду Java. Він дозволяє розробникам захистити свій код від небажаних дій шляхом перетворення зрозумілого іменування класів, методів та полів на скорочені, незрозумілі для людини символи.

Щоб використовувати ProGuard в проекті Maven, можна скористатися плагіном maven-proguard-plugin. Цей плагін дозволяє легко інтегрувати обфускацію коду в процес збирання Maven-проекту. Для конфігурації ProGuard використовується файл **proguard.pro**, де можна вказати правила обфускації, включити чи виключити певні класи, методи або поля.

При використанні ProGuard важливо враховувати деякі ключові параметри:

- ***-dontwarn:*** Цей параметр вказує ProGuard не виводити попередження про відсутність об'єктів у стеку викликів.
- ***-dontnote:*** Використовується для відключення попереджень про можливі виправлення.
- ***-keep:*** Цей параметр вказує ProGuard зберегти вказані класи, методи або поля незмінними, тобто не обфускувати їх.

- ***-keepclasseswithmembers:*** Цей параметр дозволяє зберегти класи, які містять вказані методи чи поля.
- ***-flattenpackagehierarchy:*** Параметр *-flattenpackagehierarchy* дозволяє "сплющувати" ієрархію пакетів, що також може сприяти обфускації.
- ***-keepattributes:*** Цей параметр дозволяє зберігати вказані атрибути в результаті обфускації.
- ***-adaptresourcefilecontents:*** Вказує ProGuard адаптувати вміст файлів ресурсів під час обфускації.
- ***-dontshrink:*** Забороняє зменшення розміру коду, тобто зберігає всі класи, методи та поля без змін.
- ***-dontoptimize:*** Відключає оптимізацію коду, забезпечуючи збереження його у вихідному вигляді.

Процес обфускації допомагає збільшити безпеку програмного забезпечення шляхом ускладнення процесу аналізу коду та зменшення ймовірності розуміння змісту для потенційних зловмисників. Використання інструментів, таких як ProGuard, у поєднанні з належною конфігурацією, є ефективним методом для захисту додатків від атак та зловживань.[13]

4. Програмна реалізація

Під час роботи над кваліфікаційною роботою бакалавра, було розроблено прототип інтерфейсу та його функціонал для майбутньої програми криптозахисту даних на вбудованих носіях. Для його реалізації було використано фреймворк JavaFX та додаток SceneBuilder, який дозволяє швидко розробляти візуальний макет інтерфейсу без кодування.

