Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Методи та засоби стеганографічного захисту конфіденційної інформації»

Виконав: студент 2-го курсу групи 1ПІ-20м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Білоконь Владислав Васильович
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Майданюк В.П.
(прізвище та ініціали)

«_____»_____2021р.

Опонент: д.т.н.,проф.каф.КН Василевський О.М.
(прізвище та ініціали)

«_____»_____2021р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н.,проф. Романюк О. Н.
(прізвище та ініціали)

«____»_____2021р.

ВНТУ – 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 13 » вересня 2021 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Білоконю Владиславу Васильовичу

1. Тема роботи – методи та засоби стеганографічного захисту конфіденційної інформації.

Керівник роботи: Майданюк Володимир Павлович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від « 24 » вересня 2021 р. № 277.

2. Строк подання студентом роботи
1 грудня 2021 р.

3. Вихідні дані до роботи : контейнер – файли зображень у форматах без ущільнення даних (BMP, TIFF, PNG та інші); спосіб приховування - метод LSB (Least Significant Bit); додатковий захист – шифрування даних перед приховуванням та використання технологогії JSON Web Token (JWT); мова програмування – C#; середовище розробки – Microsoft Visual Studio; кількість рівнів захисту - 3.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану питання та постановка задач дослідження; розробка методу, моделі та алгоритмів системи; розробка програмних засобів; тестування програми; економічна частина; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): мета, об'єкт, предмет дослідження, завдання дослідження, наукова новизна отриманих результатів, приховування інформації в цифрових контейнерах, порівняльний аналіз аналогів, аналіз існуючих методів кодування, порівняльний аналіз аналогів, стенографія для кодування даних,

Стеганографічний захист даних в зображенні, розробка алгоритмів роботи програми, загальний алгоритм роботи програми, порівняльний аналіз середовищ розробки, розробка структури інтерфейсу для програмного засобу, розробка блоку шифрування даних, розробка блоку стеганографічного захисту, стек технологій, мінімальна конфігурація, тестування програмного забезпечення.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Майданюк В.П., к.т.н, доцент кафедри ПЗ		
5	Ратушняк О. Г., к.т.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання 14 вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання і вибір методів його вирішення	14.09.2020 р. - 01.10.2020 р.	Вик.
2	Аналіз існуючих аналогів та постановка задач дослідження	02.10.2020 р.– 10.10.2020 р.	Вик.
3	Розробка методу та моделей системи	11.10.2020 р. - 20.10.2020 р.	Вик.
4	Розробка структур і алгоритмів програмного продукту	21.10.2020 р. - 5.11.2020 р.	Вик.
5	Розробка програмного забезпечення	6.11.2020 р. - 15.11.2020 р.	Вик.
6	Тестування розробленого програмного продукту	16.11.2020 р. - 20.11.20202 р.	Вик.
7	Економічна частина	21.11.2020 р. - 28.11.2020 р.	Вик.
8	Оформлення матеріалів до захисту	29.11.2020 р. - 30.11.2020 р.	Вик.

Студент _____ **Білоконь В. В.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Майданюк В. П.**
(підпис) (прізвище та ініціали)

Рецензент магістерської кваліфікаційної роботи _____ **Василевський О. М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК. 004.4:004.92

Білоконь В. В. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2021. с.

На укр. мові. Бібліогр.: 44 назв; рис.: 34; табл. 9.

Магістерська кваліфікаційна робота спрямована на дослідження методів і засобів стеганографічного захисту даних.

У магістерській кваліфікаційній роботі удосконалено методи і засоби стеганографічного захисту даних. Запропоновано метод приховування інформації в зображенні, як в контейнері для зашифрованого файлу з додатковим захистом. Розроблено модель шифрування даних авторським алгоритмом, з додаванням додаткового ключа для підвищення безпеки та збільшення часу на розшифрування повідомлення користувачем без потрібного доступу. Запропоновано принцип стеганографічного проховування в зображення, який зменшує можливість знаходження прихованої інформації. Розроблено алгоритму роботи програмного забезпечення для стеганографічного захисту даних.

Розробка виконана мовою програмування C# під операційну систему MS Windows у середовищі розробки MS Visual Studio 2019 Community. Працеспроможний додаток показує простоту процесу та швидкість опрацювання даних, що забезпечує всі вимоги щодо захисту даних. Характеризується надійністю та стабільністю роботи, а також широким спектром можливих застосувань.

Отримані в ході магістерської кваліфікаційної роботи результати можна використати для побудови систем приховування інформації.

Ключові слова: шифрування, приховування, стеганографічне приховування, стенографічне шифрування , технології приховування інформації.

ABSTRACT

Bilokon VV Master 's thesis on specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2021. p.

In Ukrainian language. Bibliogr .: 44 titles; fig .: 34; table 9.

The master's qualification work is aimed at researching methods and means of steganographic data protection.

Methods and means of steganographic data protection have been improved in the master's qualification work. A method of hiding information in the image as in a container for an encrypted file with additional protection is proposed. A model of data encryption by the author's algorithm has been developed, with the addition of an additional key to increase security and increase the time for the user to decrypt the message without the required access. The principle of steganographic storage in the image is proposed, which reduces the possibility of finding hidden information. The algorithm of software for steganographic data protection is developed.

The development is performed in the C # programming language for the MS Windows operating system in the MS Visual Studio 2019 Community development environment. The workable application shows the simplicity of the process and speed of data processing, which provides all the requirements for data protection. It is characterized by reliability and stability, as well as a wide range of possible applications.

The results obtained during the master's qualification work can be used to build systems of information concealment.

Key words: encryption, concealment, steganographic concealment, stenographic encryption, information concealment technologies.

ЗМІСТ

ВСТУП 10

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ 15

1.1 Аналіз сучасного стану питання 15
1.2 Порівняльний аналіз аналогів 17
1.3 Аналіз існуючих методів кодування 21
1.4 Аналіз методів стенографії для кодування даних в зображення 23
1.5 Аналіз існуючих методів стеганографічного захисту даних в зображенні 24
1.6 Аналіз існуючої ефективності роботи з JWT-токенами 30
1.7 Постановка задач роботи 35
1.8 Висновки 36

2 РОЗРОБКА МЕТОДУ, МОДЕЛІ ТА АЛГОРИТМІВ СИСТЕМИ 37

2.1 Аналіз архітектурних рішень у розробці програмних систем 37
2.2 Розробка блоку шифрування даних 42
2.3 Розробка блоку стеганографічного захисту 45
2.4 Розробка алгоритмів роботи програми 49
2.5 Аналіз продуктивності додаткового шифрування даних 52
2.6 Висновки 54

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ 55

3.1 Варіантний аналіз і обґрунтування вибору засобів реалізації програмного засобу. 55
3.2 Вибір середовища розробки 57
3.3 Розробка класів для стеганографічного захисту 58
4.4Розробка JWT для забезпечення ключа доступу при стеганографічному захисті
3.5 Висновки 66

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ 67

4.1 Аналіз методів та засобів тестування 67
4.2 Тестування програмного продукту 68
4.3 Розробка інструкції користувача 73
4.4 Висновки 75

5 ЕКОНОМІЧНА ЧАСТИНА 76

5.1 Оцінювання комерційного потенціалу розробки 76
5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи 79

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки 84
5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності 85
5.5 Висновки 88
ВИСНОВКИ 89
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 91
ДОДАТКИ 94
ДОДАТОК А (обов'язковий). Технічне завдання 95
ДОДАТОК Б (обов'язковий). Протокол перевірки роботи 99
ДОДАТОК В (обов'язковий). Лістинг програмних модулів 100
ДОДАТОК Г (обов'язковий). Ілюстративний частина до захисту роботи 116

ВСТУП

Обґрунтування вибору теми дослідження. Безпечне зберігання даних завжди є важливим питанням для багатьох організацій. Від нього залежить як безпека зберігання технології виробництва якогось товару, так і передача важливих повідомлень. Потреба безпеки інформаційних ресурсів, зберігання важливих даних та передача повідомлень в закодованому вигляді - це важливі заходи, які можуть зберегти не тільки важливі дані, а й навіть життя людей. Адже розвинуте промислове шпигунство лише одна з багатьох причин, по яким люди намагаються вберегти інформацію від потрапляння в чужі руки. Конкуренти часто мають бажання отримати незаконний доступ до інформації, яка є комерційною таємницею, що може серйозно впливати на розвиток виробництва та конкурентноспроможність. Тому питання по захисту інформації є вкрай важливими. Захист інформації ведеться для підтримки таких властивостей інформації як:

– Цілісність, неможливість модифікації інформації неавторизованим користувачем.

– Конфіденційність, інформація не може бути отримана неавторизованим користувачем.

– Доступність, полягає в тому, що авторизований користувач може використовувати інформацію відповідно до правил, встановлених політикою безпеки не очікуючи довше заданого (прийнятного) інтервалу часу.

Відповідно до властивостей, виділяють такі загрози безпеці інформації:

– загрози цілісності: знищення; модифікація;

– загрози доступності: блокування; знищення;

– загрози конфіденційності: несанкціонований доступ (НСД); витік; розголошення.

Кожен вид захисту інформації забезпечує окремі аспекти інформаційної безпеки:

– технічний – забезпечує обмеження доступу до носія повідомлення апаратно-технічними засобами (антивіруси, фаєрволи, маршрутизатори, токіни, смарт-карти тощо): попередження витоку по технічним каналам; попередження блокування;

– інженерний – попереджує руйнування носія внаслідок навмисних дій або природного впливу інженерно-технічними засобами (сюди відносять обмежуючі конструкції, охоронно-пожежна сигналізація);

– криптографічний – попереджує доступ за допомогою математичних перетворень повідомлення: попередження несанкціонованої модифікації; попередження несанкціонованого розголошення;

– організаційний – попередження доступу на об'єкт інформаційної діяльності сторонніх осіб за допомогою організаційних заходів (правила розмежування доступу).

Важливим фактором в збереженні безпеки для інформації, що передається чи зберігається, є приховування самого факту знаходження там таємної інформації. Стеганографія (у перекладі з грецького - «тайнопис»; steganos – таємниця, секрет; graphy – запис) на відміну від криптографії і розглядає методи приховування не лише інформації, але і самого факту її передачі. З цією метою приховувана інформація деякого об'єму вкладається в яке-небудь загальнодоступне повідомлення, наприклад в текст, зображення, звук та інше. При вкладенні приховуваного повідомлення відбувається як би його розсіювання за всім обсягом основного або покриваючого повідомлення. При цьому останнє не повинне зазнавати видимих змін (спотворень), які можуть бути легко виявлені сторонніми особами.

Якщо криптографічні методи, швидше за все, слід відносити до оборонних інструментів інформаційного протиборства, то стеганографія через свою специфіку може стати в XXI столітті технологією створення нової високоефективної інформаційної зброї. Хоча такий поділ є досить умовним.

Тому обрана тема магістерської кваліфікаційної роботи є повністю актуальною та доречною для дослідження.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення рівня захисту даних від несанкціонованого доступу за рахунок додаткового шифрування даних перед приховуванням.

Основними задачами дослідження є:

- провести аналіз існуючих методів і засобів стеганографічного захисту даних;
- запропонувати нові методи або удосконалити існуючі методи підвищення безпеки збереження даних в контейнері за допомогою додаткового шифрування;
- розробити програмні компоненти та систему приховування даних на основі запропонованих методів;
- провести експериментальні дослідження розроблених засобів приховування даних.
- обґрунтувати економічну доцільність розробки системи приховування даних.

Об'єкт дослідження – процес стеганографічного захисту даних від несанкціонованого доступу.

Предмет дослідження – методи та програмні засоби для стеганографічного захисту даних від несанкціонованого доступу.

Методи дослідження. У процесі досліджень використовувались: теорія інформації та кодування, теорія алгоритмів, теорія криптографії, методи стеганографії для розробки моделей та методів приховування даних у файлах зображень; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод стеганографічного захисту даних, у якому, на відміну від існуючих, використано додаткове шифрування даних методом гамування, що дозволило підвищити рівень захисту даних у 2 рази.

2. Запропоновано нову схему передачі ключа шифрування, яка відрізняється від відомої використанням технології JWT, що дозволяє зберігати ключ шифрування в контейнері у прихованому і зашифрованому вигляді.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби приховування даних в файлах зображень.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованій праці, опублікованій у співавторстві, автору належать такі результати: загальна схема приховування даних у файлах зображень [1]; модель роботи програми приховування даних [2]; комп'ютерна програма для шифрування даних перед приховуванням [3].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення магістерській кваліфікаційній роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях:

- Використання інформаційних та комунікаційних технологій в сучасному цифровому суспільстві (4-5 червня 2020 р., м. Херсон)

- L Науково-технічна конференція підрозділів Вінницького національного технічного університету (2021);

- Міжнародної науково-практична Інтернет конференція «Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова» (9-10 листопада 2021 р., Суми/Вінниця).

Публікації. Основні результати досліджень опубліковано в 3 наукових працях у збірниках матеріалів конференцій: Використання інформаційних та комунікаційних технологій в сучасному цифровому суспільстві (4-5 червня 2020 р., м. Херсон)[1]; L Науково-технічна конференція підрозділів Вінницького національного технічного університету (2021) [2]; Міжнародної науково-практична Інтернет конференція «Електронні інформаційні ресурси: створення, використання, доступ. Пам'яті Олексія Петровича Стахова» (9-10 листопада 2021 р., Суми/Вінниця) [3].

Структура та обсяг роботи. Робота містить вступ, 5 розділів, висновки, перелік посилань, додатки. В першому розділі розглянуто загальний стан питання стеганографічного захисту інформації від несанкціонованого доступу, сформульовано задачі роботи. В другому розділі обґрунтовано вибір засобів для вирішення поставленої задачі, розроблено алгоритми шифрування, дешифрування та стеганографічного захисту, а також інтерфейс користувача. У третьому розділі обґрунтовано вибір мови програмування та середовища розробки, розроблено програмний засіб для стеганографічного захисту даних. У четвертому розділі виконано тестування програми приховування даних у файлах зображень, розроблено інструкцію користувача. У п'ятому розділі обгрунтовано економічні характеристики розробки.

Перелік посилань містить 25 джерел. У додатках міститься технічне завдання на роботу, лістинг коду та ілюстративна частина до захисту роботи.

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз сучасного стану питання

Витік якоїсь комерційної інформації завжди призводить до небажаних наслідків, а буває стає причиною великих проблем в компанії. Витрати від такої діяльності конкурентів, що незаконно використовують методи шпигунства своїх цілях, є дуже великими.

Таким чином, питання і задачі безпеки будь-яких видів, навіть на малому виробництві, доводиться вирішувати кожного разу розглядаючи всіляких аспектів нашої людської діяльності.

Проте, як бачимо, абсолютно всі види безпеки тісно переплетені з інформаційною безпекою і кажучи, більш того, їх неможливо повноцінно забезпечити без забезпечення інформаційної безпеки.

Інформація – це збірка даних, відомості про осіб, дії зроблені ними, предмети, їх властивості, факти, події, записи місць, процеси та явища, незалежно від самої форми їх представлення [4].

Захист інформації – являє собою такий комплекс ряду заходів, що проведено із простою метою зниження та запобігання до безпечного рівня всіх можливостей витікання, розкрадання, поширення, втрати, знищення, підробки, перекручування або блокування інформації.

Над будь-якою інформацією, будь-якого стану зберігання, можна проводити доволі різноманітні дії, що дають можливість проводити, незаконні чи законні, різноманітні позитивні або негативні заходи для маніпуляції інформацією в потрібному ракурсі для маніпулюючого.

Блокування інформації, один з варіантів, користувач не може дістати доступ до інформації; за відсутності доступу сама інформація не втрачається.

Порушення цілісності являє собою втрату чи вихід з ладу носія. Спотворення являє собою порушення смислової значимості та порушення логічної послідовності та втрат достовірності інформації, так щоб наявна інформація не відповідала об’єкту в реальному стані.

Порушення конфіденційності настає тоді, коли з інформацією має змогу чи вже ознайомлюються особа, яка не повинна мати доступу до цих даних. Рівень допуску до такої інформації визначає її власник і законодавство. Порушення конфіденційності також може відбутися через неправильної роботи системи обмеження доступу або наявності побічного каналу доступу, який навмисно чи випадково створений.

Несанкціоноване тиражування, під даним захистом розуміється захист авторських прав і прав власності на інформацію. Для унеможливлення продачі матеріалів третіми особам для незаконного збагачення і без дозволу на те, самого власника інформації.

Автоматизована система – являє собою таку організаційно-технічну систему, що об'єднує в собі оброблювану інформацію, фізичне середовище, персонал та обчислювальну систему [5].

Захист інформації в автоматизованої систем, інакше information security, чи computer system security – діяльність, що спрямована на забезпечення безпеки оброблюваної в автоматизованій системі інформації та автоматизованій системі у цілому і дозволяє запобігти або ускладнити можливість реалізації загроз, а також знизити величину потенційних збитків внаслідок реалізації загроз.

Комплексна система захисту інформації – сукупність певних організаційних, інженерних і програмно-апаратних заходів, що забезпечують потрібний захист інформації в розробленій автоматизованій системі.

Загроза – представляє собою потенційно можливу дію, процес, подію або явище, яке може привести до створення збитку інтересам життєдіяльності

певної юридичної або фізичної особи. Реалізацією певних загрози є саме порушення роботи системи вибраних елементів.

В свою чергу загрози поділяються на природні та штучні. Природні загрози – загрози, бувають викликані дією різноманітних фізичних процесів на автоматизовану систему або низки стихійних природних явищ, які незалежні від людини. До таких відносяться: магнітні бурі, опади, радіоактивне випромінювання, стихійні лиха тощо. Також загрози можуть бути виражені у технічному характері, вони пов'язані з надійністю низки технічних засобів оброблення інформації і підсистем забезпечення автоматизованих систем.

Штучні загрози – такі, що викликані діяльністю людини чи доведене до цього наслідками людської діяльності в світі. Вони поділяються на такі види:

- ненавмисні – загрози, які пов'язані з абсолютно випадковими діями людей, через своє незнання, халатність в діях, просту цікавість, але зовсім без злого наміру і без мотивів.
- навмисні – дії людини, здійснюють умисні дії для дезорганізації роботи інформаційної системи, виведення її з ладу, для незаконного проникнення в систему і несанкціонованого доступу до інформації.

1.2 Порівняльний аналіз аналогів

Серед програм для шифрування даних можна виділити наступні:

Recolition.exe (рисунок 1.2) – проста програма для шифрування малих файлів, яка використовує алгоритм AES.

Алгоритм Advanced Encryption Standard, також відомий під назвою Rijndael – симетричний алгоритм блочного шифрування, де розмір блока 128 біт, ключ 128/192/256 біт, фіналіст конкурсу AES і прийнятий як американський стандарт шифрування урядом США. Вибір припав на AES з розрахуванням на широке використання і активний аналіз алгоритму, як це було із його попередником, DES. Станом на 2009 рік AES був одним із

найпоширеніших алгоритмів симетричного шифрування. Recolition шифрує файли і видає файл з розширенням .encrypt, а потім розшифровує їх назад [6].

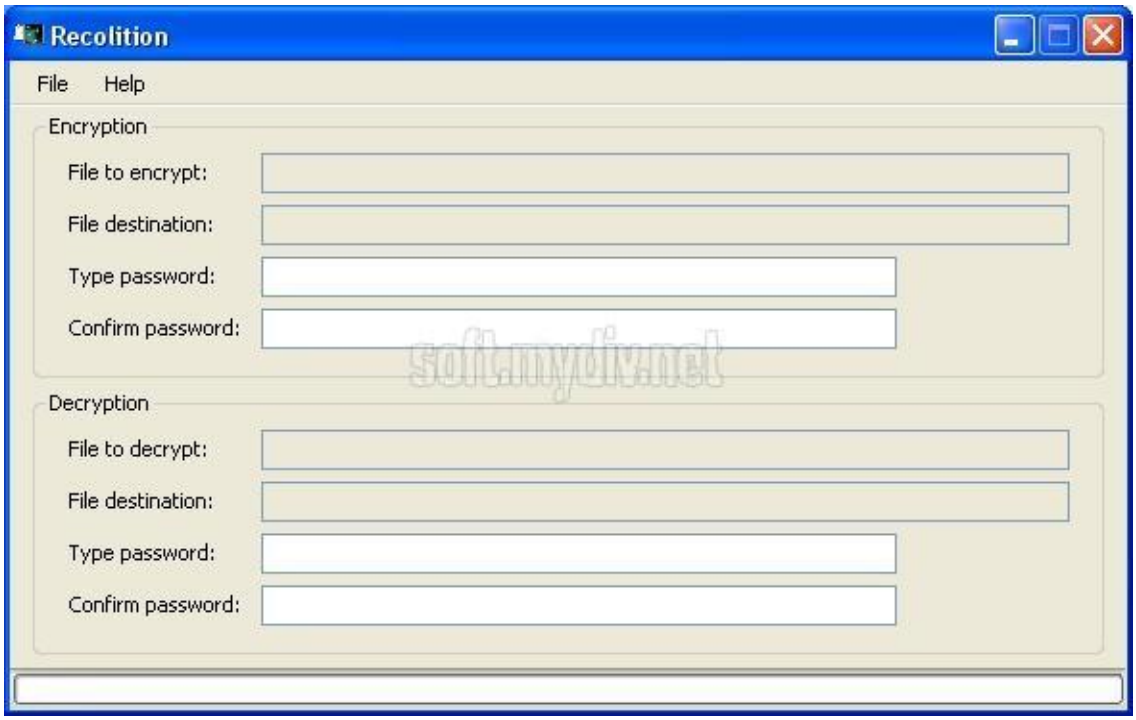

Рисунок 1.2 – Вікно програми Recolition

R-Crypto.exe (рисунок 1.3) – проста у використанні і безкоштовна програма для шифрування даних. За допомогою цієї програми можна захистити свої цінні дані від несанкціонованого доступу, або просто зробити копії даних на випадок їх втрати. R-Crypto створює зашифровані віртуальні диски. При копіюванні даних на такий диск, вони шифруються, при читанні даних з диска, вони декодуються. Доступ до таких даних можливий тільки в тому випадку, якщо користувач ввів пароль [7].

Зашифровані дані зберігаються разом з мета-даними диска в одному файлі-контейнері. Це фактично означає, що віртуальний диск є файлом, який можна зберігати в будь-якому розділі реального жорсткого диска або на змінних носіях.

Для шифрування R-Crypto використовує криптографічний інфраструктуру Windows, яка дозволяє використовувати різні криптографічні

продукти, встановлені в системі. Наприклад, Microsoft AES Cryptographic Provider - це засіб криптографії, яке встановлено в Windows XP і Vista за замовчуванням. Воно дозволяє шифрувати дані за алгоритмом AES з 128, 192 і 256-бітовим ключем.

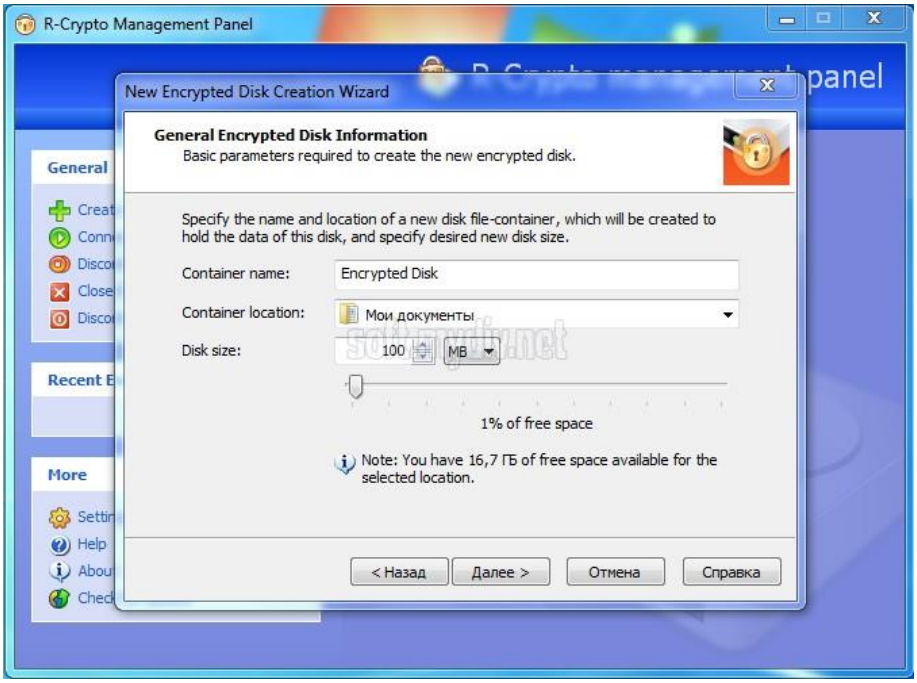

Рисунок 1.3 – Вікно програми R-Crypto

Ключові особливості та функції:

– покроковий майстер створення зашифрованих дисків;

– легка зміна розміру віртуального диска;

– можливість використовувати різні алгоритми шифрування;

– покроковий майстер розшифровки віртуального диска;

– гарячі клавіші для підключення до зашифрованого диска і для його закриття;

– закриття всіх відкритих сховищ при виході користувача з авторизації Windows;

– підтримка командного рядка.

TrustPort Tools.exe (рисунок 1.4) - містить в собі модулі для онлайн і офлайн шифрування даних, а також модуль для безпечного розміщення та

зберігання даних в хмарних сховищах. За допомогою даної програми можна з легкістю синхронізувати дані з мобільними пристроями, роблячи їх доступними для Вас в будь-який час в будь-якому місці. Також до складу програми входить утиліта ефективного знищення даних [8].

