МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА ЕЛЕКТРОНІКИ, РОБОТОТЕХНІКИ І ТЕХНОЛОГІЙ МОНІТОРИНГУ ТА ІНТЕРНЕТУ РЕЧЕЙ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри
_____ Шутко В.М.
«____»_____ 2021 р.

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА
ЗІ СПЕЦІАЛЬНОСТІ 171 «ЕЛЕКТРОНІКА»
ОПП «ЕЛЕКТРОННІ СИСТЕМИ»

Тема: «Стеганографічний захист інформації на основі стохастичного переставлення пікселів контейнера»

Виконавець
студент групи ЕС-413Б _____ Полторацький Дмитро Анатолійович

Керівник
д.т.н., професор _____ Білецький Анатолій Якович

Нормоконтролер _____ Сініцин Р.Б.

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки і телекомунікацій

Кафедра електроніки, робототехніки і технологій моніторингу та інтернету речей

Напрям (спеціальність) 171 «Електроніка»
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Шутко В. М.
«_____»_____2021р.

ЗАВДАННЯ

на виконання дипломної роботи

Полторацькому Дмитру Анатолійовичу
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи: «Стеганографічний захист інформації на основі стохастичного переставлення пікселів контейнера» затверджена наказом ректора від « 01 » квітня 2021р. № 526/ст.
2. Термін виконання роботи : з « 01 » квітня 2021р. по « 14 » червня 2021р.
3. Вихідні дані до роботи: створення програмної реалізації стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера.
4. Зміст пояснювальної записки: Теоретичний опис способу. Розробка програмного рішення. Результати аналізу та тестування.
5. Перелік обов’язкового ілюстративного матеріалу: Структура BMP-файлу. Згенерована псевдовипадкова послідовність. Загальний приклад таблиці перестановок. Згенерована таблиця перестановок. Таблиця розрахунків для перевірки на незвідність. Обрахунки мультиплікативних груп. Прототип інтерфейсу програми у режимі приховування інформації. Прототип інтерфейсу програми у режимі зчитування інформації. Діаграма виклику основних функцій. Діаграма виклику основних під-функцій при приховуванні повідомлення.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Написання заяви на написання кваліфікаційної роботи	12.03.2021	
2	Ознайомлення та обґрунтування актуальності обраної теми	13.03.2021-01.04.2021	
3	Ознайомлення з загальними положеннями про дипломне проектування та оформлення	06.05.2021-07.05.2021	
4	Формулювання мети і завдання кваліфікаційної роботи	08.05.2021-09.05.2021	
5	Бібліографічний пошук за темою кваліфікаційної роботи	10.05.2021	
6	Написання вступу та загальних відомостей	11.05.2021-19.05.2021	
7	Розробка програмного рішення. Написання в 2-го розділу	20.05.2021-28.05.2021	
8	Аналіз та тестування програмного рішення	29.05.2021-01.06.2021	
9	Написання 3-го розділу, базуючись на отриманому результаті тестування та аналізу	02.06.2021-09.06.2021	
10	Оформлення роботи. Подання на кафедру.	12.06.2021	

7. Консультація з окремого(мих) розділу(ів):

Назва розділу	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Теоретичний опис способу	д.т.н., професор, Білецький А. Я.	11.05.2021	11.05.2021

8. Дата видачі завдання: «01» квітня 2021р.

Керівник дипломної роботи (проекту)_____ Білецький А. Я.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання_____ Полторацький Д. А.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Стеганографічний захист інформації на основі стохастичного переставлення пікселів контейнера»: 47 с., 12 рис., 9 табл., 8 літературних джерел, 5 додатків.

Об’єкт дослідження: алгоритм стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера.

Мета роботи: дослідження алгоритму стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера.

Методи дослідження: аналіз практичного застосування алгоритму через розробку програмного рішення.

Даний спосіб дозволяє вирішувати сучасні задачі, що ставляться перед стеганографією.

СТЕГАНОГРАФІЯ, ЗАХИСТ ІНФОРМАЦІЇ, ПРИХОВАНА ПЕРЕДАЧА ДАНИХ, ПРИХОВАНЕ ЗБЕРІГАННЯ ІНФОРМАЦІЇ, РАСТРОВЕ ЗОБРАЖЕННЯ.

ЗМІСТ

ПВП – псевдовипадкова послідовність;

НП – незвідний поліном;

BMP – bitmap-формат растрових зображень;

CRC – контрольна сума, розрахована за допомогою циклічного надлишкового коду;

ПК – персональний комп’ютер;

LSB – last significant bit, алгоритм приховування інформації;

MC/DC – критерій покриття коду тестами, modified condition/decision coverage.

ВСТУП

Питання стеганографічного захисту інформації є актуальним ще з давніх часів. Перші згадки про приховану передачу інформації датуються V ст. до н.е. З часом технології розвивались і тягнули за собою розвиток нових методів стеганографічного захисту інформації. На сьогоднішній день стеганографія налічує кілька основних напрямків:

- класична стеганографія;
- комп'ютерна стеганографія;
- мережева стеганографія;
- цифрова стеганографія.

Даний спосіб належить до напрямку цифрової стеганографії. Цифрова стеганографія базується на прихованні секретного повідомлення у контейнері, що є цифровими даними. Зазвичай у ролі контейнера використовується дані, що мають аналогову природу, такі як зображення, відео, аудіо. Серед способів приховання можна виділити:

- оперування над самим цифровим сигналом;
- накладання секретного повідомлення на контейнер;
- використання мета-даних файлу.

Спосіб, що розглядається у роботі, оперує над самим цифровим сигналом. Такий спосіб передбачає наявність мінімальних змін у контейнері. Одним із методів модифікації контейнера для приховування секретного повідомлення є метод «LSB». Цей метод полягає у заміні наймолодшого біта в байті контейнера на біт секретного повідомлення. Нехай байт контейнера $A = 254$, тоді двійковий формат такого байту $A_{bin} = 1111\,1110$. А також потрібно приховати ненульовий біт секретного повідомлення. Після підміни молодшого біту на біт секретного повідомлення утвориться байт $C_{bin} = 1111\,1111$. При заміні молодшого біта ніяких візуальних змін не прослідковується. Отже, людські органи чуття не в змозі розпізнати, наприклад, різницю між пікселем, який містить стороній молодший біт. Але існують спеціальні методи, що дозволяють проаналізувати контейнер на наявність повідомлення.

Проста заміна молодшого біта є нестійкою до усіх, наведених нижче, типів атак:

- атака на основі відомого заповненого контейнера;
- атака на основі обраного заповненого контейнера;
- атака на основі відомого порожнього контейнера;
- атака на основі обраного порожнього контейнера;
- атака на основі відомої математичної моделі контейнера або його частини;
- атака на основі відомого прихованого повідомлення;
- атака на основі обраного прихованого повідомлення;
- адаптивна атака на основі обраного прихованого повідомлення.

Тому дана робота пропонує модифікацію методу заміни молодшого біту, яка робить спосіб стійким до атак.

Модифікація полягає у, додатковому, криптографічному захисті самого секретного повідомлення, а також у захисті самого контейнера за допомогою стохастичних перестановок. У якості контейнера використовуються растрові зображення. Для захисту контейнера від атак, пікселі зображення стохастично переставляються блоками за роздільною схемою, а повідомлення заноситься у переставлені блоки. Після повернення пікселів на місця, біти повідомлення стохастично перемішані по зображенню. Саме повідомлення шифрується перед приховуванням, за допомогою гамування байтів секретного повідомлення. Процес гамування полягає у накладанні псевдовипадкових чисел на байти відкритого тексту. Зворотний процес розшифрування отримується шляхом накладання байтів зашифрованого тексту на туж саму послідовність псевдовипадкових чисел. Завдяки використанню генератора псевдовипадкової послідовності на базі матриць Галуа в якості генератора псевдовипадкових чисел, що використовуються і для перестановок і для гамування, алгоритм не потребує у збережені таблиць перестановок та гама-послідовностей для подальшого використання їх у якості ключів. Ключами для алгоритму виступають три бінарні вектори, а саме: незвідний поліном, примітивний утворюючий елемент, вектор ініціалізації. Основний ключ –

незвідний поліном. Саме складність підбору НП великої степені, серед всієї множини можливих поліномів, робить алгоритм захищеним.

Швидкодія, універсальність та простота реалізації дозволяє даному способу вирішувати всі задачі, до яких може бути застосована стеганографія. Серед них задача прихованої передачі інформації, яка є потрібною для сфери авіації, а також багатьох інших сфер де є потреба у прихованій передачі інформації. Також задача прихованого збереження інформації, як додатковий захист інформації при її викраденні. Багато веб-ресурсів зазнають викрадення інформації з баз даних, коли викрадена інформація стеганографічно прихована, то зловмисник отримає зовсім інші дані, що використовуються для імітації. Також стеганографічний алгоритм дає змогу зберігати секретні дані у недекларованих сховищах, наприклад текстове повідомлення може бути сховане у зображення та зберігатися на хостингу для зображень. Поміж задач прихованої передачі інформації та прихованого збереження, існують задачі збереження цифрових відбитків та стеганографічних водяних знаків. Цифрові відбитки містять певну ідентифікуючу інформацію, яка повинна бути прихована і дає змогу ідентифікувати контейнер. Цифрові відбитки широко застосовуються для відстеження порушень авторських прав. Подібними є і стеганографічні водяні знаки, вони застосовуються зазвичай для ідентифікації джерела. Наприклад більшість цифрових фотоапаратів проставляють такі водяні знаки на кожне фото для можливості ідентифікації фотоапарату.

Розроблена програмна реалізація даного стеганографічного алгоритму дає змогу перевірити його практичне використання, а також використовувати її для навчальних та демонстраційних цілей.

РОЗДІЛ 1
ТЕОРЕТИЧНИЙ ОПИС СПОСОБУ

1.1 Алгоритм наповнення та зчитування контейнера

Даний спосіб полягає у заміні молодшого біта кожного байту контейнера бітом повідомлення, що повинне бути схованим. Таким чином коефіцієнт заповнення дорівнює 1/8. Для уникнення можливості виявлення прихованих даних, перед наповненням, растровий контейнер шифрується шляхом стохастичної перестановки байтів блоками розміром N, який визначається виразом (1.1). Після повернення пікселів на свої місця, біти повідомлення будуть стохастично розподілені по зображенню.

$$N = 2^k, k \in \{ 8, 16 \} \quad (1.1)$$

У випадках, коли кількість байтів контейнера некратна розміру блока, до контейнера додаються, так звані, «баластні» байти, щоб вирівняти розміри контейнера. Байти для сервісної інформації зберігають контрольну суму для перевірки цілісності повідомлення, а також кількість даних, що було приховано. Сервісні байти доповнюють розмір повідомлення, тому потрібно враховувати додаткові сімдесят два байти у розмірі контейнера.

Перш ніж приступити до опису алгоритму наповнення та зчитування контейнера розберемо термін «контейнер». У стеганографії, під терміном «контейнер» мається на увазі інформація, яка приховує секретне повідомлення. Це може бути зображення, аудіо-файл, відео та ін. В даному способі використовуються растрові зображення у форматі BMP, які не використовують стиснення даних. Контейнер має зберігати монохромні та кольорові зображення довільного розміру. У монохромному зображені на кожен піксель виділяється один байт, а в кольоровому зображені кожен піксель кодується трьома байтами за схемою RGB.

Структура файлу BMP складається з чотирьох частин: заголовок файлу (BitMap File Header), інформаційний заголовок (BitMap Info Header), палітра кольорів (Color Table), дані зображення (BitMap Array). У перших трьох частинах, назвемо їх метаданими, приховування інформації неможливе, бо файл перестане бути доступним для відкриття у програмах для роботи з графікою.

Параметр типу компресії має дорівнювати нулю, а також кількість використаних та основних кольорів має дорівнювати нулю. Такий контейнер буде використовувати стандартну палітру, а також алгоритм компресії не буде впливати на масив байтів, що представляє саме зображення. Детальна структура контейнера наведена у таблиці 1.1.1.

Таблиця 1.1.1

Структура BMP-файлу

Параметр	**Розмір**	**Зміщення**
Слово «BM», яке вказує на BMP формат файлу	2	0
Загальний розмір файлу в байтах	4	2
Зарезервовані нульові байти	4	6
Кількість сервісних байт	4	10
Розмір заголовку в байтах	4	14
Ширина зображення в пікселях	4	18
Висота зображення в пікселях	4	22
Кількість вимірів	2	26
Кількість біт на піксель	2	28
Тип компресії	4	30
Кількість байт в масиві зображення	4	34
Кількість пікселів на метр по осі X	4	38
Кількість пікселів на метр по осі Y	4	42
Кількість використаних кольорів	4	46
Кількість основних кольорів	4	50
Масив зображення	-	54

Серед параметрів контейнера виділимо ті, що знадобляться нам у ході наповнення контейнера та введемо для них умовні позначення: загальний розмір вхідного файлу в байтах L_{in}. Також введемо параметри, які з'являться у результаті наповнення: розмір блоку перестановки N, розмір повідомлення в байтах L_{msg}, контрольна сума повідомлення CRC, загальний розмір вихідного файлу в байтах L_{out}. Алгоритм наповнення може бути поділений на наступні кроки:

- розрахунок кількості біт повідомлення;
- перевірка кількості байт в масиві вхідного зображення на кратність розміру блока перестановки та можливість вмістити потрібну кількість біт;
- розширення масиву вхідного зображення у випадку невідповідності хоча б одній з умов;
- формування контейнера;
- розрахунок контрольної суми повідомлення;
- наповнення сервісних байтів контейнера значеннями контрольної суми повідомлення та розміру повідомлення в байтах відповідно;
- стохастична перестановка байтів контейнера блоками розміром N;
- шифрування повідомлення шляхом гамування байтів;
- наповнення інших байтів контейнера бітами повідомлення;
- формування вихідного зображення.

