Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програмного забезпечення для приховування інформації»

Студента 4 курсу, 7 групи,
спеціальності 121 «Інженерія програмного забезпечення» _____ Демченко Вадим Андрійович
підпис студента

Науковий керівник
кандидат технічних наук,
старший науковий співробітник _____ Звєрєв Володимир Павлович
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент _____ Цензура Микола Олександрович
підпис керівника

КИЇВ – 2020

ЗМІСТ

ВСТУП

Сьогодні життя без доступу до інформації в будь-який час, у будь-якому місці через безлічі типів пристроїв неможливо уявити. Однак її безпека стала важливішою, ніж сама інформація. Насправді, сьогодні інформаційна безпека є однією з найважливіших тем.

Перше що ми робимо на початку дня - перевіряємо телефон з підключенням до Інтернету, шукаємо інформацію, користуємося соціальними мережами, банківською діяльністю, покупками та багатьма іншими функціями в Інтернеті. Ми не досконало захищаємо безпеку персональних комп'ютерів, де зберігаються конфіденційні дані, такі як документи, особисті фотографії, електронні листи, бесіди, важливі номери та багато іншої інформації.

На протязі усього дня нас супроводжують смартфони, щоб завжди залишатися на зв'язку. За допомогою бездротового Інтернету ми купуємо квиток на автобус або платимо за паркування, користуємося кредитними картками, які також містять важливу для нас інформацію.

Все це стало можливим завдяки великим вдосконаленням, які відбулися останнім часом у сфері технологій. Однак останнім часом ми все менше чуємо про нововведення, щодо збереження використаної, оброблюваної інформації в електронному вигляді, але новини про несанкціонований доступ, кібератаки, хакерські порушення, порушення конфіденційності тощо, ми чуємо усе частіше. Це явища вже не на рівні окремих випадків, компаній чи підприємств, а і проблеми, які стають актуальними навіть на державному рівні чи урядової та міжнародної установи. Найпопулярніші загрози: хакерство, віруси, трояни, підробка, відмови в

послугах, шпигунство, зловмисне програмне забезпечення, мобільне зловмисне програмне забезпечення, криптовірологія тощо.

Пошкодження можуть бути жахливими, скориставшись прогалинами у безпеці, зловмисники можуть отримати доступ до комп'ютерної системи без усвідомлення власника, змусивши комп'ютерну систему не працювати належним чином, змінивши джерело / IP-адрес, щоб показати, що вона походить з захищеного джерела, але насправді це може надходити від хакера, який має доступ до всих пакетів бездротової мережі, таким чином, знижуючи цільову мережу і відмовляючи у послугах для законних користувачів.

Для протистояння таким діям співробітники інформаційної безпеки протягом цих років розробили системи для захисту інформації за допомогою таких понять, як: антивірус, антишпигунське програмне забезпечення, програмне забезпечення, Windows та його додатки оновлення системи, брандмауери, фільтрація вмісту і батьківський контроль, коди та алгоритми шифрування, такі як Стеганографія.

Об’єктом дослідження є захист інформації.

Предметом дослідження є розробка програми для приховування інформації з використанням сучасних технологій.

Мета роботи: вирішення проблеми безпеки персональних даних, набуття навичок застосування мов програмування, середовищ розробки для приховування інформації.

Поставлені завдання:

1. Визначити поняття шифрування даних, його особливості, функціонування та переваги.
2. Дослідити підходи для приховування інформації.
3. Скласти технічне завдання на розробку програми для приховування інформації
4. Розробити програму для приховування інформації, застосовуючи актуальні технології. Дотримуватись послідовності етапів розробки

5. Виконати тестування розробленої програми, перевірити коректність її працездатності.

РОЗДІЛ 1
ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРИХОВУВАННЯ ІНФОРМАЦІЇ

1.1. Загальні відомості про програму та методи для приховування інформації

Приховування інформації - це частина інформаційної безпеки. Інформаційна безпека стає важливою частиною сьогоднішнього життя, через необмежене використання Інтернету. Нам завжди доводиться передавати дані у вигляді тексту, звуку, зображення і відео. Інформація в основному передається через мережу, але в мережі дані не захищені. Тож для захисту конфіденційних даних в мережі від третьої сторони, застосовується техніка стеганографії.

Стеганографія – це методика приховування інформації, яка зосереджена на приховуванні існування таємних повідомлень. Зазвичай використовується техніка LSB для захисту інформації у засобах спілкування. У системі пропозицій використовуєтсья алгоритм Huffman Encoding для стиснення даних і приховування в зображенні, аудіо та відео за допомогою техніки LSB.

А також для зображень PSNR та для SNR аудіо, THD, середні та розрахункові середні коефіцієнти частоти.

1.2. Застосування стеганографії та її типи

Слово "стеганографія", як і багато інших важливих термінів, має грецьке походження. Він походить від двох грецьких слів *steganos*, що означає "прикритий", і *graphein*, що означає "писати", і надає можливості прихованого спілкування.

Стеганографія тягне за собою мистецтво писати приховані повідомлення таким чином, що тільки відправник і призначений одержувач знає про наявність повідомлення. Є чотири типи стеганографії:

- текстова стеганографія
- стеганографія зображення
- аудіо стеганографія
- відео стеганографія

Приховування інформації в тексті найбільше важливий і основний метод стеганографії.

Стеганографія зображення - це процес, який приховує повідомлення в образ зображення і генерує стего-зображення. Цей стего-образ потім надсилається до одержувача, не знаючи того, третя особа не зможе помітити, що зображення містить приховане повідомлення. Одержувач може витягти повідомлення з наявністю або без стего-ключа - це залежить від схеми приховування.

Аудіо стеганографія - один з популярних методів приховування даних, що вбудовують секретні дані в аудіосигнали. Таємні дані приховані таким чином, що не усвідомленим особам не відомо про існування вбудованих даних і не мають змін якості звуку в аудіосигналі. Дані для приховування в аудіосигнали мають численні програми, такі як захист захищених авторським правом аудіосигналів, приховане спілкування, приховування даних, які можуть впливати на безпеку уряду та людей. Найменш важливе бітове (LSB) кодування є найпростішим способом вбудовування інформації в цифровий аудіофайл. Замінивши найменш значущий біт кожен пункт відбору з двійковим повідомленням, кодування LSB дозволяє кодувати велику кількість даних. Вбудовування таємних повідомлень в цифровий звук відомі як аудіо стеганографія.

У відеостеганографії – послідовність високої роздільної здатності зображення називають кадри. У конкретному кадрі, ми можемо приховати дані за допомогою алгоритму кодування Хаффмана.

1.2.1. Стеганографія зображення

Існує два різні методи стеганографії зображення:

- Просторові методи
- Методи трансформації

У просторовому методі найбільш розповсюдженим методом є метод заміщення LSB.

Метод найменш значущого біту (LSB) - це загальний простий підхід до вбудовування інформації у файл «обкладинку». У стеганографії використовується метод заміщення LSB. I.e. , оскільки кожне зображення має три компоненти (RGB). Ця піксельна інформація зберігається в закодованому форматі в одному байті. Перші біти, що містять цю інформацію для кожного пікселя, можуть бути змінені для зберігання прихованого тексту. Для цього попередньою умовою є те, що текст, який потрібно зберігати, повинен бути меншим або рівним розміром для зображення, яке використовується для приховування тексту. Метод на основі LSB - метод просторової області. Але він вразливий для створення шуму. У цьому способі MSB (найбільш значущі біти) зображення повідомлення, яке потрібно приховати, зберігаються в LSB (найменш значущі біти) зображення, що використовується як зображення обкладинки. Відомо, що пікселі на зображенні зберігаються у вигляді бітів. У зображенні масштабу сірого кольору інтенсивність кожного пікселя зберігається у 8 біт (1 байт). Аналогічно для кольорового (RGB-червоного, зеленого, синього) зображення на кожен піксель потрібно 24 біта (8 біт на кожен шар). Візуальна система людини (HVS) не може виявити зміни кольору або інтенсивності пікселя при зміні біта LSB. Це психовізуальна надмірність, оскільки це може бути використане як перевагу для зберігання інформації в цих бітах, але при цьому не буде помітно жодної суттєвої різниці в зображенні.