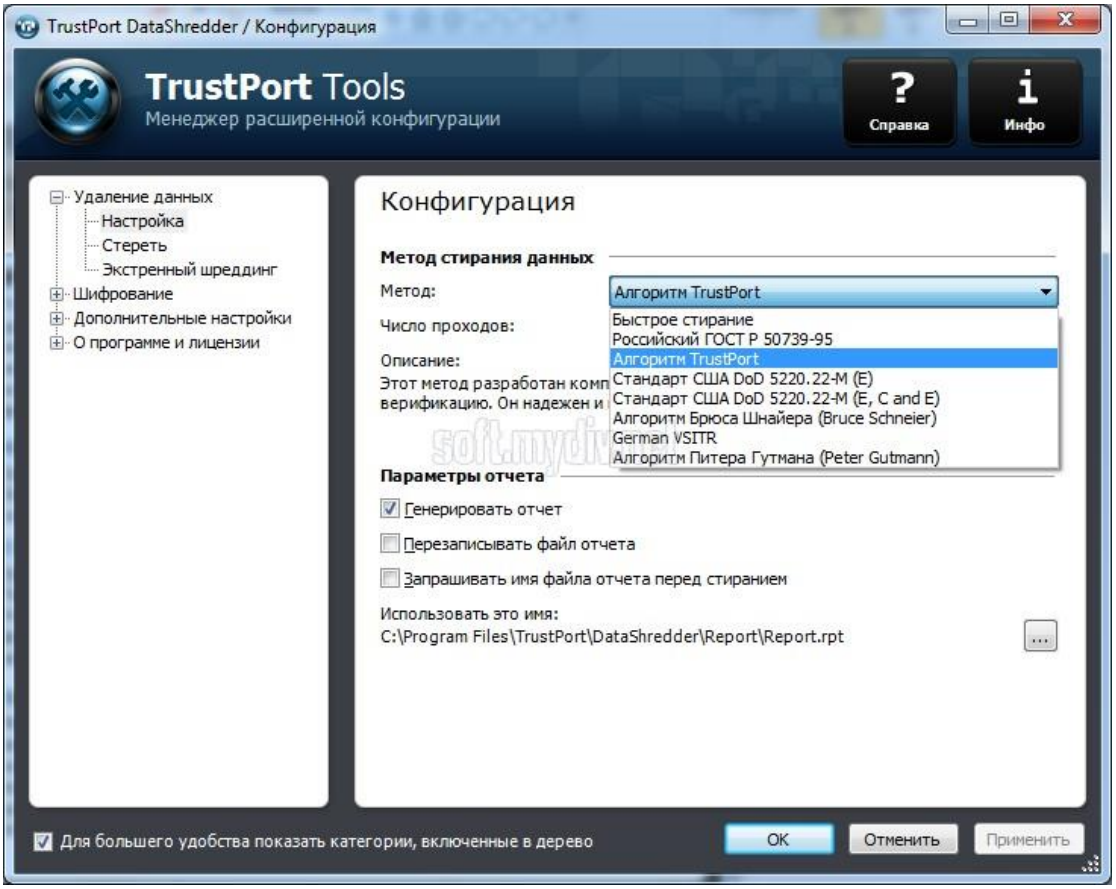

Рисунок 1.4 – Вікно програми TrustPort Tools

Ключові особливості та функції:

– шифрування архівів з високим рівнем захисту;

– шифрування дисків;

– настройка різних профілів доступу до даних;

– надійне видалення даних з можливістю вибору варіантів швидкості і надійності;

– швидке очищення системи без можливості відновлення файлів зловмисниками;

– екстрене видалення даних за допомогою гарячих клавіш.

Обмеження безкоштовної версії – 30-денний пробний період.

Проаналізувавши усі аналоги, визначено їхні можливості та недоліки, які враховувались при створенні власного програмного забезпечення з назвою «Bikod» (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Recolition	R-Crypto	TrustPort Tools	Bikod
Використання персонального ключа	+	-	+	+
Гарячі клавіші	-	+	+	+
Шифрування	+	+	-	+
2 рівні захисту зашифрованої інформації	+	-	-	+
Можливість застовувати різні алгоритми шифрування	-	+	+	+
Швидка обробка	+	+	-	+
Робота із зображенням	-	-	-	+

Таблиця порівняльних характеристик показала, що розробка програмного продукту є доцільною. В результаті отримаємо продукт, що покриває недоліки існуючих рішень і забезпечує більшу ступінь захисту даних більш простим шляхом та більш легкого передавання повідомлень зі збільшеною безпекою.

1.3 Аналіз існуючих методів кодування

Асиметричні криптосистеми – являють собою ефективні схеми криптографічного захисту інформації, що називаютьсь криптосистемами

використовуючи відкритий ключ. В системах такого типу для зашифровування інформації використовують ключ – первинний, а для розшифровування – вторинний, звідси походить і назва – асиметричні. Первинний ключ є відкритим, повністю доступним та може бути розповсюдженим для масового використання серед необмеженої кількості користувачів системи, які шифрують інформацію. Розшифровування інформації за допомогою такого ключа просто неможливе. Щоб розшифровувати інформацію отримувач такої зашифрованої інформації використовує інший ключ, він уже є секретним і називається закритим. Звісно, ключ для розшифровування не може бути створений з ключа зашифровування по одній схемі [9].

Одним з найбільших досягнень асиметричного шифрування полягає в тому, що дозволяє людям, які зовсім не мають наявної на те домовленості про захищеність, обмінюватися секретними інформаційними повідомленнями. Потреба відправникові та одержувачеві між собою погоджувати таємний персональний ключ по спеціалізованому захищеному каналу повноцінно відпала.

В популярному виді симетричного шифрування один і той самий ключ (що зберігається в секреті) використовується як для шифрування, так і для розшифрування. Розроблено ефективні (швидкі й надійні) методи шифрування.

Переваги:

– Шифри із симетричним ключем спроектовані так, щоб мати велику пропускну здатність

– Ключі для шифрів із симетричним ключем відносно короткі

– Шифри із симетричним ключем можна використати як примітиви для побудови різних криптографічних механізмів включно з псевдовипадковими генераторами чисел, обчислювально ефективних схем підпису та інших.

– Шифри із симетричним ключем можна комбінувати для отримання сильніших шифрів

Недоліки:

– При зв'язку між двома особами, ключ потрібно тримати в секреті на обох кінцях

– У великій мережі потрібно опікуватись багатьма ключами

– У зв'язку між двома особами криптографічна практика вимагає частої зміни ключів.

Проте, будь-який тип захисту не є ідеальним, тому все що розділяє зловмисника від зчитування даних – лиш час. Тому, окрім шифрування інформації, потрібно ще приховувати сам факт її розміщення

1.4 Аналіз методів стенографії для кодування даних в зображення

Стенографія – це швидкісне листування особливими знаками чи комбінаціями звичних символів, настільки короткими, що ними можна зручно записати живу мову чи скоротити текст. Фундаментом будь-якого повідомлення, навіть стенографічного, є його власний алфавіт – букви. Стенографічні алфавітні букви, чи, як прийнято називати їх, знаки, більш інформативніші звичайних букв, а сам стенографічний лист в 4-7 разів коротший звичайного.

Насамперед, стенографія представляє собою систему спеціальних значків та позначень: геометричних, які використовуються для різних систем стенографії в багатьох країнах світу та елементів літер та символів рукописного письма, які в свою чергу застосовуються в більшості саме по слов'янських країнах, тому що дана система пристосована для максимально швидкого запису за складами та словотвірними морфемами, а ще й словами усної мови.

При цьому, стенографія дуже вдало застосовується в текстах, адже дає змогу скорочувати значно, написаний текст. Одним з наглядних прикладів є «Слово о полку Ігоревім», де символ міг означати цілі слова і це було своєрідним кодування, як це показано на рисунку 1.5.

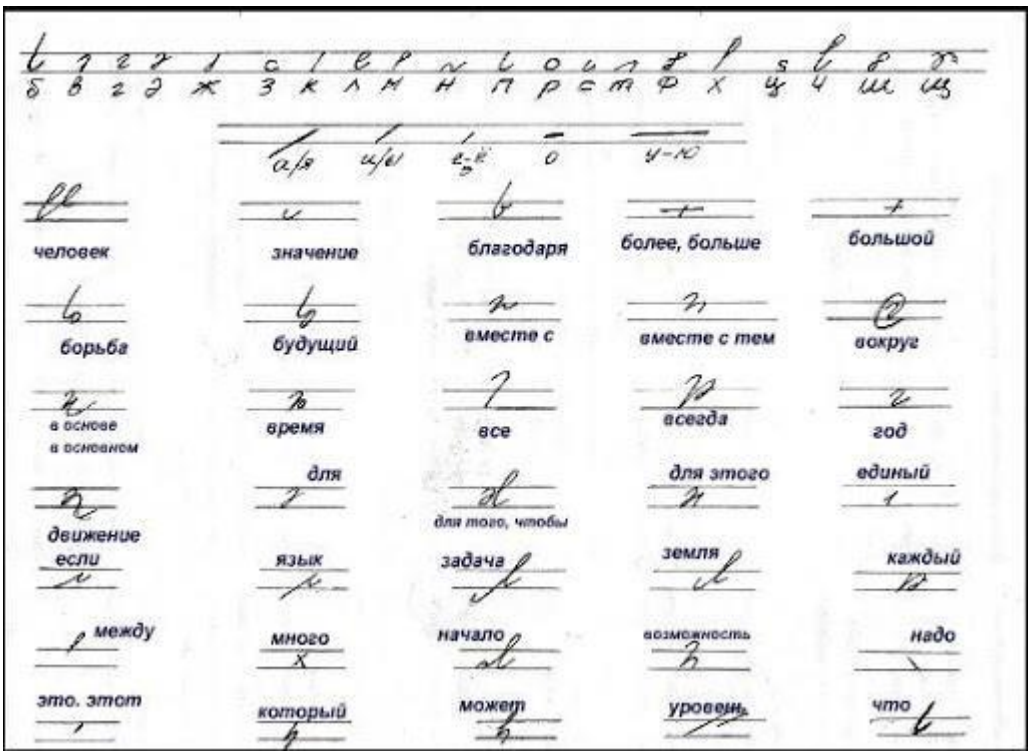

Рисунок 1.5 – Головне вікно програми

1.5 Аналіз існуючих методів стеганографічного захисту даних в зображенні

Найбільш ефективна на даний час задача стеганографії – захист інформації. Наприклад, одна секунда оцифрованого звуку з визначеною частотою дискретизації в 44100Гц та 8-бітним рівнем відліку у активному стереорежимі дає змогу приховати за рахунок заміни найменш значимих в контейнері молодших розрядів повідомлення в 10 Кбайт. Підхід такого роду дозволяє досягти змін лише у 1%, яких проста людина при прослуховуванні навіть не здатна помітити, проте при розшифруванні чи зчитуванні визначеним чином, дозволяє зберегти інформацію в потрібній формі. Цей простий приклад дає ілюстрацію певних можливостей стеганографії. Також потрібно зазначити, що без спеціальних засобів чи пристроїв, збережену інформацію в контейнері, не можливо отримати [10].

Як відомо, мета криптографії полягає в блокуванні несанкціонованого доступу до інформації шляхом шифрування змісту секретних повідомлень. Стеганографія має інше завдання, і її мета - приховати сам факт існування секретного повідомлення. При цьому обидва способи можуть бути об'єднані і використані для підвищення ефективності захисту інформації (наприклад, для передачі криптографічних ключів).

Щоб не викликати підозр у стороннього спостерігача, який передається стегоконтейнер практично нічим не повинен відрізнятися від вихідного контейнера. Але цього мало. Щоб стегосистеми була надійною, при її побудові необхідно виходити з припущення, що противник має повне уявлення про застосовуваної стеганографічної системі і деталях її реалізації. Єдиною невідомої йому величиною є стегоключа.

Виходячи з цього припущення, стегосистеми повинна бути сконструйована таким чином, щоб тільки власник стегоключа мав можливість виділити з стегоконтейнер вбудоване повідомлення і, головне, щоб тільки власник стегоключа мав можливість встановити факт присутності прихованого повідомлення.

Стеганографічні методи спрямовані на протидію системам моніторингу, сканування, відстежування та управління іншими мережевими ресурсами промислового шпигунства, також дозволяють протистояти намаганням контролю над інформаційним потоком при проходженні даних через сервери керування глобальних та локальних обчислювальних мереж. Для державних відомств і корпорацій, що зберігають в себе важливі масиви даних, це одна із найнеоціненних функцій стеганографії.

Оскільки спроби наслідування первісному шуму або ведуть до сумнівної безпеки або до занадто малому діапазону робочих частот для більшості практичних застосувань, найбільш привабливою залишається наступна базова процедура.

Вибирається клас досить гучних контейнерів і ідентифікуються біти шуму. Потім визначається, яку порцію шумових бітів контейнера можна замінити псевдовипадковими даними без значної зміни його статистичних характеристик. Так, якщо контейнер являє собою цифрову фотографію, нас повинні цікавити молодші біти градацій сірої шкали або RGB-значень при кольоровому зображенні, або коефіцієнти Фур'є в JPEG-форматі зображень. Змінюючи в середньому, припустимо, тільки 100-й піксель зображення, в одному мегабайті нестислого зображення можна заховати приблизно один кілобайт таємних даних.

Для додаткової безпеки і додання таємного повідомленням виду випадкових даних воно повинно бути зашифровано сильним криптоалгоритмом. Заміна псевдовипадковими битами деяких найгучніших бітів контейнера тільки трохи збільшить рівень шуму повідомлення, Включення відкритого тексту в контейнер може помітно змінити його статистичні характеристики. Більш того, послідовність приховують бітів повинна вибиратися псевдовипадковим способом як функція секретного ключа. Інакше противник, який має алгоритм, без праці розкриє контейнер.

Але і шифрування з ключем не звільняє від проблем. Якщо приховують біти в підозрюваному повідомленні мають деякі статистичні відхилення від інших аналогічних повідомлень, то противник отримає всі підстави для висновку, що воно містить приховані дані. Тоді шляхом додаткового зашумлення він може спотворити повідомлення і цим фактично його знищити.

По аналогії з криптографією та по типу стегоключа стегосистеми можна розділити на системи з відкритим ключем і системи з секретним ключем.

У стегосистеми з секретним ключем використовується один ключ, який повинен бути визначений або до початку обміну секретними повідомленнями, або переданий по захищеному каналу. Такий варіант вибору ключа є менш ефективним, оскільки зловмисник, перехопивши цей ключ отримує подальший доступ до даних.

Стегосистеми з відкритим ключем для вбудовування інформації та отримання потрібного повідомлення використовують різні ключі, такі, що за допомогою певних обчислень неможливо вивести один ключ використовуючи інший. Тому один відкритий ключ може передаватися просто по незахищеному каналу зв'язку. Крім того, дана схема добре працює і при взаємній недовірі відправника і одержувача. Тому вибір такого ключа є більш корисним і забезпечує високий рівень захисту [11].

Отже, стегосистема має відповідати таким основним вимогам:

1) властивості контейнера мають бути модифіковані таким чином, щоб будь-які зміни неможливо було виявити у разі візуального контролю, що визначає якість приховування повідомлення (для безперешкодного проходження стегоповідомлення каналами зв'язку воно не повинно привернути увагу);

2) стегоповідомлення має бути досить стійким до спотворень і в тому числі до зловмисних (при процесі передачі звука, зображення або використання інших інформаційних контейнерів можуть відбуватися різні трансформації зі збільшення або зменшення, перетворенн формату, ущільнення, з використанням алгоритмів з втратою даних або без нього, тощо);

3) для збереження повної цілісності прихованого повідомлення потрібно використовувати коди які можуть виправляти помилки, зроблені штучно;

4) для підвищення надійності в файлі, приховане повідомлення має бути продубльовано.

Багата палітра різноманітних алгоритмів реалізації задовільняє перераховані вимоги. На наш час, найбільш поширеним, проте найменш стійким є метод заміщення найменш значущих бітів, інакше LSB-метод, тобто Least Significant Bit, як менший значущий біт. Метод базується на удеї використання помилки дискретизації, яка завжди існує в електронних цифрових зображеннях, аудіо– чи відеофайлах. Ця похибка рівна найменшому значимому розряду числа, що визначає саму величину колірної складової елемента

зображення, тобто пікселя. Тому така модифікація молодших бітів майже завжди не викликає великої трансформації зображення і зовсім не виявляється візуально.

Ще одним популярним методом вбудовування та приховування повідомлень є використання особливостей форматів даних із стисненням з втратою даних, наприклад, JPEG. Цей метод, на відміну від методу LSB, більш стійкий до простих геометричних перетворень і виявленню їх при передачі, так як є можливість в широкому діапазоні варіювати якість стислого зображення, що робить неможливим визначення походження спотворення. Для вбудовування цифрових водяних знаків використовуються більш складні методи.

Метод дописування даних в кінець BMP-файлу. Є найпростішим співвідношенням сторін шляхом приховування, що використовують той факт, що всі стандартні програми визначають кінець даних зображення виходячи із заголовка зображення, який зберігається через підрядник знизу-вгору. Його модифікацією є метод приховування даних після палітри. Він заснований на тому, що початок даних визначається за допомогою значення поля «зсув даних» (навіть у зображеннях без палітри), значення якого можна штучним чином збільшити, а отриману таким чином ділянку BMP-файлу використовувати для приховування повідомлення.

У випадках, коли в BMP-файл зберігається 16-бітове зображення без стиснення, для приховування можна скористатися фактом, що колірні інтенсивності RGB в цьому режимі кодуються за допомогою 5 біт на канал. В результаті старший біт кожного 16-бітного відліку не містить інформацію про колір і може бути використаний для приховування.

Метод приховування в палітрі. Даний метод заснований на тому, що кожен елемент палітри складається з чотирьох байт, перші три з яких використовуються для кодування кольору, а останній зазвичай дорівнює 0 і не використовується. Таким способом можливо приховати не більше 256 байтів,

не змінивши розмір вихідного BMP-файлу. Тема BMP-файлу містить чотири байти, які дорівнюють 0 і поки не використовуються в форматі, їх використання для приховування також не призводить до збільшення розмірів контейнера. Крім того, довжина будь-якої байтової послідовності, що кодує горизонтальну лінію пікселів зображення повинна бути кратною 4 разі якщо це не виконано, вона доповнюється нульовими байтами до розміру, кратного 4. На цю особливість формату BMP базується метод приховування нульових байтів.

Такі методи, які працюють з графічними елементами, використовують у роботі контейнера графічні типи. Вони дозволяють влаштовувати не тільки текстову дані, але і такі ж зображення або інші файли. Проте, умовою маємо те, що об’єм таких прихованих даних не повинен перевищувати наявний розмір зображення-контейнера. Щоб досягнути таку мету, різні програми використовують різні технології, проте всі вони мають спільне в заміні певних потрібних пікселів у контейнері. Характерним простим прикладом цієї великоюї групи методів буде метод LSB.

Метод псевдовипадкового інтервалу.

У розглянутому вище найпростішому випадку виконується заміна НЗБ всіх послідовно розміщених пікселів зображення. Інший підхід - метод випадкового інтервалу, полягає у випадковому розподілі бітів секретного повідомлення по контейнеру, в результаті чого відстань між двома вбудованими битами визначається псевдовипадково. Ця методика особливо ефективна в разі, коли бітова довжина секретного повідомлення істотно менше кількості пікселів зображення.

Інтервал між двома послідовними вбудовування бітів повідомлення може бути, наприклад, функцією координат попереднього модифікованого пікселя.

Метод LSB має суть, яка полягає в заміні останніх значущих бітів у контейнері будь-якого типу, на біти приховуваного повідомлення. Різниця між порожнім і заповненим контейнерами повинна бути непомітною для органів чуття людини.

Практично в будь-якому типі мультимедіа-файлів після області з даними міститься прапор, після якого може знаходитися службова інформація. Якщо дописати приховуване повідомлення після даного прапора, то при перегляді або прослуховуванні помітити це буде неможливо.

Наглядний приклад методу Метод LSB можна побачити на рисунку 1.6.

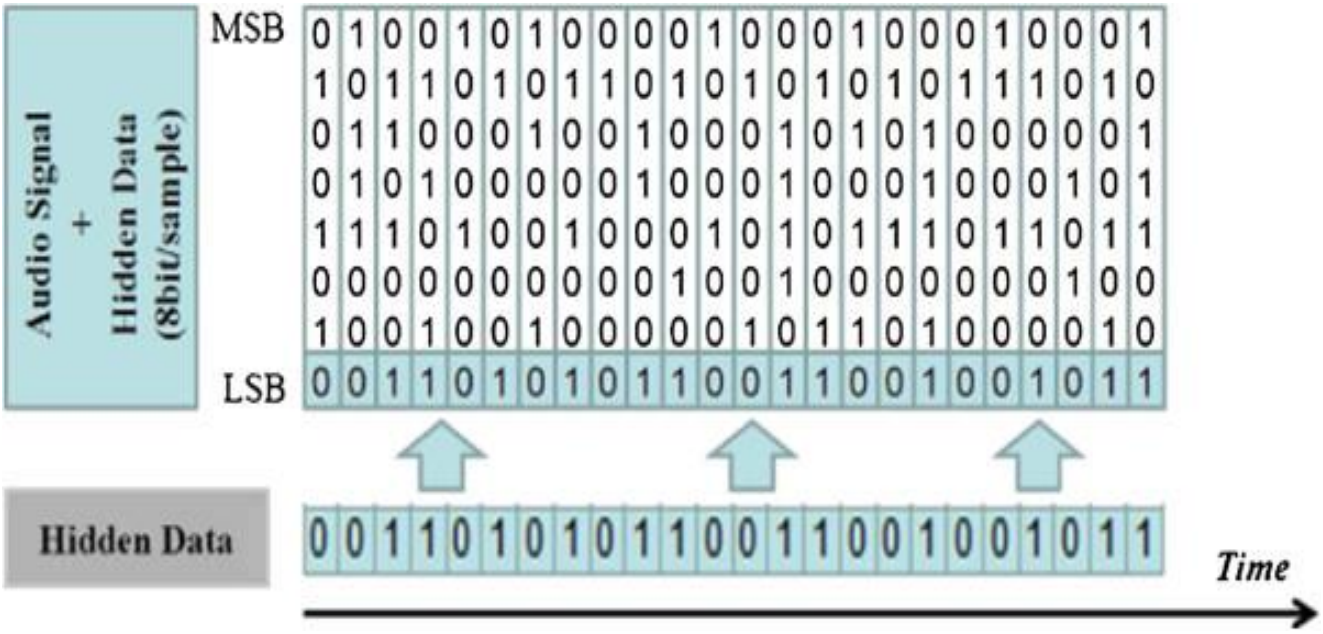

Рисунок 1.6 – Метод LSB в простій формі

Отже, для створення більш захищеного типу шифрування даних після обробки, потрібно провести ряд дій, для приховування інформації за стенографічними методами захист. Якщо використовувати шифрування без даного захисту, безпека самої передачі повідомлення чи зберігання файлу може бути під загрозою.

1.6 Аналіз існуючої ефективності роботи з JWT-токенами

Нині кіберзлочинність стала проблемою світового рівня. Наприклад, Дмитро Самарцев, директор BI.ZONE у сфері кібербезпеки, навів на Всесвітньому економічному форумі наступні цифри. У 2018 році збитки

світової економіки від кіберзлочинності склали за його словами 1.5 трильйона доларів. У 2022 році прогнозуються втрати вже 8 трильйонів, а 2030-го збиток від кіберзлочинів може перевищити 90 трильйонів доларів. Щоб зменшити втрати від кіберзлочинів, необхідно вдосконалювати методи безпеки користувачів. В даний час існує безліч методів автентифікації та авторизації, які допомагають реалізувати надійну безпекову стратегію. Серед них багато експертів виділяють як кращу авторизацію на основі токенів.

До появи токена авторизації повсюдно використовувалася система паролів та серверів. Зараз ця система все ще залишається актуальною через свою простоту та доступність. Традиційні методи, що використовуються, гарантують користувачам можливість отримати доступ до їх даних у будь-який час. Не завжди ефективно.

Розглянемо цю систему. Як правило, ідеологія їх застосування базується на таких принципах:

1. Здійснюється генерація облікових записів, тобто. люди вигадують поєднання букв, цифр чи будь-яких відомих символів, які стануть логіном та паролем.

2. Для здійснення можливості входу на сервер користувачу потрібно зберігати цю унікальну комбінацію і завжди мати до неї доступ.

3. При необхідності заново підключитися до сервера та авторизуватися під своїм обліковим записом, користувачеві потрібно заново вводити пароль та логін.

Крадіжка паролів – це далеко не унікальна подія. Один із перших задокументованих подібних випадків стався ще 1962 року. Людям не просто запам'ятовувати різні комбінації символів, тому вони часто записують усі свої паролі на папері, використовують один і той же варіант у декількох місцях, лише злегка модифікують за допомогою додавання символів або зміною регістру якийсь старий пароль, щоб використовувати його в новому місці. -за

що два паролі стають вкрай схожі. Логіни з тієї ж причини часто роблять однакові, ідентичні.

Крім небезпеки крадіжки даних та складності зі зберіганням інформації, паролі також вимагають перевірки автентичності сервера, що збільшує навантаження на згадку. Щоразу, коли користувач входить у систему, комп'ютер створює запис транзакції.

Авторизація токенів - це система, що працює зовсім інакше. За допомогою авторизації токенів вторинна служба перевіряє запит сервера. Після завершення перевірки сервер видає токен і відповідає на запит. Користувач все ще може мати один пароль для запам'ятовування, але токен пропонує іншу форму доступу, яку набагато важче вкрасти або подолати. І запис сеансу не займає місця на сервері. По суті, токен авторизації - це пристрій, призначений для забезпечення інформаційної безпеки користувача, також використовується для ідентифікації його власника. Як правило, це фізичний пристрій, який використовується для спрощення аутентифікації.

Токени авторизації розрізняються за типами. Розглянемо їх:

1. Пристрої, які потрібно підключити фізично. Наприклад: ключі, диски тощо. Той, хто будь-коли використовував USB-пристрій або смарт-карту для входу в систему, стикався з підключеним струменем.

2. Пристрої, які знаходяться досить близько до сервера, щоб встановити з ним з'єднання, але вони не підключаються фізично. Прикладом такого типу токенів може бути "magic ring" від компанії Microsoft.

3. Пристрої, які можуть взаємодіяти із сервером на великих відстанях.

У всіх трьох випадках користувач повинен щось зробити, щоб запустити процес. Наприклад, ввести пароль або відповісти на запитання. Але навіть коли ці кроки здійснюються без помилок, доступу без токена отримати неможливо.

Авторизація за допомогою токена відбувається в такий спосіб. Спочатку людина запитує доступ до сервера або захищеного ресурсу. Запит зазвичай включає введення логіна і пароля. Потім сервер визначає, чи користувач може

отримати доступ. Після цього сервер взаємодіє з пристроєм: ключ, телефон, USB або ще щось. Після перевірки сервер видає токен та відправляє користувачеві. Токен знаходиться у браузері, доки робота триває. Якщо користувач спробує відвідати іншу частину сервера, токен знову зв'язується з ним. Доступ надається або навпаки забороняється на основі виданого токена.