Кількість біт повідомлення розраховується за формулою (1.2).

$$C_{msg} = L_{msg} * 8 \quad (1.2)$$

Якщо не виконується хоча б одна з умов (1.3) або (1.4), тоді потрібно доповнити масив зображення такою кількістю байт, щоб задовольняти обидві умови. Розмір доповнений масиву зображення розраховується за формулою (1.5), коли не виконується умова (1.3), та за формулою (1.6), коли не виконується умова (1.4) або не виконуються обидві умови.

$$L_{in} \,\%\, N == 0 \quad (1.3)$$

$$L_{in} \geq C_{msg} + 72 \quad (1.4)$$

$$L_{out} = \left(\frac{L_{in}}{N} + 1\right) * N \quad (1.5)$$

$$L_{out} = \left(\frac{C_{msg} + 72}{N} + 1\right) * N \quad (1.6)$$

Контрольна сума рахується за допомогою циклічного надлишкового коду, що полягає у діленні із залишком бінарних поліномів. Значення контрольної суми є залишком від ділення на твірний поліном. У даному випадку використовується стандарт CRC-8, в основі якого лежить твірний поліном восьмої степені f_{CRC-8}.

$$f_{CRC-8} = 111010101 \quad (1.7)$$

Алгоритм розрахунку контрольної суми зводиться до наступних кроків:

- береться байт з послідовності;
- зсув вліво на один розряд;
- у випадку непарності старшого біта проводимо операцію ділення із залишком на твірний поліном;
- процедура повторюється спочатку доки послідовність біт не закінчиться. Фінальний залишок і є контрольною сумою.

Сам по собі контейнер, що наповнений бітами повідомлення, може сховати інформацію лише від людського ока, але за допомогою машинного обчислення можна з легкістю виявити приховане повідомлення. Для захисту від виявлення секретного повідомлення, стеганографічний алгоритм поєднується з криптографічним алгоритмом. Криптографічний алгоритм перетворює повідомлення у шифр за певною схемою, використовуючи спеціальний ключ, що дає змогу здійснювати обернену дію розшифрування. Існує велика кількість алгоритмів шифрування, їх поділяють на алгоритми симетричного та асиметричного шифрування. Є стандартизовані алгоритми, які є найбільш популярними, серед таких: DES, triple DES, AES, RC4, RSA, Twofish. Для криптографічного захисту повідомлення, його байти гамуються зі стохастично отриманими байтами, а пікселі зображення піддаються стохастичній перестановці. Наповнення контейнера виконується шляхом заміни молодшого біта в байті з масиву зображення відповідним бітом попередньо зашифрованого повідомлення. Після стеганографічних та криптографічних перетворень формується вихідне зображення, яке містить секретне повідомлення.

Алгоритм зчитування інформації з контейнера має такі етапи:

- розшифровка контейнера, використовуючи відповідні ключі;
- вичитування сервісної інформації повідомлення;
- вичитування повідомлення;
- розрахунок контрольної суми повідомлення;
- перевірка розрахованої контрольної суми зі збереженою.

Після перевірки контрольних сум, у разі відповідності повідомлення вважається успішно отриманим.

1.2 Генератор перестановок та байтів гамування

Шифрування виконується через стохастичні перестановки та гамування із стохастично отриманими байтами. Отже, важливим елементом є генератор псевдовипадкових чисел. Генератор псевдовипадкових чисел – це алгоритм, що створює послідовність, кожне число якої майже незалежне одне від одного і підкорюються рівномірному розподілу. У якості генератора псевдовипадкових чисел використовується генератор псевдовипадкових послідовностей на базі матриць Галуа та Фібоначчі n-ї степені над полем $GF(2)$, а також спряжених матриць Галуа та Фібоначчі. Такі генератори ПВП базуються на лінійних регістрах зсуву з лінійним зворотнім зв'язком. Узагальнені матриці Галуа утворюються способом діагонального заповнення. Спосіб діагонального заповнення полягає у тому, що утворюючий елемент записується у правій частині нижнього рядка матриці, а всі відсутні елементи цього рядка заповнюються нулями. Всі наступні рядки формуються зсувом попереднього рядка вліво на один розряд, якщо після зсуву старший розряд не дорівнює нулю, тоді рядок приводиться до залишку по модулю незвідного полінома. Отже, зі способу формування матриці можна виділити два параметри – утворюючий елемент та незвідний поліном. Як приклад візьмемо матрицю Галуа восьмої степені G_8, яка сформована утворюючим елементом $\varphi = 11100$ та незвідним поліномом $f_8 = 100011011$. Така матриця розраховується за формулою (1.8).

$$G_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix} \quad (1.8)$$

Генерація псевдовипадкової послідовності відбувається шляхом множення вектору ініціалізації на матрицю. Результатом такої дії є новий n-розрядний вектор, що використовується для у якості вектору ініціалізації для наступного такту генератора, а також стає частиною псевдовипадкової послідовності. Щоб генератор мав максимальний період $2^n - 1$, утворюючий елемент матриці має бути примітивним. Утворюючий елемент вважається примітивним, коли він породжує мультиплікативну групу максимального порядку. Таким чином для роботи генератора потрібні три ключі: незвідний поліном степені n, примітивний утворюючий елемент, вектор ініціалізації. Такий генератор здатний працювати у чотирьох основних конфігураціях: генератор на базі матриці Галуа, Фібоначчі, спряженої матриці Галуа, спряженої матриці Фібоначчі. Матриця Фібоначчі утворюється з матриці Галуа шляхом зворотного (правостороннього) транспонування $\perp$, що виконується відносно другорядної діагоналі матриці. Приклад розрахунку матиці Фібоначчі F_8 наведено у формулі (1.9).

$$F_8 = G_8^{\perp} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} \quad (1.9)$$

Спряжені матриці Галуа та Фібоначчі утворюються шляхом прямого (лівостороннього) транспонування $\top$, але прямо транспонована матриця Галуа утворює спряжену матрицю Фібоначі (1.10), а прямо транспонована матриця Фібоначчі – спряжену матрицю Галуа (1.11).

$$F_8^* = G_8^\top = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (1.10)$$

$$G_8^* = F_8^\top = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (1.11)$$

Поміж основних конфігурацій генератора існують ще додаткові конфігурації, що отримуються за рахунок перетворення подібності базових матриць. Для перетворення подібності використовуються матриці перестановки, це квадратні бінарні матриці, що налічують рівно один ненульовий елемент в кожному рядку та стовпці. До того ж для матриць перестановки легко обраховуються обернені матриці. Перетворення подібності для матриці G_8 отримується за формулою (1.12).

$$\hat{\mathrm{G}}_8 = P^{-1} * G_8 * P \quad (1.12)$$

Восьма степінь матриць Галуа використовується тільки у якості прикладу, реальні генератори базуються на матрицях високих степенів: 64, 256, 1024, 2048. Саме високі степені забезпечують складність підбору ключа. Таким чином, для генерації перестановок та гами використовується одна ПВП. Послідовність, що згенерована із використанням матриці G_8 та вектору ініціалізації $IV_{G_8} = 00000001$ зображений у таблиці 1.2.1, а для зручності ПВП представлена у вигляді байт в шістнадцятковій системі числення.

Таблиця 1.2.1

Згенерована псевдовипадкова послідовність

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	00	01	1C	4B	85	EE	C6	8B	46	09	FC	25	C1	DF	AC	BF
1	40	41	5D	16	93	7D	BB	30	76	7F	83	A6	67	B8	14	AB
2	EB	AA	F7	E1	72	0F	B4	84	F2	8D	0E	A8	CF	77	63	C8
3	23	89	7E	9F	ED	E2	56	D2	20	AD	A3	0B	C4	B3	D0	18
4	3B	B2	CC	53	BE	5C	0A	D8	F8	55	F6	FD	39	8A	5A	42
5	79	CB	07	54	EA	B6	BC	64	9C	C9	3F	C2	FB	71	2B	69
6	10	DB	DC	88	62	D4	68	0C	90	59	66	A4	5F	2E	05	6C
7	7C	A7	7B	F3	91	45	2D	21	B1	E8	8E	2A	75	5B	5E	32
8	4E	E9	92	61	F0	B5	98	B9	08	E0	6E	44	31	6A	34	06
9	48	A1	33	52	A2	17	8F	36	3E	DE	B0	F4	C5	AF	9B	9D
A	D5	74	47	15	B7	A0	2F	19	27	F9	49	BD	78	D7	4C	D1
B	04	70	37	22	95	35	1A	03	24	DD	94	29	51	86	CA	1B
C	1F	6F	58	7A	EF	DA	C0	C3	E7	3A	AE	87	D6	50	9A	81
D	9E	F1	A9	D3	3C	E6	26	E5	02	38	96	11	C7	97	0D	8C
E	12	E3	4A	99	A5	43	65	80	82	BA	2C	3D	FA	6D	60	EC
F	FE	1D	57	CE	6B	28	4D	CD	4F	F5	D9	E4	1E	73	13	FF

Кожен байт повідомлення циклічно гамуються з відповідними байтом з ПВП за формулою (1.13). В результаті отримується новий зашифрований байт.

$$Z_i = A_i \oplus B_i \qquad (1.13)$$

При використані матриці більшої степені на виході утворюється вектор, більший за довжину одного байту. Тоді вихідний вектор ділиться на частини по вісім біт, які додаються за модулем двійки, утворюючи один байт.

Перестановки отримуються з ПВП подібним способом, різниця тільки у тому, що вектор ділиться на частини розміром k. Ідея генерації перестановок зводиться до наповнення, так званої, таблиці перестановок. Таблиця перестановки зберігає відповідність індексу байту та адреси, куди його потрібно переставити. Приклад таблиці перестановок наведено в таблиці 1.2.2.

Таблиця 1.2.2

Загальний приклад таблиці перестановок

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**…**	**N – 1**
S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	…	S_{N-1}

Де адреси S_i заповнюються псевдовипадковими числами, що не містять повторень. Так як таблиця перестановок, навідміну від гами не має містити повторення, коли отримане з генератора число вже міститься у таблиці, то воно пропускається і генерується наступне, доки не заповняться всі індекси. Приклад заповненої таблиці наведено в таблці 1.2.3.

Таблиця 1.2.3

Згенерована таблиця перестановок

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	13	15	0	10	3	2	9	7	6	4	5	8	11	12	14

1.3 Алгоритми створення ключів для генератора псевдовипадкових чисел

Для генератора псевдовипадкової послідовності на базі класичних та спряжених матриць Галуа, Фібоначі використовується три ключі: незвідний поліном, примітивний утворюючий елемент та вектор ініціалізації. Цей підрозділ присвячений опису ключів, а також алгоритмів для їх створення. Ключ криптографічного алгоритму – це певна кількість цифрових даних, що тримається у секреті та використовується алгоритмом для операції шифрування, а також зворотної дії розшифрування. Алгоритм може використовувати як один ключ, так і набір ключів. Ключ може бути будь-яким типом цифрової інформації, бо незалежно від типу цифрової інформації її можна представити у вигляді послідовності бітів. Згідно принципу Керхгоффса, ключі обираються такими, щоб складність підбору ключа була теоретично дуже високою, а практично, неможливою за короткий проміжок часу. Таким чином сам криптографічний алгоритм не потребує тримання його у секреті, а є відкритим.

Головний ключ до нашого алгоритму це незвідний поліном степені n. Незвідний поліном – це поліном, що має тільки тривіальні дільники. Тому незвідні поліноми є подібними до простих чисел. Синтез незвідних поліномів відбувається за допомогою алгоритму синтезу незвідних поліномів лінійної складності. Такий алгоритм дає змогу виявити незвідність полінома за лінійну складність, в той час як класичні методи синтезу незвідних поліномів мають квадратичну складність. Лінійна складність алгоритму синтезу не тягне за собою стрімке збільшення часових витрат при збільшені степені, на відміну від квадратичної складності. Степінь полінома є одним з основних параметрів полінома, вона дорівнює максимальній степені старшого ненульового монома. Реальний генератор використовує поліноми високих степенів, де степінь n береться з виразу (1.14).

$$n \in \{ 64, 256, 1024, 2048 \} \tag{1.14}$$

Приклад полінома восьмої степені наведено у виразі (1.15).

$$f_8(x) = x^8 + x^4 + x^3 + x^1 + 1 \tag{1.15}$$

Поліном з виразу (1.15) наведений у алгебраїчній формі, але існує, також, і векторна форма представлення поліномів, яка в нашому випадку буде зручнішою, так як ми будемо мати справу тільки з бінарними поліномами. Бінарні поліноми можуть мати коефіцієнти, що дорівнюють тільки нулю або одиниці. Векторна форма поліному даного наведена у виразі (1.16).

$$f_8 = 100011011 \tag{1.16}$$

Ще одним важливим параметром полінома є його період. Період полінома є найменшим натуральним числом m, при якому поліном є дільником двочлена (1.17).

$$x^m - 1 \tag{1.17}$$

Період незвідного полінома дорівнює періоду елемента $\varphi = 10$ поля $GF(2^n)$, яке утворене даним поліномом. Ця властивість є критерієм перевірки незвідного полінома на примітивність, бо незвідний поліном є примітивним, коли його період є максимальним.