Рис. 1.1. Бітовий вид вихідного і зміненого зображення (пікселя)

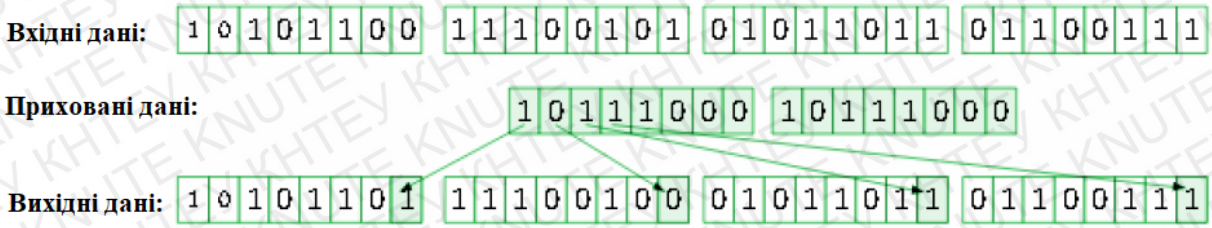


Рис. 1.2. Шифрування даних

Етапи, використовувані в стеганографії LSB:

а) Крок для приховування зображення повідомлення:

1. Додається зображення, яке використовуватиметься як зображення обкладинки. Шум додається для полегшення маскування змін через вбудовування в зображення повідомлення.

2. Додається зображення, яке подається, як зображення повідомлення.

3. Відокремлюються розрядні площини кожного зображення. Як відомо, що площина LSB містить найменшу інформацію, пов'язану з будь-яким зображенням, а площина MSB містить більшу частину форми, кольорової інформації зображення. Зазвичай ідеально буде замінити до 4 найменших бітових площин обкладинкового зображення, причому 4 верхніх бітових площин без виявлення змін у отриманому зображенні. Менша кількість бітових площин із зображення повідомлення може

бути використана, але отримане зображення стане пошкодженим і втратить інформацію.

4. Замінюється щонайменше 4 бітові площини зображення обкладинки на 4 найбільш значущі бітові площини із зображення повідомлення.

5. Отримується результатне стеганографічне зображення шляхом рекомбінації цих біт-площин.

b) Отримання зображення повідомлення:

1. Читається зображення стеганографії.

2. Дістається потрібна кількість бітових площин зображення.

3. Йде рекомбінація нижніх чотирьох бітових площин та надає отримане зображення повідомлення.

Рис. 1.3. Метод найменш значущого біту наглядно

1.2.2. Алгоритм кодування Хаффмана

Алгоритм кодування Хаффмана є оптимальним алгоритмом стиснення, коли для стискання даних використовується лише частота окремих літер. Ідея алгоритму полягає в тому, що якщо у вас є деякі літери, які частіше за інші, мають сенс використовувати менше біт для кодування цих літер, ніж для кодування рідших букв. Окрім того, сучасні схеми кодування повинні мати бітовий шаблон для кожної можливої літери. Для обліку іноземних букв деякі схеми кодування (тобто Unicode) використовують 16 біт на символ. Будь-який заданий файл або набір

файлів, ймовірно, використовуватиме лише невеликиу підмножину цього набору з 2^{16} символів. В алфавіті всього 52 символи (рахуючи малі та великі літери), 10 цифр (цифри 0–9) та деякий невеликий набір розділових знаків, який, ймовірно, буде використаний. Таким чином, реальна кількість символів, використовуваних у типовому простому текстовому файлі, ближче до 75, ніж 256 (для 8-бітного кодування) або 2^{16} (для 16-бітного кодування).

Наприклад, у файлі з такими даними:

XXXXXXYYYYZZ

Частота "X" - 6, частота "Y" - 4, а частота "Z" - 2. Якщо кожен символ представлений з використанням коду фіксованої довжини з двох біт, то кількість бітів потрібних для зберігання в цей файл буде 24, тобто (2 x 6) + (2x 4) + (2x 2) = 24.

Якби вищезазначені дані були стислі, за допомогою стиснення Хаффмана, більш часто зустрічаючі числа будуть представлені меншими бітами, такими як:

X за кодом 0 (1 біт)

Y за кодом 10 (2 біт)

Z за кодом 11 (2 біт)

Тому розмір файлу стає 18, тобто (1x 6) + (2 x 4) + (2 x 2) = 18.

У наведеному вище прикладі символам, що частіше зустрічаються, присвоюються менші коди, що призводить до меншої кількості бітів у остаточному стисненому файлі.

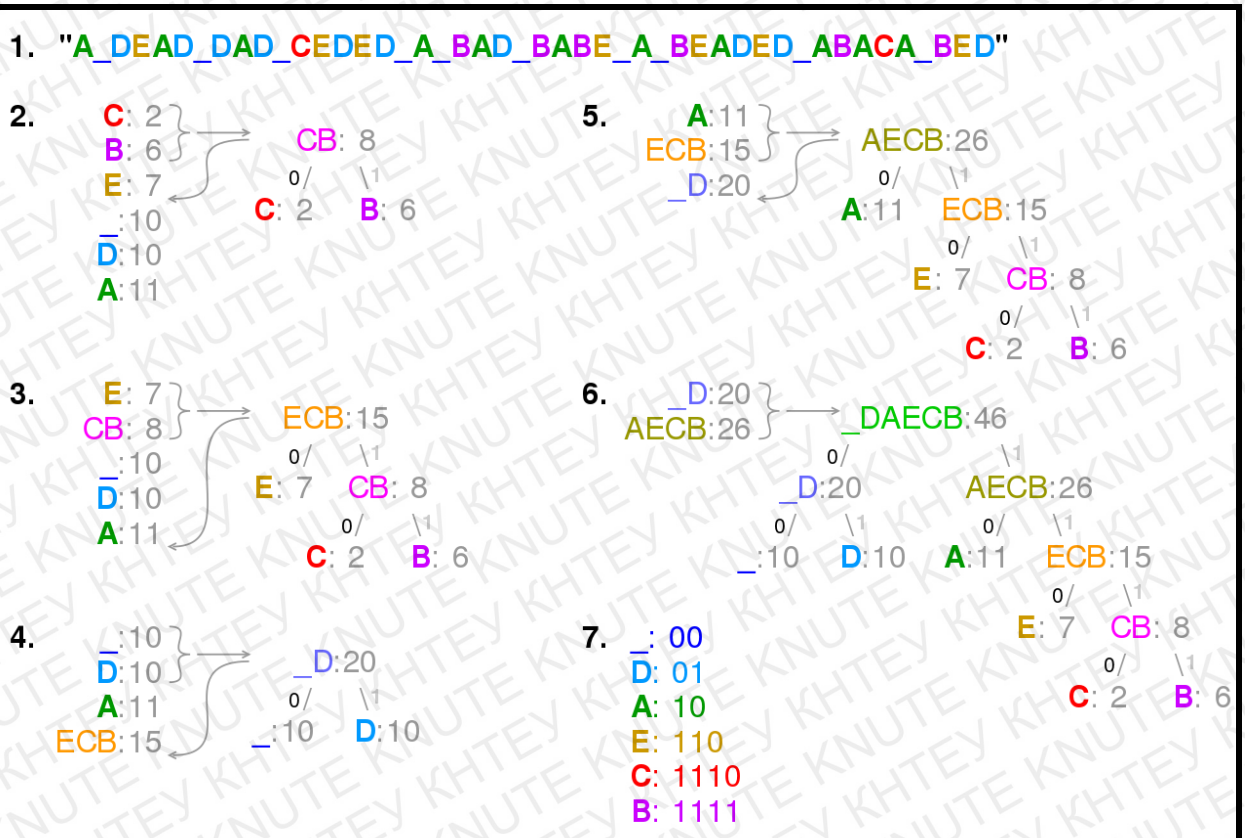


Рис. 1.4. Візуалізація використання кодування Хаффмана для кодування повідомлення

Чому алгоритм Хаффмана вважають жадібним?

