Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет електроніки та комп’ютерних технологій

Кафедра оптоелектроніки та інформаційних технологій

КУРСОВА РОБОТА
на тему: “ Стеганографічні системи для зображень”

Виконав:
студент групи ФеІм-13
Ліпковський Н.Р.
Науковий керівник:
проф. Половинко І. І.

Львів – 2023

Зміст

Вступ 3
Розділ 1. Комп’ютерна стеганографія 4
1.1 Застосування комп'ютерної стеганографії 4
1.2 Проблема стійкості стеганографічних систем 6
1.3 Структурна схема і математична модель типової стеганосистеми 7
1.4 Класифікація стеганографічних методів приховання даних 11
1.5 Приховування даних у статичних зображеннях 14
1.5.1 Приховування даних у просторовій області 15
зображення 15
1.5.2 Приховування даних у частотній області зображення 17
Розділ 2. Особливості обробки зображень в C# 18
2.1. Загальне поняття про мову програмування C# 18
2.2. Середовище Visual Studio 20
2.2.1 Загальне поняття про середовище. 20
2.2.2. Windows Form Application у Visual Studio. 20
2.3. Переваги та недоліки обробки зображення в C# 24
2.4 Загальне поняття про мову програмування Python 24
Розділ 3. Результати роботи 25
3.1 Приховування повідомлення з допомогою методу заміни найменш значущого біта 25
3.2 Приховування повідомлення з допомогою частотного методу 27
3.3 Приховування повідомлення з допомогою частотного методу вейвлетового перетворення 29
Висновки 31
Список використаної літератури 32
Додаток А 33
Додаток В 34
Додаток С 35

Вступ

Ще з давних часів люди намагалися таємно передати певну інформацію. Це можна спостерігати протягом всієї людської історії. Для цього використовувалося доволі багато різних методів та способів, кожен з яких мав своє поле використання та переваги і недоліки. Серед них є і стеганографія.

Саме цей метод я вирішив розглянути у даній роботі. Він використовується довго і еволюціонував протягом історії з розвитком технологій. Його використовують для передавання таємних повідомлення та затвердження власності, приховують у різних типах даних та інформації. Приховування повідомлення в зображеннях буде основою мого дослідження.

Для прикладу я взяв 3 типи стеганографічних систем для порівняння між собою. Це один метод з просторової області, а саму заміни найменш значущого біта, та два методи з частотної області, швидким перетворенням фур'є та вейвлетовим перетворенням. Порівнявши ці методи я хочу дослідити стеганографічні системи для зображень.

Розділ 1. Комп’ютерна стеганографія

1.1 Застосування комп'ютерної стеганографії

Під час вивчення стеганографії стає очевидним, що це не нове поняття. Протягом історії людство знаходило різні способи захисту інформації від несанкціонованого доступу. Однак із широким використанням мультимедійних технологій і телекомунікаційних засобів за останні два десятиліття стеганографія вийшла на новий етап, відомий як комп’ютерна стеганографія (КС).

Комп’ютерна стеганографія в основному передбачає приховування повідомлень у цифрових даних, які зазвичай походять з аналогових джерел, таких як мова, зображення, аудіо чи відеозаписи. Текстові або виконувані програмні файли також можна використовувати як контейнери, які часто називають «носіями». У простому сценарії молодші біти певних елементів у мультимедійному контейнері, наприклад пікселі, таймінги або коефіцієнти розподілу частот, можна замінити бітами конфіденційного повідомлення без помітного впливу на якість контейнера. За звичайних обставин сторонній спостерігач не зможе виявити втрату якості чи наявність прихованого повідомлення.

Стеганографічна система, або стеганосистема, — це сукупність засобів і методів, що використовуються для встановлення прихованого (невидимого) каналу передачі інформації. По суті, це система, яка дозволяє вбудовувати та витягувати певні інформаційні послідовності, зберігаючи їх окремо від інших даних. Процес вбудовування даних, подібний до стиснення, відрізняється від шифрування, оскільки його мета полягає не в обмеженні доступу до сигналу контейнера, а в тому, щоб гарантувати, що вбудовані дані залишаються невидимими, непошкодженими та підлягають відновленню, коли це необхідно.

При створенні стеганосистеми зазвичай враховуються такі вимоги:

- Система повинна мати прийнятну обчислювальну складність для реалізації, посилаючись на кількість кроків або операцій, необхідних для вбудовування або отримання прихованої інформації всередині структури контейнера.
- Необхідно забезпечити адекватну пропускну здатність, особливо для стеганосистем, які використовуються для прихованої передачі даних.
- Методи приховування мають гарантувати достовірність та цілісність секретної інформації для уповноважених осіб.

- Потенційні зловмисники повинні мати повне розуміння стеганосистеми та деталей її реалізації, причому єдиним невідомим елементом є ключ, який може використовувати лише власник, щоб визначити існування та зміст прихованого повідомлення.
- Якщо зловмисник дізнається про існування прихованого повідомлення, він не зможе його витягти, поки ключ залишається секретним.
- Порушники повинні бути позбавлені будь-яких технічних переваг або засобів для виявлення наявності або розголошення змісту секретних повідомлень.

Стеганографія охоплює різні напрямки, включаючи вбудовування інформації для прихованої передачі, вбудовування цифрових водяних знаків, вбудовування ідентифікаційних номерів і вбудовування заголовків.

В даний час стеганосистеми активно використовуються для вирішення кількох ключових завдань:

- Захист конфіденційної інформації від несанкціонованого доступу.
- Захист прав інтелектуальної власності, зокрема у боротьбі з піратством шляхом нанесення невидимих міток на захищені дані, які розпізнаються спеціальним програмним забезпеченням.
- Протидія системам моніторингу та управління мережевими ресурсами та промисловому шпигунству, тим самим протидіючи спробам контролювати потік інформації через локальні та глобальні сервери керування комп’ютерною мережею.
- Програмний камуфляж, коли небажане використання певного програмного забезпечення незареєстрованими користувачами може бути приховано під стандартним універсальним програмним продуктом або приховане в мультимедійних файлах.

Хоча області застосування стеганографії можуть відрізнятися, багато вимог залишаються незмінними. Однак бувають і відхилення від прийнятих правил. Наприклад, різниця між прихованою передачею даних і вбудовуванням конфіденційного водяного знака (КВЗ) полягає в тому, що в першому випадку злочинець повинен докласти зусиль для встановлення присутності та вилучення прихованого повідомлення, тоді як у другому випадку наявність такого повідомлення (КВЗ) не приховується. Крім того, порушник може законно володіти пристроєм для його виявлення, наприклад частиною BD-програвача.[9]

На малюнку 1.1.1 представлені потенційні сфери застосування стеганографії.

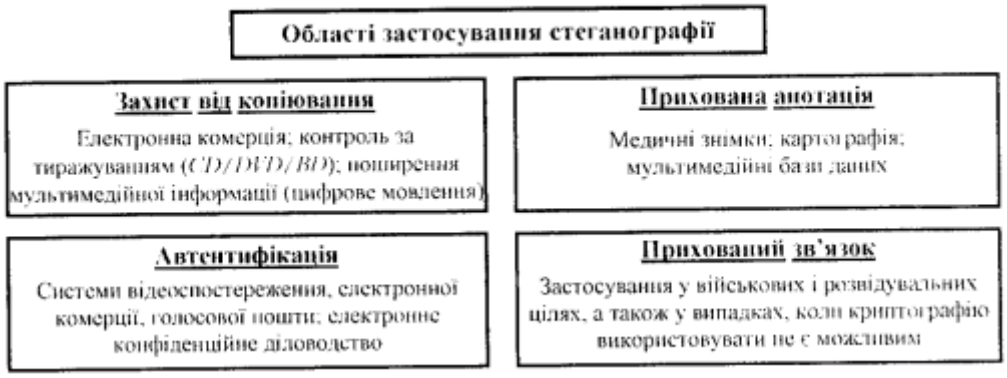


Рис.1.1.1 Потенційні області застосування КС

1.2 Проблема стійкості стеганографічних систем

Кожне із завдань, згаданих на малюнку 1.1.1, вимагає певного балансу між розміром (обсягом) вбудованого повідомлення та його здатністю протистояти зовнішнім впливам. У сучасних методах комп'ютерної стеганографії (КС), які використовуються для приховування повідомлень у різних мультимедійних файлах, малюнок 1.2.1 демонструє взаємозв'язок між надійністю стеганосистеми та обсягом вбудованих даних. На графіку видно, що зі збільшенням обсягу вбудованих даних надійність системи значно знижується.

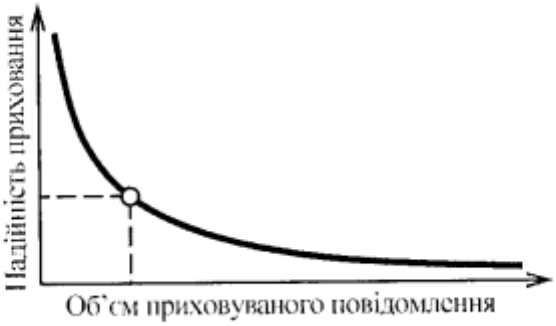


Рис.1.2.1 Взаємозв'язок між стійкістю стеганосистеми та об'ємом прихованого повідомлення при незмінній ємності контейнера

Тому існує компроміс при виборі між обсягом прихованих даних і рівнем опору або приховування всередині контейнера. Обмежуючи погіршення якості контейнера, яке може сприймати людина під час стеганографічної обробки, можна досягти або великого обсягу вбудованих даних, або високої стійкості до модифікації чи аналізу, але не

обох одночасно. Підвищення одного з цих показників неминуче призводить до зниження іншого. Хоча це твердження можна математично продемонструвати для певних методів стеганографії (наприклад, розширення спектру), воно, як правило, вірне і для інших методів комп'ютерної стеганографії. Надлишковий характер інформації контейнера дозволяє підвищити надійність приховування за рахунок зменшення ємності даних.

Обсяг вбудованих даних і рівень модифікації контейнера змінюються залежно від використовуваного методу стеганографії. Крім того, вимоги до стійкості системи до модифікації контейнера відрізнятимуться залежно від конкретних цілей приховування стеганографічних даних. Отже, різні методи стеганографії є оптимальними для різних проблем.