Отже, розібравши основні параметри незвідних поліномів, перейдемо до опису алгоритму синтезу. Спочатку потрібно обрати параметри полінома, який будемо генерувати, а саме степінь полінома та вага полінома.

Вага полінома – це кількість ненульових коефіцієнтів цього полінома. Вага незвідного полінома завжди є непарним числом. Алгоритм генерації незвідного полінома зводиться до наступних кроків:

- обрання початкового полінома для перевірки, такий поліном може бути або підібраний вручну, або за допомогою стохастичного моделювання;
- перевірка обраного полінома на незвідність за допомогою алгоритму синтезу незвідних поліномів лінійної складності;
- у разі підтвердження незвідності на попередньому кроці, протестований поліном можна використовувати, бо він є незвідним, у іншому випадку, обраховується найближчий можливий поліном та перевіряється на незвідність.

Процес вибору початкового полінома є нескладним, якщо поліном обирається вручну, то достатньо сформувати вектор потрібної довжини, молодший та старший коефіцієнти якого дорівнюють одиниці, а інші ненульові коефіцієнти розташовані у довільному порядку. Формування полінома шляхом стохастичного моделювання є подібним до ручного способу, окрім того, що ненульові коефіцієнти розподіляються за допомогою програмного алгоритму. Початковий поліном потрібен для ініціалізації процесу синтезу незвідного полінома, так звана відправна точка. Коли початковий поліном сформований, він підлягає перевірці на незвідність. В основі алгоритму перевірки лежать прості рекурентні обрахунки залишків координатного вектору по модулю полінома, що тестується. Якщо залишок на ітерації n дорівнює одиниці, тобто виконується умова (1.18), то поліном вважається незвідним. Однак зазначимо, що у разі отримання одиниці на будь-якій попередній ітерації, поліном незвідним не вважається.

$$S_k = ((S_{k-1})^2 \ll 1) \; mod \; f_k \equiv 1 \qquad (1.18)$$

Початковий координатний вектор S_0 завжди дорівнює одиниці. Кожен наступний вектор є залишком піднесеного до квадрату попереднього вектору, що доповнений нулем, по модулю полінома. У таблиці 1.3.1 наведено приклад перевірки незвідного полінома $f_8 = 100011011$, який ми вже використовували в попередньому підрозділі для побудови матриці Галуа.

Таблиця 1.3.1

Таблиця розрахунків для перевірки на незвідність

№	$f_8 = 100011011$
0	$S_0 = 1$
1	$S_1 = ((S_0)^2 \ll 1)\ mod\ f_8 = (1^2 \ll 1)\ mod\ 100011011 = 10$
2	$S_2 = ((S_1)^2 \ll 1)\ mod\ f_8 = (10^2 \ll 1)\ mod\ 100011011 = 1000$
3	$S_3 = ((S_2)^2 \ll 1)\ mod\ f_8 = (1000^2 \ll 1)\ mod\ 100011011 = 10000000$
4	$S_4 = ((S_3)^2 \ll 1)\ mod\ f_8 = (10000000^2 \ll 1)\ mod\ 100011011 = 101111$
5	$S_5 = ((S_4)^2 \ll 1)\ mod\ f_8 = (101111^2 \ll 1)\ mod\ 100011011 = 1110010$
6	$S_6 = ((S_5)^2 \ll 1)\ mod\ f_8 = (1110010^2 \ll 1)\ mod\ 100011011 = 10101011$
7	$S_7 = ((S_6)^2 \ll 1)\ mod\ f_8 = (10101011^2 \ll 1)\ mod\ 100011011 = 1111101$
8	$S_8 = ((S_7)^2 \ll 1)\ mod\ f_8 = (1111101^2 \ll 1)\ mod\ 100011011 = 1$

Другорядним ключем алгоритму є примітивний утворюючий елемент. Для генерації псевдовипадкової послідовності максимального періоду утворюючий елемент має бути, обов'язково, примітивним. Утворюючий елемент – це елемент, що утворює деяку множину елементів, базуючись на певній асоціативній арифметичній дії. Таку множину називають групою, якщо в ній існує нейтральний елемент, а також кожен елемент цієї множини має обернений йому елемент, що також належить даній множині. Група може налічувати скінченну кількість елементів, така група називається скінченною, або мати нескінченну кількість елементів. Кожна скінченна група має певну кількість елементів, що називається періодом і є дуже важливою характеристикою. Утворюючий елемент, що може бути використаний у якості ключа утворює скінченну мультиплікативну групу G в полі $GF(2^n)$, утвореним незвідним поліномом f_n. Період $|G|$ мультиплікативної групи G повинен відповідати співвідношенню (1.19).

$$|G| = 2^n - 1 \qquad (1.19)$$

Саме співвідношенням (1.19) перевіряється примітивність утворюючого елемента. Для прикладу розглянемо мультиплікативні групи в полі $GF(2^4)$, що утворені елементами $\varphi = 1101$ та $\varphi = 111$ відповідно. У якості незвідного полінома використовується поліном $f_4 = 10011$. Утворені мультиплікативні групи наведено у таблиці 1.3.2.

Таблиця 1.3.2

Обрахунки мультиплікативних груп

№	$\varphi = \mathbf{1101}$	$\varphi = \mathbf{111}$
0	$1101^0 = 0001$	$111^0 = 0001$
1	$1101^1 = 1101$	$111^1 = 0111$
2	$1101^2 = 1110$	$111^2 = 0110$
3	$1101^3 = 1010$	$111^3 = 0001$
4	$1101^4 = 1011$	-
5	$1101^5 = 0110$	-
6	$1101^6 = 1000$	-
7	$1101^7 = 0010$	-
8	$1101^8 = 1001$	-
9	$1101^9 = 1111$	-
10	$1101^{10} = 0111$	-
11	$1101^{11} = 0101$	-
12	$1101^{12} = 1100$	-
13	$1101^{13} = 0011$	-
14	$1101^{14} = 0100$	-
15	$1101^{15} = 0001$	-

З прикладу бачимо, що елемент $\varphi = 1101$ є примітивним, бо утворює мультиплікативну групу максимального періоду. А елемент $\varphi = 111$, навпаки, не є примітивним.

Отже, алгоритм генерації примітивного утворюючого елемента може бути зведений до наступних кроків:

- формування бінарного вектору довжиною n, який і є утворюючим елементом для перевірки на примітивність;
- перевірка утворюючого елементу на примітивність шляхом побудови мультиплікативної групи та оцінки її періоду;
- збільшення початкового вектору на одиницю, якщо примітивність не підтверджена, у іншому випадку цей крок пропускається.

Формування початкового утворюючого елементу (вектору) може відбуватись як вручну, так і за допомогою стохастичного моделювання.

Останнім ключем є вектор ініціалізації, його генерація є найпростішою, адже у ролі вектору ініціалізації може виступати будь-який бінарний вектор довжиною n, окрім нульового вектору. Вектор може бути сформований або вручну, або за допомогою стохастичного моделювання.

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ

2.1 Архітектура програмного рішення

Програмний продукт, опис якого наведено у даному розділі, дозволяє провести перевірку способу стеганографічного захисту на вирішення задач прихованої передачі інформації, а також прихованого збереження інформації, які є одними з основних задач, до яких може бути застосована стеганографія. Архітектура будь-якої системи має описувати принципи її роботи, а також взаємозв'язки її елементів між собою та зовнішньою середою. Етап створення архітектури є важливим і не може бути пропущеним. Адже наявність системної архітектури дає змогу: провести попередній аналіз системи ще до початку процесу розробки, спланувати та вибудувати найбільш оптимальний процес розробки, задокументувати процеси розробки та обслуговування, створити необхідні інструменти для розробки, подальшого аналізу, а також тестування. У цьому підрозділі описано архітектуру програмного продукту, яка розділена на три групи опису: функціональну, логічну та фізичну.

До функціональної групи входять описи програмного продукту на найвищому рівні, а саме взаємодію користувача з ним. На цьому етапі визначається задача чи ряд задач, що може вирішити користувач за допомогою системи. Отже формується основна функція програмного продукту. Даний програмний продукт виконує функцію перевірки практичного застосування способу стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера. Програма має показати наскільки даний спосіб підходить для вирішення задач прихованої передачі та збереження інформації, а також допомогти провести аналіз які додаткові задачі можуть бути вирішені.

Результат, як раз, і має сформувати вектори розвитку та вдосконалення програмного продукту. Користувач має змогу зашифрувати певну інформацію та приховати її у растровому зображенні довільного розміру. В роботі програми, теоретично, можливі наступні сценарії:

- розмір секретних даних менший чи дорівнює розміру контейнера, а також розмір контейнера може бути поділений на ціле число блоків;
- розмір секретних даних більший за розмір контейнера;
- контейнер не вміщує цілої кількості блоків.

Програма опрацьовує кожен з наведених сценаріїв. Взаємодія з програмою відбувається через графічний інтерфейс. Програмний продукт має повну підтримку операційних систем Windows, починаючи з версії 7, а також вимоги, не більші ніж характеристики стандартного ПК.

Логічна група описів налічує описи інтерфейсів взаємодії, поведінку системи відносно зовнішніх та внутрішніх даних, а також описує потоки цих даних в системі. До зовнішніх даних програми входять: текст, який приховується, растрове зображення у форматі BMP, яке є контейнером для зберігання секретного повідомлення, розмір блоку перестановки, розрядність генератора псевдовипадкових чисел, ключі відповідної розрядності для генератора, спосіб отримання ключа, шлях для збереження вихідного зображення, яке містить приховане зашифроване повідомлення. Внутрішні дані налічують: ключі, що формуються за допомогою стохастичного моделювання, контрольна сума секретного повідомлення та його розмір, саме зашифроване секретне повідомлення, вихідний контейнер, наповнений бітами секретного повідомлення, а також розмір вихідного контейнера. Користувач має змогу вводити вхідні дані за допомогою спеціально відведених вікнах графічного інтерфейсу. Потік даних та поведінка програми являє наступну послідовність. Першим кроком користувач завантажує зображення у форматі BMP, програма декодує завантажене зображення. Після успішного декодування програма перевіряє зображення на відповідність умовам відсутності алгоритму стиснення та наявності сторонньої палітри кольорів. Якщо зображення не відповідає хоча б одній з умов, то програма повідомляє про це

користувача. У випадку, коли зображення не вдалось декодувати, програма також має повідомити користувача про несправність файлу зображення. Наступними даними після вхідного зображення, користувачем обираються розмір блоку перестановки, а також розрядність генератора псевдовипадкових чисел. Коли обрана розрядність генератора псевдовипадкових чисел, користувач обирає спосіб формування ключів, серед яких, ручний спосіб формування або за допомогою стохастичного моделювання. Якщо обраний ручний спосіб формування ключів, то необхідно додатково задати у відповідних вікнах інтерфейсу самі ключі. У випадку стохастичного моделювання ключів, програма сама сформує потрібні ключі. Останнім кроком користувач вводить текст, що потрібно приховати та обирає шлях для збереження вихідного зображення. Стадія опрацювання зовнішніх даних замінюється стадією формування внутрішніх даних. На другій стадії програма створює внутрішні дані для поточної сесії. Спочатку розраховується розмір вихідного зображення за формулами (1.5) та (1.6), щоб задовольняти умови (1.3) та (1.4) відповідно. Якщо вхідне зображення одразу задовольняє умови (1.3) та (1.4), то розмір вихідного зображення дорівнює розміру вхідного. Виходячи з отриманого розміру вихідного зображення, формується контейнер. Якщо був обраний спосіб формування ключів, то програма розраховує потрібні ключі. Маючи ключі, повідомлення та контейнер, програма спочатку шифрує зображення, потім контейнер підлягає перестановкам, після чого зашифроване повідомлення підставляється до контейнера. Контейнер повертається у початкове положення, чим стохастично розподіляє повідомлення по контейнера. Коли вихідне зображення готове, то програма зберігає його по заданому шляху. Зворотній процес, отримання прихованих даних, являє обернену процедуру, єдиною відмінністю є те, що на вхід подається зображення, яке містить приховані дані, а ключі мають відповідати початковим.

Фізична група описів представляє програмний продукт з точки зору розробки. Описується саме рішення, використовуючи функціональні вимоги до програмного продукту. Функціональні вимоги описують поведінку програми, базуючись на цих вимогах здійснюється розробка програмного продукту. Функціональні вимоги

розподіляються на: вимоги, що описують функції програми, вхідні дані та їх потік, вихідні дані, опис умов, необхідних для виконання функції, опис опрацювання помилок та даних, що не відповідають вимогам. Функції програмного продукту розподілені на функції обробки даних та функції стеганографічного алгоритму. Програма приймає шлях до вхідного зображення, який є строковим типом даних. Якщо вхідне зображення недоступне по заданому шляху, програма оброблює цю помилку та повідомляє користувача. Програма завантажує вхідне зображення та перевіряє його на відповідність умовам, якщо зображення не підходить, то програма повідомляє користувача про неможливість використання поточного зображення. Програма надає можливість обрати розмір блоку перестановки та розрядність генератора псевдовипадкових чисел за допомогою випадаючого списку, з переліченим у ньому можливими варіантами. Розміри блоку перестановки налічують два варіанти: 256, 65536. Варіантів розрядності генератора псевдовипадкових чисел налічується чотири: 64, 256, 1024, 2048. Вибір способу формування ключів відбувається за допомогою відповідного перемикача в інтерфейсі. Виходячи із заданої розрядності генератора та способу формування ключів, програма має блокувати можливість введення ключів вручну при способі стохастичного моделювання, а при ручному способі перевіряти відповідність введених ключів вимогам. У режимі отримання прихованих даних, на вхід програма очікує шлях до зашифрованого контейнера, а вікно вводу тексту замінюється вікном виводу секретного повідомлення. Перемикач способу формування ключів завжди залишається в положенні ручного формування, адже потрібно використовувати попередньо сформовані ключі. Описавши функції обробки даних, перейдемо до опису функцій стеганографічного алгоритму. Функції стеганографічного алгоритму поділяються на: функцію стохастичного моделювання ключів генератора, функцію генератора псевдовипадкових чисел, функцію перестановок пікселів, функцію шифрування секретного повідомлення, функцію наповнення контейнера, функцію перевірки цілісності повідомлення. Програма має змогу стохастично змоделювати ключі до генератора псевдовипадкових чисел, а також, обов’язково виконується перевірка ключів на відповідність вимогам до кожного ключа. Якщо програма

працює з вручну введеними ключами, то вони також проходять таку перевірку. У разі невідповідності вручну введених ключів, програма повідомляє користувача.