Алгоритм Хаффмана є прикладом жадібного алгоритму. Взагалі жадібні алгоритми використовують дрібнозернистий або локальний мінімальний / максимальний вибір, щоб сформувати глобальний мінімум / максимум. На кожному кроці алгоритм робить найближчий вибір, який, здається, призводить до досягнення мети в довгостроковій перспективі. У випадку Хаффмана розуміння його стратегії полягає в тому, що поєднання двох найменших вузлів робить обидва кодування цих символів на трохи довше (через доданий батьківський вузол над ними), але враховуючи, що це найрідкісніші символи, це кращий вибір присвоїти їм довші бітові шаблони, ніж частіші символи. Стратегія Хаффмана насправді призводить до загального оптимального кодування символів. Навіть коли жадна

стратегія може не призвести до загального найкращого результату, вона все одно може бути використана для визначення приблизних моментів, коли оптимальне рішення вимагає вичерпного або дорогого обходу. У ситуації, обмеженій часом або простором, ми, можливо, будемо готові прийняти жадне рішення, яке швидко і легко знайти, як наближення.

У цьому алгоритмі коду змінної довжини призначається для введення різних символів. Довжина коду пов'язана з тим, як часто використовуються символи. Найчастіше символи мають найменші коди та довші коди для найменш частих символів.

Процес кодування Хаффмана:

Схема Хаффмана використовує таблицю частоти зустрічей для кожного символу у введенні. Ця таблиця може бути отримана з самого введення даних або з даних, що є репрезентативними для вхідних даних. Наприклад, частота появи букв у звичайній англійській мові може бути отримана при обробці великої кількості текстових документів і потім використана для кодування всіх текстових документів. Потім нам потрібно призначити бітовий рядок змінної довжини кожному символу, який однозначно представляє цей символ. Це означає, що кодування кожного символу повинно мати унікальний префікс. Якщо символи, що кодуються, розташовані у бінарному дереві:

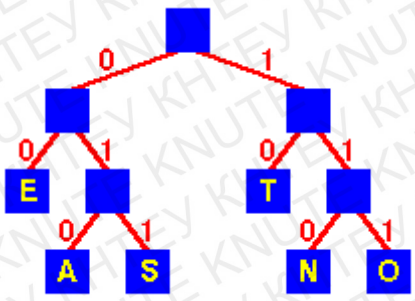


Рис. 1.5. Розташування символів в бінарному дереві

Дерево кодування для кодування *ETASNOAn* для кожного символу знаходить, слідуючи за деревом від маршруту до символу в листі: кодування - це рядок символів на кожній гілці, за якою слідує.

Рядок	Кодування
TEA	10 00 010
SEA	011 00 010
TEN	10 00 110

Рис. 1.6. Кодування символів

За бажанням літери найвищої частоти - E і T - мають двоцифрове кодування, тоді як всі інші мають трицифрове кодування. Кодування проводиться за допомогою таблиці пошуку.

Підхід «розділяй і володарюй» може нас змусити запитати, які символи повинні з’являтися в лівій і правій гілках і намагатися побудувати дерево зверху вниз. Як і у випадку оптимального дерева бінарного пошуку, це призведе до експоненціального алгоритму часу.

Жадібний підхід розміщує наші n символи у n під деревах і починається з об'єднання двох найменш вагових вузлів у дерево, якому присвоюється сума ваг двох листкових вузлів як ваги для його кореневого вузла.

Часова складність алгоритму Хаффмана становить O (nlogn). Використовуючи купу для зберігання ваги кожного дерева, кожна ітерація потребує часу O (logn) для визначення найдешевшої ваги та вставки нової ваги. Існує O (n) ітерацій, по одному для кожного елемента.

Розшифровка Хаффмана кодованих даних:

Процедура розшифровки оманливо проста. Починаючи з першого біту в потоці, потім використовуються послідовні біти потоку, щоб визначити, чи слід йти вліво або вправо в дереві декодування. Коли ми дістаємось до листя дерева, ми розшифрувуємо символ, тож розмістимо його на (нестисненому) вихідному потоці. Наступний біт у вхідному потоці - це перший біт наступного символу.

Рис. 1.7. Візуалізація розшифровки даних

Передача та зберігання кодованих даних Хаффмана:

Якщо ваша система постійно працює з даними, у яких символи мають однакові частоти зустрічань, то і кодери, і декодери можуть використовувати стандартну таблицю кодування / дерево декодування. Однак навіть текстові дані з різних джерел матимуть зовсім інші характеристики. Наприклад, звичайний англійський текст, як правило, має "e" в корені дерева, з короткими кодуваннями для "a" і "t", тоді як програми C зазвичай мають ";" в корені, з короткими кодуваннями для інших пунктуаційні знаки, такі як '(' і ')' (залежно від кількості та довжини коментарів!). Якщо дані мають змінні частоти, то для оптимального кодування нам необхідно створити дерево кодування для кожного набору даних і зберігати або передавати кодування з даними. Додаткова вартість передачі дерева кодування означає, що ми не отримаємо загальної вигоди, якщо потік даних, що кодується, досить довгий - так що заощадження за рахунок стиснення більше, ніж компенсують витрати на передачу дерева кодування також.

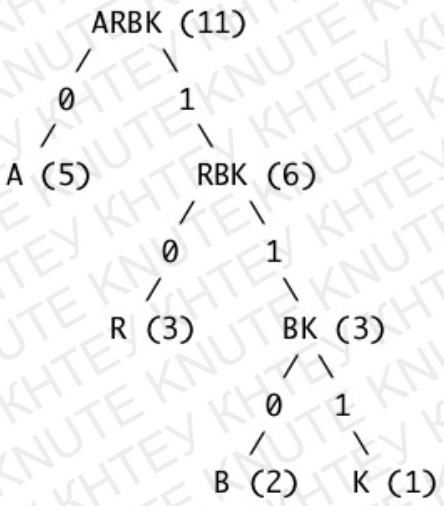
Приклад кодування Хаффмана з деревом:

Дані: ABRRKBAARAA (11 байт)

Підрахунок частоти:

```
A  5
R  3
B  2
K  1
```

Форма дерева Хаффмана, із частотними вагами в дужках:



Кодування кожного символу - це простий шлях до цього символу:

```
A    0
R   10
B  110
K  111
```

Тут закодовані оригінальні дані:

01101010 11111000 1000 (вміщується в 3 байти)

Рис. 1.8. Кодування з деревом Хаффмана

Таким чином, використовуючи техніку кодування Хаффмана, ми можемо досягти стиснення даних без втрат майже на 80%. У наведеному вище прикладі вхідні дані 11 байтів стискаються всього в 3 байти.

1.2.3. Аудіо стеганографія

Сьогодні обмін інформацією обмежений через посягання на інформаційне листування. Безпека інформації може бути здійснена шляхом виконання стратегій стеганографії. Більша частина поточних стеганографічних стратегій використовує комп'ютеризовані зорові та звукові документи як розповсюджені засоби для

приховування потрібної інформації. Існує більше потенційних результатів, щоб приховати величезну кількість інформації у вдосконаленому звуковому документі. Сигнали та комп'ютеризовані звукові документи надають нам відповідні носії для стестенографії, завдяки аномальному стану, надлишку та високій швидкості передачі інформації.

Аудіо стеганографія - це метод приховування фактів внутрішнього звукового сигналу. Програмне забезпечення стеганографії може вставляти повідомлення у звукові дані Wav, Au та, можливо, Mp3. Вбудовування прихованого повідомлення в аналоговий звук, як правило, більш складний спосіб, ніж вбудовування повідомлення в різні дані разом з віртуальними фотографіями. Важливо отримати підпрограми, які також обмежують доступ до цих звукових документів для його безпеки. Зазвичай інформація вставляється в аудіозаписи з кінцевою метою страхування авторських прав або для підтвердження комп'ютерних носіїв інформації. У системі аудіостенографії на базі ПК, приховані повідомлення налаштовуються в автоматичний звук. В аудіостеганографії недолік LSB використовується для приховування даних усередині звуку.

План приховування інформації потребує внесення додаткової інформації, не впливаючи на інтуїтивний характер звукового знаку приймаючої сторони.

Техніка просторового домену:

Техніку просторових доменів також називають технікою часової області та технікою заміщення. У стратегії просторової області прихована інформація спеціально зафіксована в хост-записі, в якому метод стеганографії є основним і простим у застосуванні.