Адміністратори встановлюють обмеження на токени. Можна дозволити одноразовий токен, який негайно знищується, коли людина виходить із системи. Іноді встановлюється маркер самознищення в кінці певного періоду часу.

Аутентифікація на основі токенів - це один із багатьох методів веб-аутентифікації, які використовуються для забезпечення безпеки процесу перевірки. Існує автентифікація по паролю, біометрії. Хоча кожен метод автентифікації є унікальним, всі методи можна розділити на 3 категорії:

1. аутентифікація за паролем (звичайне запам'ятовування комбінації символів)

2. аутентифікація по біометрії (відбиток пальця, сканування сітківки ока, FaceID)

3. аутентифікація токенів

Аутентифікація токенів вимагає, щоб користувачі отримали згенерований комп'ютером код (або токен), перш ніж їм буде надано доступ до мережі. Аутентифікація токенів зазвичай використовується у поєднанні з аутентифікацією паролів для додаткового рівня безпеки (двофакторна аутентифікація (2FA)). Якщо зловмисник успішно реалізує атаку грубої сили, щоб отримати пароль, йому доведеться обійти рівень аутентифікації токенів. Без доступу до токену отримати доступ до мережі стає складніше. Цей додатковий рівень відлякує зловмисників та може врятувати мережі від потенційно катастрофічних порушень.

У багатьох випадках токени створюються за допомогою донглів або брелоків, які генерують новий токен аутентифікації кожні 60 секунд відповідно

до заданого алгоритму. Через потужність цих апаратних пристроїв користувачі повинні постійно тримати їх у безпеці, щоб вони не потрапили до чужих рук. Таким чином, члени команди повинні відмовитися від ключа або брелока, якщо команда розпадається.

Найбільш поширені системи токенів містять заголовок, корисне навантаження та підпис. Заголовок складається з типу корисного навантаження, а також алгоритму підпису, що використовується. Корисне навантаження містить будь-які твердження, що стосуються користувача. Підпис використовується для доказу того, що повідомлення не наражалося на небезпеку при передачі. Ці три елементи працюють разом, щоб створити високоефективну та безпечну систему аутентифікації.

Хоча ці традиційні системи аутентифікації токенів все ще діють сьогодні, збільшення кількості смартфонів зробив аутентифікацію на основі токенів простіше, ніж будь-коли. Смартфони можуть бути доповнені, щоб служити генераторами кодів, надаючи кінцевим користувачам коди безпеки, необхідні для отримання доступу до їх мережі в будь-який момент часу. У процесі входу в систему користувачі отримують криптографічно безпечний одноразовий код доступу, обмежений за часом 30 або 60 секундами, залежно від налаштувань на стороні сервера. Ці м'які токени генеруються або програмою-автентифікатором на пристрої, або надсилаються на запит через SMS.

Поява аутентифікації на основі токенів смартфонів означає, що більшість співробітників вже мають обладнання для генерації кодів. В результаті витрати на впровадження та навчання персоналу зведені до мінімуму, що робить цю форму автентифікації на основі токенів зручним та економічно вигідним варіантом для багатьох компаній.

У міру зростання кіберзлочинності та ускладнення методів атак мають удосконалюватися методи та політика захисту. Через зростання атак "грубою силою", перебору за словником і фішингу для захоплення облікових даних

користувачів стає цілком очевидно, що автентифікації по паролю вже недостатньо, щоб протистояти зловмисникам.

Аутентифікація на основі токенів, коли вона використовується в тандемі з іншими методами аутентифікації, створює бар'єр 2FA, призначений для того, щоб зупинити навіть найпросунутого хакера. Оскільки токени можуть бути отримані тільки з пристрою, який їх виробляє - брелок або смартфон, системи авторизації токенів вважаються дуже безпечними і ефективними.

Але, незважаючи на безліч переваг, пов'язаних із платформою токенів, завжди залишається невеликий ризик. Звичайно, токени на базі смартфонів неймовірно зручні у використанні, але смартфони також є потенційними вразливістю. Токени, надіслані як текстів, більш ризиковані, оскільки їх можна перехопити під час передачі. Як і у випадку з іншими апаратними пристроями, смартфони також можуть бути втрачені або вкрадені та опинитися в руках зловмисників.

1.7 Постановка задач роботи

Після аналізу питання можливих ризиків та захисту від несанкціонованого доступу, а також методів його вирішення, було визначено наступні завдання, які необхідно виконати для розробки програмного продукту:

– розробити алгоритм шифрування даних;

– розробити алгоритм стеганографічного захисту та взаємодії з програмою;

– розробити інтерфейс програмного продукту, який буде легким для розуміння та інтуїтивним користувачу;

– розробити програмний продукт, призначений для візуалізації роботоспроможності методу, для вирішення проблеми захисту;

– провести тестування програмного продукту для перевірки всіх можливих варіантів використання.

1.8 Висновки

1. Аналіз стану захисту інформації в комп’ютерних системах від несанкціонованого доступу показав, що розробка програмного продукту приховування даних у файлах мультимедіа є актуальною та доцільною, оскільки більшість систем захисту орієнтовані на парольний або криптографічний захист.

2. Порівняльний аналіз методів приховування інформації в контейнерах показав, що найбільш придатним типом контейнера є зображення, які дозволяють приховувати інформацію розміром до 25 % від розміру контейнера.

3. Визначено та розроблено перелік основних завдань, які необхідні для розробки програмного продукту.

2 РОЗРОБКА МЕТОДУ, МОДЕЛІ ТА АЛГОРИТМІВ СИСТЕМИ

2.1 Аналіз архітектурних рішень у розробці програмних систем

Архітектура програмного забезпечення (англ. software architecture) – це представлення структури обчислювальної системи або програмної програми, що містить програмні компоненти, зовнішні, відкриті властивості цих компонентів, а також зв'язки між ними [12].

Також цей термін відноситься до безпосереднього документування архітектури самого програмного забезпечення. Документування загальної архітектури ПЗ сприяє спрощенню процесу комунікації та взаємодії між усіма зацікавленими особами (англ. stakeholders), дозволяє формалізувати рішення, прийняті на початкових етапах проектування, про загальний дизайн системи та забезпечує можливість подальшого використання компонентів та шаблонів даного повторно у майбутніх проектах.

Розглянемо найпопулярніші види архітектури, що використовуються для розробки програмного забезпечення, а саме монолітну, сервіс-орієнтовану та модульну архітектуру.

Монолітна архітектура програмного забезпечення – це архітектурний підхід, який полягає в тому, що різні компоненти системи об'єднані в одну програму на одній платформі. Звичайна структура монолітного додатка складається з бази даних, клієнтського інтерфейсу користувача і серверного відношення. Усі частини програмного забезпечення уніфіковані, а контроль над усіма його функціями відбувається з одного місця. Загальне представлення монолітної системи наведений на рисунку 2.1.

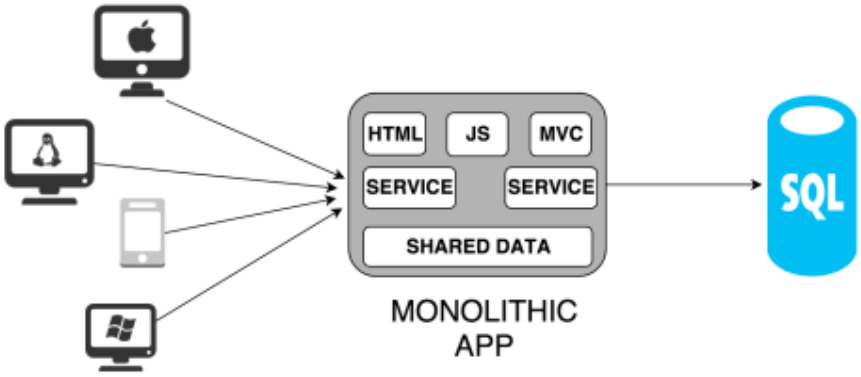

Рисунок 2.1 – Приклад монолітної архітектури

Монолітна архітектура зазвичай виступає зручним підходом для роботи невеликої групи розробників, тому безліч стартапів схиляються до вибору саме цього архітектурного рішення при створенні програних додатків. Усі компоненти монолітного програмного забезпечення виявляють ознаки міцного взаємозв'язку та взаємозалежності, в чому і проявляється самодостатність такого ПЗ. Такий дизайн системи є традиційним рішенням для створення програмних за стосунків, незважаючи на те, що на сьогодні, багато спеціалістів вважають його застарілим.

Перевагами монолітної архітектури є:

– Спрощено розробку та розгортання системи. В даний час існує безліч програмних засобів, які можна інтегрувати для спрощення розробки. Більш того, розгортання та оновлення всіх компонентів системи відбувається одним процесом, що забезпечує економію часу.

– менша кількість наскрізних проблем. У більшості програмного забезпечення спостерігається залежність від значної кількості між компонентними завданнями, наприклад, ведення логів, контрольні журнали, обмеження швидкості та інші. У додатках із монолітною структурою такі нюанси легко дозволяються завдяки єдиній кодовій базі. Підключення нових

компонентів для вирішення таких проблем легко виконувати всередині системи.

– висока продуктивність. Монолітні системи забезпечують швидкий зв'язок між програмними компонентами за рахунок загальної бази та пам'яті.

Недоліками монолітної архітектури є:

– Згодом кодова база стає надто громіздкою. Протягом життєвого циклу програмні продукти продовжують збільшуватися за обсягами, обростаючи все новим і новим функціоналом, які структура стає дедалі нечіткішою. Кодова база стає складною та незрозумілою, що ускладнює внесення змін, особливо нових розробників. Інша складність – виявлення побічних та небажаних ефектів та залежностей.

– Складність впровадження нових технологій. Залучення нової технології може вимагати внесення змін у деякі або навіть усі компоненти програмного відношення, що може виявитися досить дорогою та годинною роботою.

– Обмежена гнучкість. При найменших оновленнях монолітна система потребує суцільного повторення операції розгортання. Що вимагає від розробників припинення роботи на виконання цієї процедури. При значній кількості команд та розробників, що працюють над проектом, це може радикально зменшити гнучкість роботи.

Сервіс-орієнтована архітектура (далі SOA – service-oriented architecture) – це архітектурний підхід до розробки програмного забезпечення, що передбачає поділ системи на безліч дискретних та не взаємно пов'язаних сервісів, кожен з яких відповідає за чітко визначений функціонал. SOA передбачає поділ сервісів на дві основні ролі: постачальник та споживач послуг. Виконавцями цих ролей можуть бути програмні агенти. Головна концепція SOA полягає в тому, що програмне забезпечення може бути спроектоване та розроблене таким чином, що всі його компоненти здатні легко інтегруватися та легко використовуватись. Схематичне зображення сервіс-орієнтованої системи наведено на рисунку 2.2.

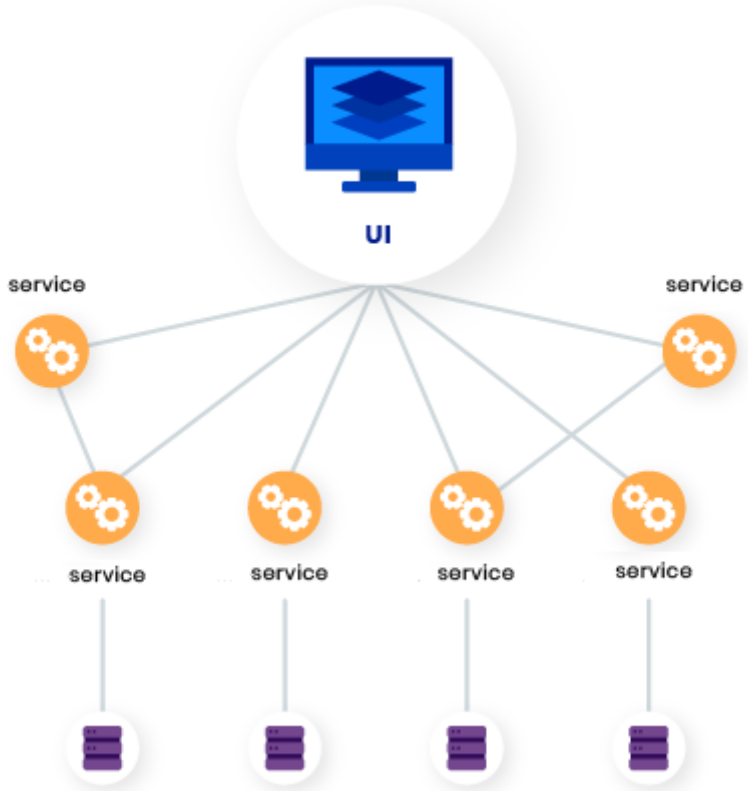

Рисунок 2.2 – Сервіс-орієнтована архітектура

Перевагам сервіс-орієнтованої архітектури є:

– Повторне використання сервісів. Завдяки тому, що усі компоненти такої системи, по своїй суті є автономними й слабо зв’язаними функціональними одиницями, то їх повторне використання для підтримки декількох програмних продуктів може бути виконане з легкістю та без негативного впливу на решту сервісів.

– Простота у супроводі. Так як кожен сервіс виступає як окрема незалежна функціональна одиниця, до неї легко можна вносити зміни, оновлювати та супроводжувати, без втручання в роботу інших сервісів.

– Висока надійність. На відміну від програмних продуктів, побудованих з використанням монолітної архітектури, добре спроектовані сервіси є не перенасичені кодом, саме тому вони є прості для відлагодження та тестування.

Що в свою чергу сприяє тому, що програмні системи на основі SOA є більш надійними.

– Паралельна розробка. Завдяки тому, що сервіс-орієнтована архітектура поділена на прошарки, вона підтримує паралелізм в розробці. Сервіси можуть розроблятися незалежно один від одного і можуть бути закінчені одночасно.

Недоліками сервіс-орієнтованої архітектури є:

– Складність контролю. Головним недоліком SOA є її безпосередня складність. Задачею кожного сервісу є своєчасна передача даних. Об’єми передачі даних можуть бути величезними, і як результат, це призводить до складності встановлення належного контролю за кожним сервісом.

– Високі інвестиційні витрати. Проектування та розробка сервіс-орієнтованих систем потребує значних попередніх інвестицій, як на людські ресурси, так і на технології та і на саму розробку.

– Додаткове навантаження. Валідація даних в сервіс-орієнтованих системах, відбувається перед прямою взаємодією між сервісами. Це призводить до того, що при використанні декількох сервісів час отримання результату збільшується, тим самим знижуючи загальну продуктивність системи.

Модульна архітектура програмного забезпечення – це підхід до розробки, у якому програма розбивається на незалежні компоненти, які можуть взаємодіяти між собою у вигляді чітко описаних контрактів. Ці компоненти – модулі – повинні мати кілька ознак. Перший – інкапсуляція, вони повинні приховувати свою реалізацію від зовнішнього оточення; другий – слабка зв'язність, модулі взаємодіють лише за допомогою заздалегідь обумовлених контрактів; третій – динамічність, можливість заміщатися «на льоту», без необхідності зупинення всього додатка.

Також це забезпечує легкість у тестуванні та виявленні помилок. Роль модулів можуть відігравати структури даних, бібліотеки функцій, класи, сервіси та інші програмні одиниці, які призначені для виконання певного, чітко визначеного функціонала і надають до нього інтерфейс. При такому

архітектурному підході програмний код часто розділяється на окремі файли, кожен з яких компілюється незалежно. Така модульність програмного коду сприяє зменшенню затрат час на повторну компіляцію при внесенні змін, як впливають не невелику кількість вихідних файлів, таким чином спрощуючи розробку. До того ж, це надає змогу замінювати певні компоненти кінцевого програмного продукту, без потреби його повторної компіляції.

Оцінивши переваги та недоліки даних архітектурних рішень, було вирішено обрати модульну архітектуру, адже вона забезпечує просту модель програмного продукту, гнучкість у внесенні змін та зручність у тестуванні.

2.2 Розробка блоку шифрування даних

Шифр XOR – це алгоритм шифрування даних з використанням винятковою диз'юнкції. Алгоритм XOR шифрування полягає в "накладення" послідовності випадкових чисел на текст, який необхідно зашифрувати. Послідовність випадкових чисел називається гамма-послідовність, і використовується для шифрування і розшифровки даних [13].

Формула для отримання закодованого тексту: Cn = Mn xor Kn.

На рисунку 2.3 проста модель даного типу шифрування.

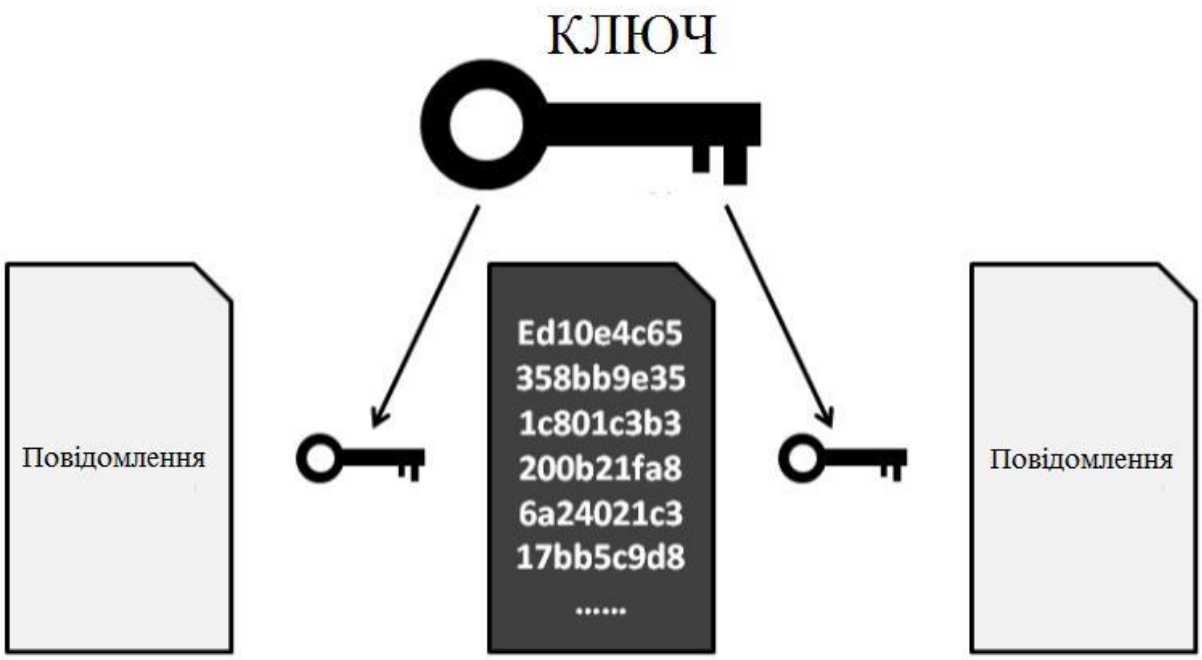

Рисунок 2.3 – Простий алгоритм шифрування

Ключ шифрування можна отримати двома способами:

Повторювати ключове слово поки довжина гами не дорівнюватиме довжині повідомлення;

Згенерувати послідовність псевдовипадкових чисел, що дорівнює по довжині тексту повідомлення.

Даний алгоритм принято за основу створеня варіації алгоритму шифрування для захисту важливої інформації, простота базового алгоритму значно зменшують розмір кодованого файлу, що дуже позитивно відображається на самій роботі програми.

На рисунку 2.4 та 2.5 наведено код функцій шифрування, яка була взята за основу.

```
using System;
public class XORCipher
{
    private string GetRepeatKey(string s, int n)
    {
        var r = s;
        while (r.Length < n)
        {
            r += r;
        }         return r.Substring(0, n);
    }
    private string Cipher(string text, string secretKey)
    {
        var currentKey = GetRepeatKey(secretKey, text.Length);
        var res = string.Empty;
        for (var i = 0; i < text.Length; i++)
        {
            res += ((char)(text[i] ^ currentKey[i])).ToString();
        }
        return res;
    }
    public string Encrypt(string plainText, string password)
        => Cipher(plainText, password);

        public string Decrypt(string encryptedText, string password)
        => Cipher(encryptedText, password);
}
```

Рисунок 2.4 – Головна частина коду шифрування

```
class Program
{
    static void Main(string[] args)
    {
        var x = new XORCipher();
        Console.Write("Текст: ");
        var message = Console.ReadLine();
        Console.Write("Пароль: ");
        var pass = Console.ReadLine();
        var encryptedMessageByPass = x.Encrypt(message, pass);
        Console.WriteLine("Зашифровано {0}", encryptedMessageByPass);
        Console.WriteLine("Розшифровано  {0}",
        x.Decrypt(encryptedMessageByPass,
        pass));
        Console.ReadLine();
    }}
```

Рисунок 2.5 – Додаткова частина коду шифрування

2.3 Розробка блоку стеганографічного захисту

За аналогією з криптографією, за типом стегоключа стегосистеми можна поділити на системи з секретним ключем та системи з відкритим ключем.

При використанні контейнерів фіксованої довжини, які вільні від недоліків потокових контейнерів, відправник заздалегідь знає розмір файлу і може вибрати приховують біти в підходящої псевдовипадкові послідовності. Оскільки контейнер відомий заздалегідь, є час оцінити його ефективність стосовно до обраного алгоритму приховування інформації. З іншого боку, контейнери фіксованої довжини мають обмежений обсяг і іноді вбудовується повідомлення може не поміститися в файл-контейнер.

Інший недолік полягає в тому, що відстані між приховують бітами рівномірно розподілені між найбільш коротким і найбільш довгим заданими відстанями, в той час як справжній випадковий шум буде мати експоненціальне

розподіл довжин інтервалу. Звичайно, можна породити псевдовипадкові експоненціально розподілені числа, але цей шлях зазвичай занадто трудомісткий. Однак на практиці найчастіше використовуються саме контейнери фіксованої довжини, як найбільш поширені і доступні.

Для більшості сучасних методів, використовуваних для приховування повідомлення в цифрових контейнерах, має місце залежність надійності системи від обсягу вбудованих повідомлень.

Дана залежність показує, що при збільшенні обсягу вбудованих повідомлень знижується надійність системи (при незмінності розміру контейнера). Таким чином, використовуваний в стегосистеми контейнер накладає обмеження на розмір вбудованих даних.

У будь-якому випадку контейнер без вбудованого повідомлення - це порожній контейнер, а контейнер, що містить вбудовану інформацію, - це заповнений або стегоконтейнер.

Вбудоване, тобто приховане повідомлення, яке перебуває в стегоконтейнер, передається від відправника до одержувача по каналу передачі, який називається Стеганографічні каналом або просто стегоканалом.

Вбудовування повідомлень в контейнер відбувається з використанням спеціального стегоключа. Під ключем розуміється секретний елемент, який визначає порядок занесення повідомлення в контейнер.

У стегосистеми з секретним або закритим ключем, що використовується один ключ, який повинен бути визначений або до початку обміну секретними повідомленнями, або переданий по захищеному каналу. Такий варіант вибору ключа є менш ефективним, оскільки зловмисник, перехопивши цей ключ отримує подальший доступ до даних.

У стегосистеми з відкритим ключем, він може бути оприлюдненим, для дій вбудовування і процесу отримання повідомлення потрібно застосовувати різні ключі, такі, що за допомогою обчислень ніяк неможливо вирахувати один ключ з іншого. Тому ключ відкритого типу може передаватися вільно по

незахищеному каналу зв'язку. Крім того, дана схема добре працює і при взаємній недовірі відправника і одержувача. Тому вибір такого ключа є більш корисним і забезпечує високий рівень захисту [14].

Будь-яка стегосистеми повинна відповідати наступним вимогам:

– Властивості контейнера повинні бути модифіковані, щоб зміна неможливо було виявити при візуальному контролі. Ця вимога визначає якість приховування впроваджуваного повідомлення: для забезпечення безперешкодного проходження стегоповідомлення по каналу зв'язку воно жодним чином не повинно привернути увагу атакуючого.

– Стегоповідомлення має бути стійко до спотворень, в тому числі і зловмисним. В процесі передачі зображення (звук або інший контейнер) може зазнавати різні трансформації: зменшуватися або збільшуватися, перетворюватися в інший формат і т. Д. Крім того, воно може бути стисло, в тому числі і з використанням алгоритмів стиснення з втратою даних.

– Для збереження цілісності вбудованого повідомлення необхідно використання коду з виправленням помилки.

– Для підвищення надійності вбудовується повідомлення повинно бути продубльовано.

Код стеганографічного методу наведено нижче на рисунку 2.6 так 2.7.

```
BinaryReader bText = new BinaryReader(rText, Encoding.ASCII);
List<byte> bList = new List<byte>();
while (bText.PeekChar() != -1)
{     bList.Add(bText.ReadByte());            }
int CountText = bList.Count;             bText.Close();
rFile.Close();
if (CountText > ((bPic.Width * bPic.Height)) - 4 ) {
    MessageBox.Show("Вибрана картинка мала для розміщення вибраного
   тексту", "Інформація", MessageBoxButtons.OK);
    return;     }         if (isEncryption(bPic))          {
    MessageBox.Show("Файл зашифрований",
    "Інформація", MessageBoxButtons.OK);
    return;
}

byte [] Symbol = Encoding.GetEncoding(1251).GetBytes("/");
BitArray ArrBeginSymbol = ByteToBit(Symbol[0]);
Color curColor = bPic.GetPixel(0, 0);
BitArray tempArray = ByteToBit(curColor.R);
tempArray[0] = ArrBeginSymbol[0];
tempArray[1] = ArrBeginSymbol[1];
byte nR = BitToByte(tempArray);

tempArray = ByteToBit(curColor.G);
tempArray[0] = ArrBeginSymbol[2];
tempArray[1] = ArrBeginSymbol[3];
tempArray[2] = ArrBeginSymbol[4];
byte nG = BitToByte(tempArray);

tempArray = ByteToBit(curColor.B);
tempArray[0] = ArrBeginSymbol[5];
tempArray[1] = ArrBeginSymbol[6];
tempArray[2] = ArrBeginSymbol[7];
byte nB = BitToByte(tempArray);

Color nColor = Color.FromArgb(nR, nG, nB);
bPic.SetPixel(0, 0, nColor);

WriteCountText(CountText, bPic);
int index = 0;
bool st = false;
for (int i = 4; i < bPic.Width; i++) {
```

Рисунок 2.6 – Головна частина коду стеганографічного приховування

```
            for (int j = 0; j < bPic.Height; j++) {
            Color pixelColor = bPic.GetPixel(i, j);
            if (index == bList.Count) {
                st = true;
                break;
            }
            BitArray colorArray = ByteToBit(pixelColor.R);
            BitArray messageArray = ByteToBit(bList[index]);
            colorArray[0] = messageArray[0];
            colorArray[1] = messageArray[1];
            byte newR = BitToByte(colorArray);

            colorArray = ByteToBit(pixelColor.G);
            colorArray[0] = messageArray[2];
            colorArray[1] = messageArray[3];
            colorArray[2] = messageArray[4];
            byte newG = BitToByte(colorArray);

            colorArray = ByteToBit(pixelColor.B);
            colorArray[0] = messageArray[5];
            colorArray[1] = messageArray[6];
            colorArray[2] = messageArray[7];
            byte newB = BitToByte(colorArray);

            Color newColor = Color.FromArgb(newR, newG, newB);
            bPic.SetPixel(i, j, newColor);
            index ++;
        }
        if (st) {
            break;
        }
    }
```

Рисунок 2.7 – Додаткова частина коду стеганографічного приховування

2.4 Розробка алгоритмів роботи програми

Розглянемо S - внутрішній стан ГПВЧ, g(K) – функція перетворення стану, f(K) – функція шифрування.