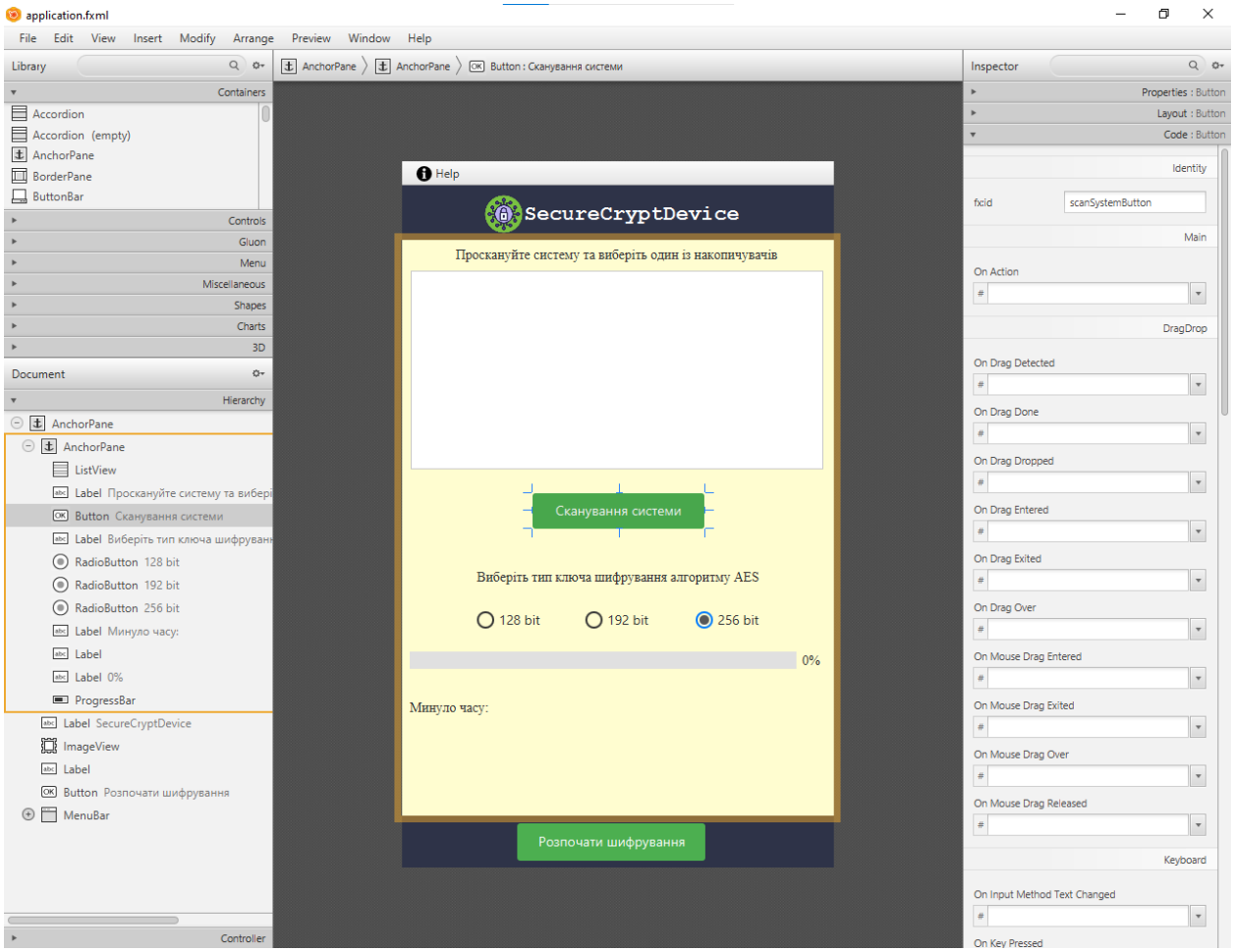

Рисунок 6. Візуал додатку SceneBuilder

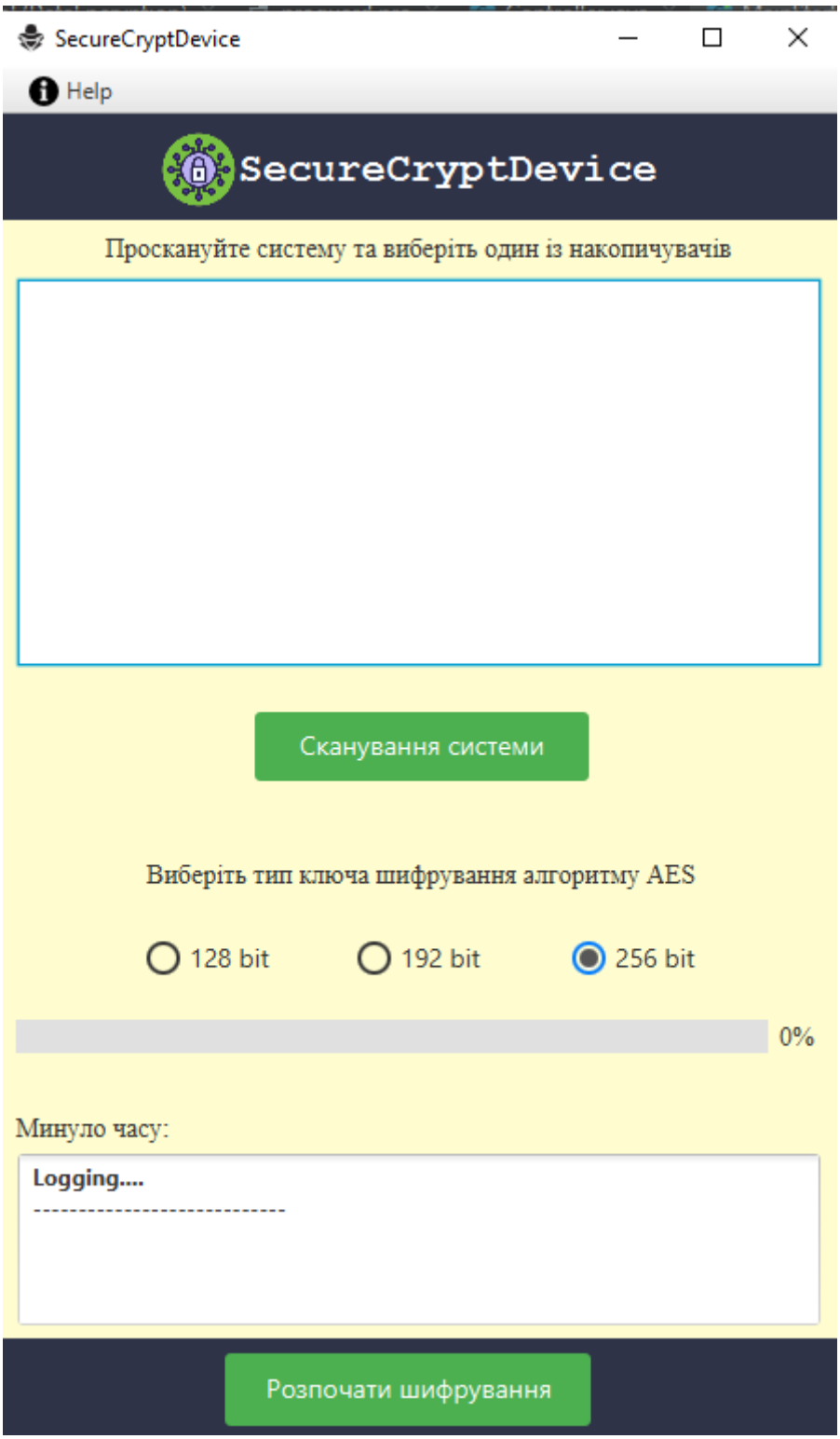

Рисунок 7. Інтерфейс головної "сторінки" програми

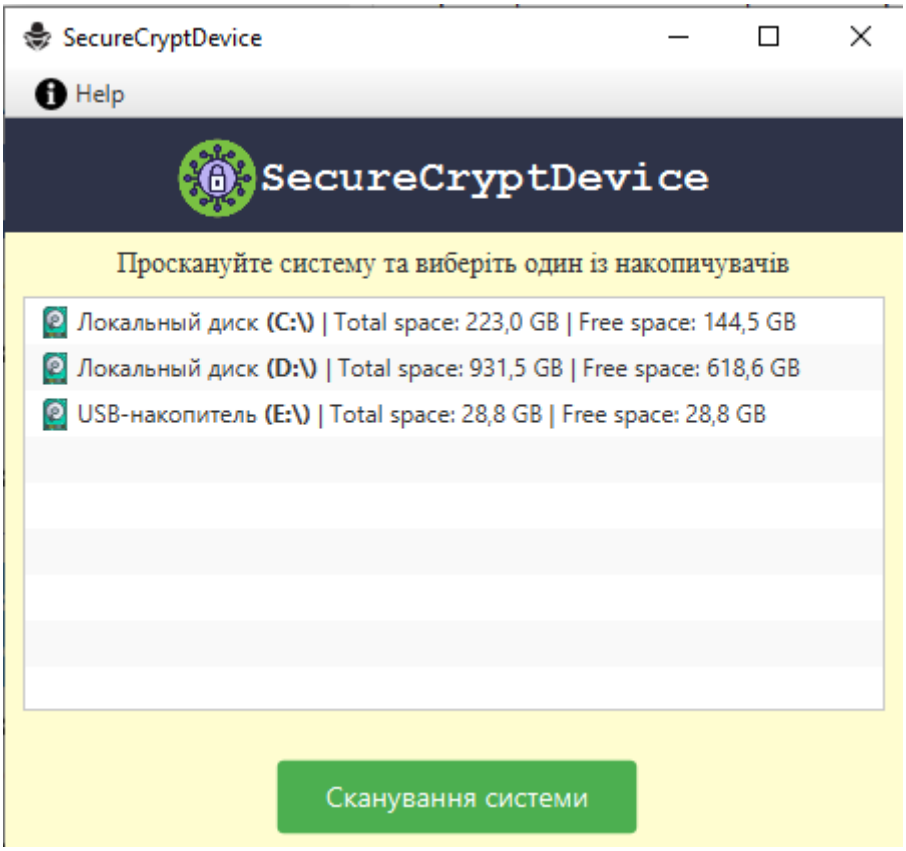

Рисунок 8. Сканування системи на наявність накопичувачів

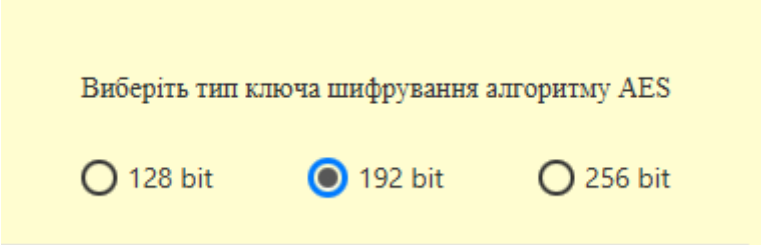

Рисунок 9. Вибір типу ключа шифрування

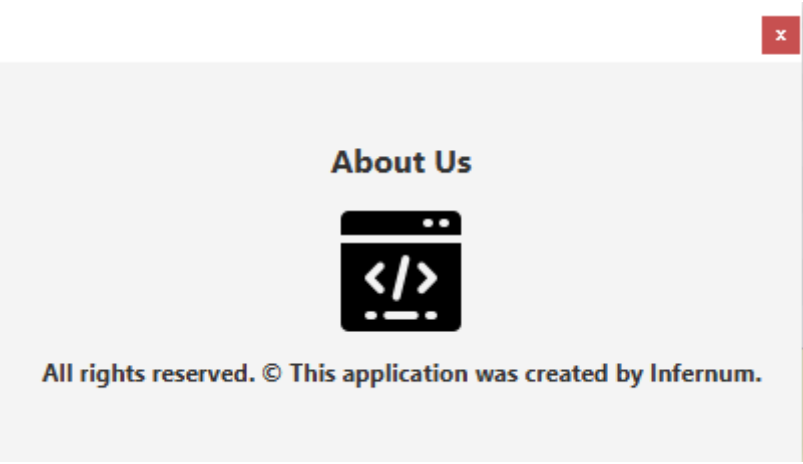

Рисунок 10. About Us

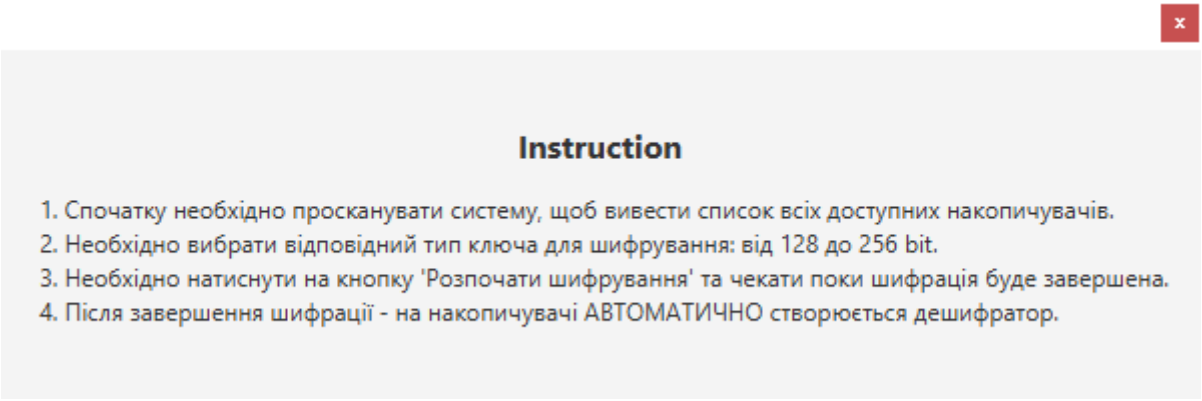

Рисунок 11. Instruction

Також, за допомогою інструментів мови програмування Java, фреймворку JavaFX та бібліотеки Java Cryptography Extension (далі – JCE) було розроблено метод шифрування даних на вбудованому носії та автоматичне створення дешифратора у вигляді виконуваного файлу на ньому з попереднім захистом у вигляді введення логіна та пароля для запуску дешифрації всіх даних на носії, якщо ж введені дані не вірні – виводиться помилка та автоматичне закриття програми дешифрації. Окрім цього, була проведена оптимізація програми для більш швидкої обробки байтів інформації під час шифрації та дешифрації.

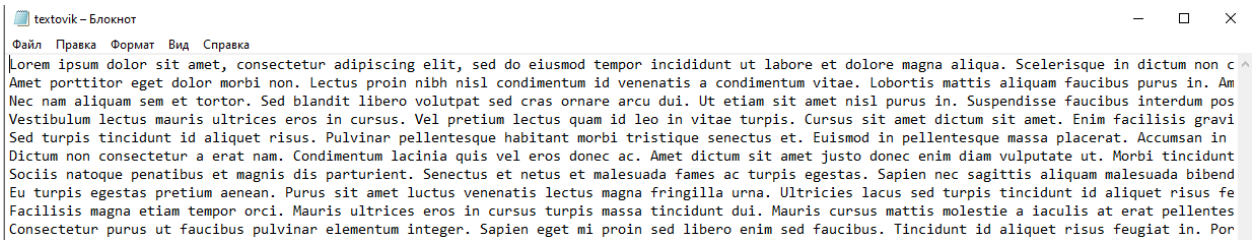

Рисунок 12. Перевірка відкриття файлу textovik.txt перед шифрацією

Рисунок 13. Перевірка відкриття файлу decrypt.txt перед шифрацією

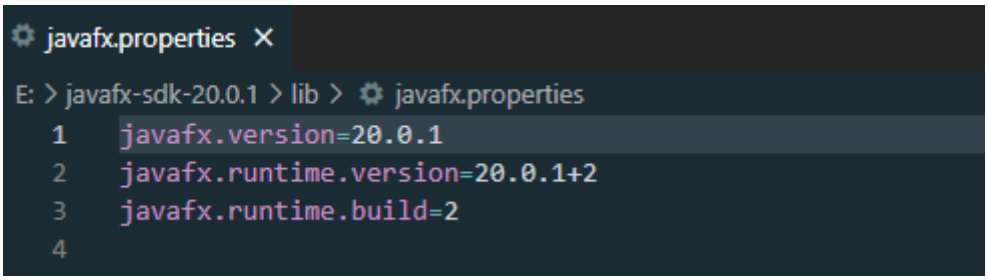

Рисунок 14. Перевірка відкриття рандомного файлу з папки модулів JavaFX

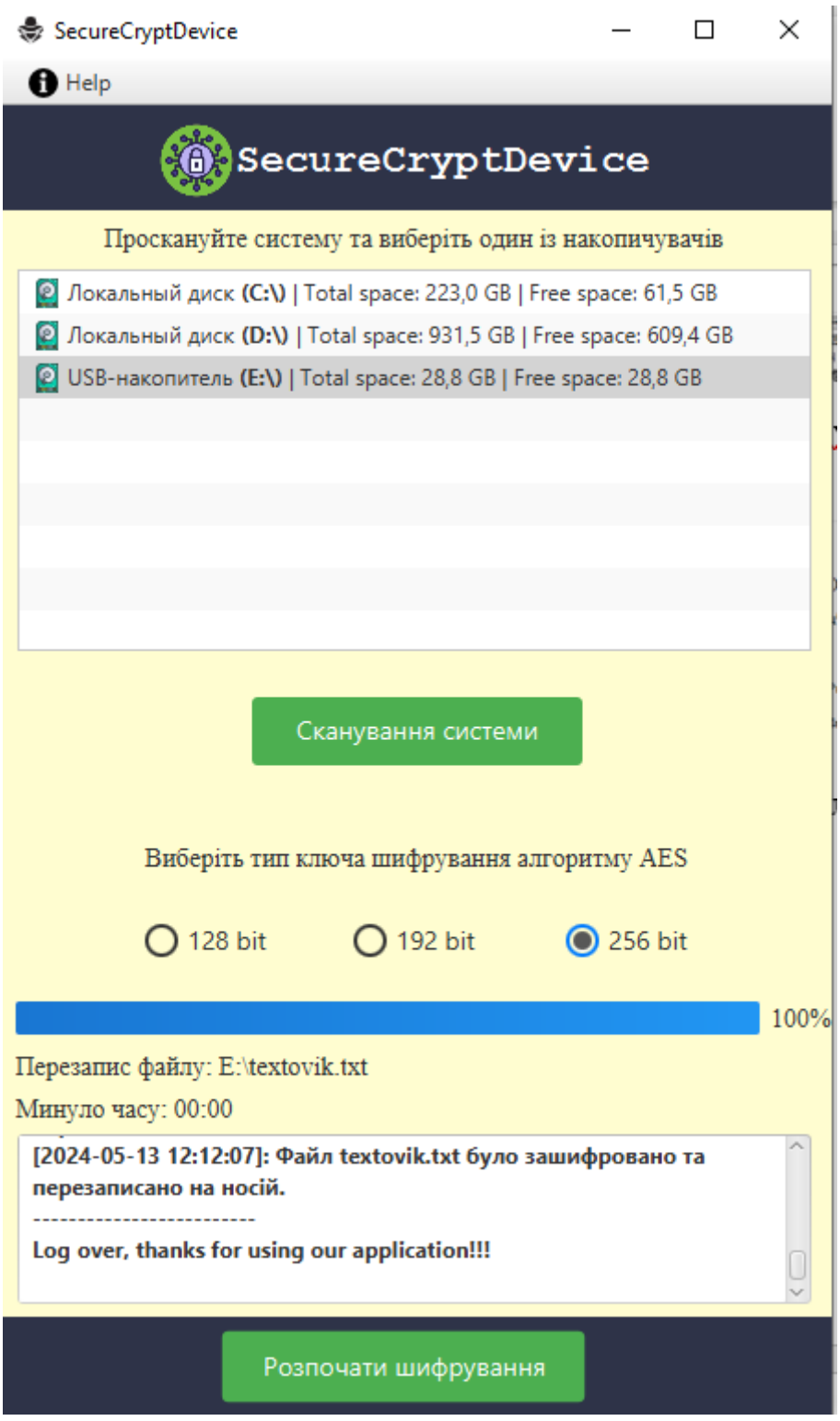

Рисунок 15. Шифрація даних на вбудованому носії після натискання кнопки "Розпочати шифрування"