Стеганоаналіз передбачає дослідження перехопленого контейнера для виявлення прихованого повідомлення в ньому. Акт вбудовування інформації в цифровий носій призводить до зміни або реструктуризації властивостей контейнера, що призводить до погіршення його характеристик або набуття нетипових значень. Ці змінені характеристики можуть слугувати «маркерами», що вказують на певні маніпуляції, які використовуються для заповнення контейнера повідомленням, тим самим порушуючи основний принцип стеганографії, який полягає в приховуванні існування секретної інформації.

Стеганоаналіз може приймати різні форми, включаючи виявлення прихованого повідомлення, вилучення прихованих даних і, зрештою, видалення або знищення прихованої інформації. Крім того, зловмисник може вирішити замінити існуючу приховану інформацію дезінформацією.

1.3 Структурна схема і математична модель типової стеганосистеми

У загальному випадку стеганографічна система може бути представлена як система зв'язку. Узагальнена структурна схема типової стеганосистеми зображена на рис. 1.3.1

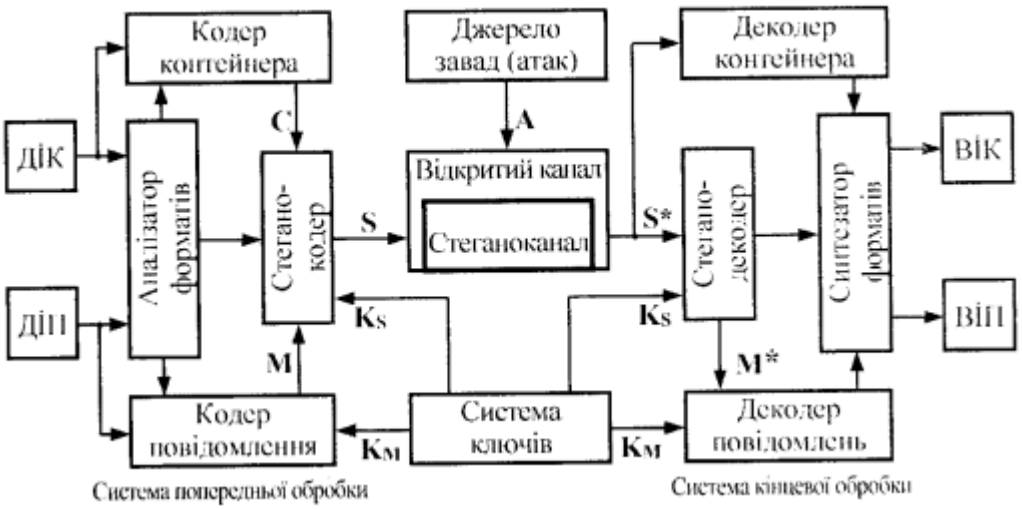


Рис.1.3.1 Структурна схема комп’ютерної стеганосистеми як системи зв’язку

Основними компонентами стеганографії є повідомлення та контейнер. Стеганографічним повідомленням (стеганоповідомленням) me M є секретна інформація, наявність якої необхідно приховати, M = {m, m. mi множина всіх повідомлень. Стеганографічним контейнером (стеганоконтейнером) се C зветься несекретна (відкрита широкому загалу) інформація, яку можна використати для приховання повідомлення шляхом маскування, C= {c, c2, ..., cul множина всіх контейнерів, причому між потужностями множин C та M має бути дотримане наступне співвідношення: q >> n. Тобто приховання інформації, яка здебільшого має досить великий об'єм, висуває суттєві вимоги до контейнера, розмір якого повинен щонайменше в декілька разів перевищувати розмір вбудовуваних даних. Зрозуміло, що для підвищення прихованості зазначене співвідношення повинне бути якомога більшим.[9]

Загалом, мультимедійні дані в будь-якому форматі або комбінації можуть служити як повідомленням, так і контейнером. Порожній контейнер відноситься до контейнера без прихованої інформації, тоді як заповнений контейнер (контейнер результатів) містить приховане повідомлення (стеганограму). Мета полягає в тому, щоб контейнер результатів візуально, на слух або іншим чином не відрізнявся від оригінального контейнера. Файл або об’єкт, отриманий після вбудовування повідомлення в себе, називається стеганофайлом або стеганооб’єктом. Наприклад, вбудовування повідомлення у файл

зображення призводить до стеганозображення, вбудовування в текстовий файл призводить до стеганотексту тощо.

Існує два основних типи контейнерів: потокові та фіксовані. Потоковий контейнер складається з безперервної послідовності змінних бітів без будь-якої структури фіксованої довжини. Повідомлення упаковуються в такі контейнери в режимі реального часу, і невідомо, чи буде контейнер достатньо великим, щоб вмістити все повідомлення. Кілька повідомлень можна вставити в один великий контейнер. Інтервали між заповненими структурними елементами в контейнері можна визначити за допомогою генератора псевдовипадкової послідовності (ПВП) на основі заданого закону розподілу. Синхронізація стає проблемою в потокових контейнерах, оскільки необхідно визначити початок і кінець послідовності. Приклади потокових контейнерів включають мобільні програми, які дозволяють передавати конфіденційні розмови або дані, замасковані під звичайні телефонні розмови. Синхронізацію важко досягти без знання секретного ключа, що перешкоджає виявленню як вмісту, так і факту прихованої передачі.

З іншого боку, фіксовані контейнери мають заздалегідь відомі розміри та характеристики, що дозволяє оптимальніше вбудовувати дані. У решті частини книги головним чином обговорюватимуться фіксовані контейнери.

Контейнери можуть бути обраними, випадковими або накладеними. Вибраний контейнер залежить від вбудованого повідомлення та, у виняткових випадках, є функцією самого повідомлення. Цей тип контейнерів більш характерний для стеганографії. Нав’язаний контейнер надає особа, яка підозрює стеганографічну комунікацію та має намір порушити її чи використати для власних цілей. На практиці найчастіше зустрічаються випадкові (ситуативні) контейнери.

У стеганосистемі, зображеній на рисунку 1.3.1, сигнал від джерела інформації-контейнера (ДІК) надходить в аналізатор формату (АФ) і кодер контейнера в систему попередньої обробки (СПО). Подібним чином сигнал від джерела інформації про повідомлення (ДІП) подається на АФ і кодер повідомлень СПО. АП визначає структуру інформації на своїх входах і видає керуючі сигнали на відповідні кодери щодо необхідності додаткових операцій кодування. Кодери контейнерів і повідомлень забезпечують сумісність форматів даних ДІЛ і ДІП із засобами подальшої цифрової обробки в стеганокодері, який використовується для вбудовування повідомлення М в контейнер С.

Кодер повідомлень може виконувати додаткові функції, такі як узгодження формату та шифрування важливих повідомлень за допомогою криптографічних кодів. Ключі шифрування Km генеруються заздалегідь, реєструються та розповсюджуються між авторизованими учасниками. Крім того, для процесу вбудовування та вилучення може знадобитися система стеганоключів Ks. Стеганосистеми можна класифікувати на два типи на основі типу використовуваного ключа: секретний ключ і стеганосистеми з відкритим ключем. Стеганосистеми із секретним ключем використовують єдиний ключ, який визначається перед обміном стеганограмами або передається по захищеному каналу. Стеганосистеми з відкритим ключем використовують різні ключі для вбудовування та вилучення повідомлень, і один ключ може вільно передаватись по незахищеному каналу, не допускаючи виведення іншого ключа шляхом обчислень.

Можна використовувати секретний алгоритм, такий як генератор псевдовипадкових бітів (ПВП). Високоякісний ПВП, який використовується в системах захисту інформації, повинен відповідати певним вимогам, включаючи криптографічну стабільність, хороші статистичні властивості, тривалий період генерованої послідовності та ефективну реалізацію апаратного/програмного забезпечення. Статистично (криптографічно) захищений ПВП не повинен демонструвати жодних виявлених шаблонів і повинен генерувати статистично незалежні псевдовипадкові послідовності при ініціалізації випадковими значеннями.[1]

У стеганографії прихована інформація вводиться шляхом модифікації бітів у контейнері, які суттєво не спотворюють його вміст. Ці модифіковані біти утворюють стеганопат. Відсоток замінених бітів стеганоконтейнера відносно його загального об’єму називається інформаційною ємністю стеганопатії.

У стеганокодері стеганографічний алгоритм (CA) і, можливо, система ключів (Ks) визначають елементи контейнера C, в який вбудовано повідомлення M. Модифіковані елементи контейнера утворюють стеганографічний канал (стеганоканал), який створюється на основі відкритого каналу, із заповненим контейнером S як входом. Стратегія вбудовування елементів повідомлення в контейнер визначає методи стеганографії, які можна згрупувати на основі обраного формату файлу як контейнера.[9]

Щоб зберегти цілісність вбудованого повідомлення, можна застосувати такі заходи, як перевірка кодом CRC або методи кодування,

захищені від втручання, особливо коли контейнер зазнає перетворень під час передачі, таких як стиснення або перетворення формату.

У стеганодетекторі визначається наявність прихованих даних у можливо спотвореному контейнері. Спотворення можуть виникати через помилки каналу зв'язку, операції обробки сигналу, навмисні атаки та інші фактори. Хоча розгляд сигнального контейнера як додаткового шуму є поширеним у багатьох моделях стеганосистем, він не враховує невипадкову природу контейнера або вимоги щодо підтримки його якості. Тому більш ефективні стеганосистеми можна побудувати, враховуючи ці фактори.

Стеганодетектори можуть бути розроблені виключно для виявлення наявності вбудованого повідомлення або для вилучення повідомлення з контейнера стеганодекодера, залежно від передбачуваної мети.

1.4 Класифікація стеганографічних методів приховання даних

Більшість методів комп'ютерної стеганографії базуються на двох фундаментальних принципах. По-перше, такі файли, як зображення чи аудіо, можна певною мірою змінювати без втрати їх функціональності. По-друге, почуття середньої людини не в змозі надійно виявити незначні зміни в таких змінених файлах, і немає спеціальних інструментів для ефективного виконання цього завдання.