2.2 Опис програмної реалізації

Для забезпечення підтримки операційних систем Windows, починаючи з сьомої версії, у якості програмної технології використаний .NET Framework 4.5. Код програми написаний мовою C#. Використана парадигма – об’єктно орієнтоване програмування. Графічний інтерфейс створений за допомогою фреймворку WPF (Windows Presentation Foundation). Код програми поділений на модулі, кожен модуль представлений у вигляді окремого класу. Таке розділення є зручним для розробки програмного забезпечення, а також його налагодження та подальшої підтримки. Прототип інтерфейсу програми в режимі приховування інформації зображено на рис. 2.2.1.

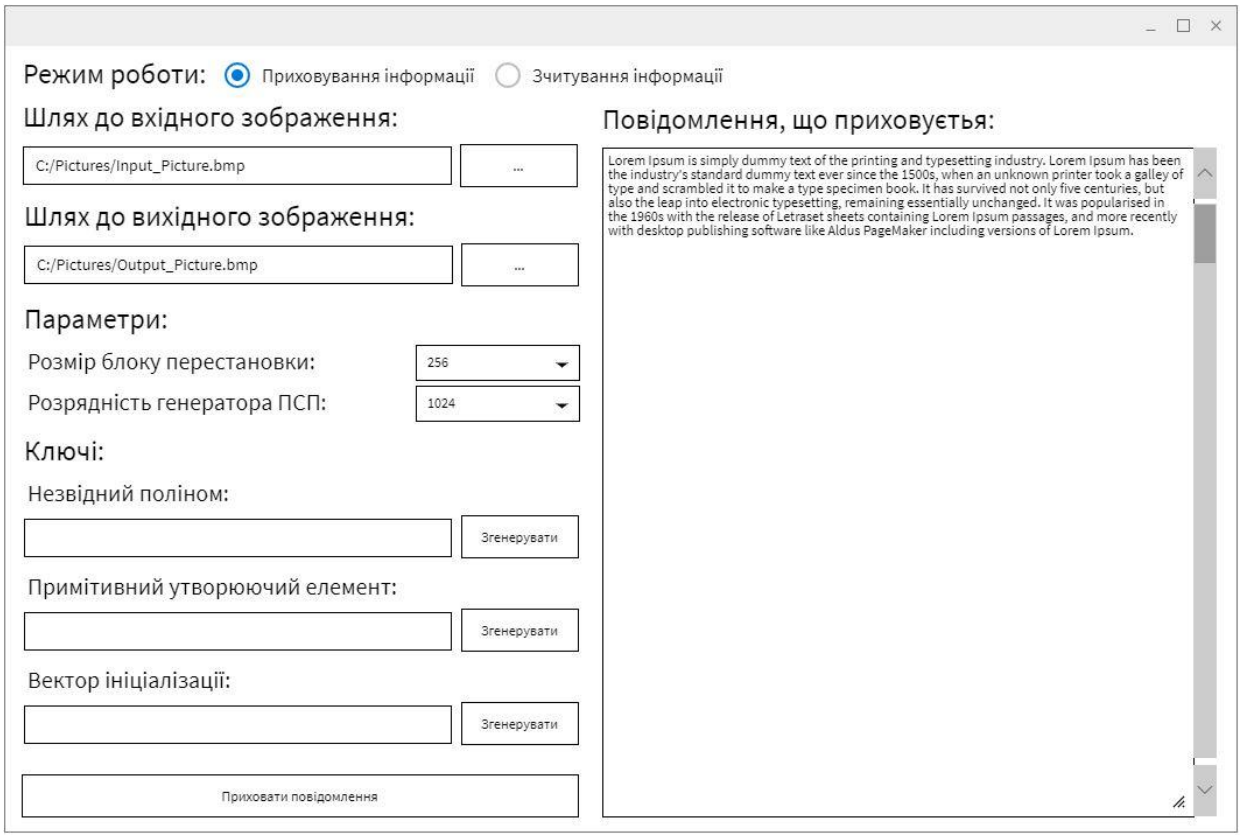


Рис. 2.2.1. Прототип інтерфейсу програми у режимі приховування інформації

Перемикач режиму роботи є елементом під назвою «радіокнопка», який представлений у просторі імен «System.Windows.Controls», класом «RadioButton». Важливо задати властивість «GroupName», де зазначається ім’я групи, до якої входять певні радіокнопки. При перемиканні радіокнопки змінюється властивість «IsChecked», яка є важливою властивістю, бо саме вона дає змогу відслідковувати активне положення радіокнопки. Вікна, де вказуються шляхи до вхідного та вихідного файлів відрізняються лише внутрішньою властивістю, яке зберігає відповідний шлях. Отже, достатньо розглянути тільки одне вікно. Текстове поле, що містить шлях до файлу, представлено класом «TextBox». Поруч знаходиться кнопка, яка відкриває вікно вибору файлу для відкриття. Властивість текстового поля «Text», яка містить строкове значення цього поля, допомагає відслідкувати відкриття користувачем файлу, властивість може змінюватись або при ручному вводі з клавіатури, або програмно після вибору файлу через вікно відкриття файлів. Параметри, такі як, розмір блоку перестановки та розрядність генератора ПВП обираються за допомогою випадаючого списку, клас «ComboBox». Відслідковується зміна вибраного елементу зі списку за допомогою властивості «SelectedItem». Вікна, де вказуються ключі подібні до вікон вибору шляху до файлу, але кнопка «Згенерувати» виконує дію запуску стохастичного моделювання, яка програмно створює відповідний ключ. У правій частині вікна програми розташоване текстове поле, в яке вводиться користувачем повідомлення, що потрібно приховати. Для такого текстового поля, властивостям «TextWrapping» та «VerticalScrollBarVisibility» задані значення «Wrap» та «Visible» відповідно. Кнопка «Приховати повідомлення» запускає виконання стеганографічного алгоритму та збереження вихідного зображення. В режимі зчитування інформації графічний інтерфейс програми має невеликі відмінності від графічного інтерфейсу в режимі приховання інформації. А саме, приховане вікно вибору шляху до вихідного зображення, бо в режимі зчитування повідомлення ніякого вихідного зображення не очікується. Також відсутні кнопки для стохастичного формування ключів, бо для розшифровки потрібно використовувати попередньо отримані ключі. Текстове поле з повідомленням є недоступним для ручого вводу символів з клавіатури. Його вміст

заповнюється програмно після вичитування інформації з контейнера. Прототип інтерфейсу програми в режимі зчитування інформації зображено на рис. 2.2.2.

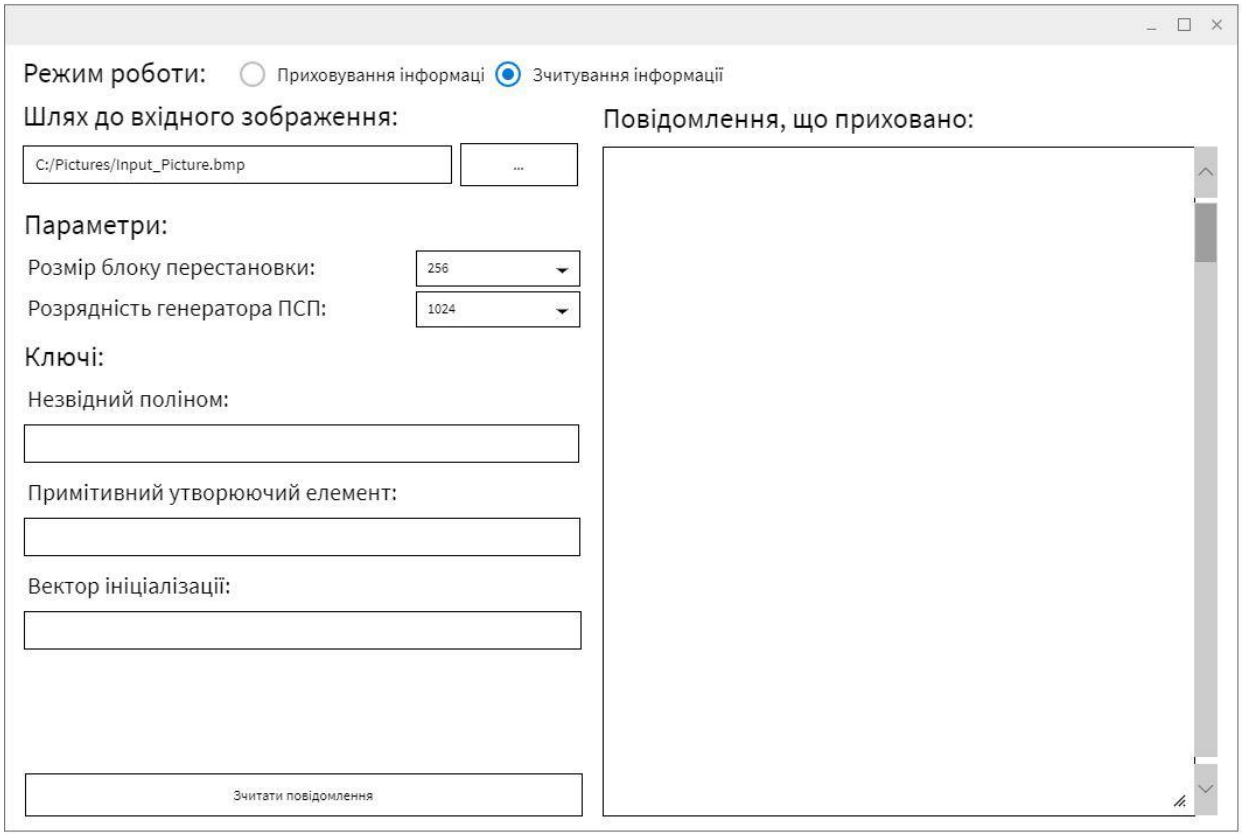


Рис. 2.2.2. Прототип інтерфейсу програми у режимі зчитування інформації

При описі елементів графічного інтерфейсу не раз зазначалося відслідковування змін у властивостях відповідних елементів. Це досягається за допомогою використання шаблону проєктування MVVM. Сам шаблон MVVM налічує три частини: графічний інтерфейс (View), дані програми (Model), які модифікуються певною логікою, а також спеціальний прошарок (ViewModel), що використовується для зв'язку, безпосередньо, графічного інтерфейсу та даних. Зв'язування властивостей елементів з приватними властивостями у коді відбувається за допомогою інтерфейсу «INotifyPropertyChanged», представленого у просторі імен «System.ComponentModel». А опрацювання команд, що подаються користувачем через графічний інтерфейс, наприклад натиск кнопки, можливе через використання інтерфейсу «ICommand» з простору імен «System.Windows.Input». Таким чином кожна властивість елементу графічного інтерфейсу, що зв'язана з даними програми, має відповідну публічну властивість в коді програмі. Ця

властивість в коді як раз і є частиною прошарку «ViewModel». Коли користувач, наприклад, змінює вміст текстового поля, то змінюється значення властивості «Text», так як до цієї властивості прив’язана властивість в коді, то зміни чіпляють і другу властивість. Отже, дані введені користувачем доступні у бекенд частині програми. Зв’язок властивостей двосторонній, а це значить, що логіка програми також може змінювати значення в графічному інтерфейсі. Подібно до елементів вводу інформації, таких як текстове поле, працюють і елементи керування, такі як кнопки, тільки при натиску на кнопку запускається прив’язана до неї лямда-функція.

У попередніх абзацах було сказано, що програма поділена на окремі частини. Кожна частину виконує певну функцію, яка є однією з основних чи допоміжних функцій або під-функцій програми. Щоб краще розподілити програму на незалежні частини, представимо процес її виконання у вигляді ланцюга викликів основних функцій, що зображений на рис. 2.2.3.

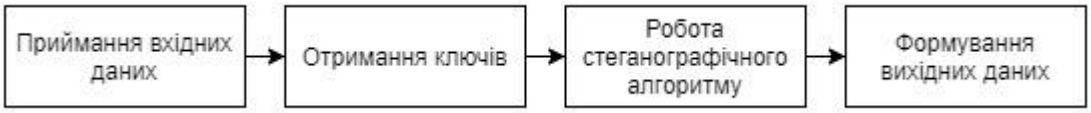


Рис. 2.2.3. Діаграма виклику основних функцій

Розглянемо основні функції та опишемо їх розподіл на незалежні частини. Першою основною функцією є приймання вхідних даних, вхідні дані програми є, завжди, зображеннями. Функція приймання вхідних даних поділяється на дві частини. Перша має необхідний набір методів для відкриття зображення дозволеного типу, а саме растрове зображення у форматі BMP. Після відкриття ми матимемо масив байтів, певної довжини. А друга містить методи для представлення цього масиву байтів у вигляді об'єкту зображення. Коли вхідні дані опрацьовані, програма отримує ключі до стегаографічного алгоритму, ця функція поділена на кілька частин: генератор незвідних поліномів, генератор примітивних утворюючих елементів, генератор векторів ініціалізації, утворювач загального ключа, який створює об'єкт загального ключа з отриманих за допомогою відповідних генераторів

або введених вручну ключів. Зі сформованим об'єктом загального ключа та об'єктом зображення ініціалізується стеганографічний алгоритм. Стеганографічний алгоритм поділений на більшу кількість частин, тому на рис. 2.2.4 наведено, додатково, діаграму викликів основних під-функцій.