Приховування LSB - це проста та швидка процедура для вставки даних у знак зведення. У LSB-підході LSB бінарних рядів кожної ілюстрації цифрових аудіоданих змінюється бінарним еквівалентом таємних даних. Межа витримує біт кожного зразка розповсюдженого звуку, який може бути меншим для деяких застосувань. У LSB-коді ідеальна швидкість передачі інформації становить 1 кбіт /

с у лінії з 1 кГц. З іншого боку, в декількох виконаннях коду LSB два біти непридатних бітів серії замінюються двома бітами даних. Це збільшить якість інформації, яка може бути закодована, однак одночасно збільшить якість додаткового шуму в аудіофайлі додатково. Більш сучасна методологія полягає у використанні генератора псевдовипадкових кількостей для розширення інформації над звуковим документом довільним чином. Одна з розповсюджених методологій полягає у використанні випадкової тимчасової стратегії, в якій секретний ключ, керований цим тендером, використовується як концепція як частина генератора псевдовипадкових кількостей, що створює нерегулярну послідовність тестових списків. Цей підхід має два недоліка, пов'язаними з використанням таких систем, як кодування LSB. Людське вухо є винятково делікатним і часто може розрізнити навіть найменший шум, що вноситься у звуковий документ, з другого боку - це недолік. У випадку, якщо звуковий запис, встановлений із таємничим повідомленням, що використовує кодування LSB, було перепроведено повторно, внесені дані втрачаються.

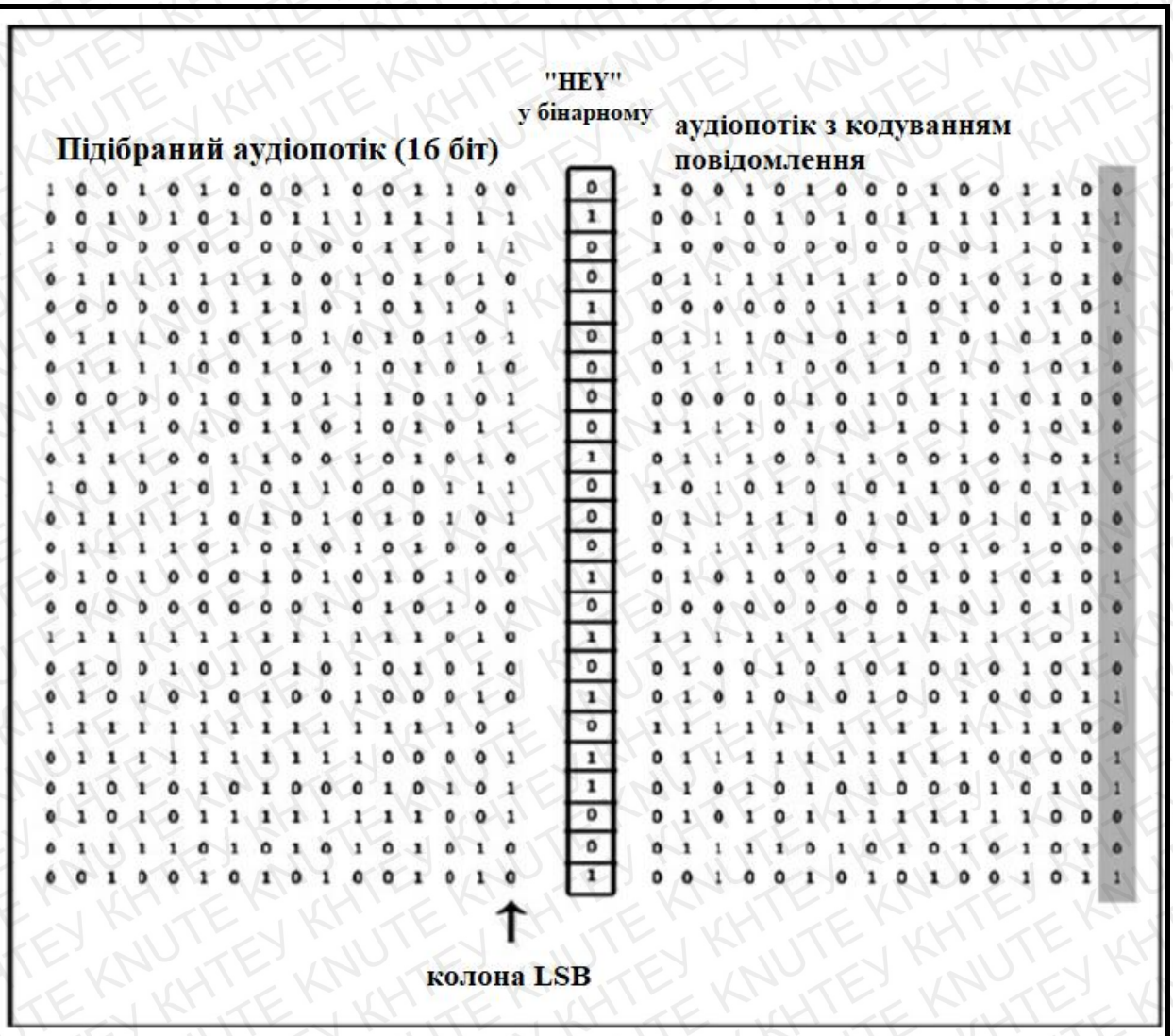


Рис. 1.9. LSB кодування аудіопотоку

Рис. 1.10. Візуалізація додання повідомлення в аудіо

1.3. *Основні етапи створення програми для приховування інформації*

Створення програми для приховування інформації – це комплексний і складний процес. Розробка та написання програмного коду ускладнюється, через те, що програма має не одну функцію кодування і приховування та має різні етапи створення.

Основні етапи створення програми:

- опрацювання технічного завдання
- написання програмного коду для кодування тексту
- написання програмного коду для кодування зображення та аудіофайлів
- перевірка працездатності та правильності написання коду
- розробка інтерфейсу для створеного функціоналу
- перевірка працездатності інтерфейсу та його функціоналу

1. **Опрацювання технічного завдання.** Один з найважливіших етапів, на якому необхідно зібрати саму необхідну інформацію та матеріал, який і буде джерелом. Правильно підібраний матеріал є запорукою написання ефективної програми. Після цього було створене технічне завдання, в якому і був написаний повний опис результату роботи. Саме в технічному завданні описані самі необхідні параметри та функції програми.
2. **Написання програмного коду для кодування тексту.** Самий первинний і основний напрямок стеганографії є приховування тексту, який і був прописаний самим першим в програмному коді. Шифрування виконується за допомогою LSB методу, як і всі подальші методи приховування в програмі. Причиною вибору цього методу є його ефективність та швидкість кодування.
3. **Написання програмного коду для кодування зображення та аудіофайлів.** Наступним кроком в створенні програми було написання програмного коду для стеганографії зображення, оскільки це є найпопулярнішим методом в приховуванні інформації за допомогою стеганографії. Звичайно, що код є набагато складніший, ніж кодування

тексту, оскільки використовуються не просто символи, а й біти зображення, яким присутній RGB код. Шифрування виконується тим самим методом LSB кодування.

Після цього було написано програмний код для кодування аудіофайлів. Самий непростий метод для створення, оскільки не має візуальних символів, які зустрічаються скрізь. На відміну від других методів, цей використовує можливість приховати дані в просторову частину аудіофайлу, за допомогою того ж LSB кодування. Важливо було створити приховування інформації без помітних змін в аудіопотоці, щоб не усвідомлена людина не мала ніяких сумнівів в тому, що файл змінено.

Це удалось виконати з методом стиснення даних та поміщення їх у формі закодованого потоку в аудіофайл, але це можливо не для всих аудіофайлів, які будуть служити файлом «обкладинкою», а лише для тих, що мають 8, 16, 24, 32 біт.