Функція шифрування може змінюватися випадковим чином з кожним символом, тому вихід цієї функції повинен залежати не лише від поточного вхідного символу, але й від внутрішнього стану S генератора. Цей внутрішній стан перетворюється функцією перетворення стану g(K) після кожного кроку шифрування. Генератор складається з компонентів S та g(K). Безпечність

такого шифру залежить від числа внутрішніх станів S й складності функції перетворення g(K). Відповідно характеристики послідовних шифрів залежать від властивостей генераторів псевдовипадкових чисел. З іншої сторони, сама функція шифрування f(K) є достатньо простою і може складатися лише з логічної операції XOR [15].

Схематично генератори ПВЧ можуть бути реалізовані у вигляді скінченних автоматів, які включають двійкові тригерні комірки пам'яті. Якщо скінченний автомат має n комірок пам'яті, тоді він може приймати різних внутрішніх станів S. Функція перетворення станів g(K) представляється за допомогою комбінаторної логіки.

У загальному, процес шифрування полягає у генерації відправником за допомогою ГПВЧ гами шифру й накладанні отриманої гами на відкритий текст таким чином, наприклад з використанням операції додавання по модулю 2, що в результаті отримується шифрований текст. Процес розшифрування зводиться до повторної генерації гами шифру отримувачем повідомлення й накладення цієї гами на зашифровані дані.

В якості ключового потоку можна використовувати нелінійно "відфільтровани" вміст зсувного регістру, а для отримання послідовності максимальної довжини - лінійний зворотний зв'язок.

Розробка шифрувальника полягає в тому, що необхідно розробити алгоритми кодування та розкодування інформації по методу шифрування XOR (рисунок 2.8), а також необхідно розробити загальний алгоритм роботи програмного засобу (рисунок 2.9).

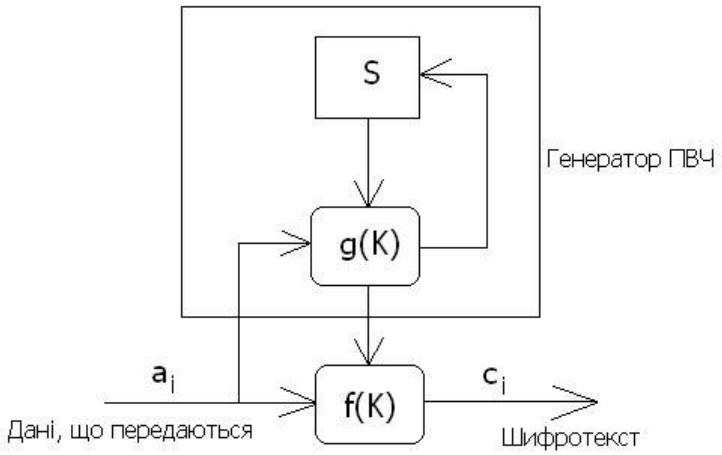

Рисунок 2.8 – Алгоритм роботи шифратора

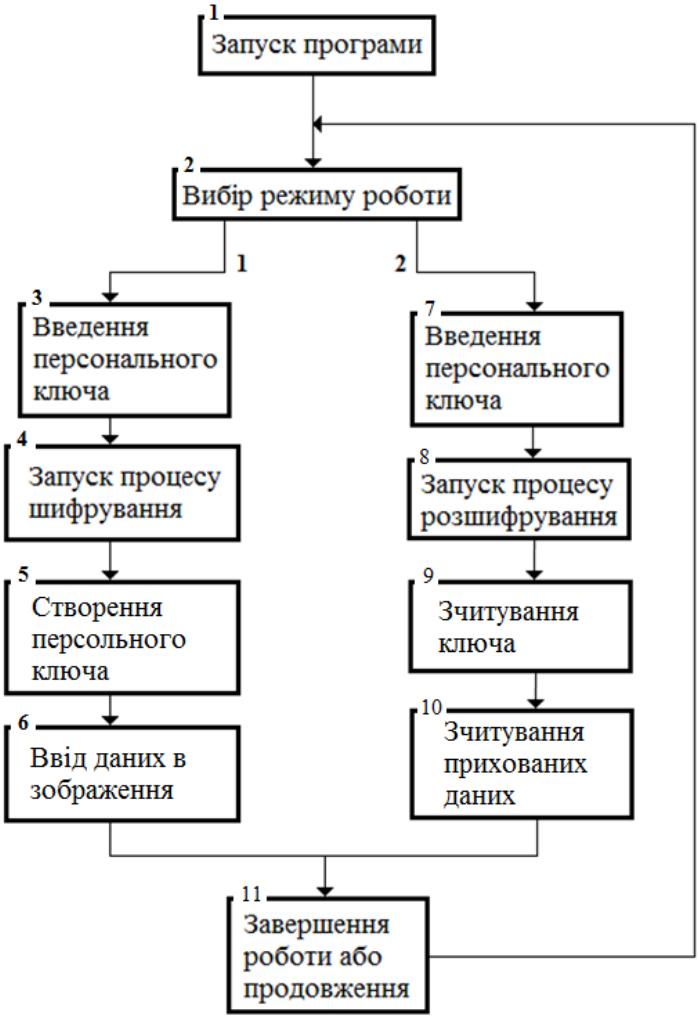

Рисунок 2.9 – Загальний алгоритм роботи програми

2.5 Аналіз продуктивності додаткового шифрування даних

При зломі складних захистів, а також при необхідності досягти максимального ефекту, застосовується комбінація перерахованих вище способів. В окремих випадках це відбувається при недостатній кваліфікованості зломщика.

Цей список не є вичерпним, а лише позначає найбільш поширені способи злому. Вигляд злому, як правило, обумовлений видом захисту. Для деяких захистів можна використовувати різні види злому, для інших спосіб може бути єдиним. Але є й такі способи організації захисту, зламати які неможливо, хоча лише з часовими мірками.

Як правило, в основі роботи крекер лежить дослідження асемблерного коду, отриманого з машинних інструкцій за допомогою спеціально призначеної для цього програми-дизассемблера. Залежно від вибраного способу злому, результат дослідження може використовуватися, наприклад, для побудови генератора ключів або для внесення необхідних змін у файл, що виконується. Останній спосіб у більшості випадків найбільш легкий, тому що не вимагає вивчення алгоритму перевірки правильності ключа: часто злом зводиться до пошуку перевірки кількох умов і заміні такої умови на безумовний перехід (goto, jmp), або, рідше, на протилежне.

Крім того, внесення змін до виконуваного файлу (патч) може здійснюватися з метою відключення небажаних дій з боку програми (наприклад, нагадування необхідності реєстрації), скорочення функціональності програми. У цих випадках часто відповідні команди процесору замінюються на байти зі значенням 90h (у шістнадцятковій системі числення), що відповідає асемблерній команді nop (No Operation), тобто «порожній команді», що не виконує жодних дій. Якщо таких команд багато, застосовується безумовний перехід (перестрибування непотрібного коду). Можливе також розширення можливостей програми написанням додаткового

коду, але, як правило, це надто трудомісткий процес, що не виправдовує часових витрат.

Тим часом, патч можливий, як правило, в тому випадку, коли файл програми, що виконується, не захищений спеціальними «пакерами» і «протекторами» – програмами, що приховують реальний код виконуваного файлу. Для останнього типу програм найчастіше використовується найінтелектуальніша частина зворотної розробки – дослідження коду програми за допомогою відладчика та створення генератора ключів, але можливі й інші рішення, наприклад створення завантажувача.

Тобто, саму швидкість розшифрування інформації в наш час визначити не просто, оскільки багато буде залежати від якості знань зловмисника та його підході до взлому. Проте базова ідея наведена на рисунку 2.10.

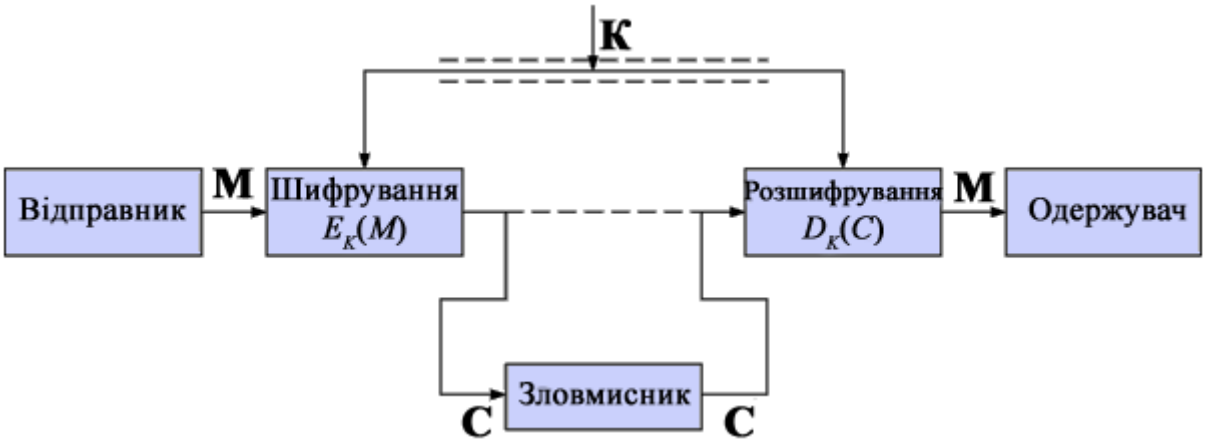

Рисунок 2.10 – Узагальнена схема дій зловмисника.

Звідси, ми можемо вважати, що якщо ми збільшимо кількість місць, які потрібно взломати в етапному порядку, щоб розсекретити інформацію. То ми можемо вивести закономірність, чим більше етапів потрібно пройти в захистів, тим більше пропорційно буде потрачено часу на розшифрування. Тобто маючи 2 етапи шифрування, зловмиснику потрібно буде вдвічі більше часу на розшифрування інформації.

Подальшого розвитку отримав метод стеганографічного захисту даних, у якому, на відміну від існуючих, використано додаткове шифрування даних

методом гамування, що дозволило підвищити рівень захисту даних у 2 рази, та впровадження технології jwt.

2.6 Висновки

1. Запропоновано ієрархічну та модульну модель архітектури системи, що спрощує налагодження та тестування програмного продукту.
2. Подальшого розвитку отримав метод стеганографічного захисту даних, у якому, на відміну від існуючих, використано додаткове шифрування даних методом гамування.
3. Обрано тип та розроблено модель інтерфейсу програмного продукту. Інтерфейс зрозумілий та простий для користувача, що покращить роботу з програмним засобом.
4. Розроблено загальний алгоритм роботи програмного продукту та алгоритм шифрування на основі генератора псевдовипадкових чисел, що дозволило збільшити надійність захисту інформації у 2 рази.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ

3.1 Варіантний аналіз і обґрунтування вибору засобів реалізації програмного засобу.

Велика різнообразність мов програмування та їх потужність в своїх сферах породжує серйозну конкуренцію між ними. Під час вибору мови програмування на якій буде реалізовано продукт, потрібно вибрати максимально потрібні характеристики і по ним проаналізувати існуючі мови програмування. Це допоможе виокремити ту мову програмування, що потрібна саме для реалізації даної задачі.

Мова програмування C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (при фірмі Microsoft) [16].

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції– члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників – мов C++, Delphi, Модула і Smalltalk – C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++).

C# розроблялась як мова програмування прикладного рівня для CLR і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR.

Нижче наведена таблиця 1.1 показує основні критері, які вплинули на вибір мови програмування.

Таблиця 3.1 − Порівняння сучасних мов програмувань

Мова / **Характеристика**	**C#**	**C++**	**Java**
Об'єктно– орієнтована	+	+	+
Garbage Collection	+	–	+
Ручне управління пам’яттю	+	+	–

1. Garbage Collection (одна з форм автоматичного керування оперативною пам'яттю комп'ютера під час виконання програм. Підпрограма – «прибиральник сміття» – вивільняє пам'ять від об'єктів, які не будуть використовуватись програмою в подальшому)

2. Ручне управління пам’яттю – можливість явного виділення і звільнення пам’яті.

3. Мова C# є засобом об'єктного програмування, нової методики проектування та реалізації програм, яка в поточному десятилітті, найімовірніше, замінить звичайне процедурне програмування. Тому за численні переваги також перед мовами Java та C++ було вибрано цю мову для створення програмного продукту.

Отже зваживши всі за і проти, в даному пункті було обрано мовою програмування для нашого програмного продукту мову C#, оскільки вона найбільш вдало може реалізувати поставлену задачу та максимально легко цю реалізацію в подальшому можна удосконалити з можливостями мови програмування C#.

3.2 Вибір середовища розробки

Microsoft Visual Studio – серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight [17].

Dev-C# – вільне інтегроване середовище розробки прикладних програм для програмного забезпечення C / C ++. В дистрибутив входить компілятор MinGW. Сам Dev-C ++ написаний на Delphi. Розповсюджується згідно до GPL.

Проект підтримується SourceForge. Засновник проекту Колін Лаплас, компанія Bloodshed Software. Один час було доступно Linux-порт, проте на сьогодні актуалізована тільки Windows-версія.

C# Builder – програмний продукт, інструмент швидкої розробки додатків (RAD), інтегроване середовище розробки (IDE), система, яка використовується програмістами для розробки програмного забезпечення на мові програмування C++.Спочатку розроблявся компанією Borland Software, а потім її підрозділом CodeGear, який сьогодні належить компанії Embarcadero Technologies. C++ Builder об'єднує в собі комплекс об'єктних бібліотек (STL, VCL, CLX, MFC та ін.), компілятор, зневаджувач, редактор коду та багато інших компонентів. Цикл розробки аналогічний Delphi. Більшість компонентів, розроблених в Delphi, можна використовувати і в C++ Builder без модифікації, але зворотнє твердження не вірне.

NetBeans IDE – вільне інтегроване середовище розробки (IDE) для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python, Groovy. Середовище може бути встановлене і для підтримки окремих мов, і у повній

конфігурації. Проект NetBeans IDE підтримувався і спонсорувався фірмою Sun Microsystems і після придбання Sun – Oracle, проте розробка NetBeans ведеться незалежно співтовариством розробників (NetBeans Community) і компанією NetBeans.Org.

Розробка додатку «Bikod» відбувалася в середовищі MS Visual Studio 2019. За рахунок застосування мови високого рівня C# у даному середовищі було досягнуто найбільш простої та ефективної реалізації даного програмного додатку. У таблиці 3.2 наведено порівняльну характеристику середовищ розробки та виявлено переваги MS Visual Studio 2019.

Таблиця 3.2 Порівняльна характеристика середовищ розробки

Критерій	MS Visual Studio 2019	Dev-C#	Net Beans	C# Builder
Розробка GUI	1	1	0	1
Автодоповнення	1	0	1	1
Статичний аналіз коду	1	1	0	1
Браузер класів	1	1	1	0
Загалом	4	3	2	3

Аналізуючи вищенаведену таблицю, можна зробити висновок, що MS Visual Studio 2019 є найбільш зручним середовищем для розробки додатку. Результати таблиці показують, що в даному середовищі більше функціоналу, ніж в приведених аналогах. Проаналізувавши отримані результати порівняння середовищ програмування було обрано MS Visual Studio 2019.

3.3 Розробка класів для стеганографічного захисту

Відповідно до архітектури додатку було прринято рішення розділити модулі програми. В основі проекту лежать базові принципи KISS and DRY.

Keep It Simple, Stupid – Роби простіше.

Цей принцип було розроблено ВМС США у 1960 році. Цей принцип свідчить, що прості системи працюватимуть краще і надійніше. Цей принцип має багато спільного з перевинуваченням колеса, яким займалися в 1970-х. Тоді він звучав як ділова та рекламна метафора.

Стосовно розробки ПЗ він означає наступне – не вигадуйте до завдання складнішого рішення, ніж їй потрібно. Іноді найрозумніше рішення виявляється найпростішим. Написання продуктивного, ефективного та простого коду – це чудово.

Однією з найпоширеніших помилок нашого часу є використання нових інструментів виключно через те, що вони блищать. Розробників слід мотивувати використовувати новітні технології не тому, що вони нові, а тому, що вони підходять для роботи [18].

Don’t Repeat Yourself – Не повторюйся.

Ця концепція була вперше сформульована у книзі Енді Ханта та Дейва Томаса «Програміст-прагматик: шлях від підмайстра до майстра». Ідея обертається навколо єдиного джерела правди (single source of truth - SSOT).

У проектуванні та теорії інформаційних систем єдине джерело істини (SSOT) – це практика структурування інформаційних моделей та схеми даних, яка передбачає, що всі фрагменти даних обробляються (або редагуються) лише в одному місці. SSOT надають достовірні, актуальні та придатні для використання дані.

Використання SSOT дозволить створити більш міцну та зрозумілу кодову базу. Дублювання коду – марна трата часу та ресурсів. Вам доведеться підтримувати ту саму логіку і тестувати код відразу в двох місцях, причому якщо ви зміните код в одному місці, його потрібно буде змінити і в іншому.

Найчастіше дублювання коду відбувається через незнання системи. Перш ніж щось писати, виявіть прагматизм: огляньтеся. Можливо, ця функція десь реалізована. Можливо, ця бізнес-логіка існує у іншому місці. Повторне використання коду завжди розумне рішення.

Реалізацію класів програмної системи було вирішено проводити відповідно до положень парадигми об'єктно-орієнтованого програмування та принципів SOLID.

Об'єктно-орієнтоване програмування (ООП) – це така парадигма, яка застосовується у галузі програмування, і полягає у тому, що програма розглядається у якості множини «об'єктів», які володіють властивостями, реалізують певну поведінку, а також виконують взаємодію між собою [19]. Експерти визначають три головні концепції, які складають фундамент ООП: успадкування, поліморфізм та інкапсуляція. ООП володіє безліччю переваг, одна з яких, а саме відмінна модульність ПЗ (в ООП десятки класів з власними наборами методів з легкістю можуть виступити у якості заміни тисячам функцій процедурної мови). Поняття об’єкта (певної програмної сутності, яка складається з полів, що володіють даними та методами, що визначають його поведінку) відіграє ключову роль в концепції об'єктно-орієнтованого програмування загалом.

SOLID — це абревіатура, яка складається з п’яти основних принципів проектування, що є основою об'єктно-орієнтованого програмування, таких як Single responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principle й Dependency inversion principle [20].

Дана абревіатура була запропонована Робертом Сесілом Мартіном, якого також часто іменують як «Дядько Боб», який являється автором багатьох книг, надзвичайно відомих та популярних у колах розробників.

Розгорнутий опис принципів SOLID наведений нижче [21].

Принцип єдиної відповідальності полягає у тому, що кожен об’єкт повинен мати лише одну відповідальність, яка неодмінно має бути цілком і

повністю інкапсульована в клас. Всі функції класу повинні бути націлені виключно на забезпечення виконання даної відповідальності.

Принцип відкритості/закритості декларує, що сутності, які є частиною програмної системи, такі як класи, модулі, функції та ін., мають бути відриті для доповнення, масштабування, але закриті для внесення змін. Тобто даний принцип значить, що зазначені програмні сутності можуть змінювати свою поведінку без зміни початкового програмного коду.

Принцип підстановки Барбари Лісков звучить, як функції, які пов’язані з використанням об’єкту певного базового класу, неодмінно зобов’язані мати можливість використовувати класи нащадки даного базового класу не володіючи знаннями про це.

Принцип розділення інтерфейсів в короткому формулюванні визначає, що класи не повинні залежати від методів, які на пряму ними не використовуються. В сутності, даний принцип проголошує наступне твердження, що занадто громіздкі або так звані «товсті» інтерфейси змушені бути розділеними на множину менших, специфічних, вузькоспеціалізованих інтерфейсів, таких щоб класи які реалізують дані маленькі інтерфейси володіли знаннями та реалізовували лише ті методи, від яких залежить безпосереднє виконання їх прямих функцій, зобов’язань. В результаті такого розподілення, при внесенні змін до методу, що належить певному інтерфейсу, така зміна не повинна спричинити редагування класів, які не використовують даний метод.

Принцип інверсії залежностей – принцип, який констатує, що модулі, що знаходяться на вищих рівнях не мають бути залежними від низькорівневих модулів, але у свою чергу обидва типи модулів зобов’язані бути залежними від абстракцій; що стосується абстракцій, то вони не мають залежати від деталей, ба більше самі деталі володіють залежність від абстракцій.

При побудові системи та в процесі реалізації класів ігрового ШІ використовувалися різні патерни проектування, зокрема широко застосовувався породжувальний патерн Singleton (одинак, синглет) [22].

Призначення патерну Singleton, полягає у тому, що часто в системах можуть існувати сутності лише у якості єдиного екземпляра, прикладами таких систем може виступати, зокрема, система ведення системного журналу повідомлень. В таких випадках необхідно вміти створювати єдиний екземпляр певного типу, надавати до нього доступ ззовні й забороняти створення декількох екземплярів цього ж типу.

Саме таке призначення та такі можливості забезпечує патерн Singleton.

Архітектура патерну Singleton заснована на ідеї використання глобальної змінної, яка володіє важливими властивостями, такими як:

– така змінна повинна завжди бути доступною;

– час існування такої змінної визначається часом роботи самої програми, тобто від запуску, а ж до її повного завершення;

– надає глобальний доступ, тобто така змінна повинна бути видима з будь-якої частини програми.

Однак, існує проблема безпосереднього використання глобальної змінної певного типу, адже неможливо забезпечити створення лише єдиного екземпляра.

Патерн Singleton забезпечує розв'язання даної проблеми, шляхом передачі відповідальності за створення єдиного об'єкта безпосередньо на клас. Доступ до об'єкта, у цьому випадку досягається завдяки статичному методу, який виступає членом самого класу і в результаті виконання надає вказівник на цей об’єкт. При такому підході, створення об’єкту відбудеться в ході виконання першого виклику цього методу, а у всіх наступних викликах буде виконуватися лише передача вказівника на нього. Також, додатковим механізмом захисту унікальності об’єкта, виступають модифікатори доступу, які забезпечують закритість конструкторів та операторів присвоєння ззовні.

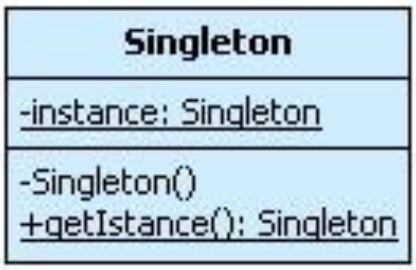

Рисунок 3.7 – UML-діаграма класів патерну Singleton

В результаті проектування та розробки запропоновано ієрархію класів програмної системи, зображену на рисунку 3.8.

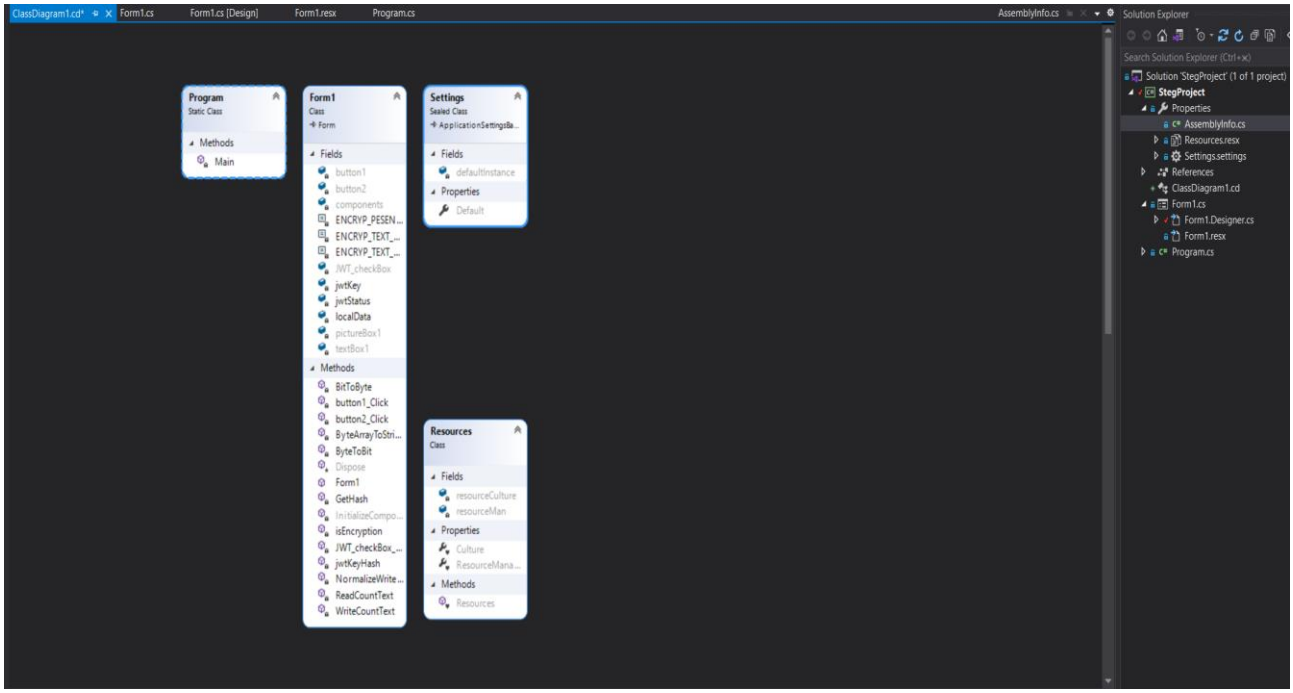

Рисунок 3.8 - Ієрархія класів

4.4 Розробка JWT для забезпечення ключа доступу при стеганографічному захисті

Реалізація надійної стратегії аутентифікації має вирішальне значення, коли йдеться про те, щоб допомогти клієнтам захистити свої мережі від

порушення безпеки. Але для того, щоб стратегія справді була ефективною, потрібне виконання кількох важливих основних умов:

1. Правильний веб-токен. Хоча існує низка веб-токенів, жоден з них не може забезпечити ту ж надійність, яку надає веб-токен JSON (JWT). JWT вважається відкритим стандартом (RFC 7519) передачі конфіденційної інформації між кількома сторонами. Обмін інформацією здійснюється цифровим підписом з використанням алгоритму або поєднання відкритого та закритого ключів для забезпечення оптимальної безпеки.
2. Приватність.
3. Використання HTTPS-з'єднань. HTTPS-з'єднання були побудовані за допомогою протоколів безпеки, що включають шифрування та сертифікати безпеки, призначені для захисту конфіденційних даних. Важливо використовувати HTTPS-з'єднання, а не HTTP або будь-який інший протокол з'єднання під час відправлення токенів, оскільки в іншому випадку зростає ризик перехоплення з боку зловмисника.