Комп'ютерна стеганографія в основному працює в медіа-середовищах, які підтримуються сучасними обчислювальними пристроями та інформаційними мережами, що дозволяє цифрове представлення всього інформаційного середовища. Фундаментальний підхід у реалізації методів стеганографії в різних мультимедійних середовищах полягає в ідентифікації незначних фрагментів і заміні їх існуючої інформації прихованою інформацією за допомогою спеціальних алгоритмів або методів. Ці фрагменти, які не є критичними для певних кадрів даних, замінюються компонентами прихованої інформації. Під фреймом у цьому контексті розуміється певна частина інформаційного середовища, що виділяється своїми характерними ознаками або семантичними характеристиками. Наприклад, окреме зображення, звуковий файл або веб-сторінку можна розглядати як фрейм.

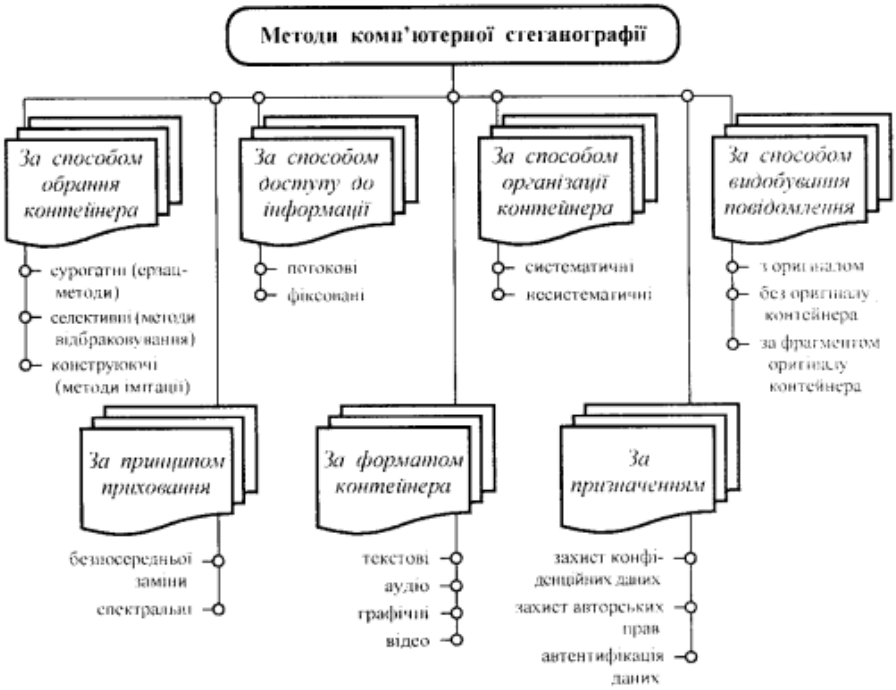
Існуючі методи комп’ютерної стеганографії можна класифікувати на три категорії на основі вибору контейнера: сурогатні методи, селективні методи та конструктивні методи.

Сурогатні методи, також відомі як неповноцінні або ерзац-методи, не надають можливості вибору контейнера. Натомість вибирається перший із найкращих доступних контейнерів, який часто може бути неоптимальним для приховання повідомлення певного формату.

Вибіркові методи припускають, що приховане повідомлення має повторювати конкретні статистичні характеристики шуму контейнера. Генерується кілька альтернативних контейнерів, і найбільш підходящий для конкретного повідомлення вибирається шляхом відхилення або порівняння хеш-функцій. Контейнер, хеш-функція якого найкраще відповідає хеш-функції повідомлення, вибирається для приховування повідомлення, у результаті чого вибраний контейнер стає стеганограмою.[1]

Конструктивні методи включають створення контейнера в самій стеганосистемі. Існують різні варіанти реалізації, наприклад імітація шуму контейнера за допомогою прихованого повідомлення. Процедури використовуються для кодування прихованого повідомлення в шум, зберігаючи вихідну модель шуму. У деяких випадках на основі моделі шуму можна сконструювати ціле повідомлення.

Таким чином, методи комп’ютерної стеганографії використовують модифікацію цифрових файлів і використовують обмеження людського сприйняття для ефективного приховування інформації. Вибір контейнера залежить від використовуваного методу, починаючи від попередньо визначених контейнерів і закінчуючи динамічно створеними.



1.4.1 Класифікація методів комп'ютерної стеганографії

Методи для потокових контейнерів і фіксованих контейнерів відрізняються способом доступу до прихованої інформації.

Контейнери можуть бути організовані систематично або несистематично, подібно до кодів виправлення. Систематичні контейнери мають бітову структуру, яка дозволяє унікально ідентифікувати конкретні місця, де приховано інформаційні біти фактичного повідомлення. Решта фрагменти контейнера служать шумом, який маскує стеганограму, як це видно в класичному методі молодших бітів. На відміну від цього, несистематичні контейнери не мають цього поділу, вимагаючи обробки всього контейнера для вилучення прихованої інформації.

Методи стеганографії можна класифікувати на два основні класи на основі принципу приховування: методи прямої заміни та спектральні методи. Методи прямої заміни використовують надмірність інформаційного середовища контейнера в просторовій (для зображень або тексту) або часовій (для звуку) області. Вони передбачають заміну незначної частини контейнера на фрагменти секретного повідомлення. З іншого боку, спектральні методи використовують спектральне представлення елементів контейнера, щоб приховати дані. Цього можна досягти шляхом маніпулювання коефіцієнтами, отриманими за допомогою дискретних косинусних перетворень, перетворень Фур'є,

перетворень Кархунена-Лоева, перетворень Адамара, перетворень Хаара та інших.

Основним фокусом комп'ютерної стеганографії є використання надмірності порожнього контейнера. Однак важливо враховувати, що приховування інформації може спотворити статистичні властивості та структуру контейнера, потенційно виявивши ознаки фальсифікації. Тому слід враховувати мінімізацію цих показань.

Окрема група стеганографічних методів використовує специфічні властивості форматів файлів. Це включає розширення зарезервованих полів у файлах, які зазвичай заповнюються нулями, спеціальне форматування даних (переміщення слів, речень, абзаців або вибір певних позицій символів), використання невикористаних областей пам'яті на магнітних, оптичних або напівпровідникових носіях, видалення заголовків ідентифікаторів файлів та більше. Ці методи зазвичай мають низьку скритність, пропускну здатність і продуктивність.

Стеганографічні методи можна класифікувати відповідно до їх призначення, розрізняючи приховану передачу чи зберігання даних і приховування даних у цифрових медіа-об'єктах з метою подальшого підтвердження своїх прав на них.

За типом використовуваного контейнера стеганографічні методи поділяються на текстові, графічні, звукові та аудіовізуальні.

Подальше вивчення буде зосереджено на стеганографічних методах приховування даних у статичних зображеннях, звукових сигналах і текстових файлах.

1.5 Приховування даних у статичних зображеннях

Дослідження комп'ютерної стеганографії в основному зосереджені на використанні зображень як контейнерів з кількох причин:

- Практична потреба: існує вимога захистити цифрові фотографії, графіку та відео від несанкціонованого копіювання та розповсюдження.
- Великий обсяг: зображення мають відносно великий розмір даних, що дозволяє вбудовувати значні обсяги прихованої інформації або підвищує надійність вбудовування.
- Фіксований розмір контейнера: зображення мають заздалегідь визначений розмір, що усуває обмеження приховування в реальному часі.

- Текстурні області: більшість реальних зображень містять текстуровані області з шумоподібною структурою, що робить їх придатними для вбудовування секретної інформації.
- Чутливість людського ока: людське око менш чутливе до незначних змін кольорів зображення, яскравості, контрасту, вмісту шуму та спотворень біля контурів, що ускладнює виявлення змін.
- Удосконалення в обробці зображень: галузь обробки цифрових зображень значно розвинулася, забезпечуючи добре розроблені методи обробки зображень.

Однак прогрес у алгоритмах стиснення зображень створив проблеми для забезпечення стабільності прихованої інформації. У міру вдосконалення техніки стиснення стає менше можливостей для вбудовування додаткових даних. Отже, підхід до стеганографії змінився від вбудовування інформації в незначущі біти до вбудовування її в найбільш важливі області зображень. Це гарантує, що втручання в ці зони призведе до повної деградації захищених контейнерів. Тому при розробці стеганографічних алгоритмів стає критично важливим враховувати не лише алгоритми стиснення зображень, але й характеристики зорової системи людини (HVS).

1.5.1 Приховування даних у просторовій області зображення

Алгоритми, розглянуті раніше, включають пряме вбудовування прихованих даних повідомлення в масив пікселів основного зображення. Перевага цих алгоритмів полягає в тому, що вони не потребують обчислювально складних і трудомістких лінійних перетворень для вбудовування.

Щоб представити кольорове зображення C, яке служить набором контейнерів для стеганографічного приховування, ми використовуємо дискретну функцію, яка призначає колірний вектор c(x, y) кожному пікселю (x, y) зображення. Значення кольору зазвичай представлено трикомпонентним вектором у просторі кольорів RGB. У моделі RGB основними кольорами є червоний, зелений і синій, і будь-який інший колір може бути виражений як комбінація цих основних кольорів. Вектор кольору c(x, y) у просторі RGB представляє інтенсивність основних кольорів.

Повідомлення вбудовується шляхом маніпулювання компонентами кольору {R(x, y), G(x, y), B(x, y)} або прямої зміни яскравості 4(x, y), яка варіюється від 1 до Lc.

Загальний принцип даних методів полягає в заміні надлишкової, малозначущої частини зображення бітами секретного повідомлення, а для видобування повідомлення необхідно знати алгоритм і/або ключ, за якими розміщувалася по контейнеру інформація приховуваного повідомлення.[9]

1.5.1.1 Метод заміни найменш значущого біта

Найбільш часто використовуваним методом приховування стеганографічної інформації в просторовій області є заміна найменш значущого біта (НЗБ, LSB).

У цьому методі змінюється молодший значущий біт коду яскравості пікселя, який еквівалентний кроку квантування. Цей біт можна порівняти з шумом або помилкою квантування. Через обмежену чутливість людського ока зміна цього біта часто залишається непоміченою. Оскільки набір шумів зображення можна використовувати для перенесення бітів стеганограми, можна вбудовувати приховані дані в зображення.

Для зображень у відтінках сірого, де кожен піксель кодується одним байтом, обсяг вбудованих даних може досягати 1/8 загального об’єму контейнера. Наприклад, у зображення розміром 512x512 пікселів (з розміром файлу 256 KiB) можна вставити 32 KiB інформації шляхом заміни LSB. Якщо одночасно змінити два молодших значущих біта комбінації коду яскравості (що також непомітно для ока), приховану ємність можна подвоїти.