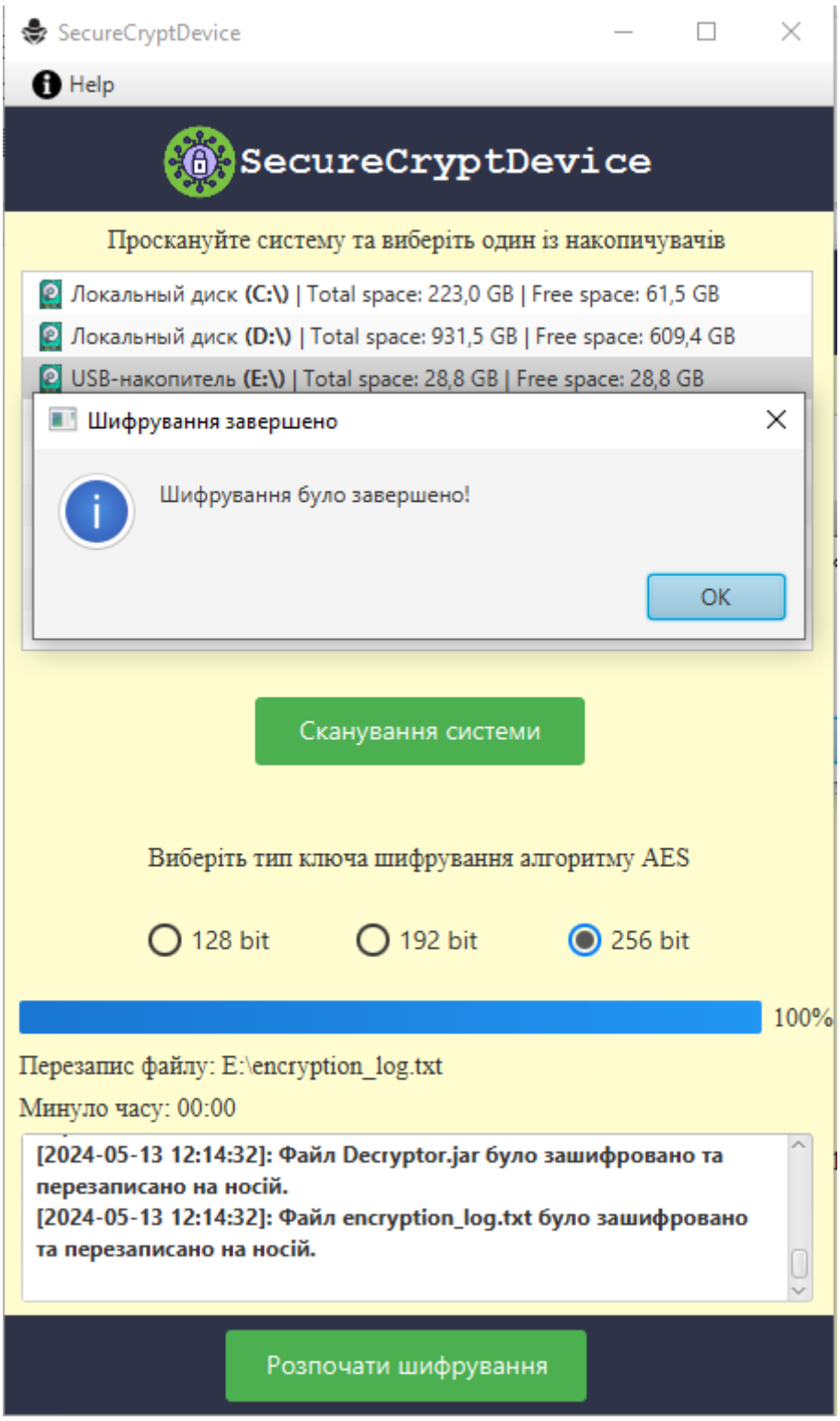

Рисунок 16. Виведення повідомлення про закінчення шифрації

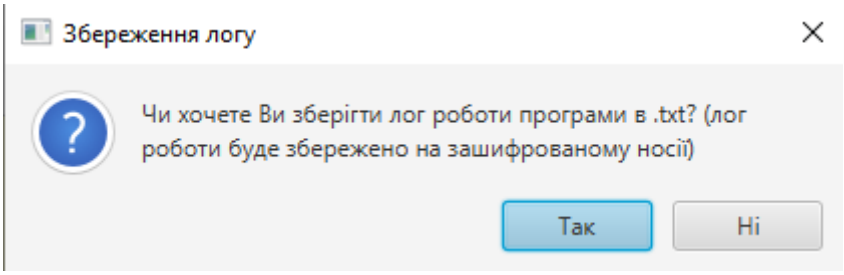

Рисунок 17. Виведення повідомлення щодо збереження логу в окремий .txt файл

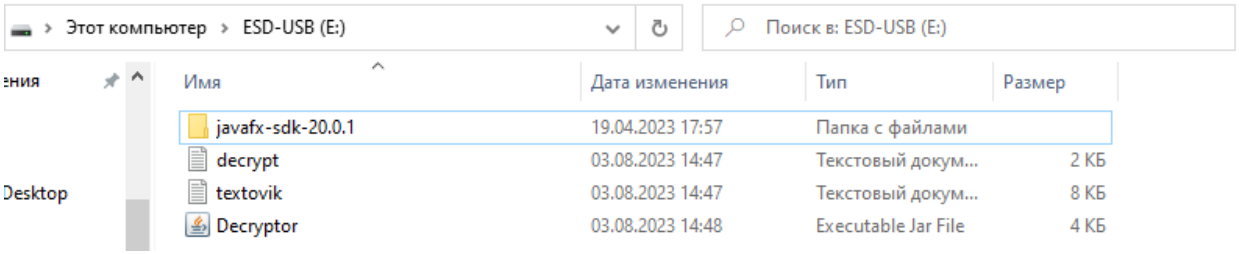

Рисунок 18. Перевірка автоматичного створення виконуваного файлу на вбудованому носії