JSON Web Token (JWT) – це відкритий стандарт (RFC 7519), який визначає компактний та автономний спосіб безпечної передачі інформації між сторонами у вигляді об'єкта JSON. Цю інформацію можна підтвердити завдяки цифровому підпису. JWT може бути підписаний секретом (за допомогою алгоритму HMAC) або іншим чином, наприклад, за схемами RSA або ECDSA [23].

У своїй компактній формі веб-токени JSON складаються із трьох частин, розділених точками: заголовок, корисне навантаження, підпис. Тому JWT виглядає зазвичай виглядає так: "xxxx.yyyy.zzzz".

Заголовок складається з двох частин: типу токена, яким є JWT, і алгоритму підпису, такого як HMAC SHA256 або RSA.

Друга частина токена - це корисне навантаження, що містить інформацію про користувача та необхідні додаткові дані. Така інформація буває зареєстрованою, публічною та приватною.

Зареєстрована – це набір ключів, які не є обов'язковими, але рекомендуються для забезпечення покращення безпеки. Наприклад, iss – унікальний ідентифікатор сторони, що генерує токен, exp – час у форматі Unix Time, що визначає момент, коли токен стане не валідним, та інші.

Приклад JWT-токена в структиризованій формі наведено на рисунку 3.9,, спрощену версію якого, використона в розбці. Оскількт данний метод не потребує забезпечення HTTPS-з'єднань.

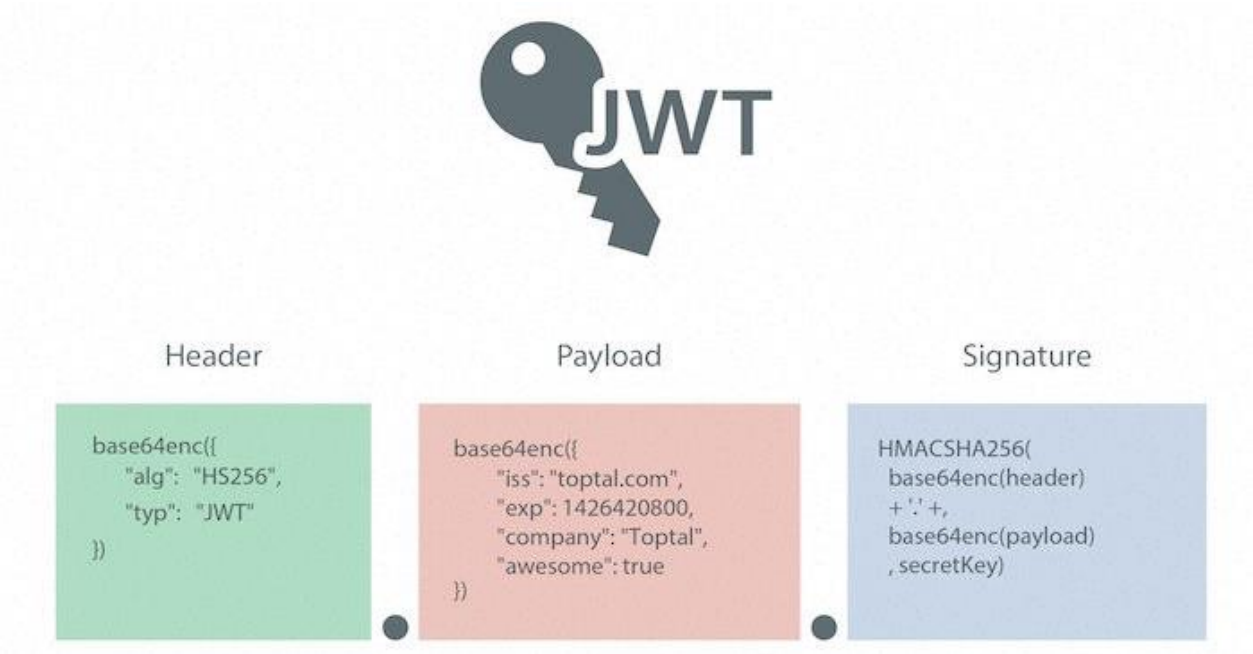

Рисунок 3.9 – Структурований JWT-токен

Вихідні дані є три рядки Base64-URL, розділені точками, які можуть бути легко передані в середовищах HTML і HTTP, будучи при цьому компактнішими порівняно зі стандартами на основі XML, такими як SAML [24].

До переваг використання JWT можна віднести розмір - токени в цій мові коду крихітні і можуть бути передані між двома користувачами досить швидко; простоту - токени можуть бути згенеровані практично з будь-якого місця, і їх не потрібно перевіряти на сервері; контроль - можна вказати, чого користувач

може отримати доступ, як довго триватиме цей дозвіл і що він може робити під час входу в систему.

До мінусів варто віднести лише один ключ - JWT покладається на один ключ, через що вся система опиниться під загрозою у випадку, якщо він буде скомпрометований; складність - JWT токени не так просто зрозуміти, через що, розробник, що не має глибоких знань алгоритмів криптографічного підпису, може ненавмисно поставити систему під загрозу; обмеження – немає можливості надсилати повідомлення всім клієнтам, і неможливо керувати клієнтами з боку сервера.

3.5 Висновки

1. Обґрунтовано вибір мови програмування C# та наведено основні її переваги. Вибрано середовище програмування Microsoft Visual Studio, як найзручніший інструмент для розробки програмного засобу.
2. Мовою програмування C# розроблені програмний засіб приховування даних у файлах зображень у форматах .BMP, .TIFF, .PNG та інших форматах, які не використовують ущільнення з втратами.
3. Розроблено код модулів та реалізована ієрархія класів у відповідності до визначеної архітектури та моделей приховування даних.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

4.1 Аналіз методів та засобів тестування

Процес тестування будь-якого програмного забезпечення – це процес перевірки правильності та відповідності заявлених до певного продукту вимог і реально реалізованої його функціональності. Даний процес здійснюється шляхом спостереження користувачем за роботою в штучно створених умовах та ситуаціях, в деяких випадках на обмеженому списку тестів, обраних попередньо. Техніка для тестування також має включати, як сам процес пошуку помилок роботи, нелогічних відкликів програмного засобу або інших несподіваних дефектів, також проходить випробування програмних компонентів з метою оцінки продукту [25].

Тестування може розпочинатись, як тільки створено продукт чи виконуваний код буде завершений, хоча б на одному етапі, навіть частково. Розробка зазвичай передбачає процес, коли буде планове тестування елементів. Наприклад, якщо проект розбитий поетапному процеси, більшість серйозних тестів проходить лише після визначення потрібних системних вимог та вони будуть реалізовані в тестових програмах.

Проте, буває відповідно до потреб гнучкої розробки програмного продукту, програмування і тестування засобу часто може відбуватись одночасно і доповнювати одне одного.

Тестування може мати такі оцінки:

– відповідність до вимог, які проектувальники та розробники виставили;

– правильна реакція, відповідь для усіх потрібних і можливих вхідних запитів чи даних;

– процеси обробки, робота функцій виконуються за прийнятний час;

– практичність програмного продукту та зручність;

– сумісність продукту з іншим програмним забезпеченням та потрібними операційними системами;

–обов’язкова відповідність результата проекта задачам поставленим замовником [26].

4.2 Тестування програмного продукту

Тестування програмного забезпечення може надавати відносно об'єктивну та незалежну інформацію про якість програмного забезпечення, ризики відмови, як для користувачів, так і для замовників. В наш час тестування програмного забезпечення – один з найбільш дорогих етапів життєвого циклу програмного забезпечення, на нього відводиться від 50% до 65% загальних витрат.

Протестуємо відкриття файлу для шифрування. Для цього необхідно, файл записати потрібну інформацію для закодування (рисунок 4.1).

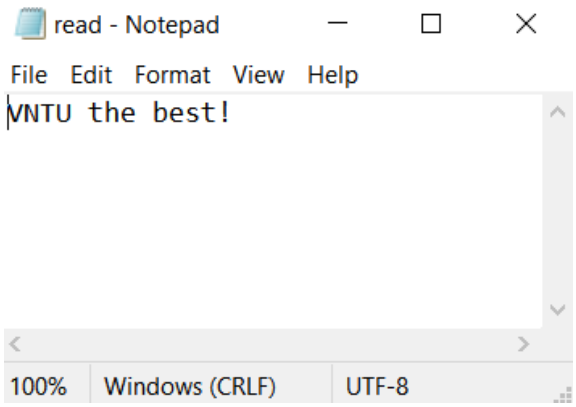

Рисунок 4.1 – Файл з текстом для приховування

Після збереження потрібної інформації в файл для зчитування, запустити програму koders.exe (рисунок 4.2).

Рисунок 4.2 – Вікно програми для приховування

Для початку шифрування та приховування інформації натиснути кнопку «Записати», та вибрати у відкрившомусь вікні файл з потрібним контейнером. (рисунок 4.3).

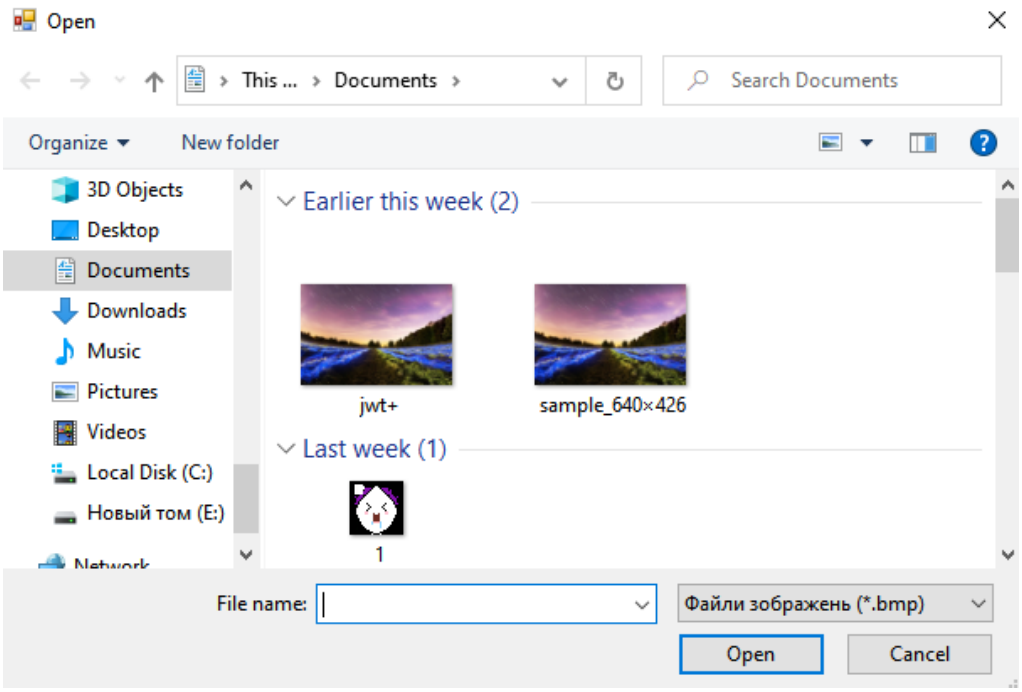

Рисунок 4.3 – Вікно вибору контейнеру для запису

Далі у відкрившомусь вікні вибрати файл з потрібною інформацією (рисунок 4.4).

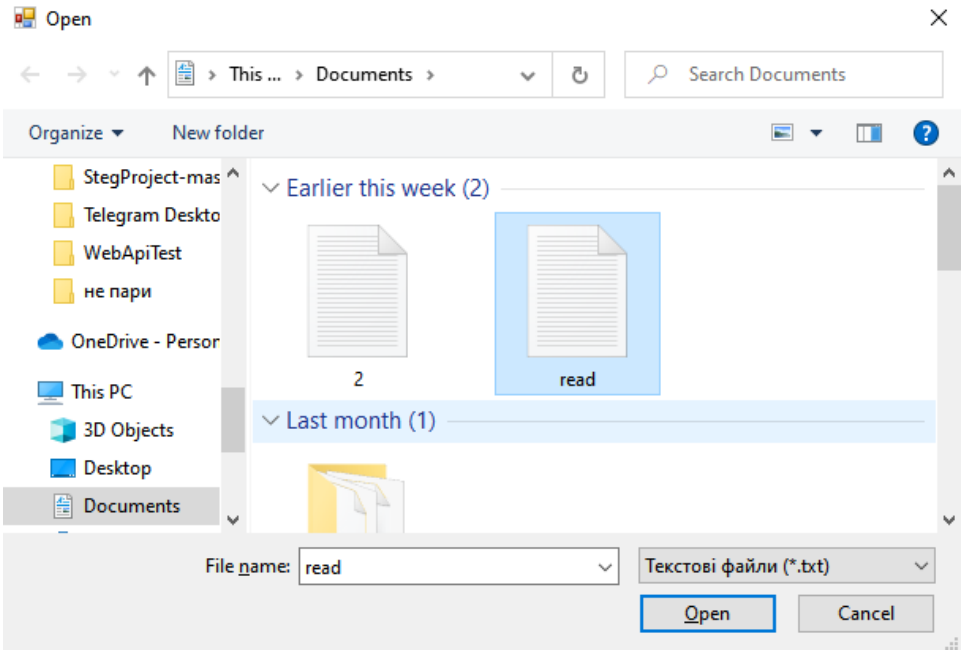

Рисунок 4.4 – Вікно вибору контейнеру для запису

Після процесу шифрування та приховування програма видасть вікно, у відкрившомусь вікні вибрати файл з потрібною інформацією (рисунок 4.5).

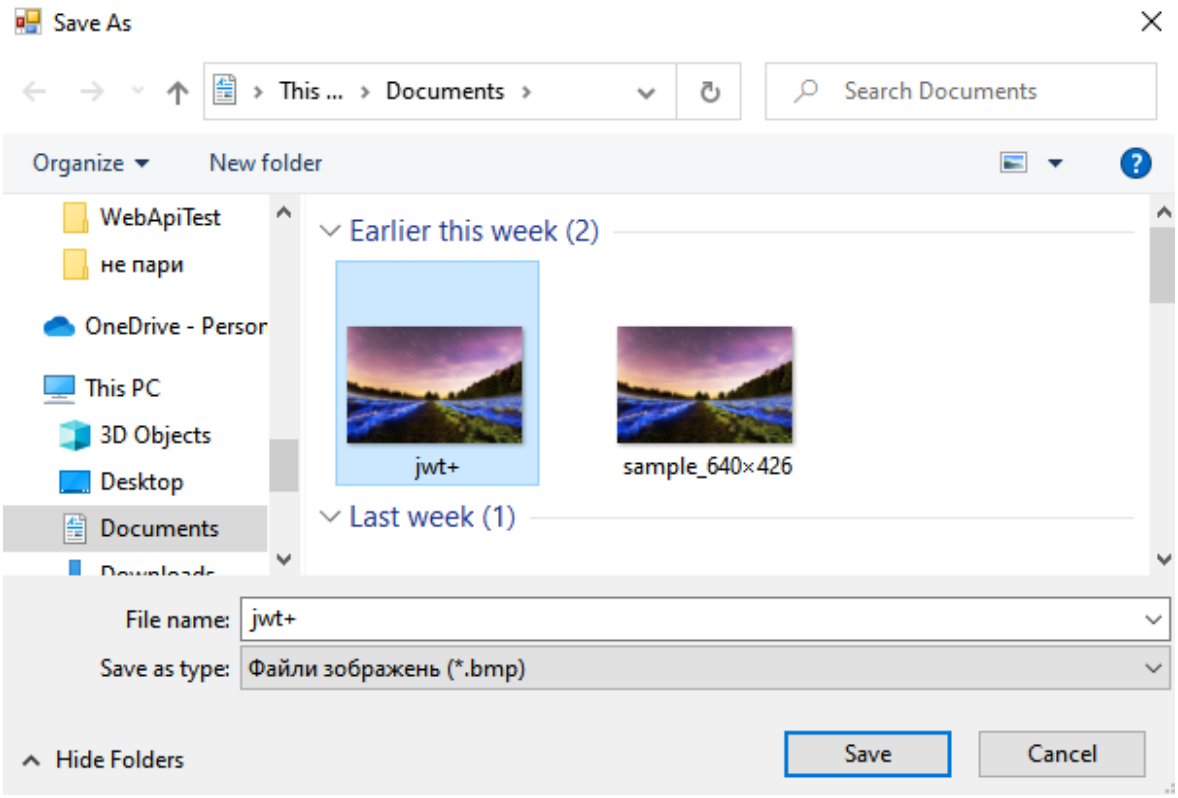

Рисунок 4.5 – Вікно вибору місця збереження файлу

Також у вікні додатку повляється вид закодованого файлу. Що свідчить про вдале приховування інформації (рисунок 4.6).

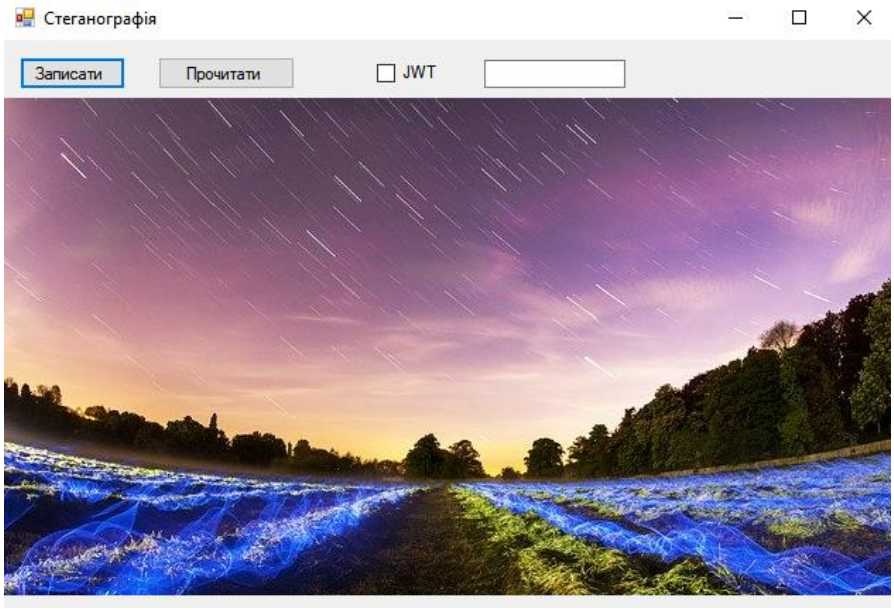

Рисунок 4.6 – Вікно додатку після приховування інформації

Для початку розшифрування та зчитування приховуваної інформації натиснути кнопку «Прочитати», та вибрати у відкрившомусь вікні файл з потрібним контейнером. (рисунок 4.7).

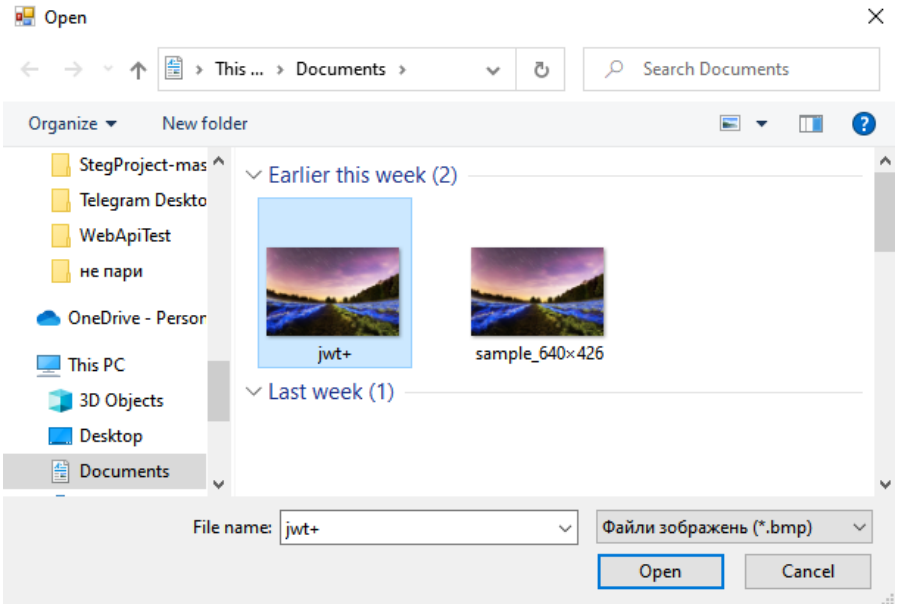

Рисунок 4.7 – Вікно файлу місця для зчитування інформації

Після процесу розшифрування та зчитування приховуваної інформації програма видасть вікно, у відкрившомусь вікні вибрати файл для збреження інформації (рисунок 4.8).

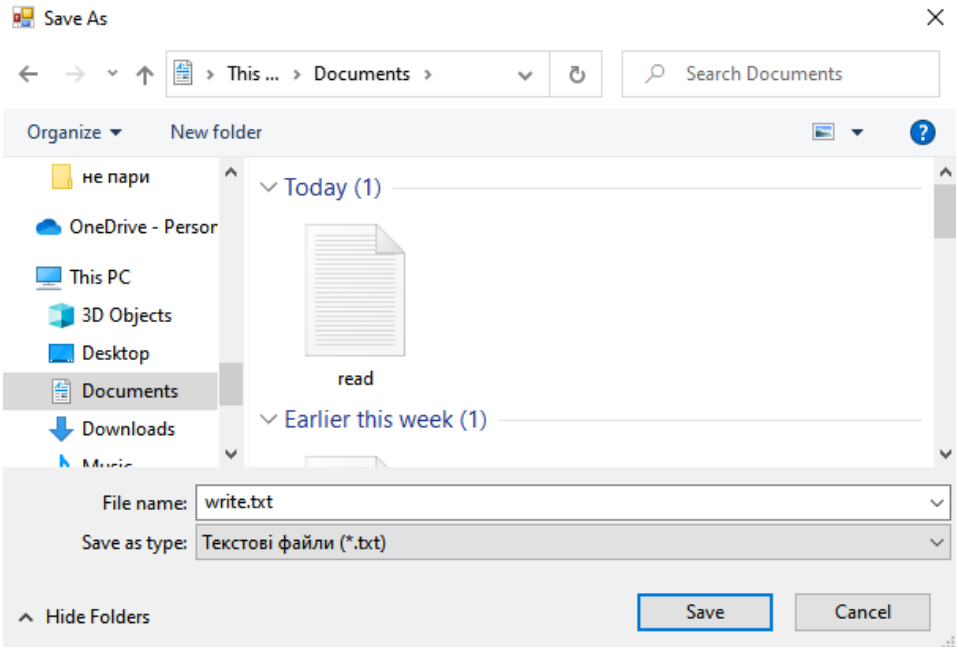

Рисунок 4.8 – Вікно вибору місця для збреження розшифрованої інформації

Проведемо такі ж дії з активованим режимом JWT та введемо додатковий пароль для шифрування (рисунок 4.9).

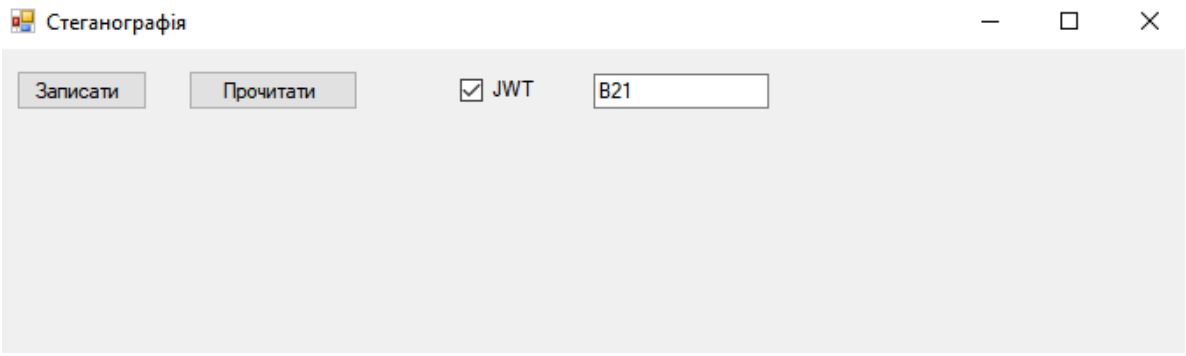

Рисунок 4.9 – Вікно з активованим режимом додаткового шифрування

Після проведеного дешифровування звіремо розшифровану інформацію з оригіналом (рисунок 4.10).