4. **Перевірка працездатності та правильності написання коду.** Після написання основних можливостей програми для приховування інформації була проведена перевірка працездатності коду. Далі після виявлення помилок та їх виправлення, була перевірка написання усього програмного коду, щоб уникнути можливих проблем з роботою програми.
5. **Розробка інтерфейсу для створеного функціоналу.** Важливим атрибутом для користування програмою був інтерфейс, який спрощує процес кодування та розшифровування файлів. Це було виконано після тестування програмного коду, щоб додати до інтерфейсу усі написані та протестовані методи приховування даних. В інтерфейсі знаходяться основні кнопки такі як: для перегляду та вибору файлів; для шифрування та розшифровування інформації. Також важливою функцією інтерфейсу є вікно, яке інформує вас про всі ваші дії та надає важливу статистику.

6. **Перевірка працездатності інтерфейсу та його функціоналу.** Звісно ж після створення інтерфейсу програми була виконана перевірка та тестування можливостей, кнопок та вірності інформативного вікна.

1.4. Технічне завдання

1. Загальні відомості

1.1. Найменування програми: Програма для приховування інформації

1.2. Планові терміни початку та закінчення робіт: 1 квітня 2020 року – 6 травня 2020 року.

1.3. Потенційні користувачі програми

Потенційні користувачі: кожна особа, яка потребує кращої безпеки для себе та своїх даних.

2. Мета та призначення

2.1. Призначення програми:

- Створити можливість покращити безпеку своїх важливих даних;
- Надати неодноманітний функціонал для користувачів програми;

2.2. Мета створення програми: приховання потрібної користувачу інформації в незвичайними методами.

3. Вимоги до програми

3.1. Вимоги до інтерфейсу

3.1.1. Інтерфейс повинен мати мінімум кнопок.

3.1.2. Функціонал програми повинен легко використовуватись, без втрати його можливостей

3.1.3. Мінімалізм інтерфейсу повинен мати інформативність.

1.5. Висновки до розділу 1

Створення програми для приховування інформації є трудомістким процесом, що потребує ознайомлення з великою кількістю нової інформації та матеріалом. Але безмежна кількість інформації дає можливість створити свій оригінальний продукт, функціонал якого не буде схожим на звичайні програми кодування.

Комбінуючи різні методи шифрування можна створити програму, що надає її користувачам корисний функціонал та безліч засобів застосування, що покращать безпеку персональних, важливих даних.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРИХОВУВАННЯ ІНФОРМАЦІЇ

2.1. Структура програми

Перед початком розробки програми потрібно добре спроектувати структуру функціоналу, що помітно спростить написання програмного коду. З дотриманням структури організованість та доцільність програми покращиться.

Приховування інформації відбувається методом внесенням стиснутих даних в обране зображення, приклад зображено на рисунку 2.1.

Сама ж методика поділяється на три різні види, які і використовуються в програмі приховування інформації - це зображено на рисунку 2.2. Кожен з них дозволяє приховати різні типи інформації в незвичайні файли «обкладинки», в яких безпека ваших персональних даних буде набагато вищою, ніж зазвичай.

2.2. Вибір середовища для розробки програми

Для створення програми було обране середовище розробки Visual Studio, через його обширний вибір мов програмування.

Visual Studio - це інтегроване середовище розробки, в якому розробники працюють під час створення програм однією з багатьох мов, включаючи C #, для .NET Framework, який і був обраний як основою для створення програми для приховування інформації. Він використовується для створення консольних та графічних програм для користувальницького інтерфейсу (GUI) разом із програмами Windows Forms або WPF (Windows Presentation Foundation), веб-додатками та веб-службами в обох рідних кодах разом із керованим кодом для всіх платформ, підтримуваних Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight. Він пропонує набір інструментів, які допоможуть вам написати та змінити код для створених програм, а також виявити та виправити помилки у ваших програмах. Visual Studio підтримує різні мови програмування за допомогою мовних сервісів, що дозволяють редактору коду та налагоджувачу підтримувати майже будь-яку мову програмування за умови існування послуги, що залежить від мови. Як і будь-який інший IDE, він включає редактор коду, який підтримує підсвічування синтаксису та завершення коду за допомогою IntelliSense для не тільки змінних, функцій та методів, але й мовних

конструкцій, таких як петлі та запити. C # призначений для створення різноманітних програм, які працюють на .NET Framework. Для роботи в цьому середовищі дуже допомагає такий набір інструментів програмування, які часто використовуються в Visual Studio IDE:

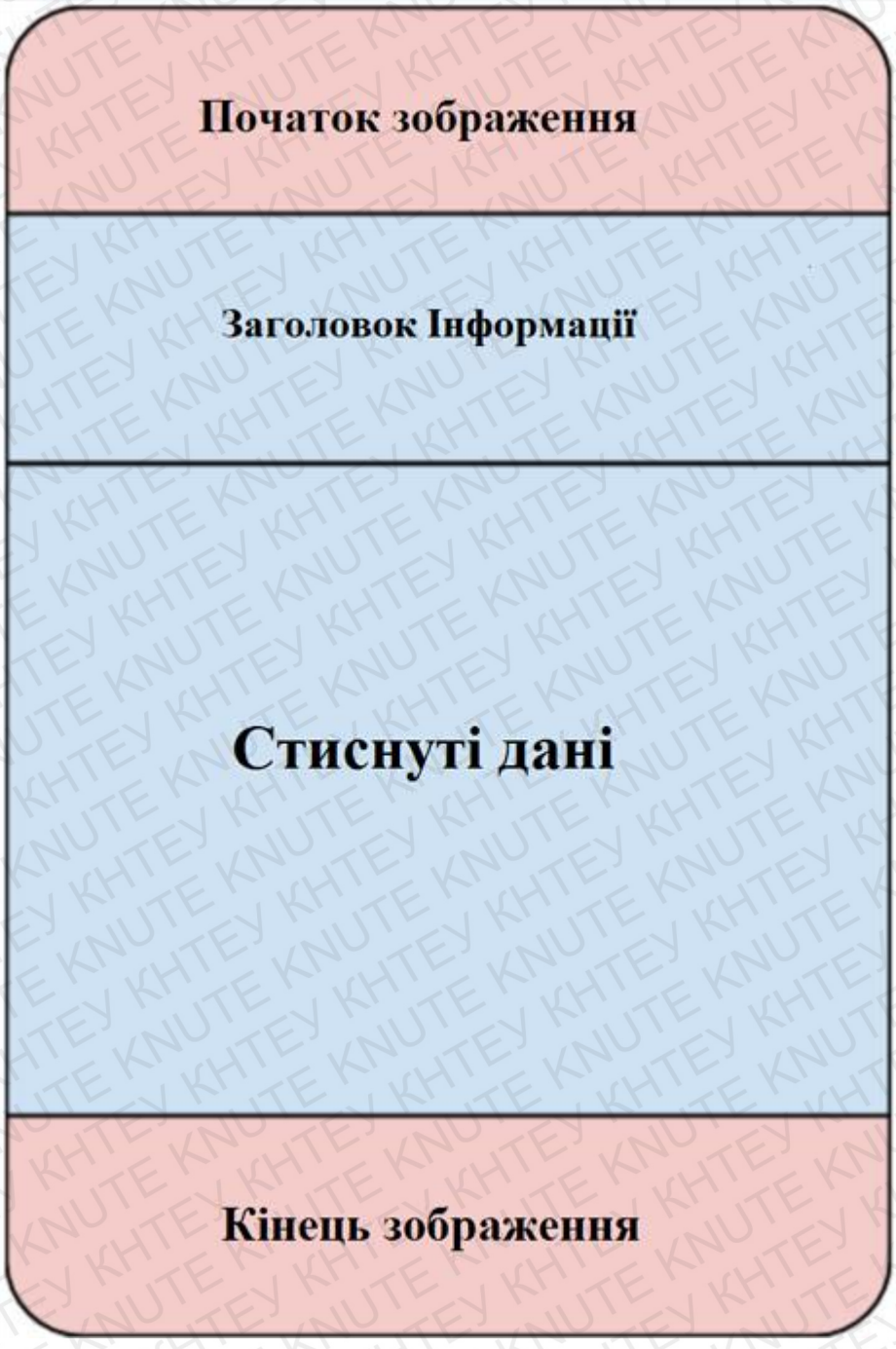


Рис. 2.1. Спрощений приклад

Рис. 2.2. Структура використаної методики

- Рядок меню
- Стандартна панель інструментів
- ToolBox
- Конструктор форм
- Вікно виводу
- Провідник рішень
- Вікно властивостей

У поєднанні з .NET Framework C # дозволяє створювати додатки Windows, веб-сервіси, інструменти бази даних, компоненти, елементи управління тощо. Visual Studio організовує вашу роботу в проектах та рішеннях. Рішення може містити кілька проектів, таких як DLL та виконуваний файл, на який посилається DLL.