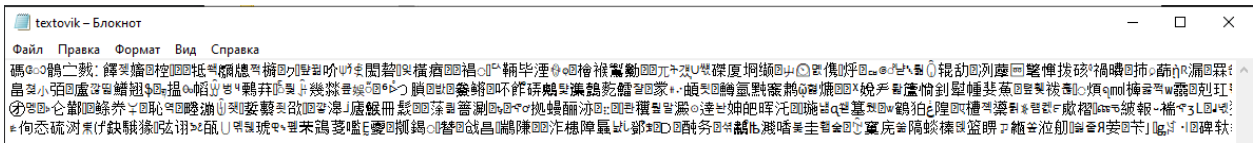

Рисунок 19. Перевірка шифрації тестового файлу textovik.txt

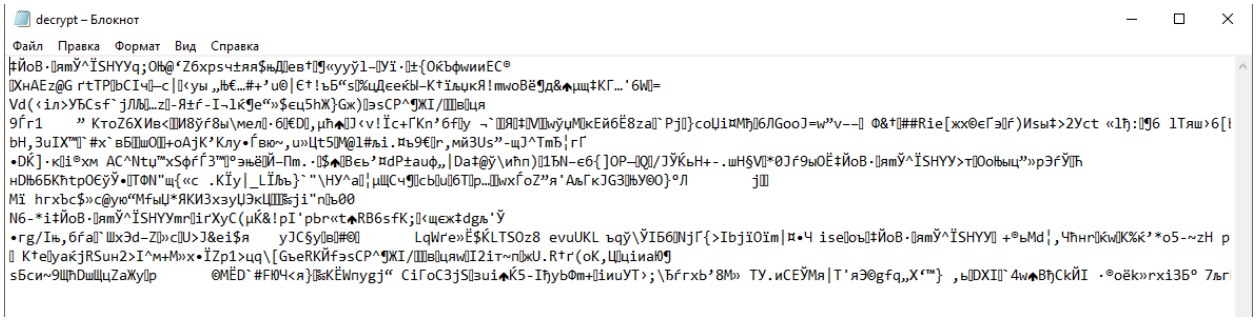

Рисунок 20. Перевірка шифрації інформації в тестовому файлі decrypt.txt

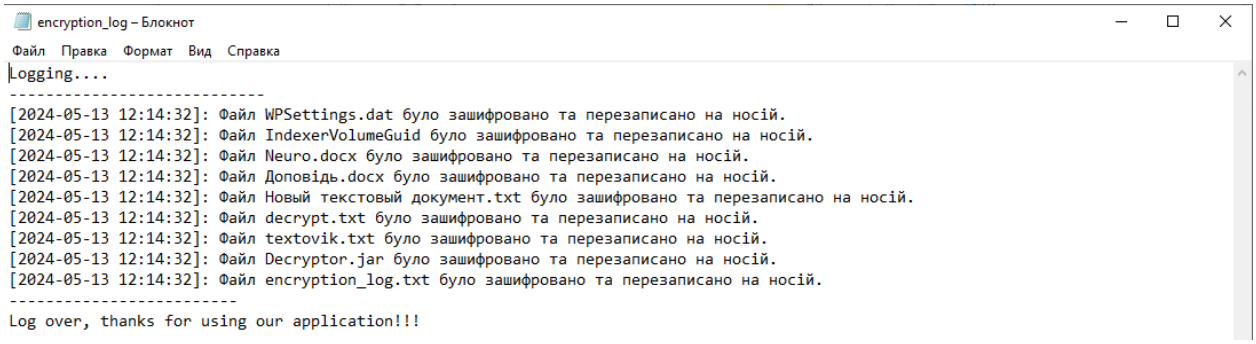

Рисунок 21. Текстовий файл логу шифрації даних

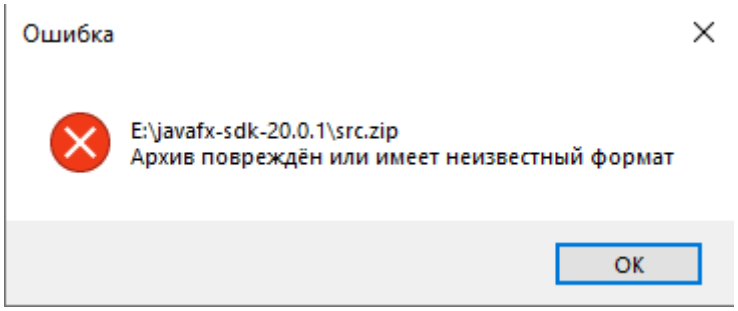

Рисунок 22. Перевірка відкриття архіву на вбудованому носії

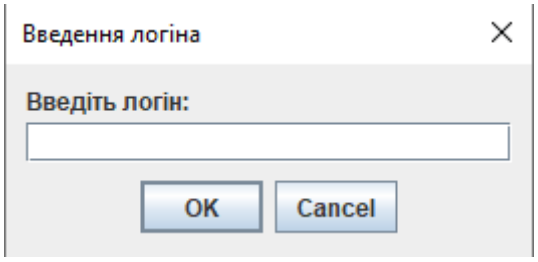

Рисунок 23. Вікно вводу логіна

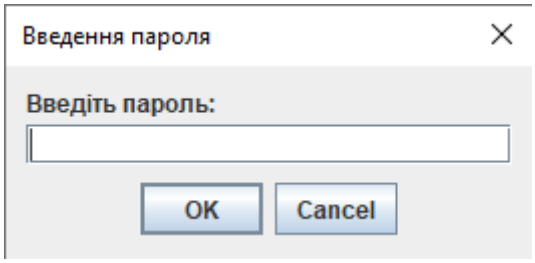

Рисунок 24. Вікно вводу пароля

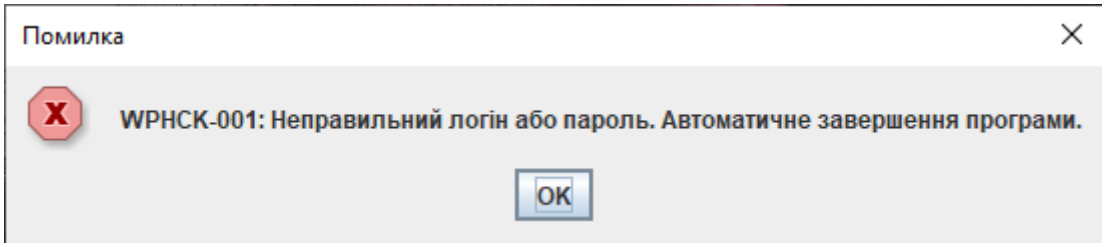

Рисунок 25. Повідомлення при невірному введені даних

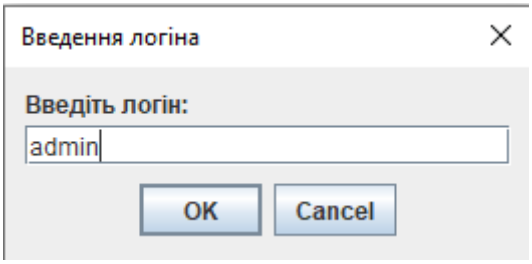

Рисунок 26. Введення корректного логіну

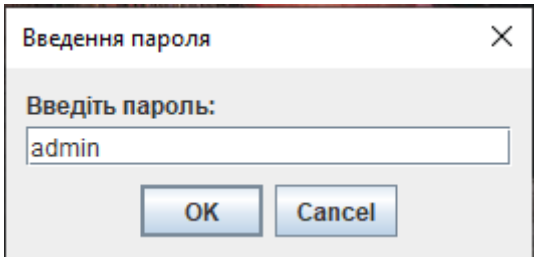

Рисунок 27. Введення корректного паролю

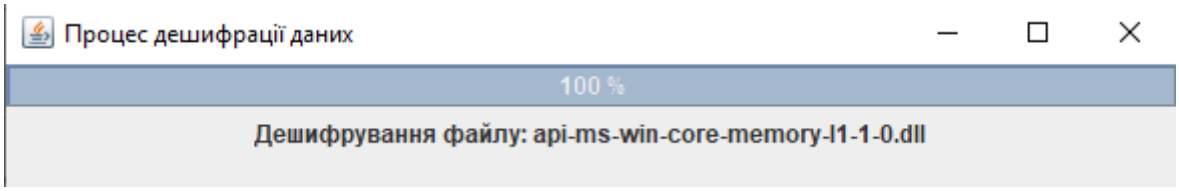

Рисунок 28. Інформаційне вікно дешифрації

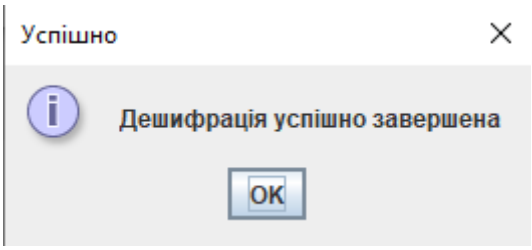

Рисунок 29. Повідомлення про успішну дешифрацію даних на носії

Рисунок 30. Перевірка дешифрації тексту в файлі decrypt.txt

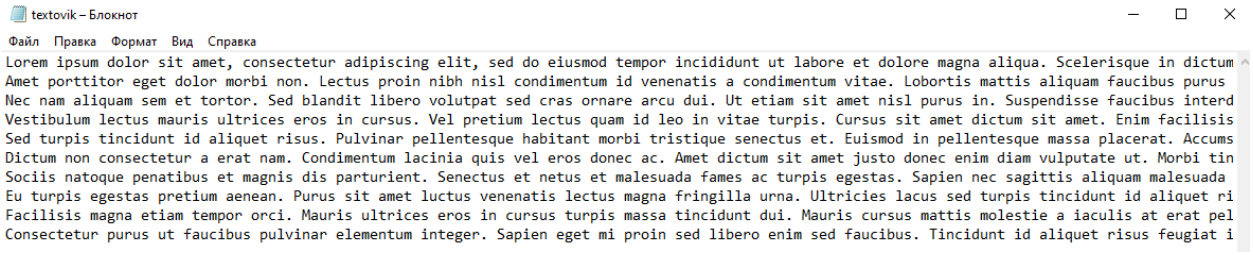

Рисунок 31. Перевірка дешифрації тексту в файлі textovik.txt

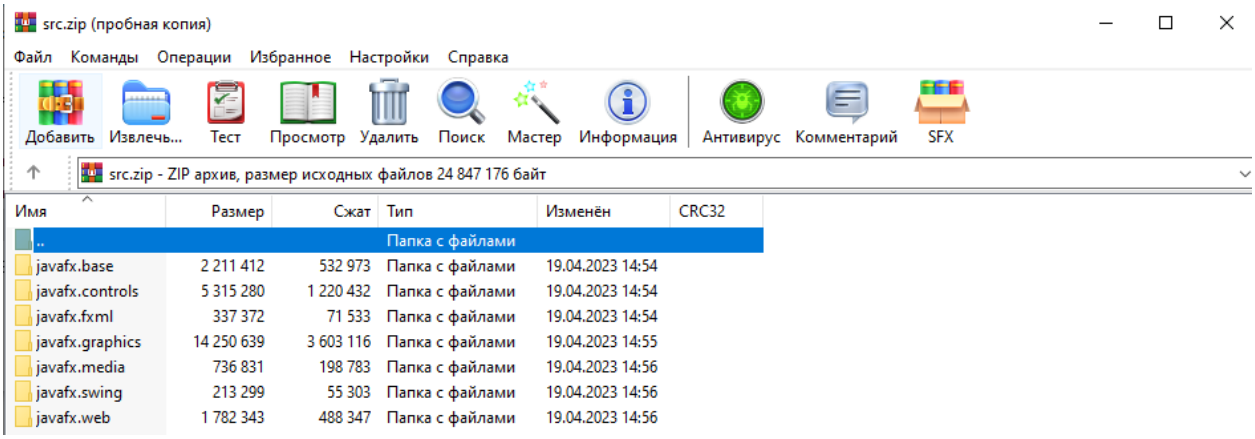

Рисунок 32. Перевірка відкриття архіву на вбудованому носії

Для підвищення рівня захисту додатку також було додано обфускацію коду на основі утиліти ProGuard за допомогою інструменту управління та автоматизації збірки проекту Maven та плагіну proguard-maven-plugin. Для коректної роботи було створено декілька текстових файлів із текстом для зміни назв змінних, методів і т.д. Окрім цього, створено файл, на основі якого плагін застосовує ті чи інші параметри для вихідного коду – конфігураційний файл proguard.pro.

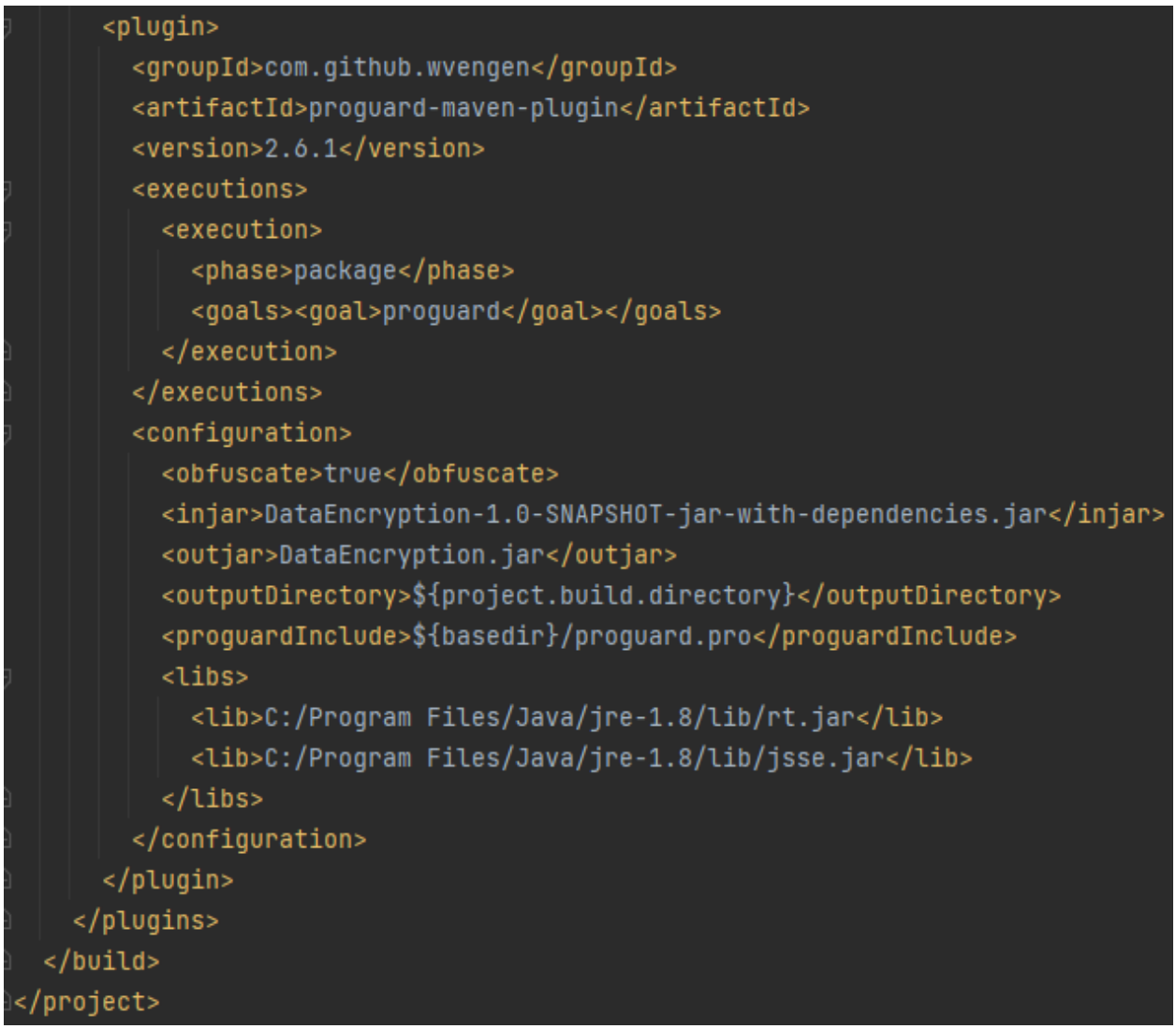

Рисунок 33. Використання плагіну ProGuard в pom.xml

Рисунок 34. Конфігураційний файл proguard.pro з параметрами для обфускації

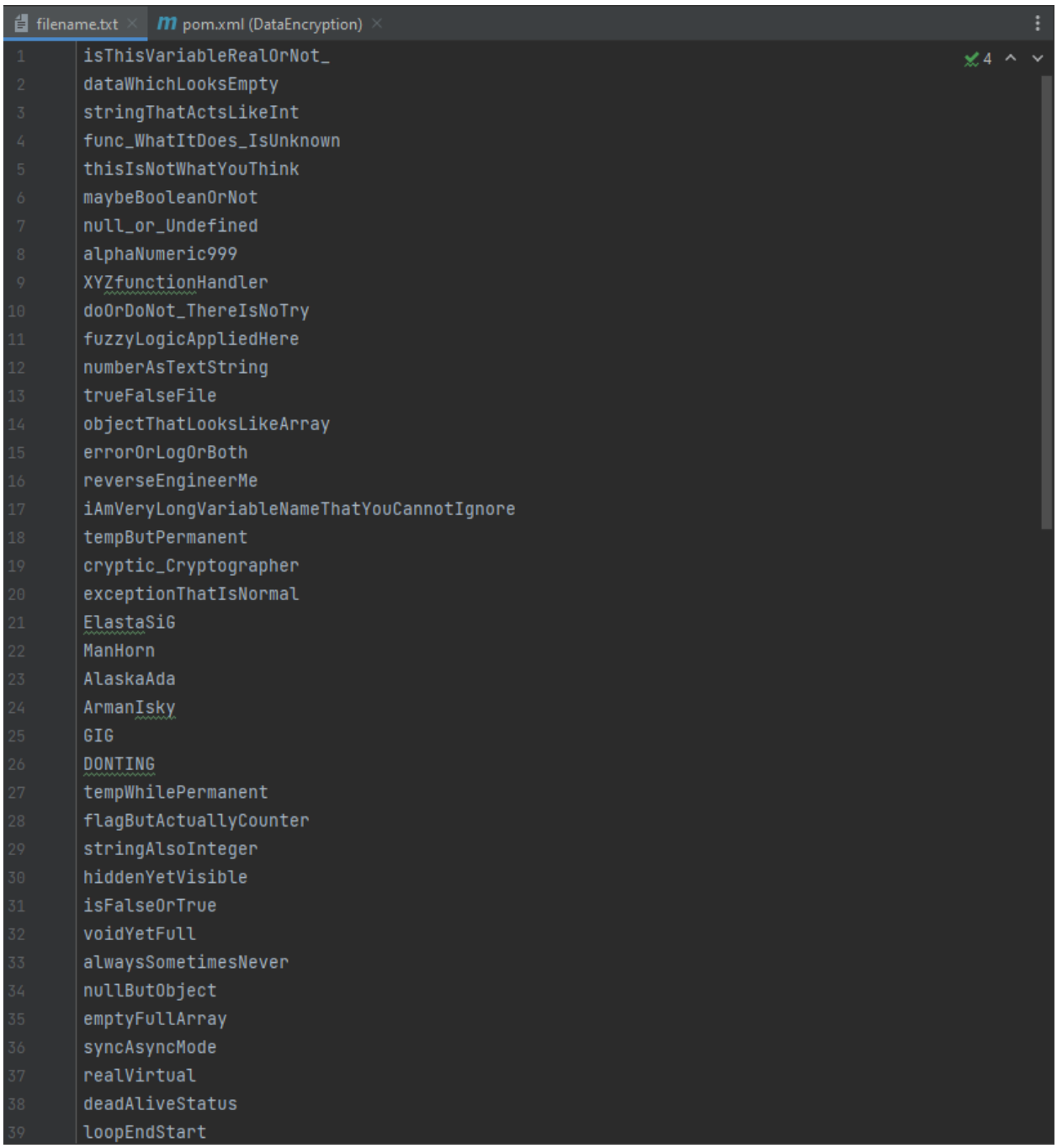

Рисунок 35. Приклад одного з декількох тестових файлів для заміни змінних

Для того, щоб обфускація була додана у додаток, необхідно зібрати проект за допомогою інструменту Maven, використовуючи команду «mvn clean package», що в свою чергу очистить папку з компільованими файлами (якщо вона до цього була створена) та перебудує проект. При цьому важливо правильно налаштувати конфігураційний файл, оскільки є можливість того, що проект не запуститься, тому необхідно дуже ретельно переглядати які

файли пропускати для обфускації, а які – ні. Після того, як DataEncryption.jar файл було створено, можливо переглянути його вміст та порівняти з іншим файлом DataEncryption-1.0-SNAPSHOT-jar-with-dependencies.jar, який використовувався як source-file перед обфускацією і містить всі оригінальні файли до їх обфусковування.