Популярність методу заміни LSB пояснюється його простотою та можливістю приховати відносно великі обсяги інформації в невеликих файлах. Створений прихований канал зв'язку може займати від 12,5% до 25% пропускної здатності контейнера, тоді як порожній і заповнений контейнери виглядатимуть майже ідентичними. Цей метод зазвичай використовується для растрових зображень у форматах стиснення без втрат, таких як GIF, BMP, PNG, PCX тощо. Однак метод дуже чутливий навіть до незначних спотворень у контейнері. Щоб зменшити цю чутливість, можна застосувати попереднє шифрування прихованого повідомлення.

Під час імпорту зображення контейнера в документ Mathcad його слід підготувати у відповідному редакторі та зберегти як файл у

поточному робочому каталозі документа Mathcad. Рекомендується уникати кириличних символів в адресі та імені файлу, щоб запобігти можливим проблемам із підтримкою кирилиці. Mathcad підтримує такі формати, як BMP, JPEG, GIF, PCX і TGA, причому для носіїв стеганограм краще використовувати формати без втрат.

Для прикладу розглянемо структуру BMP-файлу. Він складається з трьох основних розділів: заголовка файлу, заголовка растру та растрових даних. Заголовок файлу містить загальну інформацію про файл, тоді як растровий заголовок містить такі деталі, як ширина, висота, кількість біт на піксель, глибина кольору, коефіцієнт стиснення тощо. Розділ растрових даних становить головний інтерес, оскільки він містить інформацію про колір кожного пікселя на зображенні. Колір пікселя визначається комбінацією компонентів червоного, зеленого та синього кольорів, причому кожен компонент має власне значення інтенсивності в діапазоні від 0 до 255. Таким чином, кожен колірний канал відповідає 8 бітам (1 байт), в результаті чого загальна глибина кольору 24 біти (3 байти) для зображення.

1.5.2 Приховування даних у частотній області зображення

Як згадувалося раніше, стеганографічні методи, які приховують дані в просторовій області зображення, чутливі до різних спотворень, включаючи стиснення з втратами. Для підвищення стійкості до спотворень більш ефективними є методи, які використовують частотну область для приховування даних.

Існує кілька підходів до розкладання зображення в частотну область. Деякі поширені методи включають дискретне косинусне перетворення (ДКП), дискретне перетворення Фур'є (ДПФ), вейвлет-перетворення, перетворення Кархунена-Лове та інші. Ці перетворення можна застосувати як до окремих частин зображення, так і до всього зображення.[8,9]

Серед ортогональних перетворень дискретне косинусне перетворення (ДКП) і вейвлет-перетворення набули популярності в стеганографії, частково завдяки їх широкому використанню в стисненні зображень. Під час приховування стеганографічних даних до зображення бажано застосувати таке ж перетворення, яке буде застосовано під час потенційного стиснення з втратами. Стеганографічний алгоритм може бути стійким до подальшого стиснення зображення, лише якщо він враховує характеристики конкретного алгоритму стиснення. Наприклад,

ДКП є основним у стисненні JPEG, тоді як вейвлет-перетворення використовується в JPEG2000. Однак стеганографічний алгоритм, заснований на вейвлет-перетворенні, не обов'язково може бути стійким до алгоритму дискретного косинусного стиснення, і навпаки. Вибір відповідного методу стеганографічного перетворення стає ще складнішим під час приховування даних у потоковому відео, оскільки алгоритми стиснення відео включають вектори компенсації руху, які необхідно враховувати для стабільності.

Питання пошуку стабільного стеганографічного перетворення, яке залишатиметься незалежним від використовуваного алгоритму стиснення в майбутньому, залишається відкритим. Різні ортонормовані перетворення, такі як ДФП, ДКП, перетворення Хартлі, перетворення піддіапазону тощо, були досліджені з точки зору теорії інформації.

Було розроблено багато моделей для оцінки прихованої ємності стеганографічного каналу, зосереджуючись на стійкості до атак стиснення з втратами. Одна з таких моделей представлена в роботі. Він припускає вихідне зображення контейнера C, повідомлення M, яке потрібно приховати, і модифіковане зображення S (стеганографічний контейнер), задане S = C + M. Модифіковане зображення S повинно візуально не відрізнятися від оригіналу та може зазнавати стиснення з втратами в стеганографічний канал: S = O(S), де O – оператор стиснення. Завдання одержувача полягає в тому, щоб витягти вбудовані біти даних M з отриманого контейнера S і реконструювати копію вихідного повідомлення M.[9]

Розділ 2. Особливості обробки зображень в C#

2.1. Загальне поняття про мову програмування C#

Практична сторона проекту реалізована мовою програмування C# в середовищі Visual Studio 2017 Community Edition. C# — це об'єктно-орієнтована мова програмування, розроблена для платформи .NET, розроблена Андерсом Гейлсбергом, Скоттом Вілтамутом і Пітером Голдом під керівництвом Microsoft Research.

C# має синтаксис, подібний до C++ і Java. Він має сувору статичну типізацію, підтримку поліморфізму, перевантаження операторів, атрибутів, подій, властивостей, винятків і коментарів формату XML.

Базуючись на таких мовах, як C++, Delphi, Module і Smalltalk, C# усуває певні проблемні моделі в розробці програмного забезпечення, такі як множинне успадкування класів.

Історія C# тісно пов'язана з мовою програмування Java. Sun Microsystems створила Java для вирішення проблем розподілених обчислень під час розквіту Інтернету. Java, заснована на C++, представила такі поняття, як віртуальна машина та байт-код, щоб досягти машинно-незалежного виконання. Проте судовий спір між Microsoft і Sun виник, коли реалізація Microsoft Java виявилася несумісною з незалежним від платформи характером Java. Тоді Microsoft вирішила створити власну мову C#, яка успадкувала концепції віртуальної машини (середовище .NET) і байт-код (MSIL) від Java, одночасно покращуючи безпеку коду та спираючись на досвід Java.

Одним із нововведень у C# є його покращена взаємодія з кодом, написаним на інших мовах, що є корисним для великомасштабних проектів. Коли різні мови працюють на платформі .NET, .NET забезпечує сумісність програм, включаючи типи даних.

C# вважається флагманською мовою Microsoft, яка широко використовує можливості, пропоновані .NET. Хоча інші мови програмування підтримуються, вони визнані такими, що мають недоліки у використанні .NET.

Назву «C#» можна інтерпретувати як еволюцію C++ і як посилання на музичний символ для дієз (♯), поєднаний з літерою C. Стилізований символ «#» був прийнятий Microsoft як назву мови .

C# підкреслює портативність, використовуючи можливості Common Language Runtime (CLR). CLR надає такі функції, як збирання сміття, яке обробляється середовищем виконання для програм, написаних на C# та інших мовах, орієнтованих на .NET.

C# підтримує різні типи даних, включаючи суворий логічний тип. Він сприяє безпеці, дозволяючи лише безпечні неявні перетворення за замовчуванням і явно визначаючи визначені користувачем перетворення. Мова також включає коваріантність і контраваріантність у загальних типах і відокремлює члени перерахування в їх власну область.

Методи в C# визначаються в межах класів, можуть мати знак або без нього, повертати значення або бути недійсними. Мова підтримує віртуальні методи, методи розширення та динамічне введення для зв’язування методів під час виконання. Також підтримуються синхронізовані виклики методів і блокування. Властивості забезпечують зручний спосіб інкапсуляції операцій над атрибутами класу.

Синтаксис C# має структуру, подібну до інших мов у стилі C, використовуючи крапку з комою для закінчення рядків коду, фігурні дужки для групових операторів та ієрархічну структуру з методами в класах і класами в просторах імен. Квадратні дужки використовуються для роботи з масивами, їх оголошення та доступу до значень за індексом.

2.2. Середовище Visual Studio

2.2.1 Загальне поняття про середовище.

Visual Studio – інтегроване середовище розробки програмного забезпечення від фірми Microsoft. Дане середовище дозволяє створювати різноманітні програмні продукти: консольні програми, програми з графічним інтерфейсом, наприклад віконні додатки Windows Forms, а також Web-додатки тощо.

Середовище Visual Studio дозволяє розробляти додатки, використовуючи різні мови програмування: Visual C#, Visual Basic, Visual F#, Visual C++, Python і т.д. Також існує можливість розробляти додатки не тільки під Windows, а і під інші популярні платформи: Android, iOS.

Версія Visual Studio Community є абсолютно безкоштовною для учнів, студентів та розробників програм з відкритим програмним кодом.

2.2.2. Windows Form Application у Visual Studio.

Для створення графічних інтерфейсів за допомогою платформи .NET застосовуються різні технології - Window Forms, WPF, додатки для магазину Windows Store (для ОС Windows 8 / 8.1 / 10). Однак найбільш простою і зручною платформою досі залишається Window Forms або форми.

Створення графічного додатку

Для створення графічного проекту нам буде потрібно середовище розробки Visual Studio. Після установки середовища та всіх її компонентів, запустимо Visual Studio і створимо проект графічного додатку. Для цього в меню виберемо пункт File (Файл) і в підменю виберемо New -> Project (Створити -> Проект). Після цього перед нами відкриється діалогове вікно створення нового проекту. У лівій колонці виберемо Windows Desktop, а в центральній частині серед типів проектів - тип Windows Forms Application і дамо йому якесь ім'я в поле внизу. Наприклад, назвемо його HelloApp. Після цього натискаємо OK.