2.3. Створення програми

Створення програми поділяється на різні етапи розробки, оскільки функціонал повинен приховувати текст або файли всередині зображень та файлів WAV. Також програма повинна мати вибір між використанням двох алгоритмів: з випадковими показниками та лінійний алгоритм, який є більш простий, але бистродійніший.

2.3.1. *Створення методу шифрування тексту*

Щоб не привернути увагу підслуховувача, прихований вміст повинен бути непомітним. З метою підвищення безпеки даних AES (Advanced Encryption Standard) алгоритм шифрування був реалізований у запропонованому способі, безпосередньо перед вбудовою повідомлення у зображення обкладинки. AES обраний, оскільки він використовує симетричне шифрування та є універсальним з багатьма режимами роботи, - це блок-шифр, але який може працювати як лінійний шифр так і випадковий.

```
public class AESEncrypt
{
    Ссылок: 2
    static AesCryptoServiceProvider CreateAES(string key)
    {
        byte[] bytes = Encoding.ASCII.GetBytes(key);
        AesCryptoServiceProvider a = new AesCryptoServiceProvider();
        Rfc2898DeriveBytes r = new Rfc2898DeriveBytes(key, new byte[] { 1,2,3,4,5,6,7,8});
        a.Key = r.GetBytes(a.BlockSize / 8);
        a.IV = new byte[a.BlockSize / 8];
        a.Padding = PaddingMode.PKCS7;
        a.Mode = CipherMode.CBC;
        return a;
    }

    Ссылок: 4
    public string EncryptString(string text, string password)
    {
        try {
            byte[] textBytes = Encoding.Unicode.GetBytes(text);
            MemoryStream stream = new MemoryStream();
            AesCryptoServiceProvider aes = CreateAES(password);
            CryptoStream crypt = new CryptoStream(stream, aes.CreateEncryptor(), CryptoStreamMode.Write);
            crypt.Write(textBytes, 0, textBytes.Length);
            crypt.FlushFinalBlock();
            return Convert.ToBase64String(stream.ToArray());
        }
        catch
        {

        }
        return null;
    }
```

Рис. 2.3. Шифрування алгоритмом AES

```
    public string DecryptString(string text, string password)
    {
        try {
            byte[] textBytes = Convert.FromBase64String(text);
            MemoryStream stream = new MemoryStream();
            AesCryptoServiceProvider aes = CreateAES(password);
            CryptoStream crypt = new CryptoStream(stream, aes.CreateDecryptor(), CryptoStreamMode.Write);
            crypt.Write(textBytes, 0, textBytes.Length);
            crypt.FlushFinalBlock();
            return Encoding.Unicode.GetString(stream.ToArray());
        }
        catch
        {

        }
        return null;
    }
}
```

Рис. 2.4. Розшифровування алгоритмом AES

AesCryptoServiceProvider виконує симетричне шифрування(Рис. 2.3.) та розшифровування(Рис.2.4.) з використанням інтерфейсів програмування криптографічних прикладних програм (CAPI) алгоритму Розширеного стандарту шифрування (AES).

Клас Rfc2898DeriveBytes реалізує функцію виведення ключа на основі пароля, PBKDF2, використовуючи генератор псевдовипадкових чисел на основі HMACSHA1, який в свою чергу обчислює код аутентифікації повідомлень на основі хеша за допомогою хеш-функції SHA1. Після цього додається клас CryptoStream, який визначає потік, що пов'язує потоки даних з криптографічними перетвореннями. Основою цієї конструкції є CryptoStream. Будь-які криптографічні об'єкти, що реалізують CryptoStream, можуть бути ланцюговими разом з будь-якими об'єктами, що реалізують Stream, тому потоковий вихід з одного об'єкта може бути поданий на вхід іншого об'єкта та проміжний результат (вихід з першого об'єкта) не потрібно зберігати окремо, тож зберігання виконується класом MemoryStream, який створює потік, резервним сховищем якого є пам’ять. Потоки пам’яті, створені за допомогою байтового масиву, забезпечують поток даних, що не змінюється. Використовуючи байтовий масив, ви не можете ні додати, ні зменшити потік, хоча можна змінити існуючий вміст, який буде залежити від

параметрів, переданих у конструктор. Порожні потоки пам'яті можна змінювати за розміром, і їх можна записувати та читати.

2.3.2. Створення методу шифрування зображення

Спосіб приховування інформації в зображення за допомогою методу LSB показує досить вражаючі результати, оскільки використовує простий факт, що будь-яке зображення може бути розбито на окремі бітові площини, кожна з яких складається з різних рівнів інформації. Слід зазначити, що, як обговорювалося раніше, цей метод ефективний лише для растрових зображень, оскільки вони включають методи стиснення без втрат. Також цей процес був розширений, щоб використовувати його для кольорових зображень, де розрізання бітової площини робиться індивідуально для чотирьох верхніх бітових площин для кожної з R, G, B зображення повідомлення, які знову потрібно розмістити в площинах R, G, B зображення обкладинки, і витяг робиться аналогічно.

Після того як ви оберете зображення,яке буде слугувати як обкладинка – програма покаже на скільки багато ви зможете приховати інформації в нього, це було реалізовано за допомогою string FileSizeFormatProvider.GetFileSize, що показує розмір в байтах. Чим більше пікселів на обраному вами зображенні, тим більший файл ви зможете приховати в нього.

Як згадувалося раніше, програма працює, замінюючи біти непотрібних або невикористаних даних у звичайних комп'ютерних файлах бітами наших важливих даних, використовуючи спосіб випадково згенерованого коду для шифрування (Рис. 2.6.). Сам ж спосіб випадкової генерації був створений в LCG генераторі коду(Рис. 2.7.).

Легкий генератор коду (Lightweight Code Generation, LCG), він же клас DynamicMethod. Цей API можна використовувати для генерації методу без створення типу і збірки, що містять його. Для коротких методів - це найефективніший механізм генерації коду.

```
private static void EncodeFileWithColor(ref Bitmap img, byte[] file, string filename)
{
    Bitmap backup = img.Clone() as Bitmap;
    int maxLinear = img.Width * img.Height;
    OutputConsole.Write(string.Format("Розмір файлу: {0}", FileSizeFormatProvider.GetFileSize(file.Length)));
    SeedRNG generator = new SeedRNG((maxLinear + img.Width).GetHashCode(), maxLinear);
    OutputConsole.Write("Код згенеровано");
    int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
    HiddenFile f = new HiddenFile(file, filename);
    f.cipherFile((maxLinear + img.Width).GetHashCode());
    OutputConsole.Write("Шифрування файлу...");
    if (file.Length < maxLinear - extraBytes)
    {
        string fileLength = file.Length.ToString();
        OutputConsole.Write("Обробка зображення...");
        OutputConsole.Write("Написання метаданих...");
        for (int i = 0; i < fileLength.Length; i++)
        {
            Point point = LinearIndexToPoint(generator.Next, img.Width, img.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            char letter = fileLength[i];
            int value = Convert.ToInt32(letter);
            img.SetPixel(point.X, point.Y, EncodePixel(pixel,value));
        }
        //Вставте #, щоб розділити довжину файлу від імені файлу
        Point point1 = LinearIndexToPoint(generator.Next, img.Width, img.Height);
        Color pixel1 = img.GetPixel(point1.X, point1.Y);
        int value1 = Convert.ToInt32('#');
        img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1, value1));

        //Вставте ім'я файлу та закінчіть нульовою графою
        for (int i = 0; i < filename.Length; i++)
        {
            Point point = LinearIndexToPoint(generator.Next, img.Width, img.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            char letter = filename[i];
            int value = Convert.ToInt32(letter);
            img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
        }
        point1 = LinearIndexToPoint(generator.Next, img.Width, img.Height);
        pixel1 = img.GetPixel(point1.X, point1.Y);
        value1 = 0;
        img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1,value1));
        OutputConsole.Write("Запис даних про файл...");
```