Рис. 2.2.4. Діаграма виклику основних під-функцій при приховуванні повідомлення

Розберемо під-функції стеганографічного алгоритму та частини на які вони поділені. Зчитування ключів складається з однієї частини, задає ключі для ініціалізації генератора ПВП. Коли ключі до генратора задані, то утворюється, безпосередньо, об'єкт генератора ПВП. Такий об'єкт має можливість видавати псевдовипадкове число певної розрядності. Далі рахується контрольна сума повідомлення. Після отримання можливості генерації псевдовипадкового числа, починається процес шифрування повідомлення. Кожен байт повідомлення шифрується за допомогою гамування його зі згенерованим числом. Після шифрування повідомлення розраховується розмір котейнера, який зможе вмістити в собі це повідомлення. На виході маємо масив байтів потрібної довжини. Коли контейнер готовий, створюється об'єкт з алгоритмом перестановок. При створені такого об'єкту рахується кількість блоків, що вміщується у контейнері, якщо вміщується неціле число, то контейнер доповнюється потрібною кількістю байтів. Після поділення контейнера на блоки, для кожного блоку генерується таблиця перестановок, використовуючи генератор псевдовипадкових чисел. За допомогою цього об'єкту відбувається перестановка пікселів, відповідно таблиць перестановок, а також відновлення положень пікселів. Приховується зображення за допомогою звичайного циклу, який перебирає біти зашифрованого повідомлення, а потім підміняє молодший біт байту контейнера. Однак стеганографічний алгоритм може працювати

як для приховування, так і для зчитування повідомлення. Діаграма викликів основних під-функцій у режимі зчитування зображена на рис. 2.2.5.

Рис. 2.2.5. Діаграма виклику основних під-функцій при зчитуванні повідомлення

З діаграми бачимо, що частина під-функцій співпадає з тими, які використовувались для приховування. Отже, розберемо тільки ті, що не співпадають. Зчитування повідомлення являє зворотній процес до приховання, а саме, у циклі перебираються байти контейнера і з молодших бітів формується ще один масив байтів, який і є нашим повідомленням. Але зчитане повідомлення зашифроване, для того щоб розшифрувати потрібно піддати кожен байт повторній операції гамування, у результаті отримаємо вихідне повідомлення. Не завжди ми можемо бути впевненими, що зчитане повідомлення відповідає тому, що приховувалось. Тому для зчитаного повідомлення рахується контрольна сума і порівнюється з контрольною сумою прихованого повідомлення, яка була також захована у контейнері. Розглянувши функцію стеганографічного алгоритму, залишається остання, яка виконує формування вихідних даних. Ця функція поділяється на створення об’єкту вихідного зображення та об’єкту вихідного повідомлення, відповідно до режиму роботи програми. Поміж основних функцій є ще і допоміжні, такі як перетворення типів даних, відкриття та збереження файлів, контроль блокування чи приховування елементів графічного інтерфейсу, перевірка введених даних.

2.3 Стратегія тестування

Під час розробки програмного забезпечення велику роль відіграє тестування. Адже саме тестування дає змогу перевірити чи відповідає код програми заданим вимогам. Дуже важливо проробити стратегію тестування, бо стратегія тестування описує мету тестування, а також спосіб, у який буде тестуватись програма. Стратегія тестування даного програмного продукту нескладна. Тестування обмежується лише перевіркою компонентів. Такий тип тестування передбачає розбиття програми на окремі модулі, одиничні елементи програми, які не залежать один від одного. До кожного такого модуля пишеться окремий тест, який може розділятися на кілька частин, це залежить від кількості варіантів роботи модуля, які потрібно покрити тестами. Метою тестування є перевірка відповідності кожного модуля вимогам. Також важливим є критерій покриття коду тестами. Саме цей критерій дає змогу оцінити наскільки система протестована. Теоретично, щоб протестувати систему повністю, кількість тестів має дорівнювати повній кількості всіх логічних входів системи. Таке покриття на великій кількості логічних входів системи є практично неможливим, приклад розрахунку кількості тестів T_{count} відносно кількості логічних входів I_{count} наведено у виразі (2.1).

$$T_{count} = 2^{I_{count}} = 2^{32} = 4294967296 \quad (2.1)$$

З прикладу бачимо, що практично протестувати систему, у якій є тридцять два логічні входи, неможливо. Тому існують різні критерії достатнього покриття. Один з таких критеріїв називається MC/DC, саме цей критерії застосовується для покриття даного програмного коду. Існує багато різних формулювань критерія покриття MC/DC, ми ж сформулюємо його наступним чином. Якщо вплив кожного класу еквівалентності на логіку, а також кожне граничне значення протестовані, тоді ця логіка вважається покрита. Розглянемо невеликий приклад, нехай потрібно протестувати функцію $F(A, B, C, D)$, яка приведена у виразі (2.2).

$$F(A, B, C, D) = A \mid B \mid C \mid D \quad (2.1)$$

Якщо покривати функцію $F(A, B, C, D)$ за кількістю логічних входів, то потрібно шістнадцять тестів, більша частина з яких не несе реального корисного навантаження, а тільки перевіряє вже перевірене. При використанні достатнього

критерію покриття, нам знадобиться лише п’ять тестів. Потрібно протестувати, що при будь-якому вході з ненульовим значенням, функція $F(A, B, C, D)$ повертає також ненульове значення, на це нам потрібно чотири тести. А також ще один, щоб перевірити, що функція $F(A, B, C, D)$ повертає нульове значення, коли всі входи також мають нульові значення. Отже, для даного прикладу бачимо, що при використанні достатнього критерію покриття потрібно майже в три рази менше тестів, при чому результат тестування залишається незмінним.

Розібравши загальний приклад покриття коду тестами, розберемо також реальний приклад, покривши одну з частин коду. Наприклад перевірку полінома на незвідність. Для цього представимо його у вигляді блок-схеми, яка зображена на рис. 2.3.1. З блок-схеми бачимо, що вхідними даними є поліном степені n, а на виході отримуємо булеве значення, що вказує на незвідність полінома. Також є три логічні конструкції, які потрібно покрити тестами. Почнемо з першої, яка перевіряє довжину полінома, непарність його ваги, наявність ненульового молодшого та старшого коефіцієнтів. Для покриття цієї частини логіки потрібно подати чотири поліноми, які будуть задовольняти тільки одну логічну умову, після чого отримати оцінку що поліном не відповідає умові. А для покриття останніх двох логічних конструкцій, потрібно подати ще два поліноми, які задовольняють усі умови самої першої логічної конструкції. Один з цих поліномів має бути завідомо незвідним, а інший зіставним. Результат має показати для незвідного полінома позитивну відповідь, а для зіставного полінома негативну.

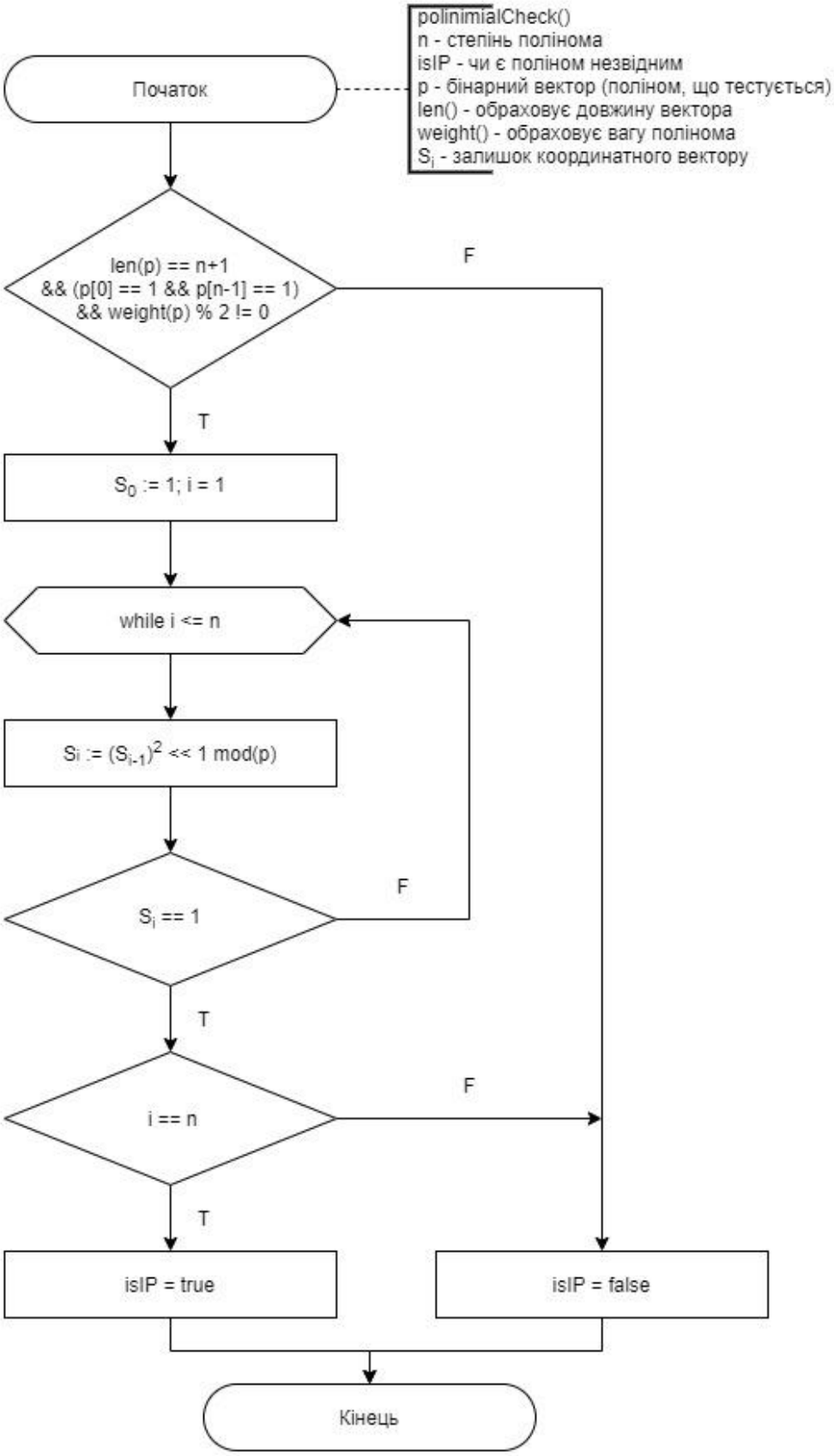


Рис. 2.3.1. Блок-схема перевірки полінома на незвідність

Частиною стратегії тестування, також, є фреймворк для тестування. Фреймворк для тестування – це певний набір інструментів та бібліотек, які дозволяють створювати автоматизовані тести для програмного забезпечення. До функцій фреймворку для тестування входять:

- завдання єдиного формату тестів;
- зручне задання тестових наборів даних;
- підключення необхідних бібліотек або їх симуляція;
- автоматизований запуск тестів;
- вивід результатів тестування.

За допомогою фреймворку для тестування збільшується швидкість написання тестів та стандартизується формат самих тестів, за рахунок чого спрощується підтримка тестів у майбутньому і повторне використання коду. Також, при стандартизованій системі створення тестів їх простіше запускати на різних платформах та читати звіт тестування. Кожен фреймворк для тестування поділений на три частини: оперування даними, створення бізнес-логіки, ядро самого фреймворку. Спрощена структура фреймворку зображена на рис. 2.3.2.

Рис. 2.3.2. Спрощена структура фреймворку для тестування

Рівень керування даними включає у себе заготовлені данні для тестування, а також користувацькі файли конфігурації. На цьому рівні задаються вхідні дані, дані, які очікуються, а також описується та налаштовується оточення. На рівні опису логіки створюються самі тести, використовуючи функції, які представлені у ядрі фреймворку. Також на цьому рівні задається симуляція оточення, якщо є потреба у ньому. Останній рівень, ядро фреймворку, містить основні методи, класи, функції для роботи з фреймворком. Існує багато готових рішень, які дозволяють створювати автоматизовані тести. Серед найпопулярніших фреймворків, що підтримують мову C# існують: xUnit, NUnit, MSTest. Для створення тестів до програмного продукту було використано фреймворк MSTest. Цей фреймворк розроблений компанією Microsoft, та підтримується у середовищі розробки Visual Studio, що робить його зручним для інтеграції у проект.

РОЗДІЛ 3
РЕЗУЛЬТАТИ АНАЛІЗУ ТА ТЕСТУВАННЯ

3.1 Результати тестування

Отримані результати в ході запуску програми підтверджують можливість захисту стеганографічного інформації на основі стохастичного переставлення пікселів контейнера, а також можливість роботи за стеганографічною моделлю Крістіана Кашена, яка зображена на рис. 3.1.1.