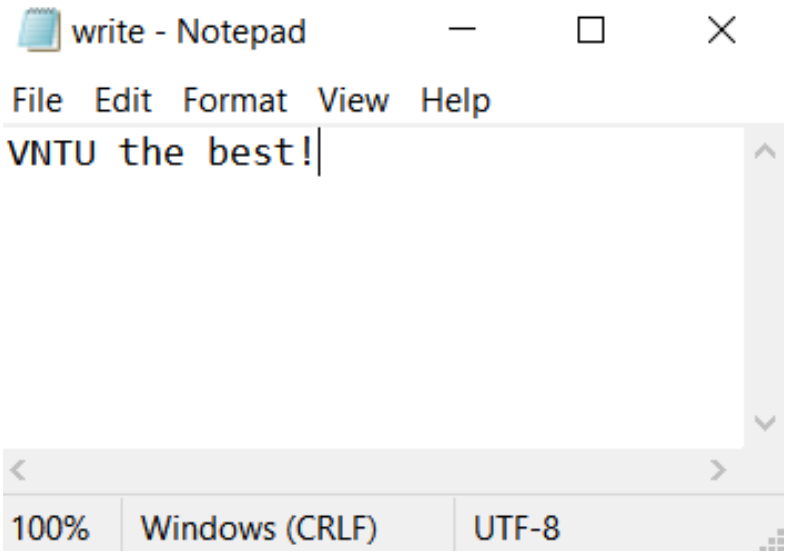

Рисунок 4.10 – Розшифруваний текст з контейнера

Тестування данного додатку виявилось успішним, та показало працеспроможність додатку. Звідси можна стверджувати, що додаток виконує всі поставлені задачі та повноцінно готовий до використання.

4.3 Розробка інструкції користувача

Вимоги до складу і параметрів технічних засобів. Система повинна працювати на IBM-сумісних персональних комп’ютерах. У таблиці 4.1 та 4.2 наведено мінімальну та рекомендовану конфігурацію персонального комп’ютера для запуску програми.

Таким чином можна досягти швидкого та безпечного зашифровування та розшифровування інформації. А також указані конфігурації забезпечують успішне приховування інформації в заданий контейнер ао успішне зчитування з контейнеру інформації з контейнеру, якщо інформація там існує.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 1,3 ГГц
Об’єм оперативної пам’яті	1,5 ГБ для 32-розрядної системи і 3 ГБ для 64-разрядної системи
Розмір жорсткого диску	20 ГБ для 32-розрядної системи і 30 ГБ для 64-розрядної системи
Графічний пристрій	графічний пристрій DirectX10

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 2,5 ГГц
Об’єм оперативної пам’яті	6 ГБ для 32-розрядної системи і 12 ГБ для 64-разрядної системи
Розмір жорсткого диску	35 ГБ для 32-розрядної системи і 55 ГБ для 64-розрядної системи
Графічний пристрій	графічний пристрій DirectX11

Програма повинна працювати під управлінням сімейства операційних систем Windows (7/8/8.1,10).

Так як програма не вимагає інсталяції, то для роботи з програмою достатньо запустити файл koders.exe.

Після цього користувач може запускати і працювати з програмним продуктом за двома можливими варіантами розвитку подій. Для вибору режиму зашифровування та приховування потрібно ввести цифру 1, якщо потрібна обернена операція ввести 2, далі натиснути «ENTER». У відкрившомусь вікну потрібно ввести персональний ключ та натиснути «ENTER». Готове зображення та текстовий файл будуть розміщені в папці з програмою.

Отже, якщо слідувати зазначеній інструкції, програма відпрацює коректно. При виникненні помилок програма сповістить про це відповідним повідомленням у вікні.

4.4 Висновки

1. Розроблено інструкцію користувача для розробленого програмного продукту.
2. Визначено мінімальні системні вимоги для використання, розробленого програмного засобу приховування даних.
3. Виконано тестування програми приховування на відповідність вимогам для усіх можливих вхідних даних. Тестування програми показало повну її працездатність та відповідність поставленому технічному завданню.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки програмного засобі з удосконаленмим методами та засобами стеганографічного захисту конфіденційної інформації. Для проведення технологічного аудиту було залучено 3-х незалежних експертів. Такими експертами будуть Майданюк Володимир Павлович (к.т.н., доц. кафедри ПЗ ВНТУ), Войтко Вікторія Володимирівна (к.т.н., доц. кафедри ПЗ ВНТУ) та Романюк Олександр Никифорович (проф., д.т.н., зав. кафедри ПЗ).

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою за таблицею 4.1.

Таблиця 4.1 – Критерії оцінювання технічного рівня та комерційного потенціалу будь-якої розробки

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження Таблиці 4.1

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
Ринкові перспективи					
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження Таблиці 4.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1. Майданюк В.П.	2. Войтко В. В.	3. Романюк О. Н.
	Бали, виставлені експертами:		
1	3	3	4
2	3	4	4
3	3	4	3
4	4	4	3
5	3	4	4
6	4	3	4
7	3	4	4
8	3	4	3
9	4	4	4
10	3	4	4
11	4	3	3
12	3	4	4
Сума балів	$СБ_1 = 41$	СБ2 = 45	$СБ_2 = 43$
Середньоариф-метична сума балів $\overline{СБ}$			$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} =$ 43

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу. Та затребувана в сфері шифруванні інформації, та прихованої її передачі. Проте також є легко доступною та простою у використанні, що дозволяє бути конкурентноспроможною і на секторі персонального використання. Що сильно збільшує ринок збуту програмного продукту, як для зашифровування інформації, так у персональних цілях з приховування інформації простим користувачем персонального елекронного пристрою.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою 5.1:

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

$Т_р$ - кількість робочих днів у місяці, $Т_р$ = 21 день;

t - число днів роботи розробника, t = 48 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2 – Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	13000	619,05	6	3 714,3
Інженер-програміст	10000	476,19	48	22 857,12
Всього:				26 571,42

Додаткова заробітна плата $З_{дод}$ виконавців роботи розраховується як (10…12)% від величини їх основної заробітної плати, тобто:

$$З_{дод} = 0.1 \cdot З_{заг} \quad (5.2)$$

де $З_{заг}$ – загальна сума заробітної плати.

Розрахуємо додаткову заробітну плату:

$$З_{дод} = 0,1 \cdot 26571,42 = 2657,14 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Нзп}{100}, \quad (5.3)$$

де Зо – заробітну плату дослідника;

Зр – заробітну плату робітника;

Нзп – норма нарахування на заробітну плату.

$$З_н = (3\,714{,}3 + 22\,857{,}12 + 2657{,}14) \cdot \frac{22}{100} = 6\,430{,}28(\text{грн.}).$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А = \frac{Ц}{Тв} \cdot \frac{Т}{12}, \quad (5.4)$$

де Ц – балансова вартість обладнання, грн;

Тв – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Т – Термін використання (Т=2 міс.).

Розрахунки для персонального комп’ютера наведено у таблиці 5.3.

Таблиця 5.3 – Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Термін корисного використання, років	Термін використання, міс.	Величина амортизаційних відрахувань,грн
Персональний комп’ютер	70000	2	2	5 833,3
Всього:				5 833,3

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n Н_i \cdot Ц_i \cdot К_i , \quad (5.5)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	200	1	200
Пачка паперу	уп.	120	1	120
Ручка	шт.	5	1	5
Всього з урахуванням транспортних витрат				357,5

Витрати на силову електроенергію розраховуються за формулою 5.5:

$$B_e = \sum_{i=1}^{n} \frac{Wyi \cdot ti \cdot Це \cdot Квпі}{\eta_i} ; \quad (5.6)$$

Де Wyi – встановлена потужність обладнання на певному етапі розробки, кВт (Wyi = 0,6кВт)

ti – тривалість роботи обладнання на етапі дослідження, год (ti = 200 год);

Це – вартість 1 кВт-години електроенергії, грн (Це = 4,1 грн/кВт);

Квпі – коефіцієнт, що враховує використання потужності, Квпі <1 ($К_{впі} = 0,8$);

ηі – коефіцієнт корисної дії обладнання, ηі<1 ($ηі = 0,8$).

$$В_е = \frac{4,1 \cdot 0,6 \cdot 200 \cdot 0,8}{0,8} = 492 \text{ (грн.)}$$

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, працівників організацій, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 5.6.

$$В_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%} \quad (5.7)$$

$$В_{св} = (3\,714,3 + 22\,857,12) \cdot \frac{20}{100\%} = 5\,314,3$$

До статті «Інші витрати» належать витрати, які не знайшли відо-браження у зазначених статтях витрат і можуть бути віднесені безпосеред-ньо на собівартість досліджень за прямими ознаками. Витрати розраховуються як 50…100% від суми основної заробітної плати дослідників (форумла 5.7).

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%} \quad (5.8)$$

$$І_в = (3\,714,3 + 22\,857,12) \cdot \frac{50}{100\%} = 13\,285,74$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$Взаг = З_о + З_р + З_{дод} + З_н + А + К + В_е + В_{св} + І_в$$

$$В_{заг} = 3\ 714{,}3 + 22\ 857{,}12 + 2657{,}14 + 6\ 430{,}28 + 5\ 833{,}3 + 357{,}5 + 492 +$$
$$+ 5\ 314{,}3 + 13\ 285{,}74 = 60941{,}68 \text{ (грн)}$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою 5.8:

$$ЗВ = \frac{Взаг}{\alpha} \quad (5.9)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 0,9.

$$ЗВ = \frac{60941{,}68}{0{,}9} = 54\ 847{,}51 \text{ (грн.)}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення коштів у розро-бку до отримання прибутку минає чимало часу.

Збільшення чистого прибутку підприємства $\Delta П_і$ для кожного із років, протягом яких очікується отримання по-зитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою (5.9):

$$\Delta П_i = \Delta П_я \cdot N + П_я \Delta N_i \,, \quad (5.10)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки в аналізованому році;

N – основний кількісний показник, який визначає обсяг діяльності підприємства у році до впровадження результатів наукової розробки;

ΔN – зміна основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технології зменшаться на 15 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 15 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 300 користувачів, протягом другого року – на 250 користувачів, протягом третього року – 175 користувачів. Реалізація інформаційної технології до впровадження результатів наукової розробки складала 800 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 250 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_{2021} = 15 \cdot 800 + (250 + 15) \cdot 300 = 91500 \text{ грн.}$$

Протягом другого року:

$$\Delta П_{2022} = 15 \cdot 800 + (250 + 15) \cdot (300 + 250) = 157750 \text{ грн.}$$

Протягом третього року:

$$\Delta П_{2023} = 15 \cdot 800 + (250 + 15) \cdot (300 + 250 + 175) = 204125 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$вкладених інвестицій розраховується за формулою:

$$Е_{абс} = (ПП - PV), \quad (5.11)$$

де ПП – приведена вартість збільшення всіх чистих прибутків від можливого впровадження науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{інв} \cdot ЗВ, \quad (5.12)$$

$k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{інв}$ = 2…5).

$$PV = 2 \cdot 54847{,}51 = 109\,695{,}02$$

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^Т \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.13)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

t – період часу (в роках) від моменту початку впровадження науковотехнічної розробки до моменту отримання підприємством збільшеної величини чистого прибутку в аналізованому році.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{91500}{(1+0,2)^2} + \frac{157750}{(1+0,3)^3} + \frac{204125}{(1+0,4)^4} = 323\,090,21 \text{ (грн.)}$$

Тоді розрахуємо $E_{абс}$:

$$Е_{абс} = 323\,090,21 - 109\,695,02 = 213\,395,19 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$Е_в = \sqrt[Т]{1 + \frac{Е_{абс}}{PV}} - 1\ , \qquad (5.14)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$Е_в = \sqrt[3]{1 + \frac{213\,395,19}{109\,695,02}} - 1 = 0,53 \text{ або } 53\ \%$$

Далі, розраховану величина $Е_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f, \qquad (5.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0{,}2 + 0{,}1 = 0{,}3$$

Оскільки $E_в$ = 53% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в} \quad (5.16)$$

$$T_{ок} = \frac{1}{0{,}53} = 1{,}88 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

5.5 Висновки

1. В результаті оцінки комерційного потенціалу розробки можна однозначно стверджувати, що розробка має високий рівень комерційного потенціалу.

2. Виконано розрахунок витрат, необхідних на проведення на науково-дослідної та конструкторсько-технологічної роботи, що становить 60941,68 грн.

3. Терміну окупності розробки складає 1,88 року, що вказує на доцільність фінансування такої наукової розробки.

ВИСНОВКИ

У магістерській кваліфікаційній роботі розроблено систему для стеганографічного захисту даних конфіденційної інформації. Основні результати отримані при виконанні даної магістерської кваліфікаційної роботи такі:

1. Аналіз стану захисту інформації в комп'ютерних системах від несанкціонованого доступу показав, що розробка програмного продукту приховування даних у файлах мультимедіа є актуальною та доцільною, оскільки більшість систем захисту орієнтовані на парольний або криптографічний захист.
2. Порівняльний аналіз методів приховування інформації в контейнерах показав, що найбільш придатним типом контейнера є зображення, які дозволяють приховувати інформацію розміром до 25 % від розміру контейнера.
3. Визначено та розроблено перелік основних завдань, які необхідні для розробки програмного продукту.
4. Запропоновано ієрархічну та модульну модель архітектури системи, що спрощує налагодження та тестування програмного продукту.
5. Подальшого розвитку отримав метод стеганографічного захисту даних, у якому, на відміну від існуючих, використано додаткове шифрування даних методом гамування.
6. Обрано тип та розроблено модель інтерфейсу програмного продукту. Інтерфейс зрозумілий та простий для користувача, що покращить роботу з програмним засобом.
7. Розроблено загальний алгоритм роботи програмного продукту та алгоритм шифрування на основі генератора псевдовипадкових чисел, що дозволило збільшити надійність захисту інформації у 2 рази.

8. Обґрунтовано вибір мови програмування C# та наведено основні її переваги. Вибрано середовище програмування Microsoft Visual Studio, як найзручніший інструмент для розробки програмного засобу.
9. Мовою програмування C# розроблені програмний засіб приховування даних у файлах зображень у форматах .BMP, .TIFF, .PNG та інших форматах, які не використовують ущільнення з втратами.
10. Розроблено код модулів та реалізована ієрархія класів у відповідності до визначеної архітектури та моделей приховування даних.
11. Розроблено інструкцію користувача для розробленого програмного продукту.
12. Визначено мінімальні системні вимоги для використання, розробленого програмного засобу приховування даних.
13. Виконано тестування програми приховування на відповідність вимогам для усіх можливих вхідних даних. Тестування програми показало повну її працездатність та відповідність поставленому технічному завданню.
14. В результаті оцінки комерційного потенціалу розробки можна однозначно стверджувати, що розробка має високий рівень комерційного потенціалу.
15. Виконано розрахунок витрат, необхідних на проведення на науково-дослідної та конструкторсько-технологічної роботи.
16. Терміну окупності розробки складає 1,09 року, що вказує на доцільність фінансування такої наукової розробки.

Результати роботи доповідалися на науково-технічних конференціях та відображені в трьох наукових публікаціях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Білоконь В.В., Майданюк В.П. Розробка програмного забезпечення для стеганографісного захисту даних. Збірник матеріалів IX Міжнародної науково-практичної конференції молодих вчених та студентів «Молодь у світі сучасних технологій» – Херсон : МССТ, 2020р. С. 115.

2. Білоконь В.В., Майданюк В.П. Методи та засоби стеганографічного захисту конфіденційної інформації. Збірник матеріалів XIII Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» – Суми/Вінниця : НІКО/ВНТУ, 9-10 листопада 2021р. С. 107.

3. Білоконь В.В., Майданюк В.П. Стеганографічний захист даних з використанням файлів зображень. Збірник матеріалів «Матеріали L науково-технічної конференції підрозділів Вінницького національного технічного університету – Вінниця : НТКП/ВНТУ, 10-12 березня 2021р. С. 778.

4. Захист інформації в інформаційних системах [Електронний ресурс] URL:https://pidruchniki.com/13670622/informatika/zahist_informatsiyi_informatsiynih_sistemah.

5. Загальні особливості автоматизованих інформаційних систем [Електронний ресурс] URL: https://studopedia.info/6-19530.html.

6. Recolition [Електронний ресурс] URL: https://soft.mydiv.net/win/download-Recolition.html.

7. R-Crypto 1.5 Build 3346 [Електронний ресурс] URL: https://soft.mydiv.net/win/download-R-Crypto.html.

8. TrustPort Tools [Електронний ресурс] URL: https://soft.mydiv.net/win/download-TrustPort-Tools.html.

9. Каплун В. А. Захист операційних систем. Навчальний посібник / В. А.Каплун, В. П Майданюк - Вінниця: ВНТУ, 2007. – 185 с.

10. Стеганографічний алгоритм захисту [Електронний ресурс] URL: http://www.economy.nayka.com.ua/?op=1&z=5584.

11. Архитектура программного обеспечения. [Електронний ресурс] URL: https://ru.wikipedia.org/wiki/Архитектура_программного_обеспечения

12. Монолитная vs Микросервисная архітектура. [Електронний ресурс] URL: https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16

13. Шифр XOR [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Шифр_XOR.

14. Дудатьєв А.В. Захист програмного забезпечення. Частина 1. Навчальний посібник./ А.В. Дудатьєв, В.А. Каплун, В.П. Семеренко -Вінниця: ВНТУ, 2005. – 140 с.

15. Поняття інтерфейсу [Електронний ресурс] URL: https://allcompositions.at.ua/temy_1-2.pdf.

16. Мова програмування C# [Електронний ресурс] URL: http://www.znannya.org/?view=csharp.

17. NET Framework. URL: https://uk.wikipedia.org/wiki/.NET_Framework.

18. Общие сведения о платформе .NET. [Електронний ресурс] URL: https://docs.microsoft.com/ru-ru/dotnet/framework/get-started/overview

19. Объектно-ориентированное программирование. [Електронний ресурс] URL: https://ru.wikipedia.org/wiki/Объектно-ориентированое_программирование

20. Архітектура та проектування програмного забезпечення. [Електронний ресурс] URL: http://dspace.wunu.edu.ua/bitstream/316497/24194/1/опорний%20конспект%20лекцій.pdf.

21. SOLID – принципы объектно-ориентированного программирования. [Електронний ресурс] URL: https://web-creator.ru/articles/solid

22. Паттерн Singleton (одиночка,синглет). [Електронний ресурс] URL: http://cpp-reference.ru/patterns/creational-patterns/singleton/

23. Майданюк В. П. Кодування та захист інформації. Навчальний посібник. / В. П. Майданюк Вінниця: ВНТУ, 2009. – 164 с.

24. Алгоритм Лемпеля — Зива — Велча. [Електронний ресурс] URL: https://ru.wikipedia.org/wiki/Алгоритм_Лемпеля_Зива_Велча.

25. Тестування програмного продукту [Електронний ресурс] URL: http://lib.mdpu.org.ua/e-book/vstup/L11.htm.

26. Рівні тестування. [Електронний ресурс] URL: https://qalearning.com.ua/theory/lectures/material/testing-levels/

ДОДАТКИ

ДОДАТОК А

(обов’язковий)

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 13 » вересня 2021 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Методи та засоби стеганографічного захисту конфіденційної інформації» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доцент Майданюк В.П.

"____"_____2021 р.

Виконав:

_____студент гр.1ПІ-20м Білоконь В.В.

"____"_____2021 р.

Вінниця – 2021 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Методи та засоби стеганографічного захисту конфіденційної інформації».

Галузь застосування - системи захисту інформації.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 277 від 24 вересня 2021 р. ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення рівня захисту даних від несанкціонованого доступу за рахунок додаткового шифрування даних перед приховуванням.

Призначення роботи – розробка методів і засобів приховування даних стеганографічним способом та збільшення безпеки за допомогою шифрування інформації.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР:

1. Білоконь В.В., Майданюк В.П. Розробка програмного забезпечення для стеганографісного захисту даних. Збірник матеріалів IX Міжнародної науково-практичної конференції молодих вчених та студентів «Молодь у світі сучасних технологій» – Херсон : МССТ, 2020р. С. 115.
2. Білоконь В.В., Майданюк В.П. Методи та засоби стеганографічного захисту конфіденційної інформації. Збірник матеріалів XIII Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» – Суми/Вінниця : НІКО/ВНТУ, 9-10 листопада 2021р. С. 107.

3. Білоконь В.В., Майданюк В.П. Стеганографічний захист даних з використанням файлів зображень. Збірник матеріалів «Матеріали L науково-технічної конференції підрозділів Вінницького національного технічного університету – Вінниця : НТКП/ВНТУ, 10-12 березня 2021р. С. 778.
4. Дудатьєв А.В. Захист програмного забезпечення. Частина 1. Навчальний посібник./ А.В. Дудатьєв, В.А. Каплун, В.П. Семеренко -Вінниця: ВНТУ, 2005. – 140 с.
5. Майданюк В. П. Кодування та захист інформації. Навчальний посібник. / В. П. Майданюк Вінниця: ВНТУ, 2009. – 164 с.

5. Технічні вимоги

Аналіз стану забезпечення захисту інформації; порівняльний аналіз аналогів; методи приховування інформації в цифрових контейнерах; аналіз існуючих методів кодування; аналіз методів стеганографічного захисту даних в зображенні; постановка задач розробки; розробка інтерфейсу програми Bikod; розробка структури інтерфейсу для програмного засобу; розробка алгоритмів роботи програми; варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу; розробка блоку шифрування даних; розробка блоку стеганографічного захисту; тестування програмного забезпечення; розробка інструкції користувача.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинг коду.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання і вибір методів його вирішення	14.09.2020 р. - 01.10.2020 р.	Вик.
2	Аналіз існуючих аналогів та постановка задач дослідження	02.10.2020 р.– 10.10.2020 р.	Вик.
3	Розробка методу та моделей системи	11.10.2020 р. - 20.10.2020 р.	Вик.
4	Розробка структур і алгоритмів програмного продукту	21.10.2020 р. - 5.11.2020 р.	Вик.
5	Розробка програмного забезпечення	6.11.2020 р. - 15.11.2020 р.	Вик.
6	Тестування розробленого програмного продукту	16.11.2020 р. - 20.11.20202 р.	Вик.
7	Економічна частина	21.11.2020 р. - 28.11.2020 р.	Вик.
8	Оформлення матеріалів до захисту	29.11.2020 р. - 30.11.2020 р.	Вик.

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б

(обов’язковий)

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Методи та засоби стеганографічного захисту конфіденційної інформації**.**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 20м

Науковий керівник: к.т.н. доц. Майданюк В.П.

Unicheck	
Оригінальність	**83,1%**
Схожість	16,9%

Аналіз звіту подібності

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

 Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

 Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлена з повним звітом подібності, який був згенерований Системою щодо роботи «Методи та засоби стеганографічного захисту конфіденційної інформації».

Автор _____ Білоконь Владислав Васильович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали, посада)

ДОДАТОК В

(обов’язковий)

Лістинг програмних модулів

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;
using System.Security.Cryptography;

namespace StegProject
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      const int ENCRYP_PESENT_SIZE = 1;
      const int ENCRYP_TEXT_SIZE = 3;
      const int ENCRYP_TEXT_MAX_SIZE = 999;
      bool jwtStatus;
      string jwtKey;
      string localData;

      private BitArray ByteToBit(byte src) {
```

```
        BitArray bitArray = new BitArray(8);
        bool st = false;
        for (int i = 0; i < 8; i++)
        {
            if ((src >> i & 1) == 1) {
                st = true;
            } else st = false;
            bitArray[i] = st;
        }
        return bitArray;
    }

    private byte BitToByte(BitArray scr) {
        byte num = 0;
        for (int i = 0; i < scr.Count; i++)
            if (scr[i] == true)
                num += (byte)Math.Pow(2, i);
        return num;
    }
```

/*Проверяет, зашифрован ли файл, возвраещает true, если символ в первом пикслеле равен / иначе false */

```
    private bool isEncryption(Bitmap scr)
    {
        byte[] rez = new byte[1];
        Color color = scr.GetPixel(0, 0);
        BitArray colorArray = ByteToBit(color.R); //получаем байт цвета и преобразуем в массив бит
        BitArray messageArray = ByteToBit(color.R); ;//инициализируем результирующий массив бит
        messageArray[0] = colorArray[0];
        messageArray[1] = colorArray[1];

        colorArray = ByteToBit(color.G);//получаем байт цвета и преобразуем в массив бит
```

```
            messageArray[2] = colorArray[0];
            messageArray[3] = colorArray[1];
            messageArray[4] = colorArray[2];

            colorArray = ByteToBit(color.B);//получаем байт цвета и преобразуем в массив бит
            messageArray[5] = colorArray[0];
            messageArray[6] = colorArray[1];
            messageArray[7] = colorArray[2];
            rez[0] = BitToByte(messageArray); //получаем байт символа, записанного в 1 пикселе
            string m = Encoding.GetEncoding(1251).GetString(rez);
            if (m == "/")
            {
                return true;
            }
            else return false;
        }

        /*Нормализует количество символов для шифрования,чтобы они всегда занимали
ENCRYP_TEXT_SIZE байт*/
        private byte[] NormalizeWriteCount(byte[] CountSymbols)
        {
            int PaddingByte = ENCRYP_TEXT_SIZE - CountSymbols.Length;

            byte[] WriteCount = new byte[ENCRYP_TEXT_SIZE];

            for (int j = 0; j < PaddingByte; j++)
            {
                WriteCount[j] = 0x30;
            }

            for (int j = PaddingByte; j < ENCRYP_TEXT_SIZE; j++)
            {
                WriteCount[j] = CountSymbols[j - PaddingByte];
            }
```

```
    return WriteCount;
}

/*Записыает количество символов для шифрования в первые биты картинки */
private void WriteCountText(int count, Bitmap src) {
    byte[] CountSymbols = Encoding.GetEncoding(1251).GetBytes(count.ToString());

    if (CountSymbols.Length < ENCRYP_TEXT_SIZE)
    {
        CountSymbols = NormalizeWriteCount(CountSymbols);
    }

    for (int i = 0; i < ENCRYP_TEXT_SIZE; i++)
    {
        BitArray bitCount = ByteToBit(CountSymbols[i]); //биты количества символов
        Color pColor = src.GetPixel(0, i + 1);
        BitArray bitsCurColor = ByteToBit(pColor.R); //бит цветов текущего пикселя
        bitsCurColor[0] = bitCount[0];
        bitsCurColor[1] = bitCount[1];
        byte nR = BitToByte(bitsCurColor); //новый бит цвета пиксея

        bitsCurColor = ByteToBit(pColor.G);//бит бит цветов текущего пикселя
        bitsCurColor[0] = bitCount[2];
        bitsCurColor[1] = bitCount[3];
        bitsCurColor[2] = bitCount[4];
        byte nG = BitToByte(bitsCurColor);//новый цвет пиксея

        bitsCurColor = ByteToBit(pColor.B);//бит бит цветов текущего пикселя
        bitsCurColor[0] = bitCount[5];
        bitsCurColor[1] = bitCount[6];
        bitsCurColor[2] = bitCount[7];
        byte nB = BitToByte(bitsCurColor);//новый цвет пиксея