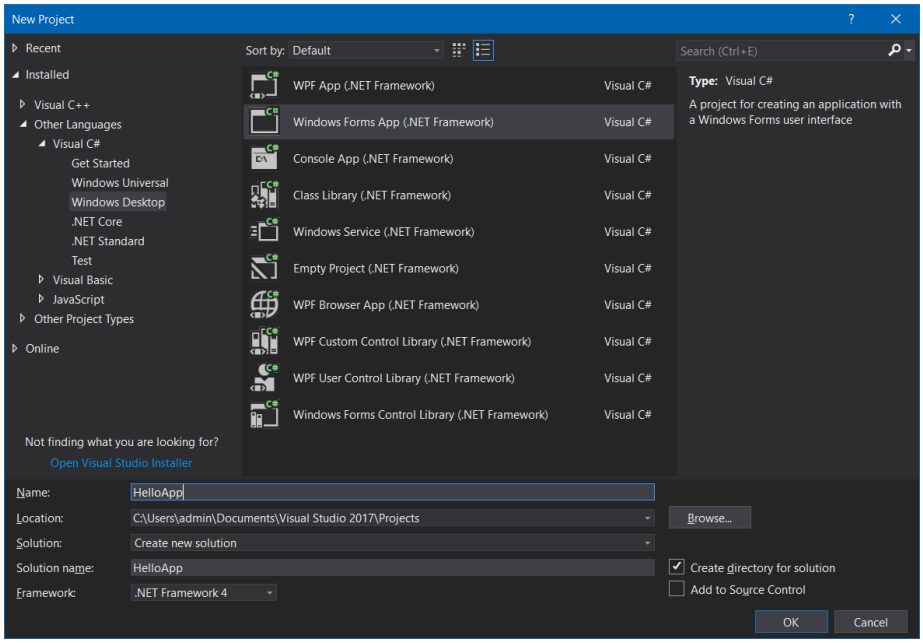


Рис. 2.1 Створення проекту

Після цього Visual Studio відкриє наш проект з створеними за замовчуванням файлами:

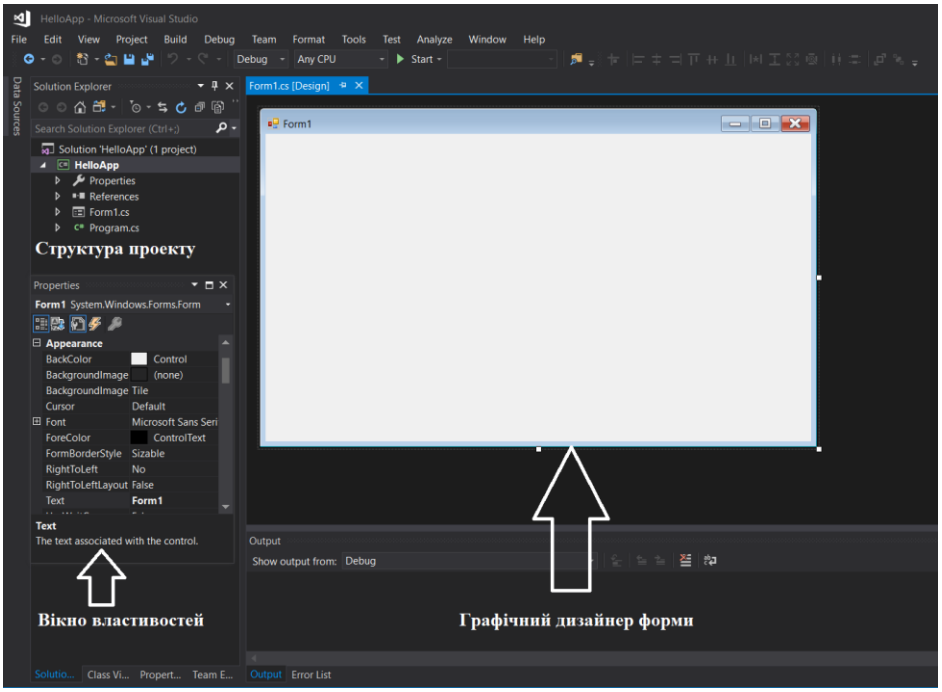


Рис. 2.2 Вигляд Windows Forms Application

Більшу частину простору Visual Studio займає графічний дизайнер, який містить форму майбутнього програми. Поки вона порожня і має тільки заголовок Form1. Справа знаходиться вікно файлів рішення / проекту - Solution Explorer (Оглядач рішень). Там і знаходяться всі пов'язані з нашим додатком файли, в тому числі файли форми Form1.cs.

Внизу справа знаходиться вікно властивостей - Properties. Так як у мене в даний момент обрана форма як елемент управління, то в цьому полі відображаються властивості, пов'язані з формою.

Тепер знайдемо в цьому вікні властивість форми Text і змінимо його значення на будь-яке інше:

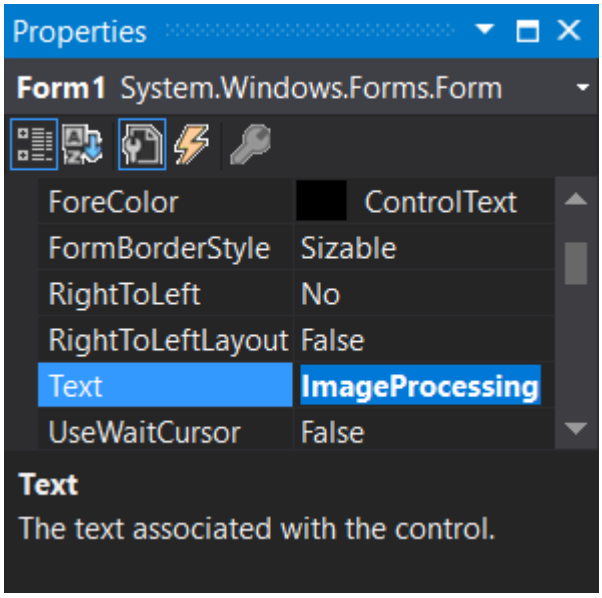


Рис. 2.3 Вікно властивостей Form

Таким чином ми поміняли заголовок форми. Тепер перенесемо на поле який-небудь елемент управління, наприклад, кнопку. Для цього знайдемо в лівій частині Visual Studio вкладку Toolbox (Панель інструментів). Натиснемо на цю вкладку, і у нас відкриється панель з елементами, звідки ми можемо за допомогою миші перенести на форму будь-який елемент:

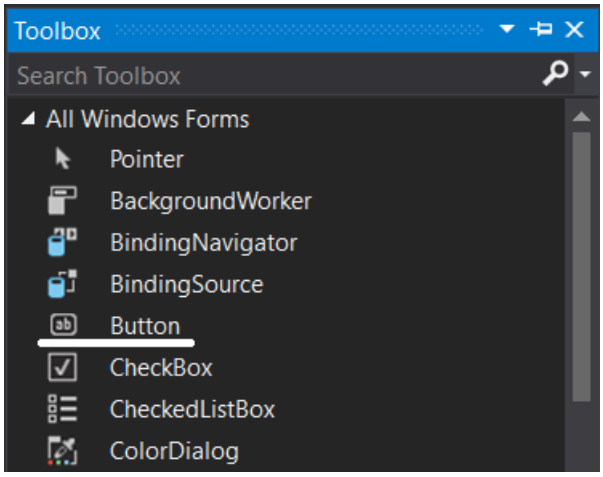


Рис. 2.4 Список елементів

Знайдемо серед елементів кнопку і захопивши її покажчиком миші, перенесемо на форму. Це візуальна частина. Тепер приступимо до самого

програмування. Додамо найпростіший код на мові C #, який би виводив повідомлення після натискання кнопки. Для цього ми повинні перейти в файл коду, який пов'язаний з цією формою. Якщо у нас не відкритий файл коду, ми можемо натиснути на форму правою кнопкою миші і в меню вибрати View Code.

Однак скористаємося іншим способом, щоб не писати багато зайвого коду. Наведемо курсор миші на кнопку і клацнемо по ній подвійним клацанням. Ми автоматично потрапляємо в файл коду Form1.cs.