Рис. 2.5. Шифрування файлу в зображення

```
public class SeedRNG
{
    ссылка: 1
    private int seed { get; set; }
    private List<int> obtained;
    ссылка: 1
    private int limit { get; set; }
    private Random r;
    private List<int> exp = new List<int>();
    private LCG lcg;
    Ссылок: 6
    public SeedRNG(int seed, int limit, bool b = false)
    {
        this.seed = seed;
        this.limit = limit;
        lcg = new LCG(limit, new Random(seed).Next(0, limit),seed);
        obtained = new List<int>();
        r = new Random(seed);
        if (b)
        {
            exp = Enumerable.Range(0, limit).ToList();
        }
    }

    Ссылок: 11
    public int Next
    {
        get
        {
            return (int)lcg.next();
        }
    }

    Ссылок: 8
    public int NextN
    {
        get
        {
            int x = 0;
            int n = r.Next(0, exp.Count);
            x = exp[n];
            exp.RemoveAt(n);
            return x;
        }
    }
}
```

Рис. 2.6. Випадкова генерація коду для шифрування

```
public class LCG
{
    private BigInteger m;
    private BigInteger a;
    private BigInteger c;
    private BigInteger x0;
    private int index = 0;
    private int seed = 0;

    Ссылок: 8
    public List<BigInteger> x { get; private set; }

    Ссылок: 0
    public LCG(BigInteger m, BigInteger x0)
    {
        this.m = m;
        this.x0 = x0 % m;
        Initialize();
    }

    ссылка: 1
    public LCG(BigInteger m, BigInteger x0, int seed)
    {
        this.m = m;
        this.x0 = x0 % m;
        this.seed = seed;
        Initialize();
    }

    Ссылок: 2
    private void Initialize()...

    Ссылок: 0
    public void setIndex(int i)...

    Ссылок: 0
    public BigInteger get(int i)...

    ссылка: 1
    public BigInteger next()...

    Ссылок: 0
    public List<double> getList()...

    ссылка: 1
    private BigInteger calcC(List<BigInteger> prime, List<BigInteger> factor)...

    ссылка: 1
    private List<BigInteger> factors(List<BigInteger> prime)...


    ссылка: 1
    private List<BigInteger> primes()...
```

Рис. 2.7. LCG генератор

Щоб не обмежуватися один лише одним способом шифрування, був створений лінійний спосіб шифрування(Рис. 2.8.), хоч він і простіший, але більш бистродійний.

```
private static void EncodeFileWithColorLinear(ref Bitmap img, byte[] file, string filename)
{
    Bitmap backup = img.Clone() as Bitmap;
    int maxLinear = img.Width * img.Height;
    int c = 0;
    OutputConsole.Write(string.Format("Розмір файлу: {0}", FileSizeFormatProvider.GetFileSize(file.Length)));
    int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
    HiddenFile f = new HiddenFile(file, filename);
    f.cipherFile((maxLinear + img.Width).GetHashCode());
    OutputConsole.Write("Шифрування файлу...");
    if (file.Length < maxLinear - extraBytes)
    {
        string fileLength = file.Length.ToString();
        OutputConsole.Write("Обробка зображення...");
        OutputConsole.Write("Написання метаданих...");
        for (int i = 0; i < fileLength.Length; i++)
        {
            Point point = LinearIndexToPoint(c, img.Width, img.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            char letter = fileLength[i];
            int value = Convert.ToInt32(letter);
            img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
            c++;
        }
        //Вставте #, щоб розділити довжину файлу від імені файлу
        Point point1 = LinearIndexToPoint(c, img.Width, img.Height);
        Color pixel1 = img.GetPixel(point1.X, point1.Y);
        int value1 = Convert.ToInt32('#');
        img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1, value1));
        c++;
        //Вставте ім'я файлу та закінчіть нульовою графою
        for (int i = 0; i < filename.Length; i++)...
        point1 = LinearIndexToPoint(c, img.Width, img.Height);
        pixel1 = img.GetPixel(point1.X, point1.Y);
        value1 = 0;
        img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1, value1));
        c++;
        //Написати файл
        OutputConsole.Write("Запис даних про файл...");
        for (int i = 0; i < file.Length; i++)
        {
            Point point = LinearIndexToPoint(c, img.Width, img.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            int value = f.file[i];
            img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
            c++;
        }
        OutputConsole.Write("Вбудовування файлу завершено");
    }
```

Рис. 2.8. Лінійний спосіб шифрування

2.3.3. Створення методу шифрування аудіо

Бібліотека libsndfile зробила роботу з даними файлів wav порівняно простою. Libsndfile - це бібліотека C для читання та запису файлів, що містять дискретизований звук (наприклад, MS Windows WAV та формат Apple / SGI AIFF) через один стандартний інтерфейс бібліотеки. Я використовував Audacity для

перетворення MP3-файлу в WAV(Рис. 2.9.). Audacity - це безкоштовний цифровий аудіоредактор з відкритим кодом та програмним забезпеченням для запису, доступний для Windows, macOS, Linux та інших Unix-подібних операційних систем. На додаток до запису аудіо з декількох джерел, Audacity можна використовувати для післяобробки всіх типів аудіо, включаючи подкасти, додаючи ефекти, такі як нормалізація, обрізка та згасання. В даний час він також використовується в Національному курсі ІКТ рівня ІКТ у Великобританії, призначений для створення звуку.

Після цього я розробив і протестував код. Як було обговорено раніше, зміна лише найменшого значущого біта у кожному зразку звуку зробило вбудовані дані дуже важкими для виявлення. MD5 хеші файлів embed-in.dat та embed-out.dat точно збігалися, тому дані, вбудовані у steg.wav, були відновлені ідеально. Програма належним чином виявляє, чи не вбудовуються дані вбудованого файлу під час кодування, і він належним чином визначає, коли під час декодування немає вбудованих даних. Я імпортував оригінальний wav файл та змінений в Audacity. І зміг помітити жодних відмінностей, навіть після збільшення масштабу. Я також побудував звуковий спектр кожного аудіофайлу, і я не міг уявити жодних відмінностей(Рис. 2.10.). Будь-яка зміна спектрів буде в дуже низьких дБ в діапазоні шуму і на різних частотах. Обидва ці тести показали, що вбудовані дані не повинні бути чутими або легко виявлятися. Я перевірив це, прослухавши два аудіофайли і я був приємно здивований результатом.

Також було створено лінійний спосіб шифрування тексту(Рис. 2.11) і даних(Рис. 2.12.), для розширення функціоналу програми.