        Color nColor = Color.FromArgb(nR, nG, nB); //новый цвет из полученных битов
```

```
            src.SetPixel(0, i + 1, nColor); //записали полученный цвет в картинку
        }
    }

    /*Читает количество символов для дешифрования из первых бит картинки*/
    private int ReadCountText(Bitmap src) {
        byte[] rez = new byte[ENCRYP_TEXT_SIZE];
        for (int i = 0; i < ENCRYP_TEXT_SIZE; i++)
        {
            Color color = src.GetPixel(0, i + 1);
            BitArray colorArray = ByteToBit(color.R); //биты цвета
            BitArray bitCount = ByteToBit(color.R); ; //инициализация результирующего массива
бит
            bitCount[0] = colorArray[0];
            bitCount[1] = colorArray[1];

            colorArray = ByteToBit(color.G);
            bitCount[2] = colorArray[0];
            bitCount[3] = colorArray[1];
            bitCount[4] = colorArray[2];

            colorArray = ByteToBit(color.B);
            bitCount[5] = colorArray[0];
            bitCount[6] = colorArray[1];
            bitCount[7] = colorArray[2];
            rez[i] = BitToByte(bitCount);
        }
        string m = Encoding.GetEncoding(1251).GetString(rez);
        return Convert.ToInt32(m, 10);
    }

    /* Открыть файл для шифрования */
    private void button1_Click(object sender, EventArgs e)
    {
```

```
        string FilePic;
        string FileText;
        OpenFileDialog dPic = new OpenFileDialog();
        dPic.Filter = "Файли зображень (*.bmp)|*.bmp|Всі файли (*.*)|*.*";
        if (dPic.ShowDialog() == DialogResult.OK)
        {
           FilePic = dPic.FileName;
        }
        else
        {
           FilePic = "";
           return;
        }

        FileStream rFile;
        try
        {
           rFile = new FileStream(FilePic, FileMode.Open); //відкриваємо поток
        }
        catch (IOException)
        {
           MessageBox.Show("Помилка відкриття файла", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
           return;
        }
        Bitmap bPic = new Bitmap(rFile);

        OpenFileDialog dText = new OpenFileDialog();
        dText.Filter = "Текстові файли (*.txt)|*.txt|Всі файли (*.*)|*.*";
        if (dText.ShowDialog() == DialogResult.OK)
        {
           FileText = dText.FileName;
        }
```

```
        else
        {
            FileText = "";
            return;
        }
        if (jwtStatus)
        {
            FileStream jwtText;
            try
            {
                jwtText = new FileStream(FileText, FileMode.Open); //відкриваємо поток
            }
            catch (IOException)
            {
                MessageBox.Show("Помилка відкриття файла", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            jwtKeyHash();
            jwtKey = textBox1.Text;
            byte[] array = new byte[jwtText.Length];
            // считываем данные
            jwtText.Read(array, 0, array.Length);
            // декодируем байты в строку
            localData = Encoding.Default.GetString(array);
            string textFromFile = jwtKey + localData;
            jwtText.Close();
            StreamWriter wText;
            try
            {
                wText = new StreamWriter(FileText, false, Encoding.Default);
            }
            catch (IOException)
            {
```

```
            MessageBox.Show("Помилка відкриття файла", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        wText.Write(textFromFile);
        wText.Close();

    }

    FileStream rText;
    try
    {
        rText = new FileStream(FileText, FileMode.Open); //відкриваємо поток
    }
    catch (IOException)
    {
        MessageBox.Show("Помилка відкриття файла", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }

    BinaryReader bText = new BinaryReader(rText, Encoding.ASCII);

    List<byte> bList = new List<byte>();
    while (bText.PeekChar() != -1) { //считали весь текстовый файл для шифрования в лист
байт
        bList.Add(bText.ReadByte());
    }
    int CountText = bList.Count; // в CountText - количество в байтах текста, который
нужно закодировать
    bText.Close();
    rFile.Close();
```

```
        //проверям, что размер не выходит за рамки максимального, поскольку для хранения
размера используется
        //ограниченное количество байт
        if (CountText > (ENCRYP_TEXT_MAX_SIZE - ENCRYP_PESENT_SIZE -
ENCRYP_TEXT_SIZE)) {
            MessageBox.Show("Розмір тексту великий для даного алгоритма, уменьшіть
розмір", "Інформація", MessageBoxButtons.OK);
            return;
        }

        //проверяем, поместится ли исходный текст в картинке
        if (CountText > (bPic.Width * bPic.Height)) {
            MessageBox.Show("Вибранна картинка занадто мала для разміщення вибраного
тексту", "Інформація", MessageBoxButtons.OK);
            return;
        }

        //проверяем, может быть картинка уже зашифрована
        if (isEncryption(bPic))
        {
            MessageBox.Show("Файл уже зашифровано", "Информация",
MessageBoxButtons.OK);
            return;
        }

        byte [] Symbol = Encoding.GetEncoding(1251).GetBytes("/");
        BitArray ArrBeginSymbol = ByteToBit(Symbol[0]);
        Color curColor = bPic.GetPixel(0, 0);
        BitArray tempArray = ByteToBit(curColor.R);
        tempArray[0] = ArrBeginSymbol[0];
        tempArray[1] = ArrBeginSymbol[1];
        byte nR = BitToByte(tempArray);

        tempArray = ByteToBit(curColor.G);
```

```
tempArray[0] = ArrBeginSymbol[2];
tempArray[1] = ArrBeginSymbol[3];
tempArray[2] = ArrBeginSymbol[4];
byte nG = BitToByte(tempArray);

tempArray = ByteToBit(curColor.B);
tempArray[0] = ArrBeginSymbol[5];
tempArray[1] = ArrBeginSymbol[6];
tempArray[2] = ArrBeginSymbol[7];
byte nB = BitToByte(tempArray);

Color nColor = Color.FromArgb(nR, nG, nB);
bPic.SetPixel(0, 0, nColor);
//то есть в первом пикселе будет символ /, который говорит о том, что картика зашифрована

WriteCountText(CountText, bPic); //записываем количество символов для шифрования

int index = 0;
bool st = false;
for (int i = ENCRYP_TEXT_SIZE + 1; i < bPic.Width; i++) {
   for (int j = 0; j < bPic.Height; j++) {
      Color pixelColor = bPic.GetPixel(i, j);
      if (index == bList.Count) {
         st = true;
         break;
      }
      BitArray colorArray = ByteToBit(pixelColor.R);
      BitArray messageArray = ByteToBit(bList[index]);
      colorArray[0] = messageArray[0]; //меняем
      colorArray[1] = messageArray[1]; // в нашем цвете биты
      byte newR = BitToByte(colorArray);

      colorArray = ByteToBit(pixelColor.G);
```

```
            colorArray[0] = messageArray[2];
            colorArray[1] = messageArray[3];
            colorArray[2] = messageArray[4];
            byte newG = BitToByte(colorArray);

            colorArray = ByteToBit(pixelColor.B);
            colorArray[0] = messageArray[5];
            colorArray[1] = messageArray[6];
            colorArray[2] = messageArray[7];
            byte newB = BitToByte(colorArray);

            Color newColor = Color.FromArgb(newR, newG, newB);
            bPic.SetPixel(i, j, newColor);
            index ++;
        }
        if (st) {
            break;
        }
    }
    pictureBox1.Image = bPic;

    String sFilePic;
    SaveFileDialog dSavePic = new SaveFileDialog();
    dSavePic.Filter = "Файли зображень (*.bmp)|*.bmp|Всі файли (*.*)|*.*";
    if (dSavePic.ShowDialog() == DialogResult.OK)
    {
        sFilePic = dSavePic.FileName;
    }
    else
    {
        sFilePic = "";
        return;
    };
```

```
        FileStream wFile;
        try
        {
            wFile = new FileStream(sFilePic, FileMode.Create); //відкриваємо поток на запись
результатов
        }
        catch (IOException)
        {
            MessageBox.Show("Помилка открытия файла на запись", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        bPic.Save(wFile, System.Drawing.Imaging.ImageFormat.Bmp);
        wFile.Close(); //закрываем поток


        StreamWriter cleaner = new StreamWriter(FileText, false, Encoding.Default);
        cleaner.Write(localData);
        cleaner.Close();
    }
    /*Открыть файл для дешифрования */
    private void button2_Click(object sender, EventArgs e)
    {
        string FilePic;
        OpenFileDialog dPic = new OpenFileDialog();
        dPic.Filter = "Файли зображень (*.bmp)|*.bmp|Всі файли (*.*)|*.*";
        if (dPic.ShowDialog() == DialogResult.OK)
        {
            FilePic = dPic.FileName;
        }
        else
        {
            FilePic = "";
```

```
            return;
        }

        FileStream rFile;
        try
        {
            rFile = new FileStream(FilePic, FileMode.Open); //відкриваємо поток
        }
        catch (IOException)
        {
            MessageBox.Show("Помилка відкриття файла", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }
        Bitmap bPic = new Bitmap(rFile);
        if (!isEncryption(bPic)) {
            MessageBox.Show("В файлі немає зашифрованної інформації", "Інформація",
MessageBoxButtons.OK);
            rFile.Close();
            return;
        }

        int countSymbol = ReadCountText(bPic); //считали количество зашифрованных
символов
        byte[] message = new byte[countSymbol];
        int index = 0;
        bool st = false;
        for (int i = ENCRYP_TEXT_SIZE + 1; i < bPic.Width; i++) {
            for (int j = 0; j < bPic.Height; j++) {
                Color pixelColor = bPic.GetPixel(i, j);
                if (index == message.Length) {
                    st = true;
                    break;
                }
```

```
            BitArray colorArray = ByteToBit(pixelColor.R);
            BitArray messageArray = ByteToBit(pixelColor.R); ;
            messageArray[0] = colorArray[0];
            messageArray[1] = colorArray[1];

            colorArray = ByteToBit(pixelColor.G);
            messageArray[2] = colorArray[0];
            messageArray[3] = colorArray[1];
            messageArray[4] = colorArray[2];

            colorArray = ByteToBit(pixelColor.B);
            messageArray[5] = colorArray[0];
            messageArray[6] = colorArray[1];
            messageArray[7] = colorArray[2];
            message[index] = BitToByte(messageArray);
            index++;
        }
        if (st) {
            break;
        }
    }
    string strMessage = Encoding.GetEncoding(1251).GetString(message);
    string sFileText;
    SaveFileDialog dSaveText = new SaveFileDialog();
    dSaveText.Filter = "Текстові файли (*.txt)|*.txt|Всі файли (*.*)|*.*";
    if (dSaveText.ShowDialog() == DialogResult.OK)
    {
        sFileText = dSaveText.FileName;
    }
    else
    {
        sFileText = "";
        rFile.Close();
        return;
```

```
        };

        FileStream wFile;
        try
        {
            wFile = new FileStream(sFileText, FileMode.Create); //відкриваємо поток на запись
результатов
        }
        catch (IOException)
        {
            MessageBox.Show("Помилка відкриття файла на запис", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            rFile.Close();
            return;
        }
        StreamWriter wText = new StreamWriter(wFile, Encoding.Default);
        if (jwtStatus)
        {
            jwtKeyHash();
            jwtKey = textBox1.Text;
            if (strMessage.IndexOf(jwtKey) == 0)
            {
                strMessage = strMessage.Substring(jwtKey.Length);
            }
            else
            {
                MessageBox.Show("Ключ не вірний", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                wText.Close();
                wFile.Close(); //закрываем поток
                rFile.Close();
                return;
            }
        }
```