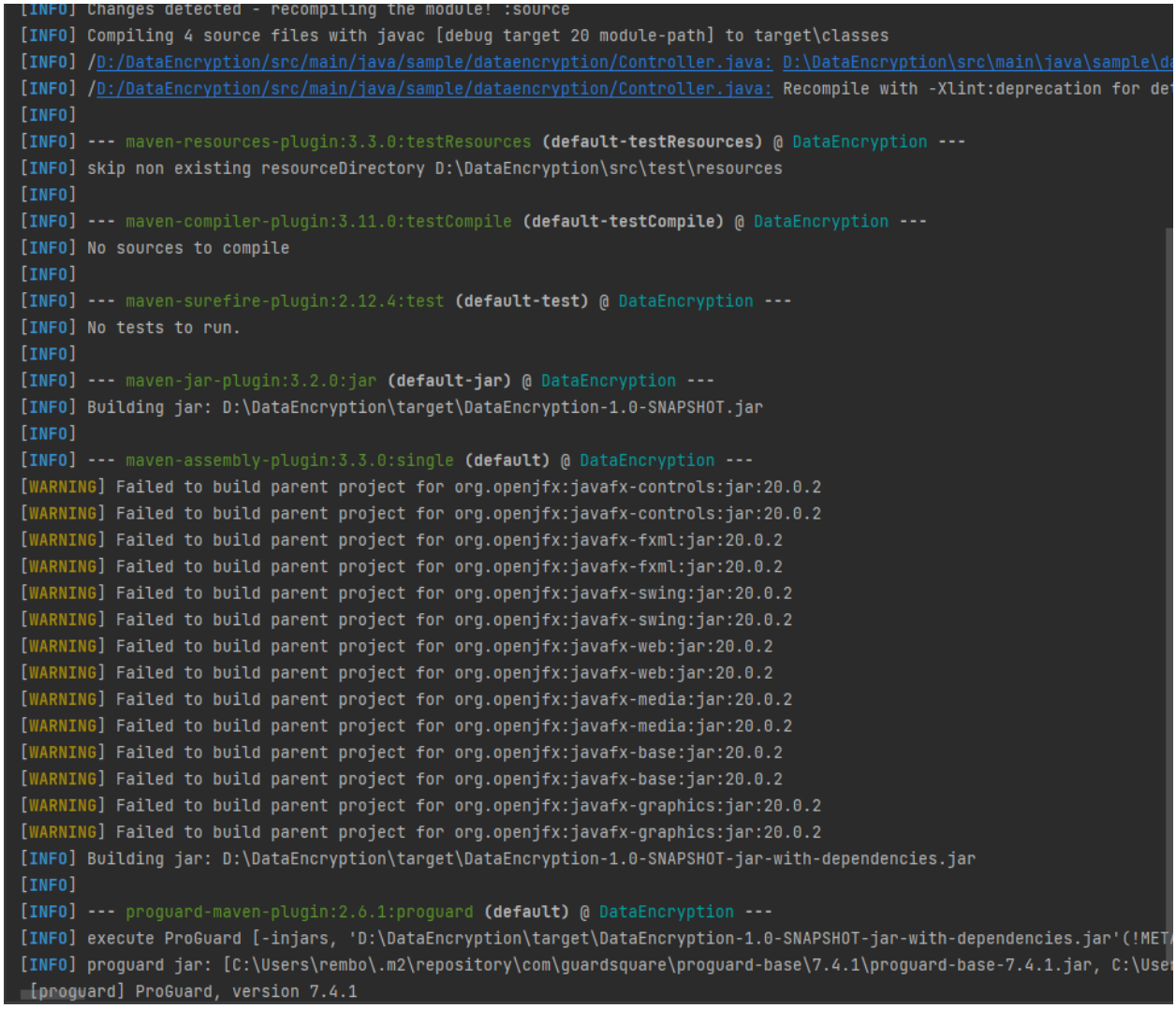

Рисунок 36. Збірка проекту за допомогою Maven

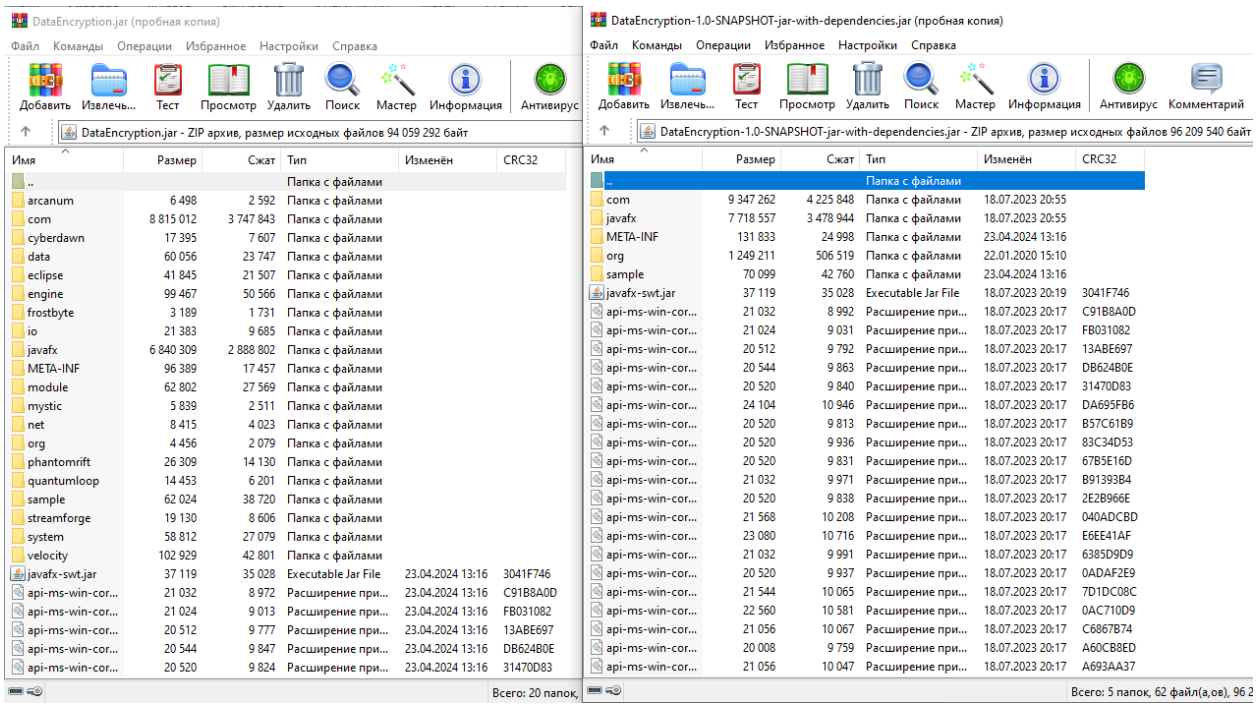

Рисунок 37. Порівняння обфускованого (ліворуч) та не обфускованого (праворуч) .jar файлів

5. Manual тестування додатку на мові Java

5.1. Інформаційний огляд Manual тестування

Manual тестування є важливою складовою процесу розробки програмного забезпечення, що полягає у ручному виконанні тестів для виявлення помилок та відхилень від очікуваної поведінки програми. У контексті Java-додатків, manual тестування вимагає ретельного аналізу функціональних та нефункціональних вимог до програми та перевірки їх відповідності.

Основні принципи Manual тестування:

- **Розуміння вимог**: Перед початком тестування важливо повністю розібратися у вимогах до програми. Це включає в себе функціональні вимоги (що програма повинна робити) та нефункціональні вимоги (якість, продуктивність, безпека тощо).
- **Планування тестів**: На основі розуміння вимог формується план тестування. План містить опис того, що буде тестуватися, які частини програми будуть використані, та які тести будуть виконані.
- **Створення тестових сценаріїв**: Для кожної частини програми розробляються тестові сценарії. Це послідовності кроків, які тестувальник повинен виконати для перевірки відповідності програми вимогам.
- **Виконання тестів**: Тестувальник вручну виконує тестові сценарії. Він аналізує реакцію програми на кожен крок тестового сценарію, щоб переконатися, що програма працює правильно та відповідає вимогам.
- **Документування тестових результатів**: Результати тестування документуються у вигляді звітів. Звіти містять інформацію про виконані тести, виявлені помилки та рекомендації щодо їх виправлення.

Техніки тестування:

- **Функціональне тестування**: Перевірка функцій програми для впевненості, що вона працює згідно з вимогами.
- **Регресійне тестування**: Перевірка того, чи не виникли нові помилки вже протестованих функцій після внесення змін в програму.
- **Інтерфейсне тестування**: Перевірка коректності та зручності інтерфейсу користувача.
- **Стрес-тестування**: Визначення меж продуктивності програми та виявлення слабких місць шляхом штучного перевантаження програми.[14][15]

5.2. Тестування розробленого додатку

Перед початком тестування програмного забезпечення необхідно виконати ряд підготовчих заходів для забезпечення ефективного та систематичного проведення тестування. У цьому розділі описано процес підготовки до тестування додатку криптозахисту даних на вбудованих носіях.

- **Розробка плану тестування:** Перед початком тестування було створено відповідний план тестування, який включає в себе перелік функціональних та нефункціональних вимог, а також опис тестових сценаріїв. План тестування був підготовлений на основі аналізу вимог до програми та стандартів криптографії.