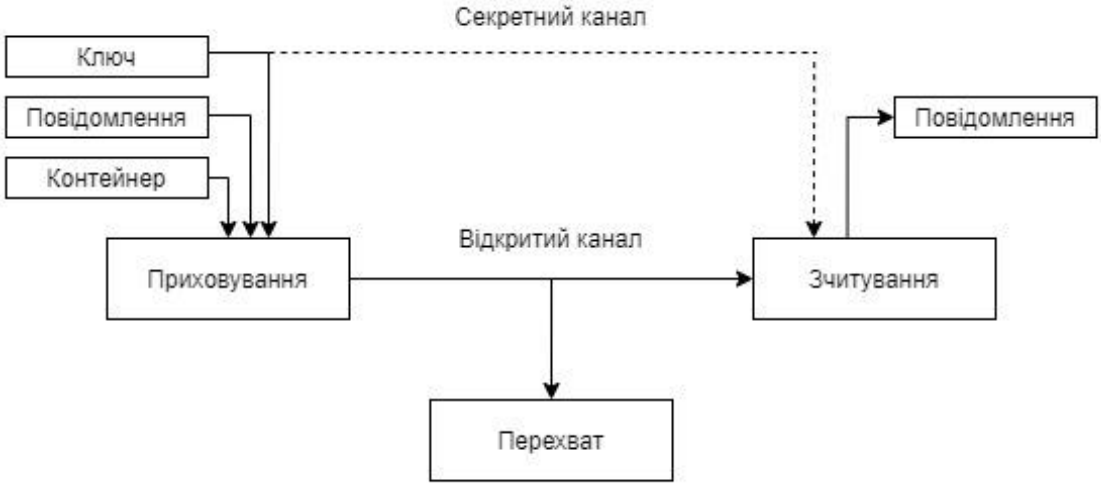


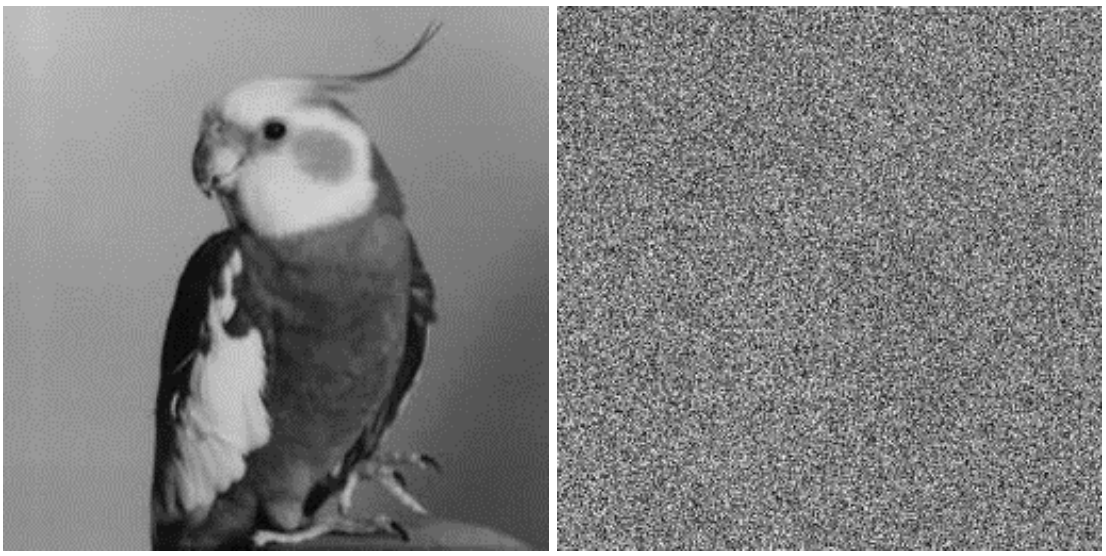
Рис. 3.1.1. Спрощена стеганографічна модель Крістіана Кашена

На схемі зображення спрощена модель, вона відрізняється від повної тим, що у повній моделі, коли система знаходиться у пасивному режимі, то по відкритому каналу зв’язку відправляються пусті контейнери. У даному випадку є сенс розглядати систему тільки у активному режимі. Система знаходиться у активному режимі, коли по відкритому каналу зв’язку відправляються контейнери з прихованим повідомленням.

В моделі є три актори: адресант, адресат та зловмисник. Задача адресанта непомітно для зловмисника передати секретне повідомлення. Адресат має зчитати секретне повідомлення, а зловмисник не повинен мати змогу виявити наявність прихованої інформації. Модель працює наступним чином: адресант генерує ключ та приховує секретне повідомлення у контейнері за допомогою ключа; готовий контейнер відправляється відкритим каналом зв'язку, а ключ передається іншим, секретним, каналом зв'язку; під час передачі контейнера, зловмисник перехоплює контейнер, але він не підозрює наявність у контейнері прихованої інформації; адресат отримує ключ і контейнер та зчитує секретне повідомлення. У такому випадку повідомлення вважається переданим. Але потрібно зазначити, що зловмисник не повинен виявити наявність секретного повідомлення.

За допомогою програмної реалізації даного способу стеганографічного алгоритму було успішно зашифровано та приховано секретне повідомлення. Візуальних змін у зображенні, яке містило приховане повідомлення, не було виявлено. Зчитування та розшифровка секретного повідомлення пройшли успішно. Контрольні суми прихованого та зчитаного повідомлень були однакові.

Щоб під час аналізу молодших бітів контейнера уникнути будь-яких підозр щодо наявності прихованої інформації, перед приховуванням повідомлення пікселі контейнера підлягають стохастичним перестановкам. Таким чином руйнується зв'язок між бітами секретного повідомлення. Ілюстрація розподілу пікселів растрового зображення при здійсненні стохастичних перестановок зображена на рис 3.1.2.



а) б)

Рис. 3.1.2. Розподіл пікселів при стохастичній перестановці

З прикладу бачимо, що при стохастичній перестановці пікселів наше зображення перетворюється на білий шум. Отже, при повернені пікселів назад, коли секретне повідомлення було приховане, то біти повідомлення будуть розподілені стохастично і представляти не зв’язану послідовність, а білий шум.

3.2 Результати аналізу

Не дивлячись на стохастичні перестановки пікселів, повідомлення шифрується криптографічним алгоритмом. Шифрування повідомлення необхідне для забезпечення додаткового рівня захищеності, а також для підняття ентропії. Адже, якщо взяти з кожного байту зображення молодший біт та утворити з отриманої послідовності біт послідовність байтів, то ентропія такого масиву байтів буде мати інший характер, ніж ентропія текстового повідомлення. Тому повідомлення шифрується перед приховуванням. Розглянемо приклад з повідомленням: «Спосіб стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера». Ентропія такого повідомлення становить приблизно 3,67. Розподіл байтів відкритого повідомлення наведено у таблиці 3.2.1.

Таблиця 3.2.1

Розподіл байтів відкритого повідомлення

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	4	8	6	1	2	2	1	2	0	0	0	0	0	0	0	1
144	0	0	0	0	0	0	7	1	0	0	0	0	0	0	0	0
160	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
176	8	1	3	4	0	7	0	1	2	1	2	2	1	10	11	3
192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
208	57	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0
224	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

З розподілу байтів відкритого повідомлення бачимо, що байти зосереджені групами у певних діапазонах значень. Це добре видно на гістограмі, яка наведена на рис. 3.2.1.

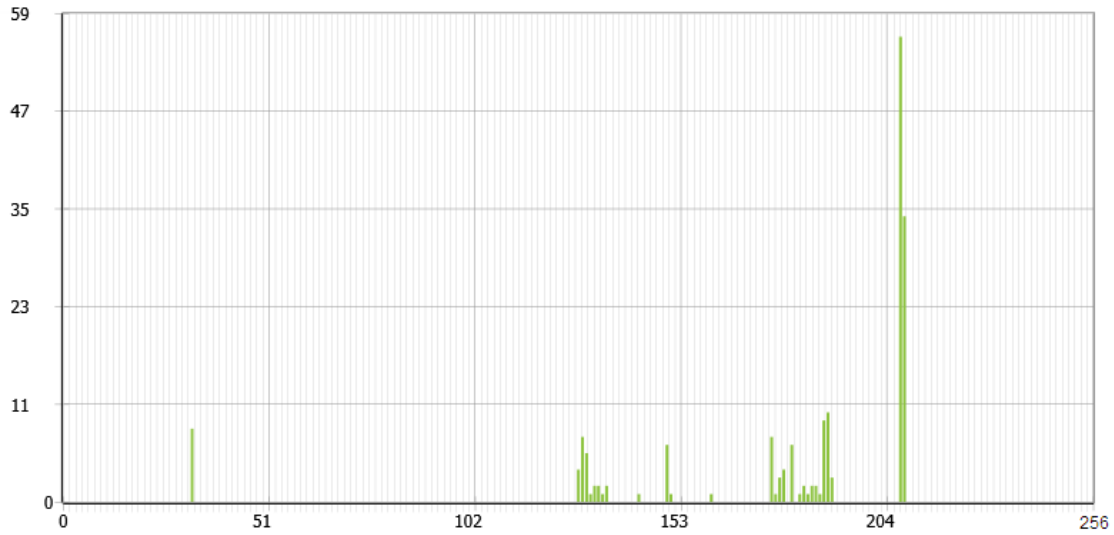


Рис. 3.2.1. Гістограма розподілу байтів відкритого повідомлення

Проаналізувавши розподіл байтів відкритого повідомлення, бачимо, що присутній, по-перше, байт зі значенням 32 у кількості дев’яти одиниць. Це значення відповідає відступу між словами у схемах кодування символів. Отже, вже спливає на думку, що це речення з десяти слів. Далі бачимо, що інші значення зустрічаються у діапазоні від 128 до 209. Саме ці значення байтів представляють кирилицю у системах кодування. Далі нескладно відновити повідомлення, хоча б частково. Тому повідомлення шифрується. Ентропія зашифрованого повідомлення, яке використане у якості прикладу, приблизно дорівнює 6,9. Бачимо, що ентропія зросла майже в два рази, а це вказує на більш рівномірний розподіл значень байтів. Розподіл байтів зашифрованого повідомлення наведено у таблиці 3.2.2.

Таблиця 3.2.2

Розподіл байтів зашифрованого повідомлення

№	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	1	2	2	1	0	0	1	1	0	0	0	2	0
16	1	1	0	0	1	1	1	0	0	2	2	1	1	0	1	1
32	1	1	0	1	0	2	1	0	0	0	0	1	0	1	1	1
48	2	3	0	0	0	0	0	0	2	0	0	2	2	1	1	1
64	1	1	1	0	2	0	0	0	1	0	0	1	3	0	1	1
80	0	0	0	2	0	0	1	0	1	1	0	1	1	2	1	1
96	4	0	2	0	0	0	0	3	2	0	2	2	0	0	0	0
112	0	0	2	1	1	0	2	0	0	0	1	0	0	2	2	0
128	1	0	0	1	0	1	0	0	0	0	0	1	1	1	0	0
144	0	0	1	1	0	0	2	3	0	0	1	2	1	3	1	2
160	0	1	0	1	0	0	1	3	0	1	1	1	1	2	0	0
176	0	0	0	1	0	0	0	2	4	1	0	0	3	0	1	1
192	0	2	1	0	1	1	1	1	0	3	0	0	0	0	0	1
208	0	0	0	0	0	0	0	1	0	2	0	1	0	0	2	2
224	1	0	1	1	0	1	4	1	1	0	2	1	0	2	1	0
240	1	1	0	0	2	1	2	1	0	2	0	1	0	1	1	0

Звісно, ентропія зашифрованого повідомлення, що має мінімум 256 символів, значно більша, вона приблизно наближується до восьми. Але так як наше повідомлення, що використане для прикладу, має меншу кількість символів, то і ентропія відповідно менша. З розподілу байтів зашифрованого повідомлення неможливо побачити будь-які особливості такої послідовності. Значення вже не зосереджені групами у певному діапазоні, а також немає домінуючого значення. Це добре видно на гістограмі, зображеній на рис. 3.2.2.

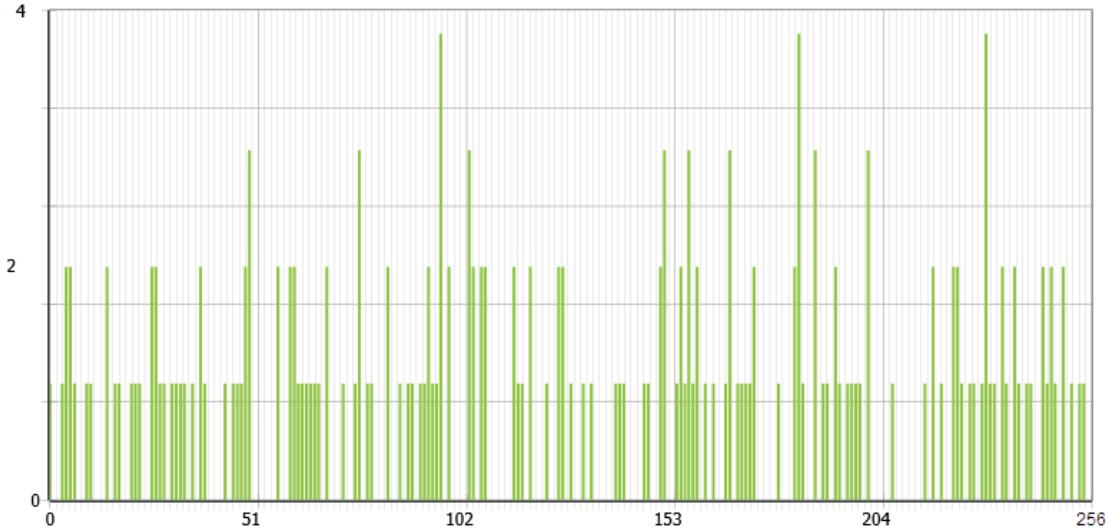


Рис. 3.2.2. Гістограма розподілу байтів зашифрованого повідомлення

Шифрування секретного повідомлення та здійснення стохастичних перестановок неможливе без генератора псевдовипадкових чисел. В попередніх розділах зазначалось, що генерація псевдовипадкових чисел базується на генераторах псевдовипадкової послідовності, які являють собою лінійні регістри зсуву з лінійним зворотнім зв’язком. Такі генератори мають легку реалізацію як програмно, так і апаратно, схема такого генератора зображена на рис. 3.2.3.