```
            wText.Write(strMessage);
            MessageBox.Show("Текст записано в файл", "Інформація", MessageBoxButtons.OK);
            wText.Close();
            wFile.Close(); //закрываем поток
            rFile.Close();
        }
        private void jwtKeyHash()
        {
            var password = textBox1.Text;
            jwtKey = GetHash(password);
        }
        private void JWT_checkBox_CheckedChanged(object sender, EventArgs e)
        {
            jwtStatus = !jwtStatus;
        }
        private string GetHash(string password)
        {
            byte[] tmpSource = ASCIIEncoding.ASCII.GetBytes(password);
            byte[] tmpChekHash = new MD5CryptoServiceProvider().ComputeHash(tmpSource);
            var result = ByteArrayToString(tmpChekHash);
            return result;
        }
        private string ByteArrayToString(byte[] arrInput)
        {
            StringBuilder sOutput = new StringBuilder(arrInput.Length);
            for (int i = 0; i < arrInput.Length - 1; i++)
            {
                sOutput.Append(arrInput[i].ToString("X2"));
            }
            return sOutput.ToString();
        }
    }
}
```

ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОДИ ТА ЗАСОБИ СТЕГАНОГРАФІЧНОГО ЗАХИСТУ КОНФІДЕНЦІЙНОЇ ІНФОРМАЦІЇ

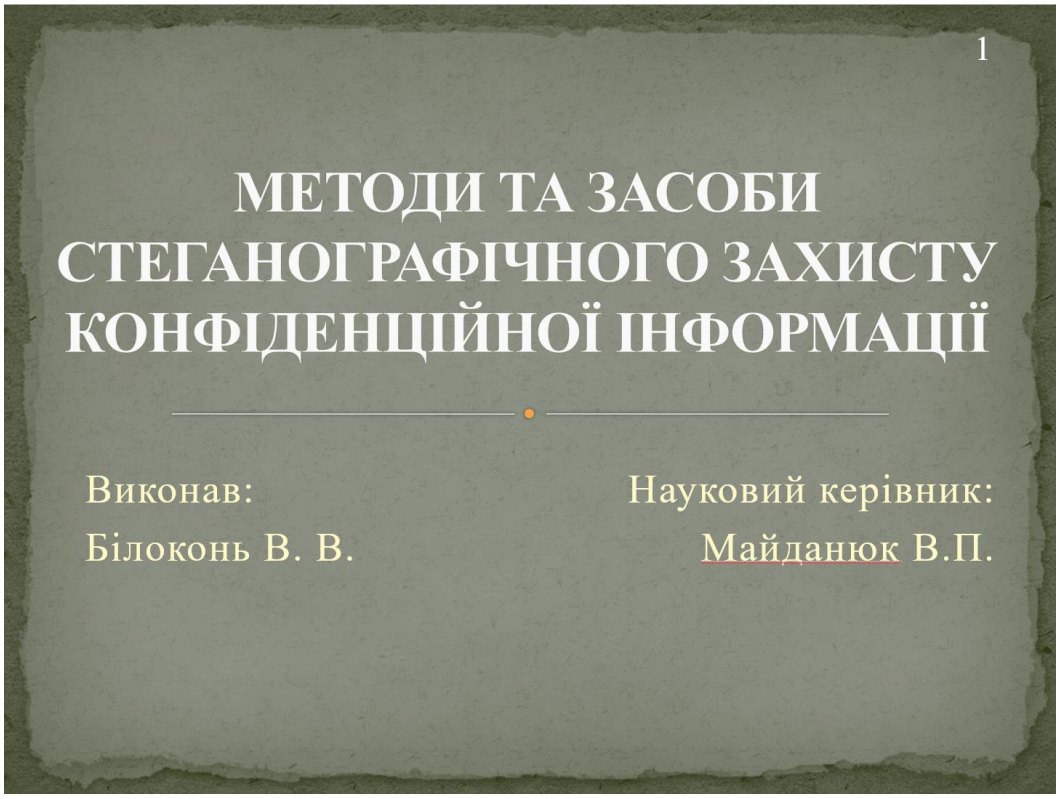

Рисунок В.1 – Тема, автор, науковий керівник

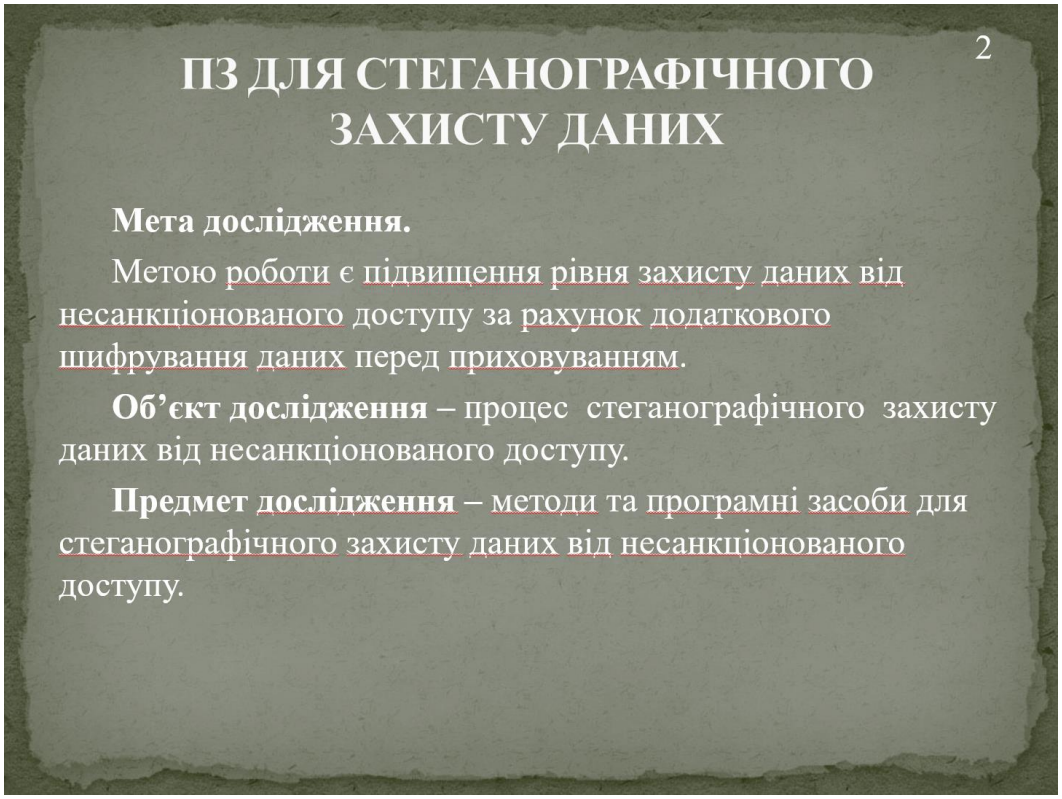

Рисунок В.2 – Мета, об'єкт, предмет дослідження

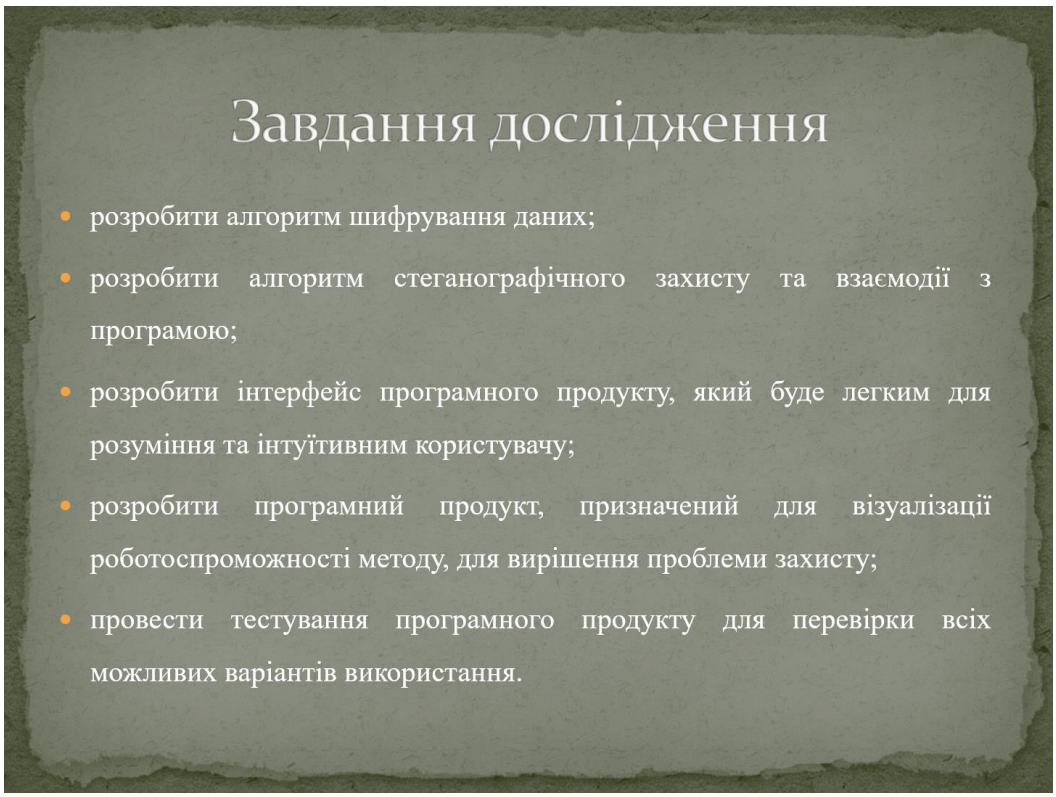

Рисунок В.3 – Завдання дослідження

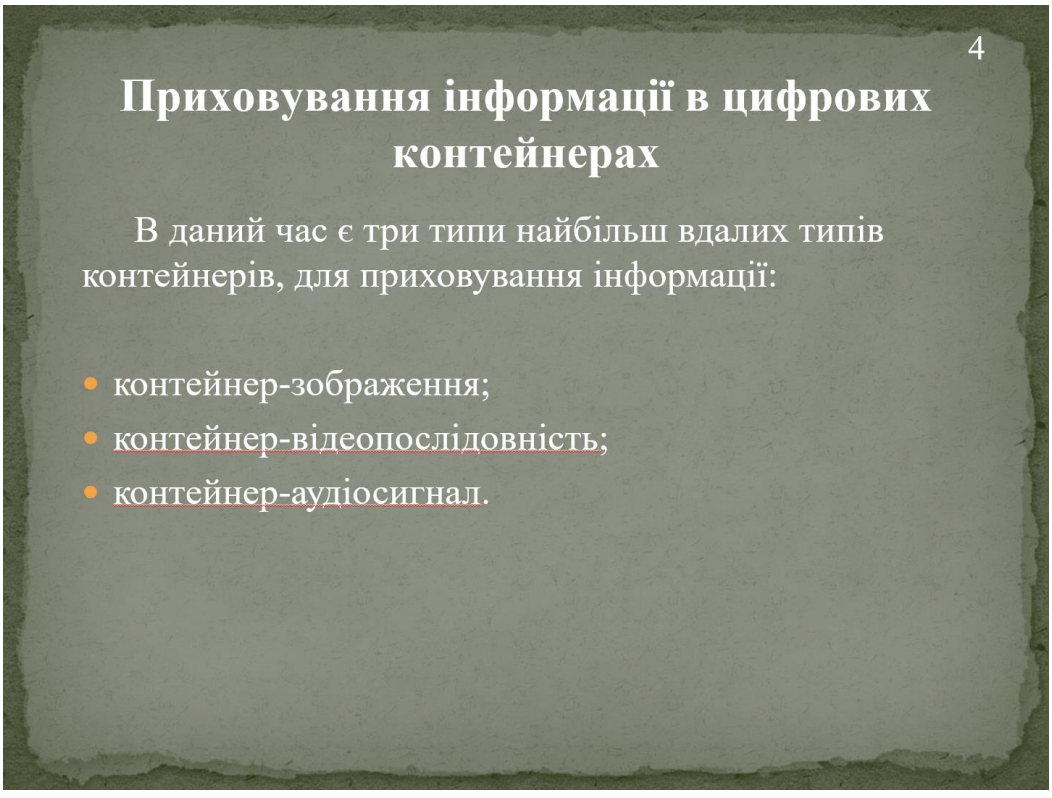

Рисунок В.4 – Приховування інформації в цифрових контейнерах

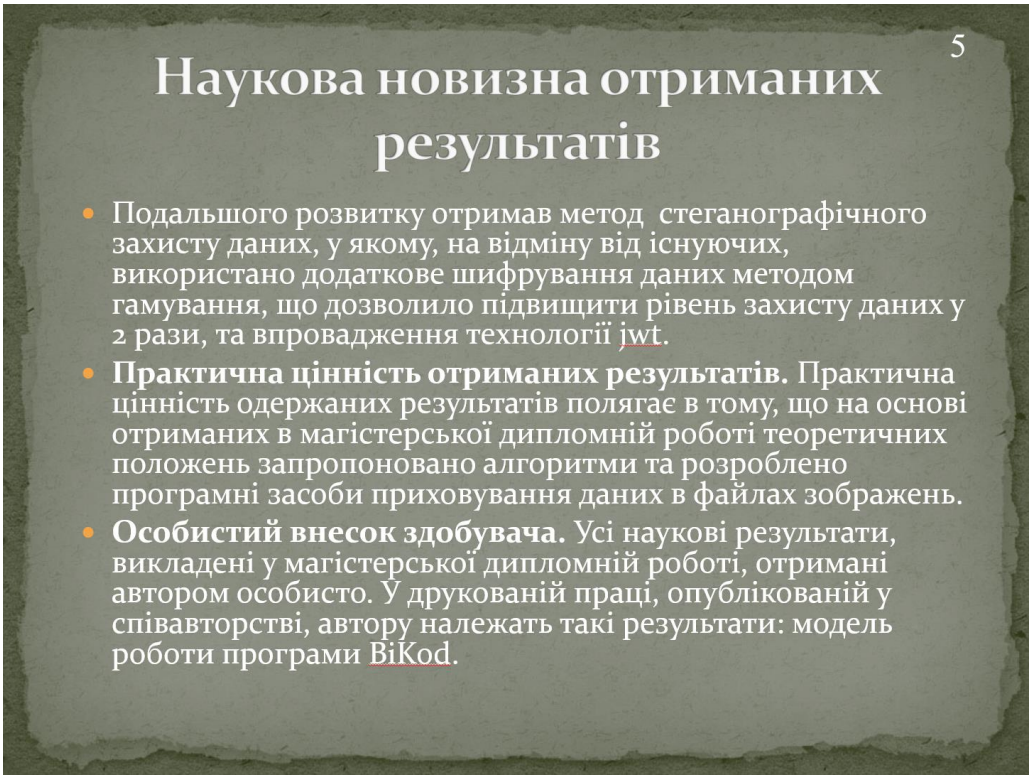

Рисунок В.5– Наукова новизна отриманих результатів

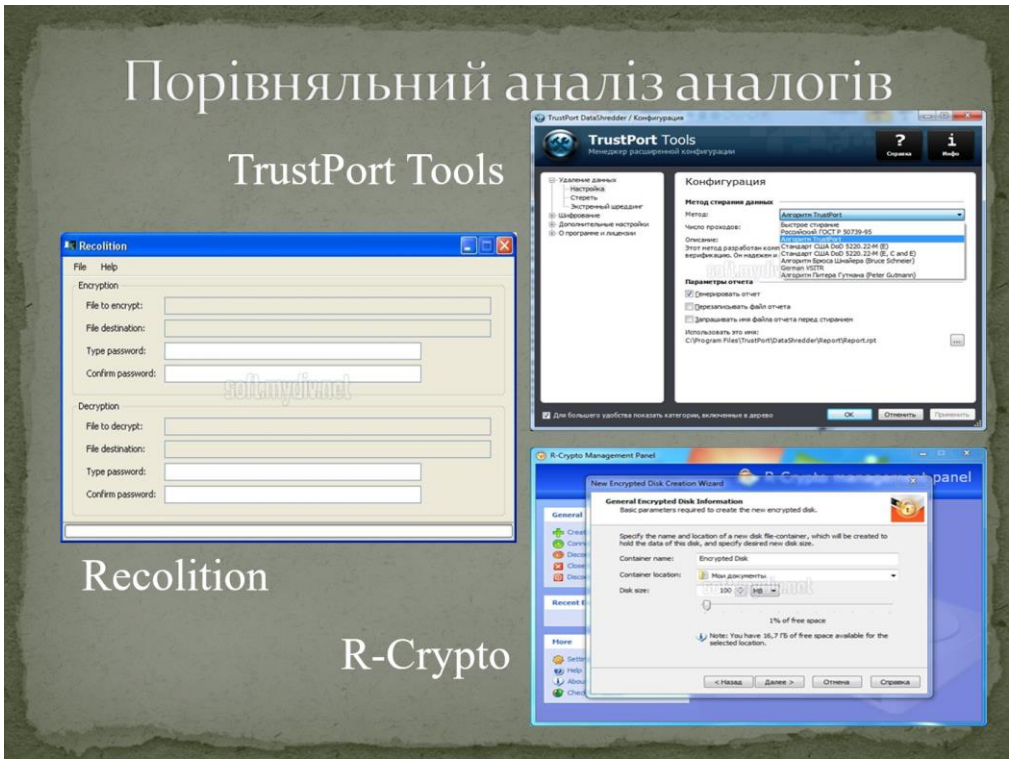

Рисунок В.6 – Порівняльний аналіз аналогів

7

Аналіз існуючих методів кодування

Шифри із симетричним ключем спроектовані так, щоб мати велику пропускну здатність

Ключі для шифрів із симетричним ключем відносно короткі

Шифри із симетричним ключем можна використати як примітиви для побудови різних криптографічних механізмів включно з псевдовипадковими генераторами чисел, обчислювальних ефективних схем підпису та інших.

Шифри із симетричним ключем можна комбінувати для отримання сильніших шифрів

Рисунок В.7 – Аналіз існуючих методів кодування

8

Стенографія для кодування даних

Насамперед, стенографія являє собою систему спеціальних значків: геометричних, які використовуються у різних системах стенографії багатьох країн та елементів літер рукописного письма, які застосовуються в більшості по слов'янських країнах, пристосована для швидкого запису за складами, словотвірними морфемами й словами усної мови.

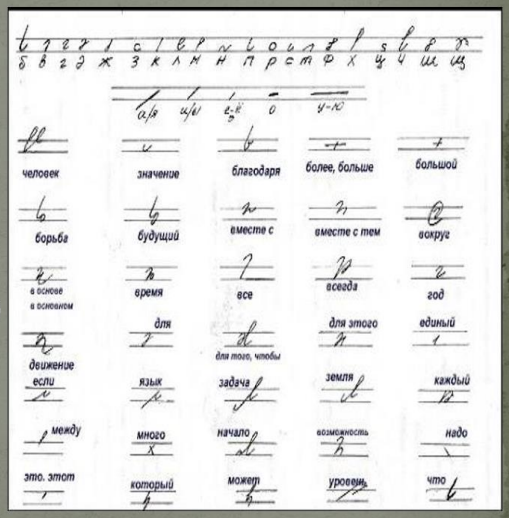

Рисунок В.8 – Стенографія для кодування даних

Рисунок В.9 – Стеганографічний захист даних в зображенні

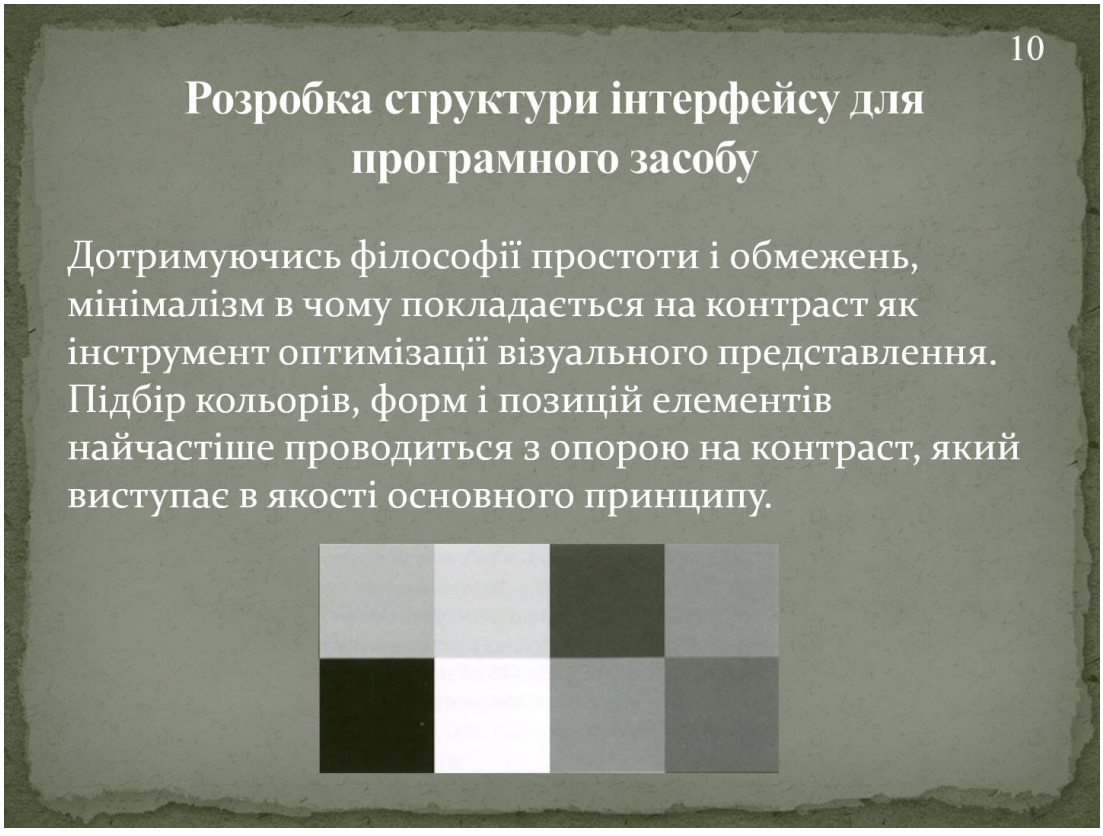

Рисунок В.10 – Розробка структури інтерфейсу для програмного засобу

Рисунок В.11 – Розробка алгоритмів роботи програми

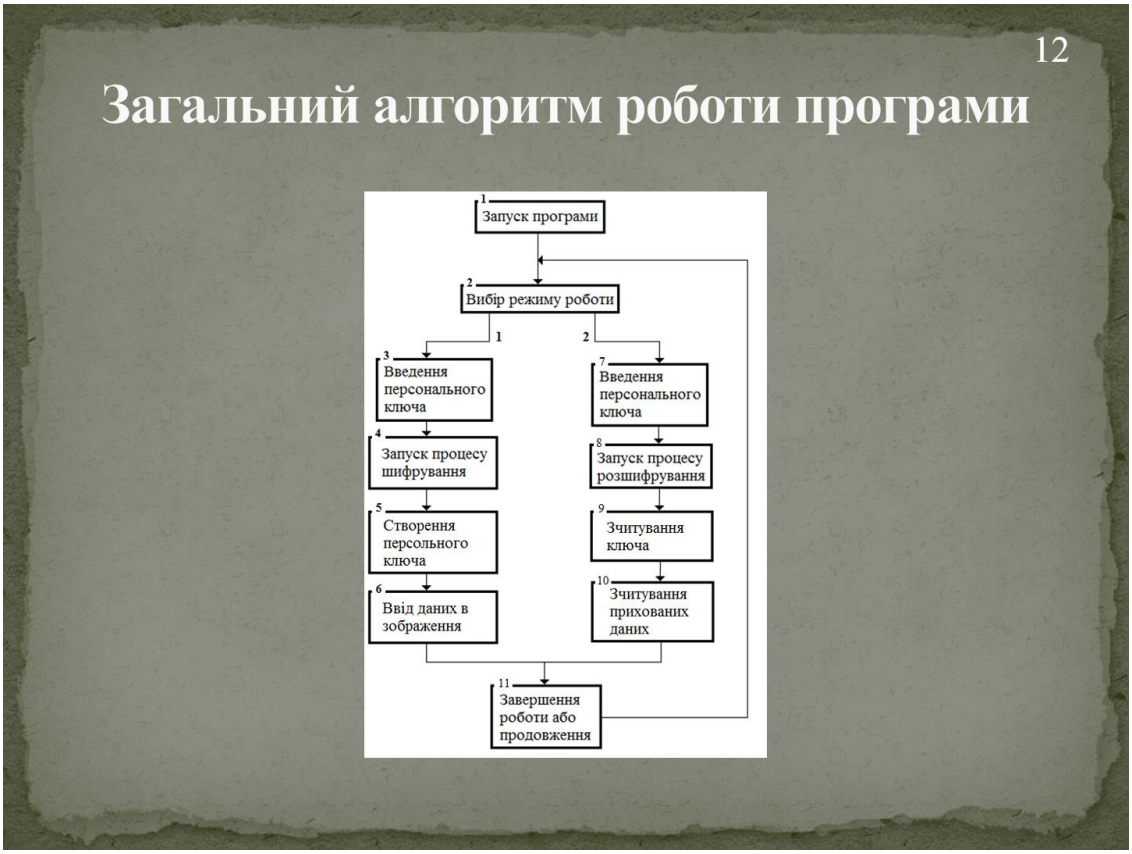

Рисунок В.12 – Загальний алгоритм роботи програми

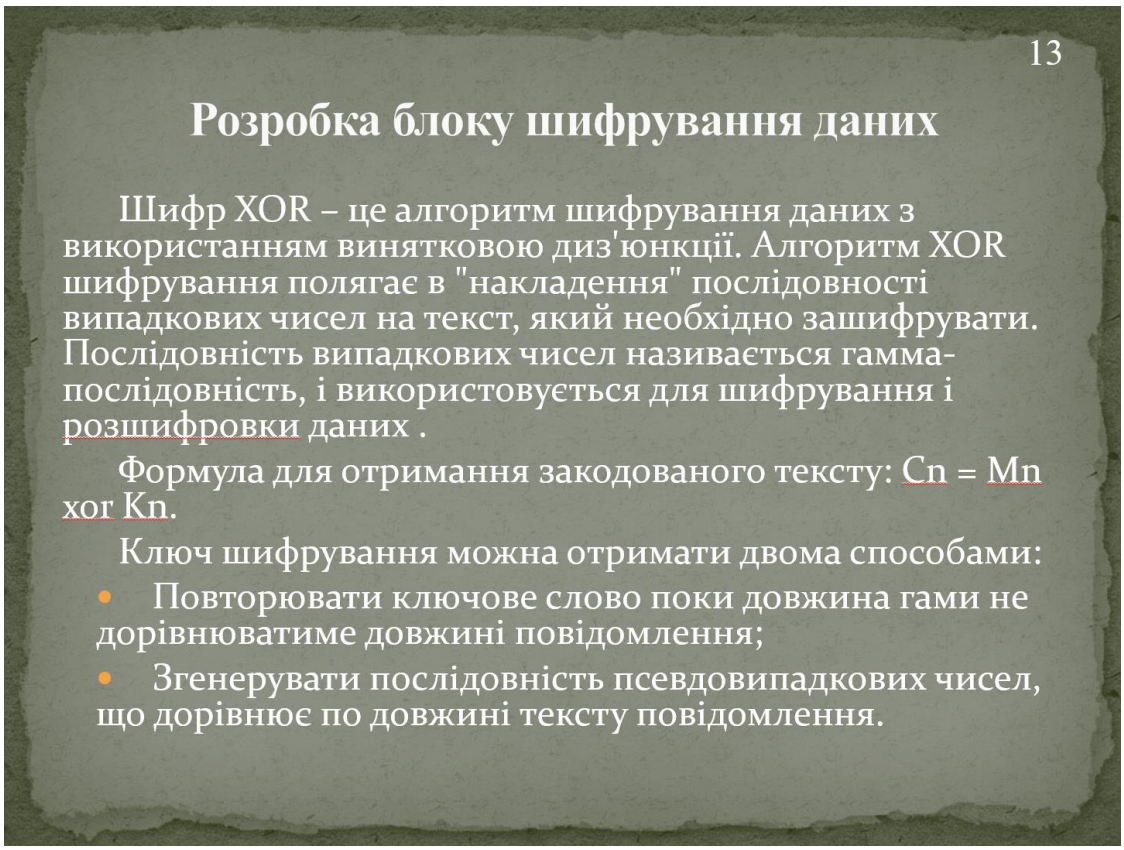

Рисунок В.13 – Розробка блоку шифрування даних

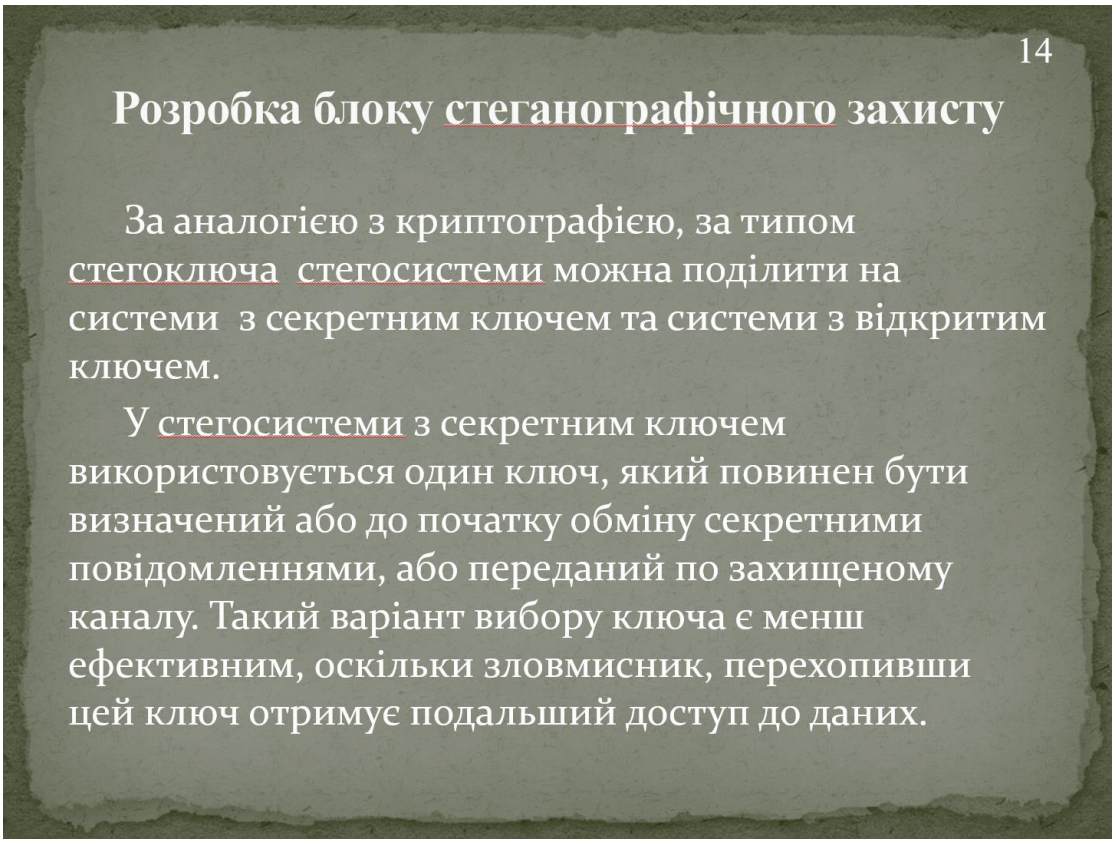

Рисунок В.14 – Розробка блоку стеганографічного захисту

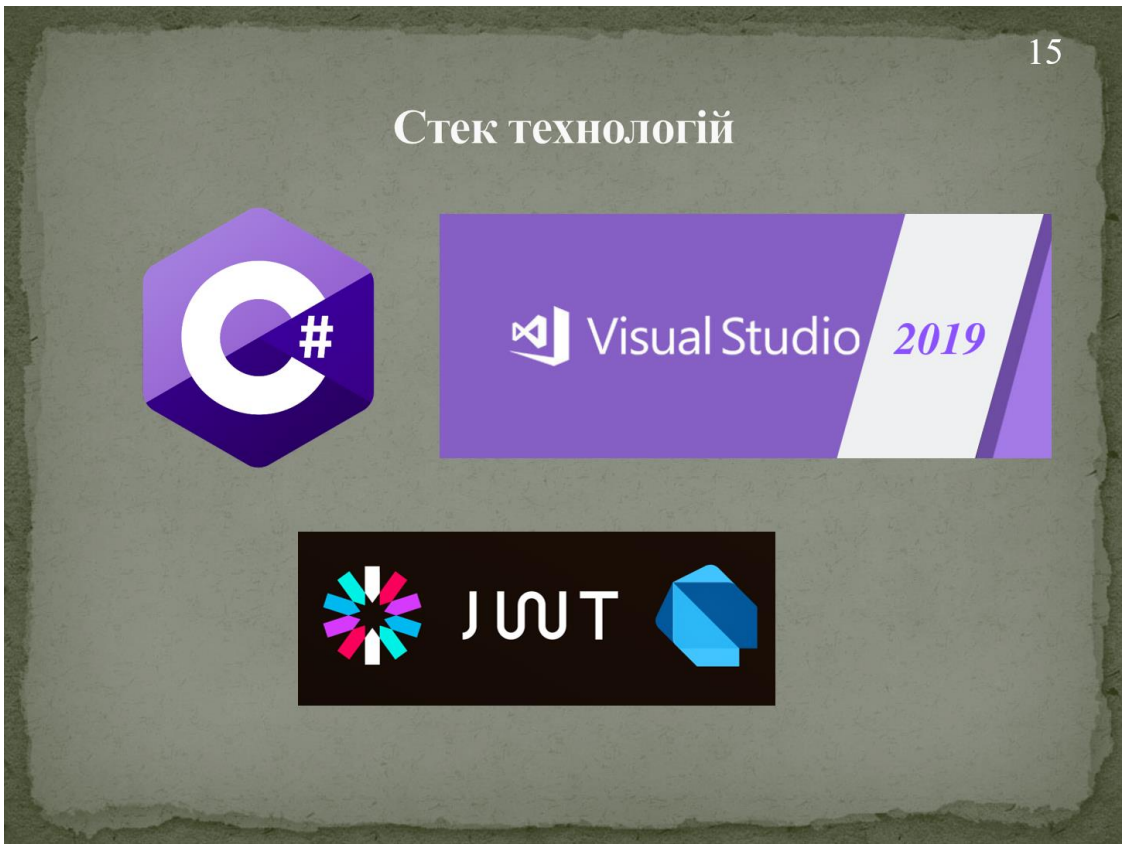

Рисунок В.15 – Стек технологій

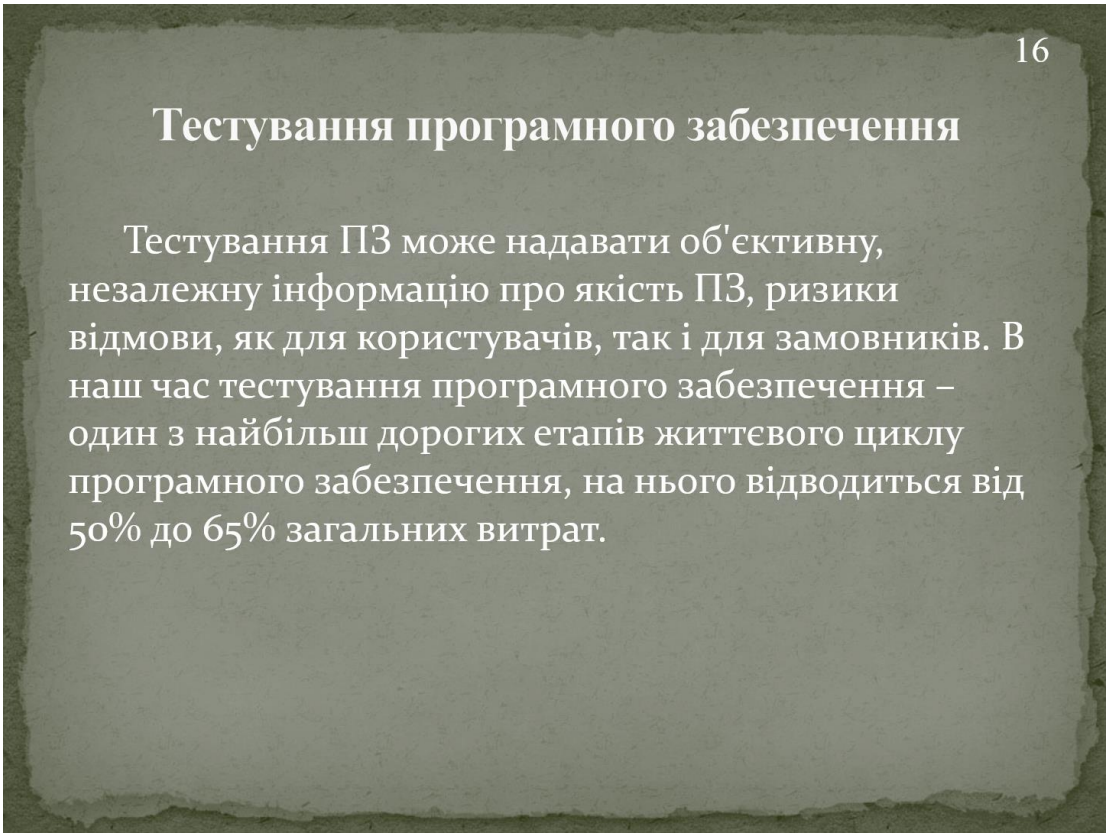

Рисунок В.16 – Тестування програмного забезпечення

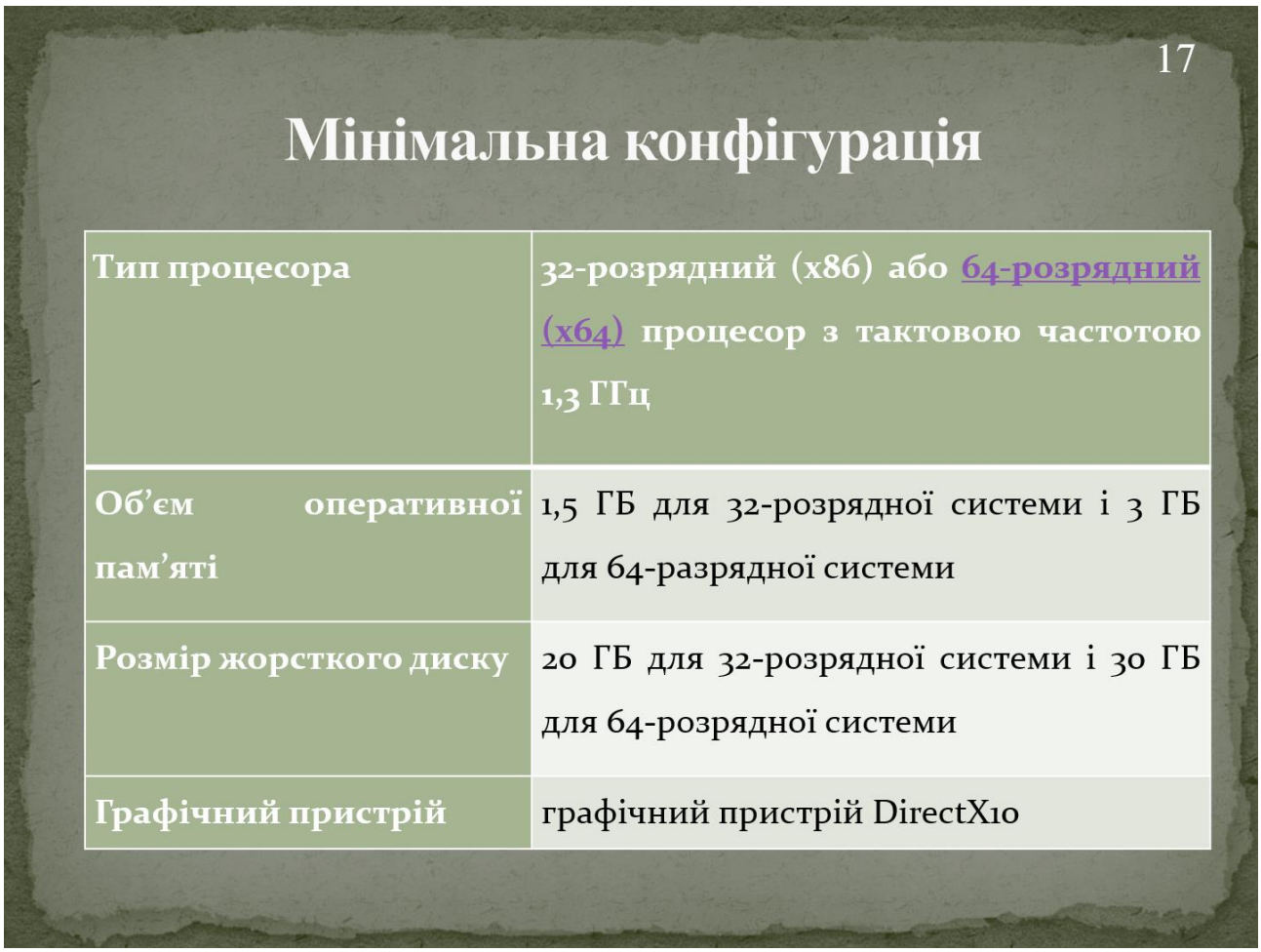

Рисунок В.17 – Мінімальна конфігурація