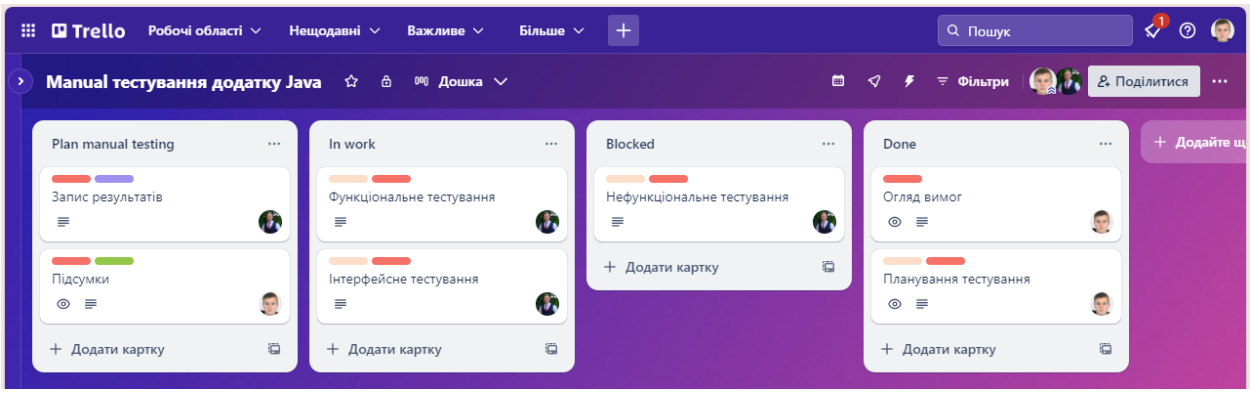

Рисунок 38. План тестування додатку

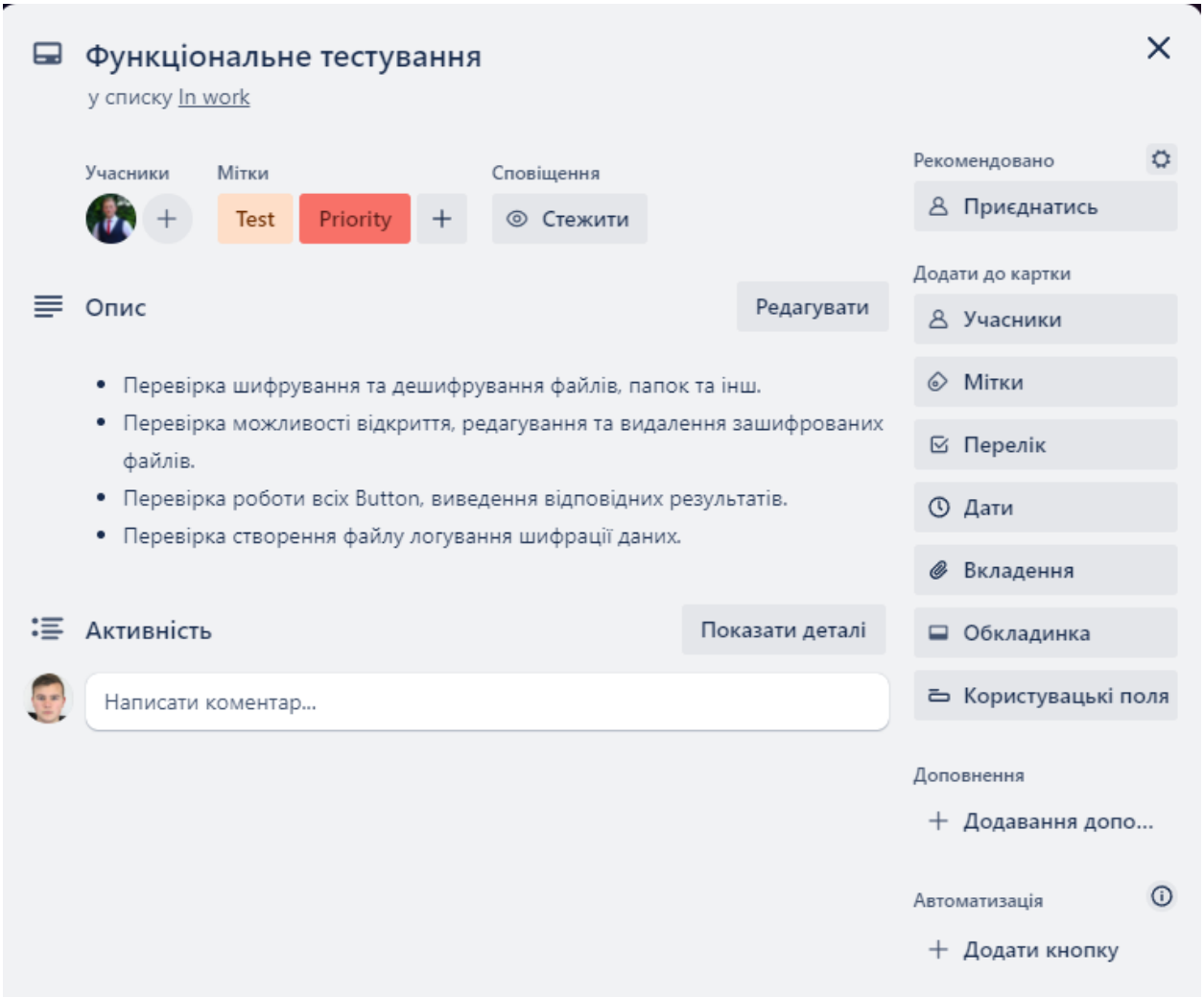

Рисунок 39. Опис одного з тестувань

- **Визначення учасників для тестування:** Учасниками процесу тестування є тестувальники, які мають досвід у проведенні тестування програмного забезпечення. Крім того, залучені розробники, які можуть надати додаткову підтримку під час тестування та виправлення виявлених помилок.
- **Підготовка тестового середовища:** Перед початком тестування було налаштовано тестове середовище, включаючи необхідне апаратне забезпечення (комп'ютери, вбудовані носії, тощо) та програмне забезпечення (операційну систему, необхідні бібліотеки, середовище розробки тощо).
- **Надання вихідного коду програми:** Для забезпечення ефективного тестування тестувальникам було надано вихідний код програми. Це

дозволило їм докладніше оцінити реалізацію програми, виявити можливі проблеми та виконати необхідні корекції.

- **Перевірка відповідності документації:** Перед початком тестування була перевірена відповідність документації програми її реальному функціоналу та можливостям. Це допомагає уникнути непорозумінь щодо очікуваної поведінки програми та правильної інтерпретації вимог.

Після визначення головних пунктів manual тестування, було проведено тестування інформаційної системи криптозахисту даних на вбудованих носіях на вразливості та неточності, та в кінцевому випадку не було знайдено критичних помилок.

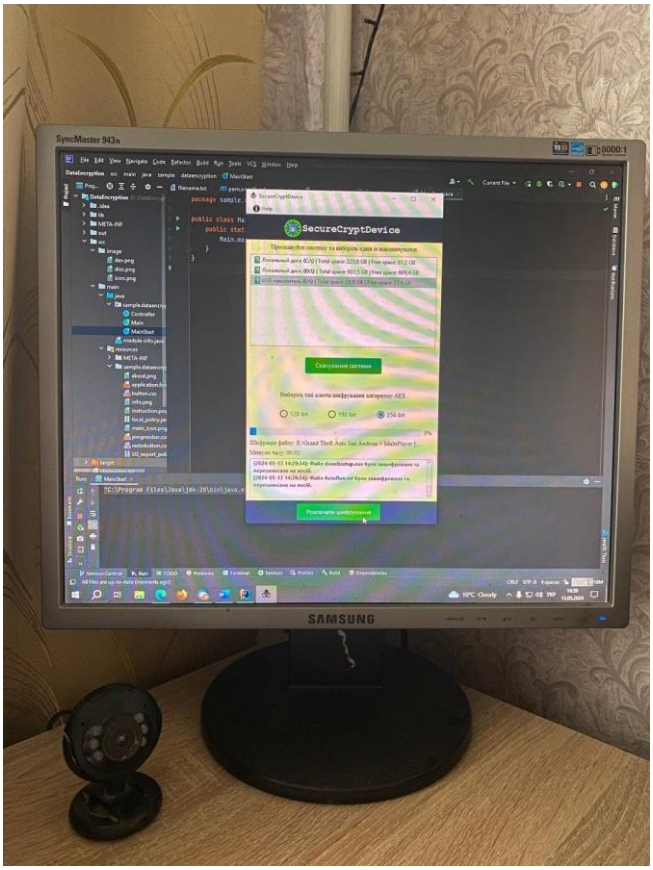

Рисунок 40. Проведення тестування одним із тестувальників

ВИСНОВОК

У даній роботі була досліджена інформаційна система криптозахисту даних на вбудованих носіях. Основними завданнями цього дослідження було розроблення та аналіз системи криптозахисту, що забезпечує надійний захист конфіденційної інформації на вбудованих носіях даних.

У процесі роботи були досягнуті наступні результати:

1. Розглянуто та проаналізовано основні принципи криптографічного захисту даних, включаючи симетричні та асиметричні шифри, хеш-функції, алгоритми електронного цифрового підпису.
2. Розроблено інформаційну систему криптозахисту даних на вбудованих носіях, яка використовує сучасні методи шифрування та підпису даних для забезпечення безпеки і конфіденційності інформації.
3. Проведено тестові дослідження ефективності розробленої системи на різних типах вбудованих носіїв даних та різних обсягах інформації.
4. Проведено manual тестування додатку для виявлення можливих вразливостей і недоліків в роботі програмного забезпечення та подальшого удосконалення.

Отже, результати дослідження свідчать про актуальність та ефективність використання інформаційних систем криптозахисту даних на вбудованих носіях. Далі цю роботу можна розширювати та вдосконалювати, додавати нові методи та технології криптозахисту, щоб підвищити рівень ефективності роботи інформаційної системи та безпеки даних на вбудованих носіях.

СПИСОК ЛІТЕРАТУРИ

1. Антонюк А.О. Теоретичні основи захисту інформації — Київ: НТУУ "КПІ", 2003. — 233 с.
2. Інформаційні технології. Криптографічний захист інформації. — Мінекономрозвитку України, 2016. — 228 с.
3. Криптографічний захист інформації і системи розподілу ключів. *Pidru4niki*. URL: https://pidru4niki.com/15070412/bankivska_sprava/kriptografichniy_zahist_informatsiyi_sistemi_rozpodilu_klyuchiv (дата звернення: 05.07.2023).
4. Daemen J., Rijmen V. The Design of RijndaeL: AES - The Advanced Encryption Standard — Springer, 2002. — 255 p.
5. JavaFX. *Axiom JDK*. URL: https://axiomjdk.ru/announcements/2022/08/18/gid-po-javafx-razrabatyvaem-gui-na-java/ (date of access: 05.07.2023).
6. Dea C. JavaFX 2.0: Introduction by Example — Apress, 2011. - 196 p. - ISBN: 1430242574
7. Weaver J.L. et al. Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology — Apress, 2012. - 620 p.
8. Security Developer Guide. *Oracle Help Center*. URL: https://docs.oracle.com/en/java/javase/11/security/java-cryptography-architecture-jca-reference-guide.html#GUID-EAB9FF73-4C69-4FD5-8A0C-5CF48211A859 (date of access: 05.07.2023).
9. Java Cryptography Extension. *Moved*. URL: https://docs.oracle.com/javase//1.5.0/docs/guide/security/jce/JCERefGuide.html (date of access: 05.07.2023).
10. Marius P. Techniques of Program Code Obfuscation for Secure Software. *ResearchGate*.

URL: https://www.researchgate.net/publication/235611093_Techniques_of_Program_Code_Obfuscation_for_Secure_Software (date of access: 23.04.2024).

11. Sengupta A. Frontiers in Securing IP Cores: Forensic detective control and obfuscation techniques. - The Institution of Engineering and Technology,2020. - 345 p.

12. Horváth M., Buttyán L. Cryptographic Obfuscation: A Survey. - Springer International Publishing : Springer, 2020. - 122 p.

13. Maven proguard Plug-In – Overview. *Site not found · GitHub Pages*. URL: https://wvengen.github.io/proguard-maven-plugin/ (date of access: 23.04.2024).

14. O'Regan O. Concise Guide to Software Testing — Springer, 2019. — 309 p.

15. Mayers G.J. The Art of Software Testing — Wiley, 2011 — 240 p.

Додаток А. Код класу Main.java програми

```
package sample.dataencryption;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;
import java.io.IOException;

public class Main extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("/sample/dataencryption/applic
ation.fxml"));
        AnchorPane root = fxmlLoader.load();
        Scene scene = new Scene(root, 450, 736);
        stage.setTitle("SecureCryptDevice");
        Image icon = new
Image("D:\\Учеба\\DataEncryption\\src\\image\\icon.png");
        stage.getIcons().add(icon);

        stage.setOnShown(event -> {
            stage.setMinWidth(stage.getWidth());
            stage.setMinHeight(stage.getHeight());
            stage.setMaxWidth(stage.getWidth());
            stage.setMaxHeight(stage.getHeight());
        });

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

Додаток Б. Код класу Controller.java програми

```
package sample.dataencryption;

import javafx.animation.AnimationTimer;
import javafx.animation.Timeline;
import javafx.application.Platform;
import javafx.concurrent.Task;
import javafx.fxml.FXML;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import org.apache.commons.lang3.StringUtils;
import javax.crypto.*;
import javax.swing.*;
import javax.swing.filechooser.FileSystemView;
import java.io.*;
import java.nio.file.Files;
import java.nio.file.StandardCopyOption;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.Optional;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Controller {
    @FXML
    private ListView<DriveInfo> flashDriveListView;
    @FXML
    private Label timeElapsedLabel;
    @FXML
    private Label pathLabel;
    @FXML
    private ProgressBar progressBar;
    @FXML
    private Label progressLabel;
    @FXML
    private RadioButton bit128;
    @FXML
    private RadioButton bit192;
    @FXML
    private RadioButton bit256;
```