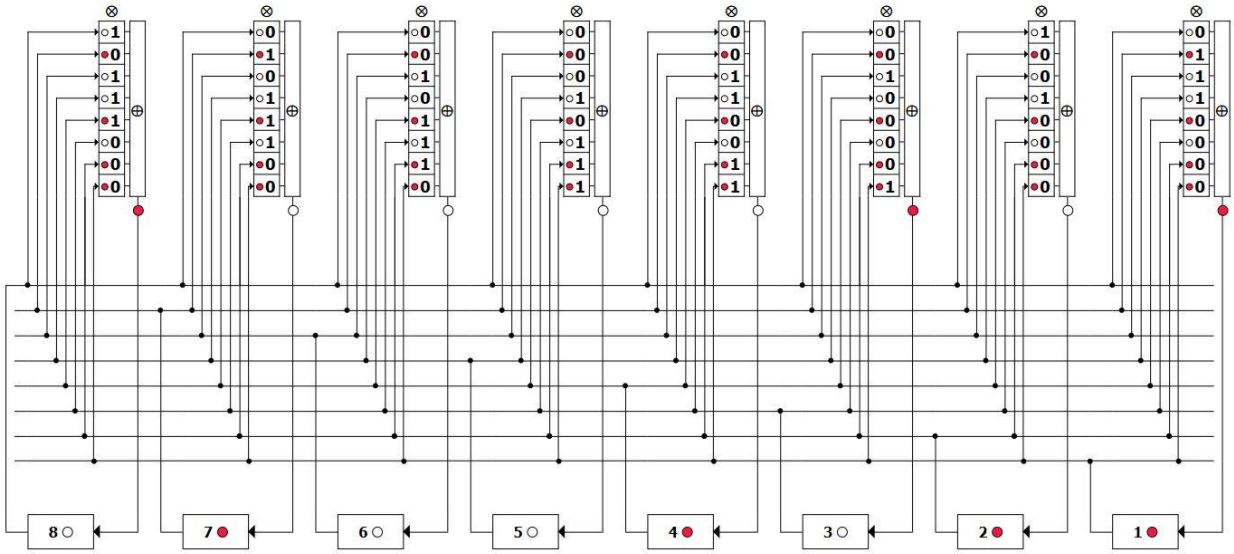


Рис. 3.2.3. Схема генератора псевдовипадкової послідовності

На схемі зображений генератор восьмого порядку. На кожному такті генератора утворюється вектор відповідної довжини, який також використовується для отримання нового вектору, активуючи потрібні верхні регістри, на схемі він записується у нижні регістри. Верхні регістри представляють матрицю Галуа, Фібоначчі або спряжену їм, яка утворена незвідним поліномом та примітивним утворюючим елементом. Верхні вертикальні регістри поділяються на два типи. Регістри першого типу виконують дію порозрядного множення по модулю двійки, такі регістри позначені символом $\otimes$. А регістри другого типу виконують операцію порозрядного додавання по модулю двійки, вони позначені символом $\oplus$.

Генератори на базі лінійних регістрів зсуву з лінійним зворотнім зв'язком здатні утворювати псевдовипадкову послідовність великого періоду з гарними статистичними властивостями. Але у таких генераторів є один недолік, який полягає у тому, що, теоретично, існує можливість знайти поліном, яким утворюється послідовність. Відомий алгоритм Берлекемпа-Мессі дає змогу знайти мінімальний поліном, що подається на вхід лінійної рекурентної послідовності над довільним полем. Після проведеного аналізу можливості знаходження полінома, за допомогою якого генерувалась псевдовипадкова послідовність, було виявлено емпіричним шляхом, що результат алгоритму Берлекемпа-Мессі є завжди примітивним поліномом. А це означає, що для кожного примітивного полінома існує сімейство

слабких ключів, примітивних утворюючих елементів. Приклад для примітивних поліномів восьмої степені наведено у таблиці 3.2.3.

Таблиця 3.2.3

Слабкі ключі у вісімковій системі числення

№	ПрП	1	2	3	4	5	6	7	8
1	100011101	002	004	020	035	114	137	205	235
2	100101011	002	004	020	053	301	351	356	373
3	100101101	002	004	020	055	135	165	345	366
4	101001101	002	004	020	115	253	343	353	370
5	101011111	002	004	020	137	206	225	232	300
6	101100011	002	004	020	143	223	341	363	364
7	101100101	002	004	020	145	213	214	231	355
8	101101001	002	003	004	005	020	021	150	151
9	101110001	002	004	020	057	133	161	340	363
10	110000111	002	004	020	035	064	157	207	326
11	110001101	002	004	020	075	140	177	215	270
12	110101001	002	004	020	030	202	251	315	351
13	111000011	002	004	020	037	226	240	303	375
14	111001111	002	004	020	126	130	166	240	317
15	111100111	002	004	020	054	113	175	347	352
16	111110101	002	004	020	050	147	176	323	365

Отже, слабкі ключі мають тільки ті незвідні поліноми, що являються примітивними, а це означає, що при використанні незвідного полінома, який не є примітивним, отримати цей його через алгоритм Берлекемпа-Мессі неможливо. Таке твердження дає можливість використання генератора псевдовипадкової послідовності, коли незвідний поліном не є примітивним. Тільки у такому випадку можна отримати максимальну крипто-стійкість алгоритму.

3.3 Вектори подальшого розвитку

Окрім цілі розробки програмного забезпечення для перевірки можливості здійснення стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера, була ще ціль отримати можливі вектори подальшого розвитку способу, а також дослідити спектр задач, які можна вирішити за допомогою даного способу. Для початку розберемо основні цілі стеганографії, до цілей стеганографії можна віднести: приховану передачу даних, зберігання цифрових відбитків, зберігання стеганографічних водяних знаків. Перша ціль, прихована передача даних, є історично найперша ціль для якої застосовується стеганографія. Задача полягає у передачі інформації таким чином, щоб третя особа і не здогадалась, що повідомлення було передано. До практичних застосунків пов'язаних до цієї цілі можна віднести: непомітну передачу інформації, приховане збереження інформації. Приховане збереження даних є одним із основних практичних застосунків стеганографії. Цей практичний застосунок було успішно підтверджено за допомогою розробленого програмного продукту, який дав змогу успішно приховати текстове повідомлення у растровому зображенні. Після збереження зображення, яке містить приховані дані, візуальних змін не було виявлено, тому можна вважати, що текст приховано зберігається у зображені. Непомітна передача даних схожа на приховане збереження даних, окрім того, що зображення, яке містить приховане повідомлення, передається будь-яким каналом зв’язку. Наступна ціль, зберігання цифрових відбитків, полягає у збереженні певної ідентифікуючої інформації у контейнері. Ця інформація використовується для захисту контенту від повторного комерційного використання. Наприклад, на торговій площадці, де продаються зображення з умовою заборони на повторне використання у комерційних цілях, користувач купляє зображення, тоді у зображення приховується інформація, яка допомагає ідентифікувати покупця, це може бути електронна адреса, номер телефону, ім’я, номер транзакції. Якщо така людина захоче перепродавати зображення, коли це заборонено, то у торгової площадки, яка має ексклюзивні права на зображення, будуть всі можливості юридично захистити себе від цифрового піратства. Приховування такого цифрового

відбитку відбувається за допомогою стеганографії. Також цифровий відбиток не повинен легко виявлятися і підмінятися. Стеганографічний спосіб захисту інформації, який описується у даній роботі, задовольняє ці умови, а це означає, що він здатний вирішувати цю задачу. Зберігання стеганографічних водяних знаків є подібним до зберігання цифрового відбитку, різниця лише в тому, що водяний знак це певна незмінна інформація, що проставляється у кожний екземпляр контейнера.

Перевагами даного способу є його простота реалізації, можливість реалізувати алгоритм як програмно так і апаратно, висока ентропія зашифрованого повідомлення, висока крипто-стійкість контейнера. У результаті перевірки програмної реалізації способу стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера, сформовані наступні вектори розвитку:

- створення апаратної реалізації для аналізу роботи даного алгоритму на базі мікроконтролера або мікропроцесора, а також програмованих логічних інтегральних схем;
- розширення спектру можливих форматів зображень, спроба створення додаткової модифікації для підтримки алгоритмів стискання;
- підтримка інших видів контейнера, таких як, відео, аудіо-файли, документи;
- створення програмних рішень для рішення задач збереження цифрових відбитків та стеганографічних водяних знаків.

ВИСНОВКИ

Отже, в результаті досліджень була підтверджена можливість практичного використання запропонованого способу стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера. Були, також, виявлені наступні переваги запропонованого способу:

- швидкість роботи алгоритму;
- простота програмної реалізації;
- можливість програмної реалізації на різних платформах;
- можливість апаратної реалізації;
- універсальність використання алгоритму;
- стійкість до атак.

Розроблена програмна реалізація дає змогу вирішити задачі прихованого зберігання та передачі інформації, а також виконує демонстраційні функції та може використовуватись у навчальних цілях. Завдяки стохастичній перестановці пікселів, зловмисник не в змозі виявити приховану інформацію, а також здогадатись про її наявність. Щоб отримати оригінальну послідовність, не маючи відповідних ключів, зловмиснику знадобиться перебрати $N!$ варіантів, що є практично неможливим. До того ж сама послідовність знаходиться у зашифрованому стані. Варіант, коли зловмисник може підібрати відповідні ключі, також практично неможливий, адже у якості основного ключа використовується незвідний поліном великої степені. Множина можливих векторів дуже велика для здійснення перебору. Навіть якщо орієнтуватися саме на перебір усіх можливих незвідних поліномів. Та навіть з відомим незвідним поліномом не буде змоги отримати таблицю перестановки та розшифрувати повідомлення, потрібно ще мати примітивний утворюючий та вектор ініціалізації, які також є практично неможливими для атаки в лоб. Вразливість генераторів псевдовипадкової послідовності на базі лінійних регістрів зсуву з лінійним зворотним зв’язком, яка полягає у можливості отримання незвідного полінома з відрізку згенерованої послідовності за допомогою алгоритму Берлекемпа-Мессі, притаманна тільки для класичних генераторів, такі генератори використовують примітивні незвідні поліноми, які мають ряд слабких ключів.

Генератори на основі матриць Галуа, Фібоначчі та спряжених їм здатні працювати з незвідними поліномами, що не є примітивними. Послідовність, яка утворена незвідним поліномом, що не є примітивним, не має слабких ключів, а це означає неможливість знаходження незвідного полінома за допомогою алгоритму Берлекемпа-Мессі.

Окрім перевірених застосувань даного способу для вирішення задач прихованої передачі та прихованого збереження інформації, існує також можливість застосування його для вирішення задач приховання цифрових відбитків та стеганографічних водяних знаків. Серед можливих векторів розвитку алгоритму є:

- використання завадостійкого кодування для можливості відновлення послідовності у разі наявності завад у каналі зв’язку;
- модифікації для сумісності з алгоритмами компресії;
- використання різних типів контейнера.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Білецький А. Я. Алгоритм синтезу незвідних поліномів лінійної складності / А. Я. Білецький, А. В. Коавльчук, К. А. Новіков, Д. А. Полторацький // Захист інформації. – 2020. – № 2. – С. 74–87.
2. Міжнародна науково-технічна конференція «АВІА» АВІА-2019 : тези доп., 23–25 квіт. 2019 р., Київ / Нац. ун-т «Національний авіаційний університет». – Київ : Вид-во Нац. ун-ту «Національний авіаційний університет», 2019. – 13.5 с.
3. Програмний комплекс потокового байт-орієнтованого Галуа шифрування : матеріали XV науково-технічної конференції студентів, аспірантів, докторантів та молодих учених, 21–22 лист. 2018 р., Київ, Україна / нац. ун-т «Національний авіаційний університет». – Київ, 2018. – 72 с.
4. Білецький А. Я. Примітивні матриці Галуа в криптографічних застосуваннях / А. Я. Білецький // Захист інформації. – 2014. – № 4. – С. 274–283.
5. Білецький А. Я. Систематичні байт-орієнтовані коди / А. Я. Білецький, Д. В. Конюший, Д. А. Полторацький // Захист інформації. – 2018. – № 1. – С. 18–31.
6. Белецкий А. Я. Алгебраические основы теории кодирования и криптографии. / А. Я. Белецкий. – Аграр Медиа Груп, 2017. – 176 с.
7. GitHub [Електронне джерело] – 2020. - GammaGenerator/GammaGaloisMatrix/MainWindow.xaml.cs at master · licurg/ GammaGenerator. – режим доступу: https://github.com/licurg/GammaGenerator/blob/master/GammaGaloisMatrix/MainWindow.xaml.cs (дата звертання 01.06.2021). – Назва з екрана
8. GitHub [Електронне джерело] – 2018. – RASTR-ENCRYPTOR/RASTR-ENCRYPTOR/RASTR-2/ at master · licurg/RASTR-ENCRYPTOR. – режим доступу: https://github.com/licurg/RASTR-ENCRYPTOR/tree/master/RASTR-ENCRYPTOR/RASTR-2 (дата звертання 01.06.2021). – Назва з екрана