Додаємо вивід повідомлення після натискання кнопки, змінивши код наступним чином:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Привіт");
}
```

Рис. 2.5 Програма виводу повідомлення

Запуск програми

Щоб запустити додаток в режимі налагодження, натиснемо на клавішу F5 або на зелену стрілку на панелі Visual Studio. Після цього запуститься наша форма з самотньою кнопкою. І якщо ми натиснемо на кнопку на формі, то нам буде відображено повідомлення з привітанням.

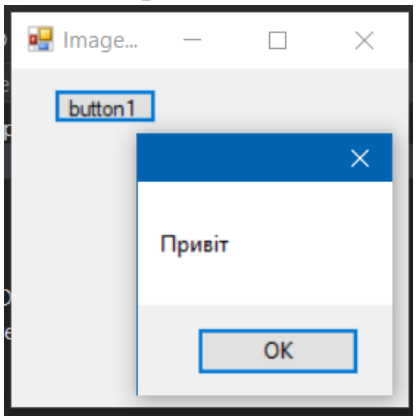


Рис. 2.6 Результат програми

Після запуску програми студія компілює його в файл з розширенням exe. Знайти даний файл можна, зайшовши в папку проекту і далі в каталог bin / Debug або bin / Release

2.3. Переваги та недоліки обробки зображення в C#

Одним з найбільших недоліків роботи із зображенням у мові C# є відсутність вбудованих методів його обробки. Через це доводиться створювати все з нуля і це доволі незручно.

Але, не зважаючи на це у C# відносно простий поріг входження, що дає можливість новачкам швидше освоїтись з роботою. Також перевагою є доволі великий функціонал, за допомогою якого можна реалізувати програми з ефективними способами обробки зображень.

2.4 Загальне поняття про мову програмування Python

Python — це інтерпретована, об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Його високорівневі вбудовані структури даних у поєднанні з динамічним типізацією та динамічним зв’язуванням роблять його дуже привабливим для швидкої розробки додатків, а також для використання в якості мови сценаріїв або склеювання для з’єднання існуючих компонентів. Простий, легкий у засвоєнні синтаксис Python підкреслює читабельність і, отже, знижує витрати на обслуговування програми. Python підтримує модулі та пакунки, що сприяє модульності програм і повторному використанню коду. Інтерпретатор Python і велика стандартна бібліотека доступні у вихідній або двійковій формі безкоштовно для всіх основних платформ і можуть вільно поширюватися.

Спочатку Python був розроблений як навчальна мова, але його простота використання та чистий синтаксис привели до того, що його сприйняли як новачки, так і експерти. Чистота синтаксису Python змусила деяких називати його «виконуваним псевдокодом», і спавді, часто набагато легше прочитати та зрозуміти сценарій Python, ніж читати подібний сценарій, написаний, скажімо, на C.

Також одна з головних переваг даної мови є її бібліотеки, яких є дуже багато і в багатьох напрямках, як часові ряди, нейронні мережі, математика, обробка зображень та багато інших. Для своєї роботи я використовую бібліотеку OpenCV.

OpenCV — це кросплатформна бібліотека, за допомогою якої ми можемо розробляти програми комп’ютерного зору в режимі реального часу. В основному він зосереджений на обробці зображень, захопленні та аналізі відео, включаючи такі функції, як виявлення обличчя та

виявлення об’єктів. Саме через її зручність роботи із зображеннями я обрав її.

Розділ 3. Результати роботи

У практичній частині я розгляну 3 методи для приховування повідомлення на зображенні та порівняю їх між собою. Для дослідження я взяв випадкове зображення з інтернету та випадкове повідомленняю

Рис. 3.1 Оригінальне зображення

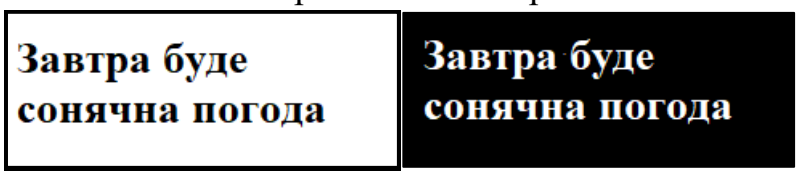


Рис. 3.2 Тексти для аналізу

3.1 Приховування повідомлення з допомогою методу заміни найменш значущого біта

Метод заміни найменш значущого біта є найпоширенішим серед існуючих методів стеганографічного приховання інформації у просторовій області. Його імплементація є відносно проста, оскільки не потребує переведення зображення у частотний простір.

Рис.3.1.1 Зображення із захованим повідомленням на світлому фоні

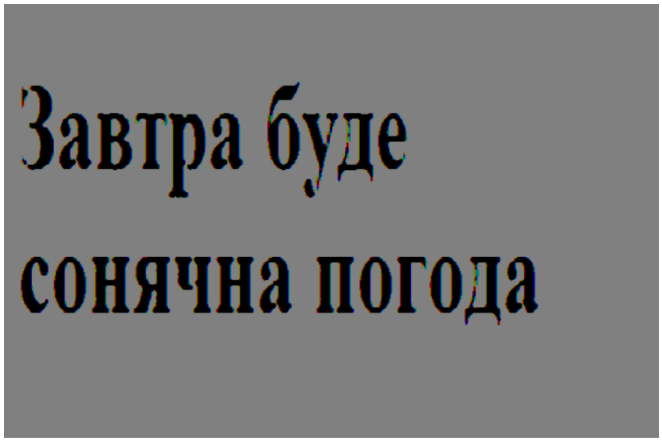


Рис.3.1.2 Вилучене повідомлення

Рис.3.1.3 Зображення із захованим повідомленням на темному фоні

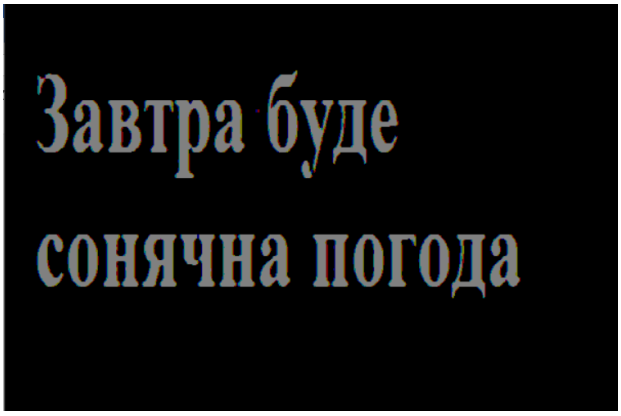


Рис.3.1.4 Вилучене повідомлення

3.2 Приховування повідомлення з допомогою частотного методу

У випадку з частотною формою, на потрібно спочатку перевести зображення і повідомлення в неї, а після внесення змін виконати зворотну трансформацію. Це можна виконати різними методами. У першому прикладі я зроблю це одним з найрозповсюдженіших способів, а саме швидким перетворенням фур’є.

Рис.3.1.4 Зображення із захованим повідомленням на світлому фоні

Рис.3.2.2 Вилучене повідомлення

Рис.3.2.3 Зображення із захованим повідомленням на темному фоні

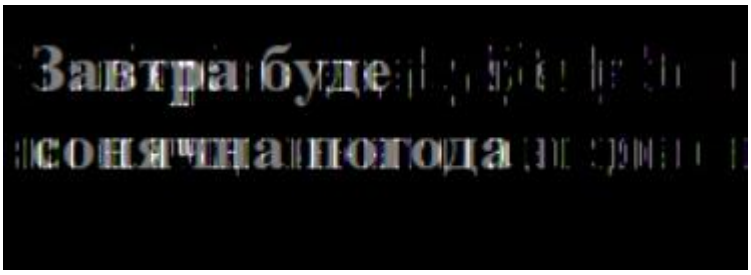


Рис.3.2.4 Вилучене повідомлення

Як можна помітити, повідомлення на темному фоні майже не створює артефакти на зображенні і зчитується краще.

3.3 Приховування повідомлення з допомогою частотного методу вейвлетового перетворення

Ще один приклад частотних методів є вейвлетове перетворення, яке є складніше за попереднє у своєму процесі перетворення даних. Дане перетворення є частіше застосована у динамічних даних через свою прив’язку до часу. Але воно є доволі ефективне і при обробці зображення.

1.Праворуч зображення до приховування, в центрі саме повідомлення і ліворуч зображення вже з нанесеним повідомленням.

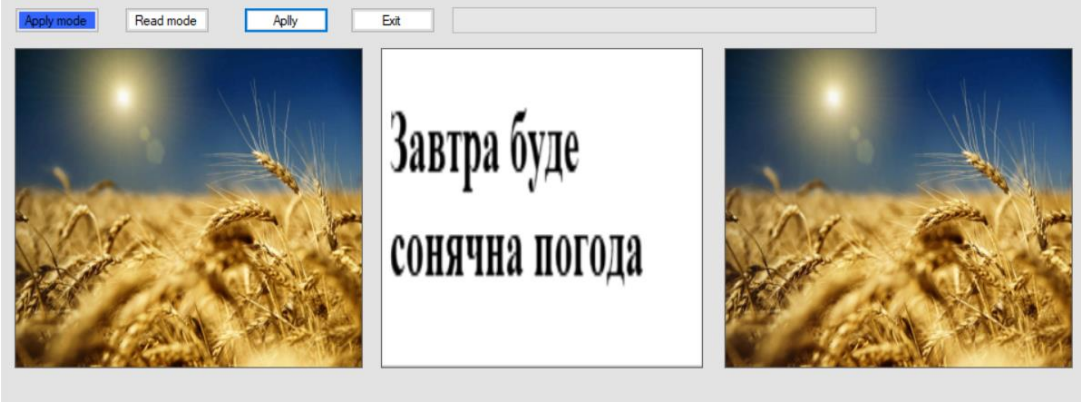


Рис.3.3.1 Приховування повідомлення з білим фоном

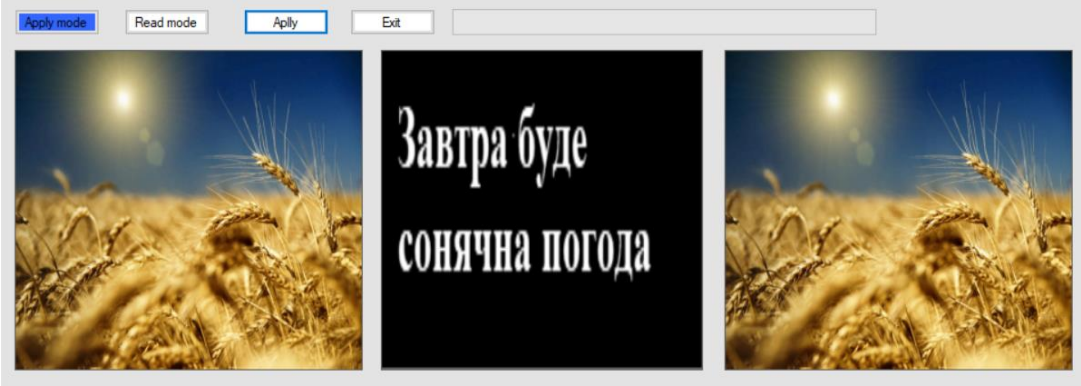


Рис.3.3.1 Приховування повідомлення з чорним фоном

2.Тут ми зчитуємо водяний знак із зображення.

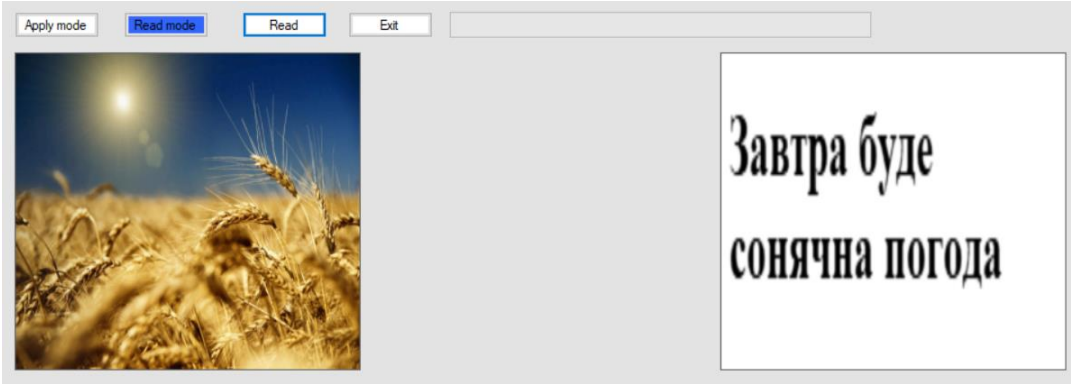


Рис.3.3.3 Вилучення повідомлення з білим фоном

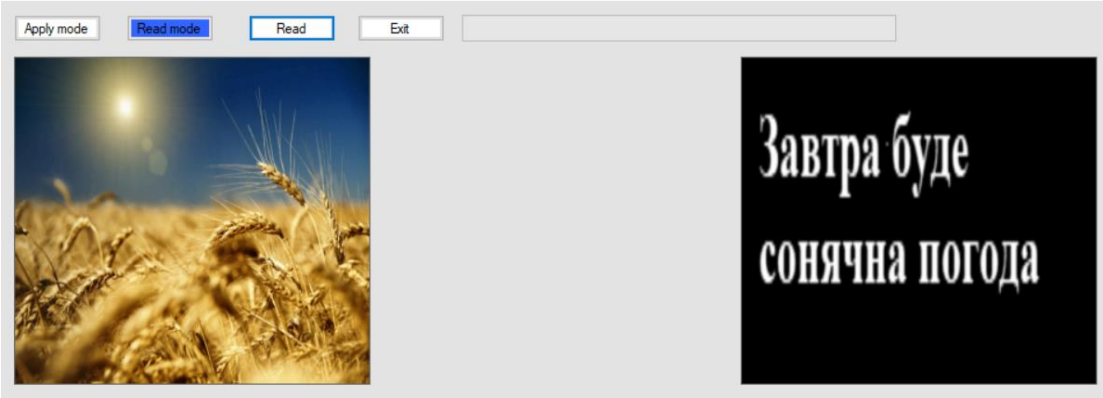


Рис.3.3.4 Вилучення повідомлення з білим фоном

3.На цьому рисунку видно, що у зображенні без знаку його не виявили.

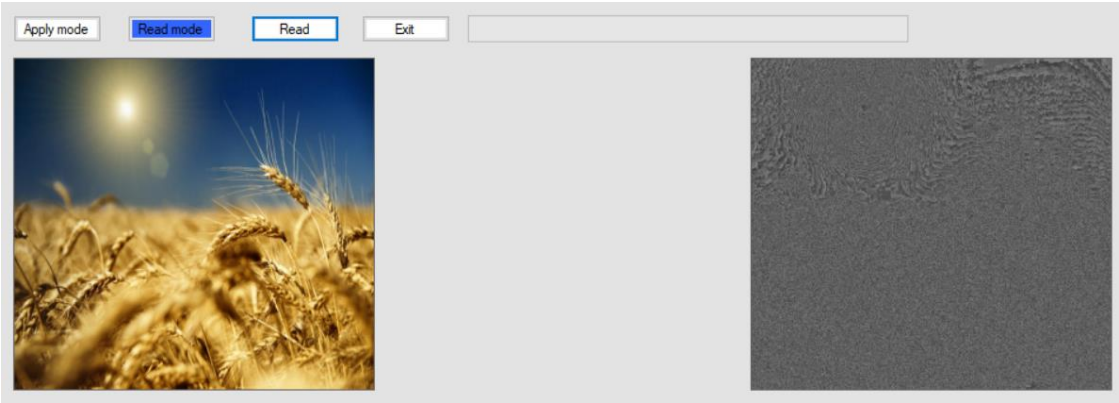


Рис. 3.3.5 Результат зчитування зображення без повідомлення

Висновки

На початку виконання даної роботи було поставлено декілька основних завдань, а саме:

- дослідження поняття стеганографія та стеганографічні системи;
- пошук та аналіз методів комп’ютерної стеганографії;
- розробка програм для реалізації даних методів;
- обробка та аналіз отриманих даних.

Далі проведено літературний аналіз даної теми та вибір методів для практичної реалізації. Я вирішив зупинитися на 3 методах стеганографії. Це просторовий метод заміни найменш значущого біта та частотні методи швидкого перетворення Фур’є та вейвлетового перетворення. Ці методи були обрані для всебічного аналізу питання. Метод заміни НЗБ було вибрано через його простоту, велику розповсюдженість та

пропускну здатність. Недоліками ж його є низька завадостійкість. З частотних методів було обрано Фур'є знову ж таки через його відносну простоту, а вейвлетове перетворення, як більш складніший метод.

Дослідженно інструментарій для роботи. Вивчено особливості обробки зображень за допомогою мови C# у середовищі Visual Studio. Показано, що ця мова програмування володіє рядом переваг. А саме: багатофункціональність, можливість швидкої обробки великого об'єму графічного матеріалу, використання модульної системи. Вивчено особливості мови програмування Python та її переваги, а саме простота та наявність великої кількості бібліотек для оброблення зображення.

Створено та використано програми для приховування та виявлення повідомлення на зображенні різними методами.

З допомогою даних програм приховано та виявлено 2 повідомлення. Одне повідомлення у вигляді чорного тексту на білому фоні, інше протилежне. Зроблено це було для виявлення, чи впливає цей аспект на кінцевий результат. Як виявилося, тільки Фур'є перетворення мало добре помітні артефакти на зображенні після нанесення білого зображення, чого не можна було помітити після чорного. Інші методи виявилися задовільного рівня, хоча було помічено певне затемнення повідомлення на НЗБ.

У подальшому досліджені планується дослідити завадостійкість даних методів та ліміт можливої для приховування інформації без спотворення носія.

Список використаної літератури

1. A.Nakonechny,R.Nakonechny,V. Pavlish"Digital Image Processing" Lviv Polytechnic Publishing House,Lviv,P.366,2010.
2. E. Bribiesca, J.R. Jimenez, V. Medina, R. Valdes, O. Yanez, "A voxel-based measu re of discrete compactness for brain imaging", in Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE Cat. No.03CH37439), 2003.
3. I.Polovynko "Marcing of the Color Imagies by Using Wavelet Transformations" , Electronics and Computer Technologies Department, 2021.

4. I.Polovynko, “Digital signature creation by using descrete cosine and wavelet transformations,” Electronics and Information Technology., 2018.
5. I.Polovynko, T. Kril, “Physics of the optical computers,” Electronics and Information Technologies. , 2014.
6. R.C. Gonzalez, E.R. Woods, “Digital Image Processing Second Edition Prentice Hall Upper Saddle River,” New Jersey 07458, 2017.
7. R.O. Duda, P.E. Hart, D.G. Stork, “Pattern Classification, 2nd ed.,” John Wiley& Sons, NY., 2001.
8. І.Половинко , Н. Ліпковський «Стійкість оптичних маркерів на зображеннях за умови зовнішніх завад». // Електроніка та інформаційні технології. – 2022. – Вип. 19– С. 38-47
9. Коханович Г.Ф., Пузиренко Ф.Ю. Комп’ютерна стеганографія. Теорія і практика-Київ:Київ- Прес,2006.-288с

Додаток А

Код програми для нанесення та зчитування повідомлення частотним методом зі швидким перетворенням фур’є