```
    @FXML
    private AnchorPane rootPane;
    @FXML
    private TextArea logArea;
    @FXML
    private Menu help;
    private Timeline timeline;
    private Timeline timer;
    private static final String ALGORITHM = "AES";
    private SecretKey secretKey;
    private Thread encryptionThread;
    private Thread decryptionThread;
    private AnimationTimer animationTimer;
    private long encryptedBytes;
    private long totalBytesToEncrypt;


    @FXML
    public void initialize() {
        logArea.appendText("Logging....\n----------------------
----- \n");
        Platform.runLater(() -> {
            Stage stage = (Stage)
rootPane.getScene().getWindow();
            stage.setOnCloseRequest(e -> {
                if (encryptionThread != null &&
encryptionThread.isAlive()) {
                    e.consume();

                    Alert alert = new
Alert(Alert.AlertType.CONFIRMATION);
                    alert.setTitle("Зупинити шифрацію даних");
                    alert.setHeaderText(null);
                    alert.setContentText("Ви точно хочете
прервати шифрацію даних? Вся зашифрована інформація
залишиться!");
                    Optional<ButtonType> result =
alert.showAndWait();

                    if (result.isPresent() && result.get() ==
ButtonType.OK) {
                        encryptionThread.interrupt();
                        try {
                            encryptionThread.join();
                        } catch (InterruptedException ex) {
                            throw new RuntimeException(ex);
                        }
                        stage.close();
                    }
                }
            });
        });
```

```
    }

    /* --- Виведення всіх накопичувачів в системі --- */
    public class DriveInfo {
        private String name;
        private String description;
        private Image icon;
        private long totalSpace;
        private long freeSpace;

        public DriveInfo(String name, String description, long
totalSpace, long freeSpace, Image icon) {
            this.name = name;
            this.description = description;
            this.totalSpace = totalSpace;
            this.freeSpace = freeSpace;
            this.icon = icon;
        }

        public String getName() {
            return name;
        }

        public String getDescription() {
            return description;
        }

        public long getTotalSpace() {
            return totalSpace;
        }

        public long getFreeSpace() {
            return freeSpace;
        }

        public Image getIcon() {
            return icon;
        }

        @Override
        public String toString() {
            return " " + description + " (" + name + ")" + " |
Total space: " + totalSpace +
                    " | Free space: " + freeSpace;
        }
    }

    public class DriveInfoCell extends ListCell<DriveInfo> {
        private HBox content;
        private ImageView iconImageView;
        private Label nameLabel;
        private Label descriptionLabel;
```

```
        private Label spaceLabel;

        public DriveInfoCell() {
            iconImageView = new ImageView();
            iconImageView.setFitWidth(16);
            iconImageView.setFitHeight(16);

            nameLabel = new Label();
            nameLabel.setStyle("-fx-font-weight: bold");

            descriptionLabel = new Label();
            spaceLabel = new Label();

            content = new HBox(3);
            content.setAlignment(Pos.CENTER_LEFT);
            content.getChildren().addAll(iconImageView,
descriptionLabel, nameLabel, spaceLabel);
        }

        private String formatSize(long size) {
            final int unit = 1024;
            if (size < unit) return size + " B";
            int exp = (int) (Math.log(size) / Math.log(unit));
            String pre = "KMGT".charAt(exp - 1) + "";
            return String.format("%.1f %sB", size /
Math.pow(unit, exp), pre);
        }

        @Override
        protected void updateItem(DriveInfo driveInfo, boolean
empty) {
            super.updateItem(driveInfo, empty);
            if (empty || driveInfo == null) {
                setGraphic(null);
            } else {
                iconImageView.setImage(driveInfo.getIcon());
                nameLabel.setText("(" + driveInfo.getName() +
")");

descriptionLabel.setText(driveInfo.getDescription());
                String formattedTotalSpace =
formatSize(driveInfo.getTotalSpace());
                String formattedFreeSpace =
formatSize(driveInfo.getFreeSpace());
                spaceLabel.setText("| Total space: " +
formattedTotalSpace + " | Free space: " + formattedFreeSpace);
                setGraphic(content);
            }
        }
    }

    @FXML
```

```
    private void scanSystemClicked() throws
FileNotFoundException {
        flashDriveListView.getItems().clear();
        File[] drives = File.listRoots();
        FileSystemView fsv = FileSystemView.getFileSystemView();
        boolean flashDriveFound = false;

        for (File drive : drives) {
            String driveName = drive.getAbsolutePath();
            if (drive.isDirectory() && drive.canRead() &&
driveName.matches(".*[A-Za-z]:\\\\$")) {
                String driveDescription =
fsv.getSystemTypeDescription(drive);
                long totalSpace = drive.getTotalSpace();
                long freeSpace = drive.getFreeSpace();
                String iconPath =
"D:/DataEncryption/src/image/disc.png";
                Image icon = new Image(new
FileInputStream(iconPath));
                DriveInfo driveInfo = new DriveInfo(driveName,
driveDescription, totalSpace, freeSpace, icon);
                flashDriveListView.setCellFactory(listView ->
new DriveInfoCell());
                flashDriveListView.getItems().add(driveInfo);
                flashDriveFound = true;
            }
        }

        if (!flashDriveFound) {
            Alert alert = new
Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Не знайдено накопичувачів");
            alert.setHeaderText(null);
            alert.setContentText("Не вдалось знайти
накопичувачі, які під'єднані до комп'ютера");
            alert.showAndWait();
        }
    }

    /* --- Показ About Us в MenuBar --- */
    @FXML
    private void showAboutDialog() {
        Stage dialogStage = new Stage();
        dialogStage.initModality(Modality.APPLICATION_MODAL);
        dialogStage.initStyle(StageStyle.UTILITY);
        dialogStage.setResizable(false);

        Label headerLabel = new Label("About Us");
        headerLabel.setStyle("-fx-font-weight: bold; -fx-font-
size: 16px");

        Image image = new Image("file:src/image/dev.png");
```

```
        ImageView imageView = new ImageView(image);

        Label aboutLabel = new Label("All rights reserved. ©
This application was created by Infernum.");
        aboutLabel.setStyle("-fx-font-weight: bold");

        VBox vbox = new VBox(headerLabel, imageView,
aboutLabel);
        vbox.setAlignment(Pos.CENTER);
        vbox.setSpacing(10);

        Scene scene = new Scene(vbox, 400, 200);
        dialogStage.setScene(scene);
        dialogStage.showAndWait();
    }

    @FXML
    private void showInstructionDialog() {
        Stage dialogStage = new Stage();
        dialogStage.initModality(Modality.APPLICATION_MODAL);
        dialogStage.initStyle(StageStyle.UTILITY);
        dialogStage.setResizable(false);

        Label headerLabel = new Label("Instruction");
        headerLabel.setStyle("-fx-font-weight: bold; -fx-font-
size: 16px");

        Label instructionLabel = new Label(" 1. Спочатку
необхідно просканувати систему, щоб вивести список всіх " +
                "доступних накопичувачів.\n 2. Необхідно вибрати
відповідний тип ключа для шифрування: від 128 до 256 bit.\n " +
                "3. Необхідно натиснути на кнопку 'Розпочати
шифрування' та чекати поки шифрація буде завершена.\n " +
                "4. Після завершення шифрації - на накопичувачі
АВТОМАТИЧНО створюється дешифратор.");

        VBox vbox = new VBox(headerLabel, instructionLabel);
        vbox.setAlignment(Pos.CENTER);
        vbox.setSpacing(10);

        Scene scene = new Scene(vbox, 600, 175);
        dialogStage.setScene(scene);
        dialogStage.showAndWait();
    }

    /* --- Показ часу та прогресу шифрації в додатку --- */
    @FXML
    private void startTime() {
        animationTimer = new AnimationTimer() {
            private long startTime;

            @Override
```

```
            public void handle(long now) {
                long elapsedTime = (now - startTime) /
1_000_000_000;
                timeElapsedLabel.setText("Минуло часу: " +
formatTime(elapsedTime));
            }

            @Override
            public void start() {
                startTime = System.nanoTime();
                super.start();
            }
        };
        animationTimer.start();
    }

    private String formatTime(double time) {
        long minutes = (long) (time / 60);
        long seconds = (long) (time % 60);
        return String.format("%02d:%02d", minutes, seconds);
    }

    /* --- Генерування ключа та створення SecretKey --- */
    private SecretKey generateSecretKey() {
        try {
            KeyGenerator keyGenerator =
KeyGenerator.getInstance(ALGORITHM);

            if (bit128.isSelected()) {
                keyGenerator.init(128);
            } else if (bit192.isSelected()) {
                keyGenerator.init(192);
            } else if (bit256.isSelected()) {
                keyGenerator.init(256);
            }

            return keyGenerator.generateKey();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return null;
    }

    private SecretKey getSecretKey() {
        if (secretKey == null) {
            secretKey = generateSecretKey();
        }
        return secretKey;
    }

    /* --- Процес шифрування даних --- */
    private void encryptFlashDrive(File flashDrive) {
```

```
        if (flashDrive.isDirectory()) {
            File[] files = flashDrive.listFiles();
            if (files != null) {
                for (File file : files) {
                    if (file.isDirectory()) {
                        encryptFlashDrive(file);
                    } else {
                        if (!encryptionThread.isInterrupted()) {
                            totalBytesToEncrypt = file.length();
                            encryptFile(file);
                        } else {
                            break;
                        }
                    }
                }
            }
        }
    }

    private void encryptFile(File file) {
        try {
            Cipher cipher = Cipher.getInstance(ALGORITHM);
            cipher.init(Cipher.ENCRYPT_MODE, getSecretKey());

            File encryptedFile = File.createTempFile("temp",
null);
            encryptedBytes = 0;
            progressBar.setProgress(0);
            Platform.runLater(() -> {
                pathLabel.setText("Шифрація файлу: " +
StringUtils.abbreviate(file.getAbsolutePath(), 50));
            });
            try (InputStream inputStream = new
FileInputStream(file);
                 OutputStream outputStream = new
FileOutputStream(encryptedFile);
                 CipherOutputStream cipherOutputStream = new
CipherOutputStream(outputStream, cipher)) {

                byte[] buffer = new byte[65536];
                int bytesRead;

                while ((bytesRead = inputStream.read(buffer)) !=
-1) {
                    cipherOutputStream.write(buffer, 0,
bytesRead);
                    encryptedBytes += bytesRead;
                    updateProgressBar(progressBar,
encryptedBytes, totalBytesToEncrypt);
                }

                cipherOutputStream.flush();
```

```
            }

            Platform.runLater(() -> {
                pathLabel.setText("Перезапис файлу: " +
StringUtils.abbreviate(file.getAbsolutePath(), 50));
            });
            Files.move(encryptedFile.toPath(), file.toPath(),
StandardCopyOption.REPLACE_EXISTING);
            String currentTime =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-
dd HH:mm:ss"));
            String logMessage = "[" + currentTime + "]: Файл " +
file.getName() + " було " +
                    "зашифровано та перезаписано на носій.\n";
            Platform.runLater(() -> {
                logArea.appendText(logMessage);
            });
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /* --- Оновлення ProgressBar на основі кількості оброблених
даних файлу */
    private void updateProgressBar(ProgressBar progressBar, long
done, long total) {
        double progress = (double) done / total;
        progressBar.setProgress(progress);
        int percentage = (int) (progress * 100);
        Platform.runLater(() -> progressLabel.setText(percentage
+ "%"));
    }

    public static Alert methodAlert(Alert.AlertType type, String
title, String content, ButtonType... buttons) {
        Alert alert = new Alert(type);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.getButtonTypes().setAll(buttons);

        Stage alertStage = (Stage)
alert.getDialogPane().getScene().getWindow();
        alertStage.setAlwaysOnTop(true);

        return alert;
    }

    @FXML
    private void startCryptClicked() {
        DriveInfo selectedDrive =
flashDriveListView.getSelectionModel().getSelectedItem();
```

```
        progressBar.setProgress(0);
        logArea.clear();
        logArea.appendText("Logging....\n----------------------
----- \n");

        if (selectedDrive != null) {
            File flashDrive = new File(selectedDrive.getName());
            startTime();
            totalBytesToEncrypt = 0;
            encryptedBytes = 0;
            Task<Void> encryptionTask = new Task<>() {
                @Override
                protected Void call() {
                    encryptFlashDrive(flashDrive);
                    return null;
                }
            };

            encryptionTask.setOnSucceeded(event -> {
                animationTimer.stop();
                progressLabel.setText("100%");
                progressBar.setProgress(1.0);

                createDecryptor(selectedDrive.getName());