Додаток 1

Вихідний код графічного інтерфейсу

```
<Window x:Class="Steganography.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:local="clr-namespace:Steganography"
      mc:Ignorable="d"
      Title="Steganography"
      Background="#ecf0f1"
      Height="600"
      Width="1200">
   <Window.DataContext>
      <local:ViewModel/>
   </Window.DataContext>
   <Grid Margin="10">
      <Grid.RowDefinitions>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="*"/>
      </Grid.RowDefinitions>
      <WrapPanel Grid.Row="0">
         <Label Content="Режим роботи:" Style="{StaticResource MainLabel}"/>
         <RadioButton Content="Приховування інформації" GroupName="mode" IsChecked="{Binding
ForwardMode}" Style="{StaticResource MainRadioButton}"/>
         <RadioButton Content="Зчитування інформації" GroupName="mode" IsChecked="{Binding
ReverseMode}" Style="{StaticResource MainRadioButton}"/>
      </WrapPanel>
      <Grid Grid.Row="1">
         <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
         </Grid.ColumnDefinitions>
         <Grid Grid.Column="0" Margin="0,0,5,0">
            <Grid.RowDefinitions>
               <RowDefinition Height="Auto"/>
               <RowDefinition Height="Auto"/>
               <RowDefinition Height="Auto"/>
               <RowDefinition Height="Auto"/>
               <RowDefinition Height="Auto"/>
               <RowDefinition Height="Auto"/>
               <RowDefinition Height="Auto"/>
               <RowDefinition Height="Auto"/>
```

```
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Label Grid.Row="0" Margin="0,5,0,0" Content="Шлях до вхідного зображення:"
Style="{StaticResource MainLabel}"/>
        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>
            <TextBox Grid.Column="0" Style="{StaticResource MainTextBox}" Text="{Binding
InputFilePath}"/>
            <Button Grid.Column="1" Content="•••" Style="{StaticResource FileButton}"
Command="{Binding OpenInputFile}"/>
        </Grid>
        <Label Grid.Row="2" Margin="0,5,0,0" Content="Шлях до вихідного зображення:"
Style="{StaticResource MainLabel}" Visibility="{Binding OutputImageVisibility}"/>
        <Grid Grid.Row="3" Visibility="{Binding OutputImageVisibility}">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>
            <TextBox Grid.Column="0" Style="{StaticResource MainTextBox}" Text="{Binding
OutputFilePath}"/>
            <Button Grid.Column="1" Content="•••" Style="{StaticResource FileButton}"
Command="{Binding OpenOutputFile}"/>
        </Grid>
        <Label Grid.Row="4" Content="Параметри:" Style="{StaticResource MainLabel}"/>
        <Grid Grid.Row="5">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Label Grid.Row="0" Grid.Column="0" Content="Розмір блоку перестановки:"
Style="{StaticResource SecondaryLabel}"/>
            <ComboBox Grid.Row="0" Grid.Column="1" Style="{StaticResource MainComboBox}"
Margin="0,0,0,5" SelectedItem="{Binding BlockSize}">
                <ComboBoxItem Content="256" IsSelected="True"/>
                <ComboBoxItem Content="65536"/>
            </ComboBox>
```

```
            <Label Grid.Row="1" Grid.Column="0" Content="Розрядність генератора ПСП:"
Style="{StaticResource SecondaryLabel}"/>
            <ComboBox Grid.Row="1" Grid.Column="1" Style="{StaticResource MainComboBox}"
SelectedItem="{Binding GeneratorDegree}">
              <ComboBoxItem Content="64" IsSelected="True"/>
              <ComboBoxItem Content="256" IsEnabled="False"/>
              <ComboBoxItem Content="1024" IsEnabled="False"/>
              <ComboBoxItem Content="2048" IsEnabled="False"/>
            </ComboBox>
          </Grid>
          <Label Grid.Row="6" Content="Ключі:" Style="{StaticResource MainLabel}"/>
          <Button Grid.Row="7" Content="Завантажити файл-ключ" Style="{StaticResource
MainButton}" Command="{Binding OpenKeys}"/>
          <Grid Grid.Row="8">
            <Grid.RowDefinitions>
              <RowDefinition Height="Auto"/>
              <RowDefinition Height="Auto"/>
              <RowDefinition Height="Auto"/>
              <RowDefinition Height="Auto"/>
              <RowDefinition Height="Auto"/>
              <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <WrapPanel Grid.Row="0">
              <Ellipse Width="12" Height="12" VerticalAlignment="Center" Fill="{Binding
IrreduciblePolynomialStatus}"/>
              <Label Content="Незвідний поліном:" Style="{StaticResource SecondaryLabel}"/>
            </WrapPanel>
            <Grid Grid.Row="1">
              <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="*"/>
              </Grid.ColumnDefinitions>
              <Button Grid.Column="0" Margin="0" Grid.ColumnSpan="{Binding
EnterKeyButtonSpan}" Content="Ввести вручну" Style="{StaticResource GenerateButton}"
Command="{Binding EnterIrreduciblePolynomial}"/>
              <Button Grid.Column="1" Content="Згенерувати" Style="{StaticResource
GenerateButton}" Command="{Binding GenerateIrreduciblePolynomial}" Visibility="{Binding
GenerateKeyButtonVisibility}"/>
            </Grid>
            <WrapPanel Grid.Row="2" Margin="0,5,0,0">
              <Ellipse Width="12" Height="12" VerticalAlignment="Center" Fill="{Binding
PrimitiveOEStatus}"/>
              <Label Content="Примітивний утворюючий елемент:" Style="{StaticResource
SecondaryLabel}"/>
            </WrapPanel>
            <Grid Grid.Row="3">
```

```
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <Button Grid.Column="0" Margin="0" Grid.ColumnSpan="{Binding
EnterKeyButtonSpan}" Content="Ввести вручну" Style="{StaticResource GenerateButton}"
Command="{Binding EnterPrimitiveOE}"/>
                <Button Grid.Column="1" Content="Згенерувати" Style="{StaticResource
GenerateButton}" Command="{Binding GeneratePrimitiveOE}" Visibility="{Binding
GenerateKeyButtonVisibility}"/>
            </Grid>
            <WrapPanel Grid.Row="4" Margin="0,5,0,0">
                <Ellipse Width="12" Height="12" VerticalAlignment="Center" Fill="{Binding
InitVectorStatus}"/>
                <Label Content="Вектор ініціалізації:" Style="{StaticResource SecondaryLabel}"/>
            </WrapPanel>
            <Grid Grid.Row="5">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <Button Grid.Column="0" Margin="0" Grid.ColumnSpan="{Binding
EnterKeyButtonSpan}" Content="Ввести вручну" Style="{StaticResource GenerateButton}"
Command="{Binding EnterInitVector}"/>
                <Button Grid.Column="1" Content="Згенерувати" Style="{StaticResource
GenerateButton}" Command="{Binding GenerateInitVector}" Visibility="{Binding
GenerateKeyButtonVisibility}"/>
            </Grid>
        </Grid>
        <Button Grid.Row="9" Content="Приховати повідомлення" Style="{StaticResource
MainButton}" Command="{Binding HideMessage}" Visibility="{Binding
HideMessageButtonVisibility}"/>
        <Button Grid.Row="9" Content="Зчитати повідомлення" Style="{StaticResource
MainButton}" Command="{Binding ExtractMessage}" Visibility="{Binding
ExtractMessageButtonVisibility}"/>
        <ProgressBar Grid.Row="10" Style="{StaticResource MainProgressBar}"
Maximum="{Binding MaxProgressValue}" Value="{Binding ProgressValue}"/>
    </Grid>
    <Grid Grid.Column="1" Margin="5,0,0,0">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Label Grid.Row="0" Margin="0,5,0,0" Content="Повідомлення, що приховується:"
Style="{StaticResource MainLabel}"/>
```

```
            <Button Grid.Row="1" Content="Записати повідомлення у файл" Style="{StaticResource
MainButton}" Command="{Binding SaveMessage}" Visibility="{Binding
MessageSaveButtonVisibility}"/>
            <Button Grid.Row="1" Content="Відкрити повідомлення з файлу" Style="{StaticResource
MainButton}" Command="{Binding OpenMessage}" Visibility="{Binding
MessageOpenButtonVisibility}"/>
            <TextBox Grid.Row="2" Style="{StaticResource MainTextArea}" Text="{Binding
MessageText}" IsEnabled="{Binding MessageTextFieldEnabled}"/>
        </Grid>
    </Grid>
  </Grid>
</Window>
```

Вихідний код генератора ПВП

```
class GaloisAlgorithm
   {
      private MainKey mainKey;
      private ulong[] matrix;
      private ulong gamma;
      private int degree;
      public GaloisAlgorithm(MainKey mainKey)
      {
         this.mainKey = mainKey;
         degree = mainKey.IrreduciblePolynomial.Length - 1;
         ulong max = 0xFFFFFFFFFFFFFFFF;
         BigInteger polynomial = NumericHelper.BinToDec(mainKey.IrreduciblePolynomial);
         matrix = new ulong[degree];
         matrix[0] = Convert.ToUInt64(mainKey.PrimitiveOE, 2);
         for (int i = 1; i < degree; i++)
         {
            BigInteger a = matrix[i - 1] << 1;
            if (a > max)
               a = a ^ polynomial;
            matrix[i] = (ulong)a;
         }
      }
      public ulong Next()
      {
         ulong vector = gamma;
         if (gamma == 0)
            vector = Convert.ToUInt64(mainKey.InitVector, 2);
         gamma = 0;
         for (int i = 0; i < degree; i++)
         {
            gamma ^= matrix[i] & matrix[i] * (vector & 0x1);
            vector >>= 1;
         }
         return gamma;
      }
      public void Refresh()
      {
         gamma = 0;
      }
   }
```

Вихідний код генератора ключів

```
class KeyGenerator
   {
      private static List<BigInteger> D = new List<BigInteger>()
      {
3,5,15,17,51,85,255,257,771,1285,3855,4369,13107,21845,65535,65537,196611,327685,983055,111412
9,3342387,5570645,16711935,16843009,50529027,84215045,252645135,286331153,858993459,143165
5765,4294967295,4294967297,12884901891,21474836485,64424509455,73014444049,219043332147,3
65072220245,1095216660735,1103806595329,3311419785987,5519032976645,16557098929935,18764
712120593,28470681808895,56294136361779,93823560602965,281479271743489,844437815230467,1
407396358717445,4222189076152335,4785147619639313,14355442858917939,23925738098196565,7
1777214294589695,217020518514230019,361700864190383365,723401728380766673,1085102592571
150095,1229782938247303441,3689348814741910323,6148914691236517205
      };
      private int degree;
      public KeyGenerator(int degree)
      {
         this.degree = degree;
      }
      public string GenerateIrreduciblePolynomial()
      {
         PolynomialGenerator polynomialGenerator = new PolynomialGenerator(degree);
         BigInteger polynomial = polynomialGenerator.Generate();
         while (CheckOE(2, polynomial))
         {
            polynomial = polynomialGenerator.Generate();
         }
         return polynomial.ToBinaryString();
      }
      public string GeneratePrimitiveOE(string polynomialString)
      {
         BigInteger OE;
         BigInteger polynomial = NumericHelper.BinToDec(polynomialString);
         using (RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider())
         {
            byte[] numBytes = new byte[degree / 8];
            rng.GetBytes(numBytes);
            OE = new BigInteger(numBytes);
         }
         while(!CheckOE(OE, polynomial))
         {
            OE += 1;
         }
```

```
            return OE.ToBinaryString();
        }
        public string GenerateInitVector()
        {
            string initVector = "";
            using (RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider())
            {
                byte[] numBytes = new byte[degree / 8];
                rng.GetBytes(numBytes);
                initVector = new BigInteger(numBytes).ToBinaryString();
            }
            return initVector;
        }
        private bool CheckOE(BigInteger OE, BigInteger polynomial)
        {
            BigInteger maximum = BigInteger.Pow(2, degree - 1);

            foreach (var divider in D)
            {
                if (ModularArithmetics.ModPower(OE, divider, polynomial, maximum) == 1) return false;
            }
            return true;
        }
    }
```

Вихідний код шифратора повідомлень

```
class MessageEncryptor
   {
      GaloisAlgorithm galoisAlgorithm;
      public MessageEncryptor(GaloisAlgorithm galoisAlgorithm)
      {
         this.galoisAlgorithm = galoisAlgorithm;
      }
      public string Encrypt(string message)
      {
         StringBuilder cipher = new StringBuilder();
         galoisAlgorithm.Refresh();
         foreach (var item in message)
         {
            byte[] gamma = BitConverter.GetBytes(galoisAlgorithm.Next());
            byte addition = 0;
            foreach (var part in gamma)
            {
               addition ^= part;
            }
            cipher.Append((char)((byte)item ^ (byte)addition));
         }
         return cipher.ToString();
      }
      public string Decrypt(string cipher)
      {
         StringBuilder message = new StringBuilder();
         galoisAlgorithm.Refresh();
         foreach (var item in cipher)
         {
            byte[] gamma = BitConverter.GetBytes(galoisAlgorithm.Next());
            byte addition = 0;
            foreach (var part in gamma)
            {
               addition ^= part;
            }
            message.Append((char)((byte)item ^ (byte)addition));
         }
         return message.ToString();
      }
   }
```