```
import cv2
import numpy as np
import random
im = cv2.imread("29.png")
cv2.imshow('Original', im)
cv2.waitKey(0)
cv2.destroyAllWindows()
def in_mes(im, mes, res_p, alpha):
    im = cv2.imread(im)
    im_f = np.fft.fft2(im)
    h, w, ch = np.shape(im)
    mess = cv2.imread(mes)
    mes_h, mes_w = mess.shape[0], mess.shape[1]
    x, y = list(range(int(h / 2))), list(range(w))
    random.seed(h + w)
    random.shuffle(x)
    random.shuffle(y)
    tmp = np.zeros(im.shape)
    for i in range(int(h / 2)):
        for j in range(w):
            if x[i] < mes_h and y[j] < mes_w:
                tmp[i][j] = mess[x[i]][y[j]]
                tmp[h - 1 - i][w - 1 - j] = tmp[i][j]
    res_f = im_f + alpha * tmp
    res = np.fft.ifft2(res_f)
    res = np.real(res)
    cv2.imwrite(res_p, res, [int(cv2.IMWRITE_JPEG_QUALITY), 100])
in_mes("29.png","text2.png","res_f.png",5)
res = cv2.imread("res_f.png")
cv2.imshow('Hidden', res)
cv2.waitKey(0)
cv2.destroyAllWindows()
def out_mes(orig, im, res_p, alpha):
    orig = cv2.imread(orig)
```

```
    im = cv2.imread(im)
    ori_f = np.fft.fft2(orig)
    im_f = np.fft.fft2(im)
    h, w = orig.shape[0], orig.shape[1]
    mess = (ori_f - im_f) / alpha
    mess = np.real(mess)
    res = np.zeros(mess.shape)
    random.seed(h + w)
    x = list(range(int(h / 2)))
    y = list(range(w))
    random.shuffle(x)
    random.shuffle(y)
    for i in range(int(h / 2)):
        for j in range(w):
            res[x[i]][y[j]] = mess[i][j]
    cv2.imwrite(res_p, res, [int(cv2.IMWRITE_JPEG_QUALITY), 100])
out_mes("res_f.png","29.png","rez1.png",5)
res = cv2.imread("rez1.png")
cv2.imshow('Message from image', res)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Додаток В

Код програми для нанесення та зчитування повідомлення з допомогою методу заміни найменш значущого біта

```
from PIL import Image
import cv2
def hide_image(source_image_path, secret_image_path, output_image_path):
    source_image = Image.open(source_image_path)
    secret_image = Image.open(secret_image_path)
    # Convert images to RGBA mode
    source_image = source_image.convert("RGBA")
    secret_image = secret_image.convert("RGBA")
    # Resize secret image to fit source image
    secret_image = secret_image.resize(source_image.size, Image.ANTIALIAS)
    # Get pixel data for source and secret images
    source_pixels = source_image.load()
    secret_pixels = secret_image.load()
    # Iterate over each pixel of source image
    for i in range(source_image.width):
        for j in range(source_image.height):
            # Get RGB values for source pixel
            r, g, b, a = source_pixels[i, j]
            # Get RGB values for secret pixel
            sr, sg, sb, sa = secret_pixels[i, j]
            # Replace least significant bit of source pixel's RGB values with secret pixel's RGB values
            modified_r = (r & 0xFE) | ((sr >> 7) & 0x01)
            modified_g = (g & 0xFE) | ((sg >> 7) & 0x01)
            modified_b = (b & 0xFE) | ((sb >> 7) & 0x01)
            # Update the source pixel with modified RGB values
            source_pixels[i, j] = (modified_r, modified_g, modified_b, a)
    # Save the modified source image with the secret image hidden
    source_image.save(output_image_path)
    print("Image hidden successfully!")
def extract_image(source_image_path, output_image_path):
    source_image = Image.open(source_image_path)
    # Convert image to RGBA mode
    source_image = source_image.convert("RGBA")
    # Get pixel data for source image
    source_pixels = source_image.load()
    # Create a new image for the extracted secret image
    extracted_image = Image.new("RGBA", source_image.size)
```

```
    # Get pixel data for extracted image
    extracted_pixels = extracted_image.load()
    # Iterate over each pixel of source image
    for i in range(source_image.width):
        for j in range(source_image.height):
            # Get RGB values for source pixel
            r, g, b, a = source_pixels[i, j]
            # Extract least significant bit of RGB values as secret pixel
            sr = (r & 0x01) << 7
            sg = (g & 0x01) << 7
            sb = (b & 0x01) << 7

            # Set extracted pixel RGB values
            extracted_pixels[i, j] = (sr, sg, sb, a)
    # Save the extracted secret image
    extracted_image.save(output_image_path)
    print("Image extracted successfully!")
# Example
source_image_path = "29.png"
secret_image_path = "text2.png"
output_image_path = "res_p.png"
hide_image(source_image_path, secret_image_path, output_image_path)
# Example
source_image_path = "res_p.png"
output_image_path = "p.png"
extract_image(source_image_path, output_image_path)
res = cv2.imread("res_p.png")
cv2.imshow('Marked', res)
cv2.waitKey(0)
cv2.destroyAllWindows()
res1 = cv2.imread("p.png")
cv2.imshow('Mark from image', res1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Додаток С

Код програми для нанесення та зчитування повідомлення з допомогою вейвлетового перетворення