                ButtonType okButton = new ButtonType("OK",
ButtonBar.ButtonData.OK_DONE);
                Alert alert =
methodAlert(Alert.AlertType.INFORMATION, "Шифрування завершено",
                        "Шифрування було завершено!", okButton);
                alert.showAndWait();
                logArea.appendText("-------------------------
\nLog over, thanks for using our application!!!\n");

                ButtonType yesButton = new ButtonType("Так",
ButtonBar.ButtonData.YES);
                ButtonType noButton = new ButtonType("Ні",
ButtonBar.ButtonData.NO);
                Alert logAlert =
methodAlert(Alert.AlertType.CONFIRMATION, "Збереження логу",
                        "Чи хочете Ви зберігти лог роботи
програми в .txt? (лог роботи буде збережено на " +
                                "зашифрованому носії)",
yesButton, noButton);

                Optional<ButtonType> result =
logAlert.showAndWait();
                if (result.isPresent() && result.get() ==
yesButton) {
                    try {
                        File logFile = new
File(selectedDrive.getName() + File.separator +
```

```
"encryption_log.txt");
                        if (logFile.exists()) {
                            logFile.delete();
                        }
                        logFile.createNewFile();
                        FileWriter writer = new
FileWriter(logFile, true);
                        BufferedWriter bufferedWriter = new
BufferedWriter(writer);
                        bufferedWriter.write(logArea.getText());
                        bufferedWriter.close();

                        Alert successAlert =
methodAlert(Alert.AlertType.INFORMATION, "Збереження логу",
                                "Лог збережено успішно на
вбудованому носії", okButton);
                        successAlert.showAndWait();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            });

            encryptionThread = new Thread(encryptionTask);
            encryptionThread.start();
        } else {
            ButtonType okButton = new ButtonType("OK",
ButtonBar.ButtonData.OK_DONE);
            Alert alert = methodAlert(Alert.AlertType.WARNING,
"WPHCK-001: Помилка у виборі носія",
                    "Необхідно обрати носій, спробуйте ще раз!",
okButton);
            alert.showAndWait();
        }
    }

    /* --- Процес створення jar-файлу дешифратора даних на
накопичувачі */
    private String getDecodeSecretKeyToJar() {
        SecretKey secretKey = getSecretKey();
        byte[] encodedKey = secretKey.getEncoded();
        return Base64.getEncoder().encodeToString(encodedKey);
    }

    private void createDecryptor(String selectedDrive) {
        try {
            String flashDriveName = selectedDrive +
File.separator;
            String decodeKey = getDecodeSecretKeyToJar();

            String decryptorCode = "import javax.crypto.*;\n" +
                    "import java.io.*;\n" +
```

```
                    "import java.nio.file.Files;\n" +
                    "import java.nio.file.StandardCopyOption;\n"
+
                    "import java.util.Base64;\n" +
                    "import javax.crypto.spec.SecretKeySpec;\n"
+
                    "import javax.swing.*;\n" +
                    "import java.awt.*;\n" +
                    "import java.awt.event.WindowEvent;\n" +
                    "import java.awt.event.WindowAdapter;\n" +
                    "public class Decryptor {\n" +
                    "    private static final String ALGORITHM =
\"AES\";\n" +
                    "    private static SecretKey secretKey;\n"
+
                    "    private static JProgressBar
progressBar;\n" +
                    "    private static JLabel fileLabel;\n" +
                    "    private static JFrame frame;\n" +
                    "    private static void
createInformationFrame() {\n" +
                    "        frame = new JFrame(\"Процес
дешифрації даних\");\n" +
                    "
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);\n" +
                    "        frame.setSize(600, 100);\n" +
                    "
frame.setLocationRelativeTo(null);\n\n" +
                    "        JPanel centerPanel = new JPanel(new
FlowLayout(FlowLayout.CENTER));\n" +
                    "        fileLabel = new
JLabel(\"Дешифрування файлу: \");\n" +
                    "        centerPanel.add(fileLabel);\n" +
                    "        progressBar = new JProgressBar(0,
100);\n" +
                    "
progressBar.setStringPainted(true);\n" +
                    "        frame.add(progressBar,
BorderLayout.NORTH);\n" +
                    "        frame.add(centerPanel,
BorderLayout.CENTER);\n" +
                    "        frame.setVisible(true);\n" +
                    "    }\n" +
                    "    private static void updateProgress(int
progress, String fileName) {\n" +
                    "        SwingUtilities.invokeLater(() ->
{\n" +
                    "
progressBar.setValue(progress);\n" +
                    "
fileLabel.setText(\"Дешифрування файлу: \" + fileName);\n" +
                    "
```

```
progressBar.setValue(progress);\n" +
                    "        });\n" +
                    "    }\n" +
                    "    private static void decryptFile(File
file) {\n" +
                    "        try {\n" +
                    "            Cipher cipher =
Cipher.getInstance(ALGORITHM);\n" +
                    "
cipher.init(Cipher.DECRYPT_MODE, secretKey);\n" +
                    "            File decryptedFile =
File.createTempFile(\"temp\", null);\n" +
                    "            try (InputStream inputStream =
new FileInputStream(file);\n" +
                    "                 OutputStream outputStream
= new FileOutputStream(decryptedFile);\n" +
                    "                 CipherOutputStream
cipherOutputStream = new CipherOutputStream(outputStream,
cipher)) {\n" +
                    "                byte[] buffer = new
byte[65536];\n" +
                    "                int bytesRead;\n" +
                    "                long fileSize =
file.length();\n" +
                    "                long totalBytesRead = 0;\n"
+
                    "                while ((bytesRead =
inputStream.read(buffer)) != -1) {\n" +
                    "
cipherOutputStream.write(buffer, 0, bytesRead);\n" +
                    "                    totalBytesRead +=
bytesRead;\n" +
                    "                    int progress = (int)
((totalBytesRead * 100) / fileSize);\n" +
                    "
updateProgress(progress, file.getName());\n" +
                    "                }\n" +
                    "            }\n" +
                    "
Files.move(decryptedFile.toPath(), file.toPath(),
StandardCopyOption.REPLACE_EXISTING);\n" +
                    "        } catch (Exception e) {\n" +
                    "            e.printStackTrace();\n" +
                    "        }\n" +
                    "    }\n" +
                    "    public static void main(String[] args)
{\n" +
                    "        String username =
JOptionPane.showInputDialog(null, \"Введіть логін:\", \"Введення
логіна\", JOptionPane.PLAIN_MESSAGE);\n" +
                    "        String password =
JOptionPane.showInputDialog(null, \"Введіть пароль:\",
```

```
\"Введення пароля\", JOptionPane.PLAIN_MESSAGE);\n" +
                    "        if (isValidLogin(username,
password)) {\n" +
                    "            String flashDriveName = \"" +
flashDriveName + "\";\n" +
                    "            secretKey = new
SecretKeySpec(Base64.getDecoder().decode(\"" + decodeKey + "\"),
ALGORITHM);\n" +
                    "            File flashDrive = new
File(flashDriveName);\n" +
                    "            createInformationFrame();\n" +
                    "
decryptFlashDrive(flashDrive);\n" +
                    "
JOptionPane.showMessageDialog(null, \"Дешифрація успішно
завершена\", \"Успішно\", JOptionPane.INFORMATION_MESSAGE);\n" +
                    "            frame.dispose();\n" +
                    "        } else {\n" +
                    "
JOptionPane.showMessageDialog(null, \"WPHCK-002: Неправильний
логін або пароль. Автоматичне завершення програми.\",
\"Помилка\", JOptionPane.ERROR_MESSAGE);\n" +
                    "        }\n" +
                    "    }\n" +
                    "    private static boolean
isValidLogin(String username, String password) {" +
                    "    return username.equals(\"admin\") &&
password.equals(\"admin\");\n" +
                    "    }\n" +
                    "    private static void
decryptFlashDrive(File flashDrive) {\n" +
                    "        if (flashDrive.isDirectory()) {\n"
+
                    "            File[] files =
flashDrive.listFiles();\n" +
                    "            if (files != null) {\n" +
                    "                for (File file : files)
{\n" +
                    "                    if (file.isDirectory())
{\n" +
                    "
decryptFlashDrive(file);\n" +
                    "                    } else {\n" +
                    "
decryptFile(file);\n" +
                    "                    }\n" +
                    "                }\n" +
                    "            }\n" +
                    "        }\n" +
                    "}";
```

```
            File decryptorFile = new File("Decryptor.java");
            try (FileWriter fileWriter = new
FileWriter(decryptorFile)) {
                fileWriter.write(decryptorCode);
            }

            Process compileProcess =
Runtime.getRuntime().exec("javac Decryptor.java");
            compileProcess.waitFor();

            String manifestContent = "Main-Class: Decryptor\n";
            File manifestFile = new File("Manifest.txt");
            try (FileWriter fileWriter = new
FileWriter(manifestFile)) {
                fileWriter.write(manifestContent);
            }

            Process jarProcess = Runtime.getRuntime().exec("jar
cfm " + flashDriveName + "Decryptor.jar Manifest.txt
Decryptor.class");
            jarProcess.waitFor();

            File decryptorClass = new File("Decryptor.class");
            decryptorFile.delete();
            manifestFile.delete();
            decryptorClass.delete();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Додаток В. Код module-info.java програми

```
module sample.dataencryption {
    requires javafx.controls;
    requires javafx.fxml;
    requires javafx.web;
    requires javafx.graphics;
    requires javafx.base;
    requires javafx.media;
    requires jdk.crypto.ec;
    requires java.desktop;
    requires org.apache.commons.lang3;
    requires java.compiler;

    opens sample.dataencryption to javafx.fxml;
    exports sample.dataencryption;
}
```

Додаток Г. Код pom.xml програми

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>sample</groupId>
  <artifactId>DataEncryption</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>DataEncryption</name>

  <properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    <junit.version>5.9.2</junit.version>      </properties>

  <dependencies>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-controls</artifactId>
      <version>20.0.2</version>
    </dependency>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-fxml</artifactId>
      <version>20.0.2</version>
    </dependency>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-swing</artifactId>
      <version>20.0.2</version>
    </dependency>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-web</artifactId>
      <version>20.0.2</version>
    </dependency>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-media</artifactId>
      <version>20.0.2</version>
    </dependency>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-base</artifactId>
      <version>20.0.2</version>
    </dependency>
```

```
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-graphics</artifactId>
      <version>20.0.2</version>
    </dependency>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.12.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.2.0</version>
        <configuration>
          <archive>
            <manifest>
              <addClasspath>true</addClasspath>

<mainClass>sample.dataencryption.MainStart</mainClass>
            </manifest>
          </archive>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-resources-plugin</artifactId>
        <version>3.3.0</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.11.0</version>
        <configuration>
          <source>20</source>
          <target>20</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-maven-plugin</artifactId>
        <version>0.0.8</version>
        <executions>
          <execution>
            <id>default-cli</id>
            <configuration>

<mainClass>sample.dataencryption.MainStart</mainClass>
            </configuration>
```

```
          </execution>
        </executions>
      </plugin>
      <plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>3.3.0</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>single</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
          </descriptorRefs>
          <archive>
            <manifest>

<mainClass>sample.dataencryption.MainStart</mainClass>
            </manifest>
          </archive>
        </configuration>
      </plugin>
      <plugin>
        <groupId>com.github.wvengen</groupId>
        <artifactId>proguard-maven-plugin</artifactId>
        <version>2.6.1</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals><goal>proguard</goal></goals>
          </execution>
        </executions>
        <configuration>
          <obfuscate>true</obfuscate>
          <injar>DataEncryption-1.0-SNAPSHOT-jar-with-
dependencies.jar</injar>
          <outjar>DataEncryption.jar</outjar>

<outputDirectory>${project.build.directory}</outputDirectory>

<proguardInclude>${basedir}/proguard.pro</proguardInclude>
          <libs>
            <lib>C:/Program Files/Java/jre-1.8/lib/rt.jar</lib>
            <lib>C:/Program Files/Java/jre-
1.8/lib/jsse.jar</lib>
          </libs>
        </configuration>
      </plugin>
```

```
    </plugins>
  </build>
</project>
```