```
using System;
using System.Drawing;
using System.Windows.Forms;
using AForge.Imaging.Filters;
using System.Collections;
using System.Linq;

namespace dyplom
{
    public partial class Form1 : Form
    {
        Color default_color = Color.FromArgb(64, 64, 64);
        Color selected_color = Color.FromArgb(94, 94, 94);

        public Form1()
        {
            InitializeComponent();
            ApplySwitch.BackColor = selected_color;
        }

        private void Clear()
        {
            ImageBox.Image = null;
            WatermarkBox.Image = null;
```

```
            ResultBox.Image = null;
        }

        private Bitmap ToGreyScale(Bitmap bitmap)
        {
            int grey, i, j;
            Color color;
            Progress.Maximum = bitmap.Width * bitmap.Height;
            Progress.Step = 1;
            for (i = 0; i < bitmap.Width; i++)
            {
                for (j = 0; j < bitmap.Height; j++)
                {
                    color = bitmap.GetPixel(i, j);
                    grey = (int)((color.R + color.G + color.B) / 3);
                    bitmap.SetPixel(i, j, Color.FromArgb(grey, grey, grey));
                    Progress.PerformStep();
                }
            }
            Progress.Value = 0;
            return bitmap;
        }

        private byte getByte(byte[] bits)
        {
            String bitString = "";
            for (int i = 0; i < 8; i++)
                bitString += bits[i];
            byte newpix = Convert.ToByte(bitString, 2);
            int dePix = (int)newpix ^ 2;
            return (byte)dePix;
        }

        private byte[] getBits(byte simplepixel)
        {
            int pixel = 0;
            pixel = (int)simplepixel ^ 2;
            BitArray bits = new BitArray(new byte[] { (byte)pixel });
            bool[] boolarray = new bool[bits.Count];
            bits.CopyTo(boolarray, 0);
            byte[]  bitsArray  =  boolarray.Select(bit  =>  (byte)(bit  ?  1  :
0)).ToArray();
            Array.Reverse(bitsArray);
            return bitsArray;
        }

        private void ReadSwitch_Click_1(object sender, EventArgs e)
        {
            ReadSwitch.BackColor = selected_color;
            ApplySwitch.BackColor = default_color;
            GoButton.Text = "Read";
            WatermarkBox.Visible = false;
            Clear();
        }

        private void ApllySwitch_Click(object sender, EventArgs e)
        {
            ApplySwitch.BackColor = selected_color;
            ReadSwitch.BackColor = default_color;
            GoButton.Text = "Aplly";
            WatermarkBox.Visible = true;
            Clear();

        }

        private void ExitButton_Click_1(object sender, EventArgs e)
```

```
            {
                Application.Exit();
            }

            private void GoButton_Click_1(object sender, EventArgs e)
            {
                if  (GoButton.Text.Equals("Aplly")  &&  !(ImageBox.Image  ==  null  ||
WatermarkBox.Image == null))
                {
                    Bitmap simple = new Bitmap(ImageBox.Image);
                    Bitmap secretGreyScale = new Bitmap(WatermarkBox.Image);
                    if      (secretGreyScale.Height      !=      simple.Height      ||
secretGreyScale.Width != simple.Width)
                    {
                        ResizeBilinear resizeFilter = new ResizeBilinear(simple.Width,
simple.Height);
                        secretGreyScale = resizeFilter.Apply(secretGreyScale);
                    }
                    Color pixelContainerImage = new Color();
                    Color pixelMsgImage = new Color();

                    byte[] MsgBits;
                    byte[] AlphaBits;
                    byte[] RedBits;
                    byte[] GreenBits;
                    byte[] BlueBits;
                    byte newAlpha = 0;
                    byte newRed = 0;
                    byte newGreen = 0;
                    byte newBlue = 0;
                    #region Encryption
                    Progress.Maximum = simple.Height * simple.Width;
                    Progress.Step = 1;
                    for (int i = 0; i < simple.Height; i++)
                    {
                        for (int j = 0; j < simple.Width; j++)
                        {
                            pixelMsgImage = secretGreyScale.GetPixel(j, i);
                            MsgBits = getBits((byte)pixelMsgImage.R);
                            pixelContainerImage = simple.GetPixel(j, i);
                            AlphaBits = getBits((byte)pixelContainerImage.A);
                            RedBits = getBits((byte)pixelContainerImage.R);
                            GreenBits = getBits((byte)pixelContainerImage.G);
                            BlueBits = getBits((byte)pixelContainerImage.B);
                            AlphaBits[6] = MsgBits[0];
                            AlphaBits[7] = MsgBits[1];
                            RedBits[6] = MsgBits[2];
                            RedBits[7] = MsgBits[3];
                            GreenBits[6] = MsgBits[4];
                            GreenBits[7] = MsgBits[5];
                            BlueBits[6] = MsgBits[6];
                            BlueBits[7] = MsgBits[7];
                            newAlpha = getByte(AlphaBits);
                            newRed = getByte(RedBits);
                            newGreen = getByte(GreenBits);
                            newBlue = getByte(BlueBits);
                            pixelContainerImage  =  Color.FromArgb(newAlpha,  newRed,
newGreen, newBlue);
                            simple.SetPixel(j, i, pixelContainerImage);
                            Progress.PerformStep();
                        }
                    }
                    ResultBox.Image = simple;
                    ResultBox.SizeMode = PictureBoxSizeMode.StretchImage;
                    Progress.Value = 0;
                    #endregion
```

```
                }
                else if (GoButton.Text.Equals("Read") && ImageBox.Image != null)
                {
                    Bitmap EncryptedImage = (Bitmap)ImageBox.Image;
                    Bitmap    hiddenImage    =    new    Bitmap(EncryptedImage.Width,
EncryptedImage.Height);
                    Color pixelToDecrypt = new Color();

                    byte[] BitsToDecrypt = new byte[8];
                    byte[] AlphaBits;
                    byte[] RedBits;
                    byte[] GreenBits;
                    byte[] BlueBits;
                    byte newGrey = 0;
                    #region Decryption
                    Progress.Maximum = EncryptedImage.Height * EncryptedImage.Width;
                    Progress.Step = 1;
                    for (int i = 0; i < EncryptedImage.Height; i++)
                    {
                        for (int j = 0; j < EncryptedImage.Width; j++)
                        {
                            pixelToDecrypt = EncryptedImage.GetPixel(j, i);
                            AlphaBits = getBits((byte)pixelToDecrypt.A);
                            RedBits = getBits((byte)pixelToDecrypt.R);
                            GreenBits = getBits((byte)pixelToDecrypt.G);
                            BlueBits = getBits((byte)pixelToDecrypt.B);
                            BitsToDecrypt[0] = AlphaBits[6];
                            BitsToDecrypt[1] = AlphaBits[7];
                            BitsToDecrypt[2] = RedBits[6];
                            BitsToDecrypt[3] = RedBits[7];
                            BitsToDecrypt[4] = GreenBits[6];
                            BitsToDecrypt[5] = GreenBits[7];
                            BitsToDecrypt[6] = BlueBits[6];
                            BitsToDecrypt[7] = BlueBits[7];
                            newGrey = getByte(BitsToDecrypt);
                            pixelToDecrypt    =    Color.FromArgb(newGrey,    newGrey,
newGrey);

                            hiddenImage.SetPixel(j, i, pixelToDecrypt);
                            Progress.PerformStep();
                        }
                    }
                    ResultBox.Image = hiddenImage;
                    ResultBox.SizeMode = PictureBoxSizeMode.StretchImage;
                    Progress.Value = 0;
                    #endregion
                }
            }

            private void ImageBox_Click_1(object sender, EventArgs e)
            {
                OpenFileDialog ofd = new OpenFileDialog();
                ofd.Filter = "Png Image (.png)|*.png| Bitmap Image (.bmp)|*.bmp| Gif
Image (.gif)|*.gif| JPG Image(.jpg) | *.jpg";

                if (ofd.ShowDialog() == DialogResult.OK)
                {
                    ImageBox.ImageLocation = ofd.FileName;
                    ImageBox.SizeMode = PictureBoxSizeMode.StretchImage;
                }
            }

            private void WatermarkBox_Click_1(object sender, EventArgs e)
            {
                OpenFileDialog ofd = new OpenFileDialog();
                ofd.Filter = "Png Image (.png)|*.png| Bitmap Image (.bmp)|*.bmp| Gif
Image (.gif)|*.gif | JPG Image(.jpg) | *.jpg";
```

```
                    if (ofd.ShowDialog() == DialogResult.OK)
                    {
                        Bitmap img = new Bitmap(ofd.FileName);
                        WatermarkBox.Image = ToGreyScale(img);
                        WatermarkBox.SizeMode = PictureBoxSizeMode.StretchImage;
                    }
                }

                private void ResultBox_Click_1(object sender, EventArgs e)
                {
                    if (ResultBox.Image == null) return;
                    SaveFileDialog sfd = new SaveFileDialog();
                    sfd.Filter = "Bitmap Image (.bmp)|*.bmp";
                    if           (sfd.ShowDialog()           ==           DialogResult.OK)
ResultBox.Image.Save(sfd.FileName);
                }
            }
        }
```