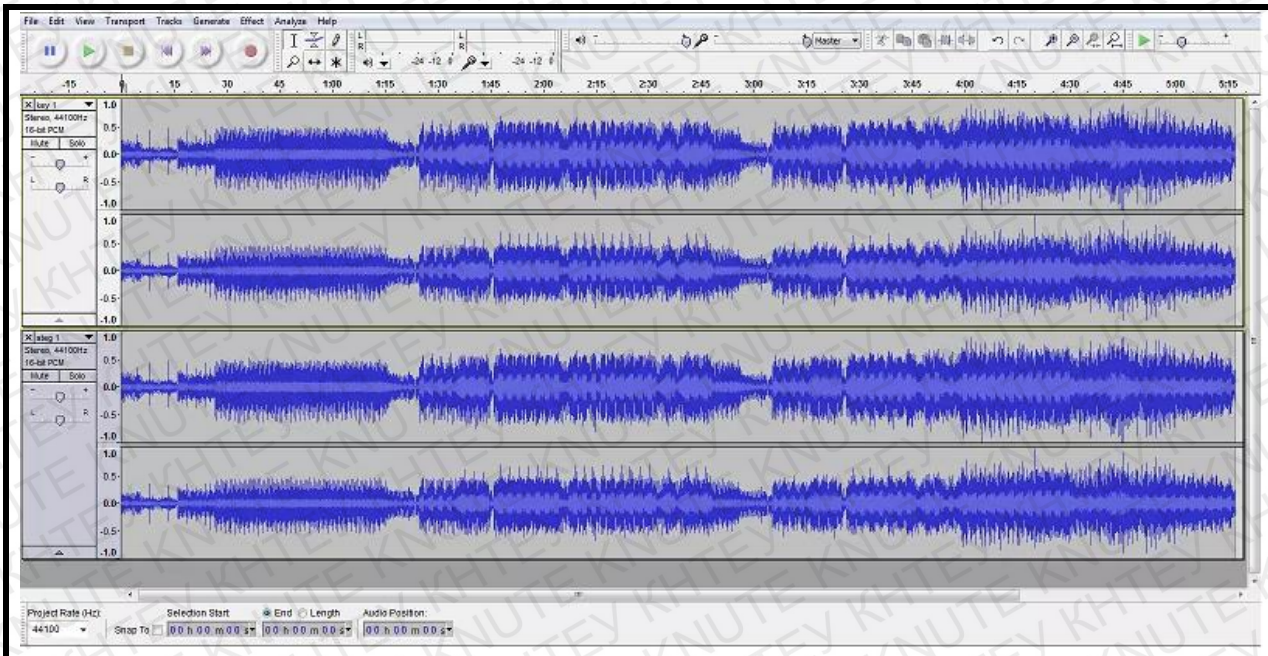


Рис. 2.9. Створення wav файлу для подальших змін

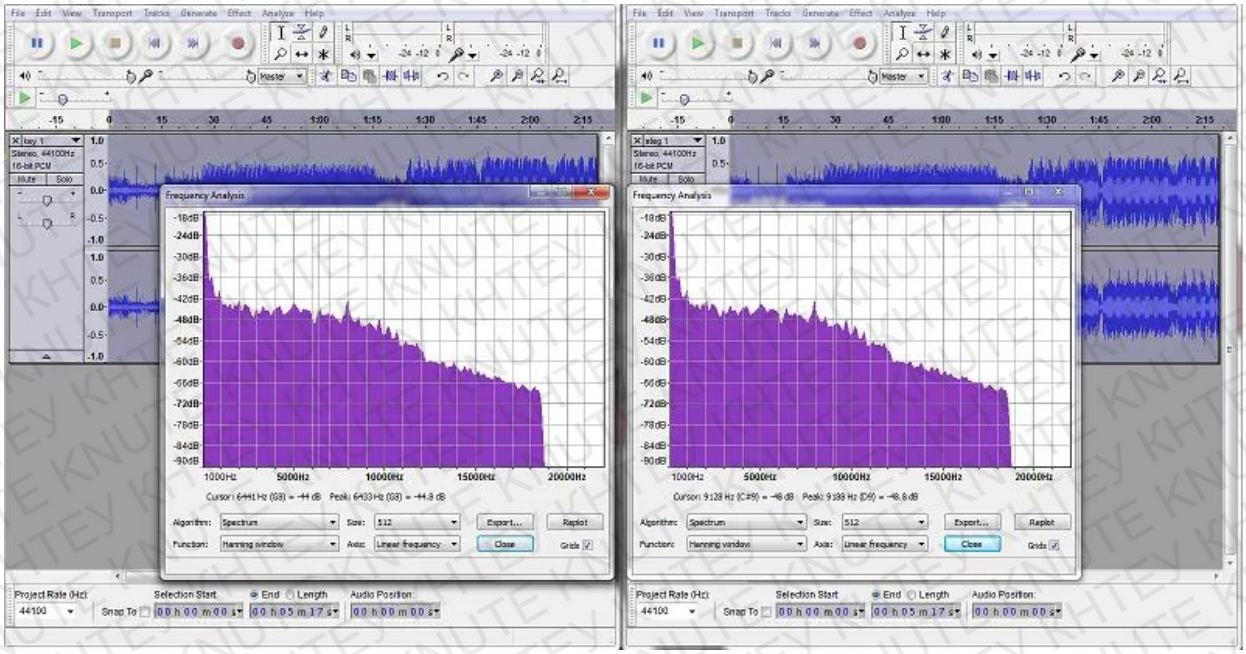


Рис. 2.10. Звукові спектри оригінального wav файлу та зміненого

```
public static byte[] EncryptTextLinear(byte[] wav, string text)
{
    WavAudio audio = new WavAudio(wav);
    uint value = 0;
    string pass = string.Format(audio.bitsPerSample.ToString());
    AESEncrypt encrypt = new AESEncrypt();
    string encrypted = encrypt.EncryptString(text, pass);
    OutputConsole.Write(string.Format("Текст зашифровано \n{0}", encrypted));
    if (encrypted.Length <= Math.Floor((double)(audio.totalSamples / 8)))
    {
        uint n = 0;
        OutputConsole.Write("Код згенеровано");
        OutputConsole.Write("Обробка WAV файлу...");
        for (int i = 0; i < encrypted.Length; i++)
        {
            value = encrypted[i];
            for (int x = 0; x < 8; x++)
            {
                uint sample = n;
                uint sampleValue = audio.samples[sample];
                sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
                audio.samples[sample] = sampleValue;
                n++;
            }

        }
        value = 0;
        for (int x = 0; x < 8; x++)
        {
            uint sample = n;
            uint sampleValue = audio.samples[sample];
            sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
            audio.samples[sample] = sampleValue;
            n++;
        }
        audio.Save();
        OutputConsole.Write(string.Format("Текст зашифровано... використано {0} аудіо зразків", encrypted.Length * 8));
        OutputConsole.Write("Збереження WAV файлу");
        return audio.data;
    }
    else
    {
        return null;
    }

}
```

Рис. 2.11. Лінійне шифрування тексту у wav файл

```
public static byte[] EncryptFileLinear(byte[] wav, byte[] file, string filename)
{
    WavAudio audio = new WavAudio(wav);
    uint value = 0;
    int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
    HiddenFile f = new HiddenFile(file, filename);
    OutputConsole.Write(string.Format("Розмір файлу: {0} байт", file.Length));
    f.cipherFile((int)audio.totalSamples);
    if (file.Length <= Math.Floor((double)(audio.totalSamples / 8)) - extraBytes)
    {
        uint n = 0;
        OutputConsole.Write("Шифрування файлу");
        OutputConsole.Write("Обробка WAV файлу...");
        OutputConsole.Write("Написання метаданих ...");
        //Написати розмір файлу
        for (int i = 0; i < file.Length.ToString().Length; i++)...
        value = '#';
        for (int x = 0; x < 8; x++)...
        //Написати ім'я файлу
        for (int i = 0; i < filename.Length; i++)...
        value = 0;
        for (int x = 0; x < 8; x++)
        {
            uint sample = n;
            uint sampleValue = audio.samples[sample];
            sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
            audio.samples[sample] = sampleValue;
            n++;
        }
        //Написати вміст файлу
        OutputConsole.Write("Запис даних файлу...");
        for (int i = 0; i < file.Length; i++)...
    }
    else...
    OutputConsole.Write("Вбудовування файлу завершено");
    OutputConsole.Write(string.Format("Використано {0} аудіо зразків", (file.Length + extraBytes) * 8));
    audio.Save();
    return audio.data;
}
```

Рис. 2.12. Лінійне шифрування даних у wav файл

2.4 Висновки до розділу 2

Використовуючи різні методи шифрування була розроблена програма з корисним та сучасним функціоналом для приховування даних, яка надає перспективні можливості її користувачам: від шифрування звичайного тексту в зображення і до приховування будь-яких типів файлів у аудіофайл. Процес розробки був у середовищі Visual Studio з використанням мови C# і її обширних бібліотек, які надали багато корисних інструментів для написання коду.

РОЗДІЛ 3
ОПИС СТВОРЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРИХОВУВАННЯ ІНФОРМАЦІЇ

3.1. Опис програми

Коли користувач відкриває програму – одразу ж попадає на основну частину, в якій і знаходиться весь функціонал(Рис. 3.1.). Щоб охватити більшу частину потенційних користувачів, усі розробники програм намагаються зробити їх у мінімалістичному вигляді. Це пов'язано не тільки з метою зробити програму простою у користуванні, а ще й з, популярним на сьогоднішній день, стильом minimal modern. Більше того, сучасний асоціюється з чистим стилем. Тож метою було створення мінімалістичного виду, щоб не додати нічого лишнього до дизайну і більше зосередитись на можливостях програми.

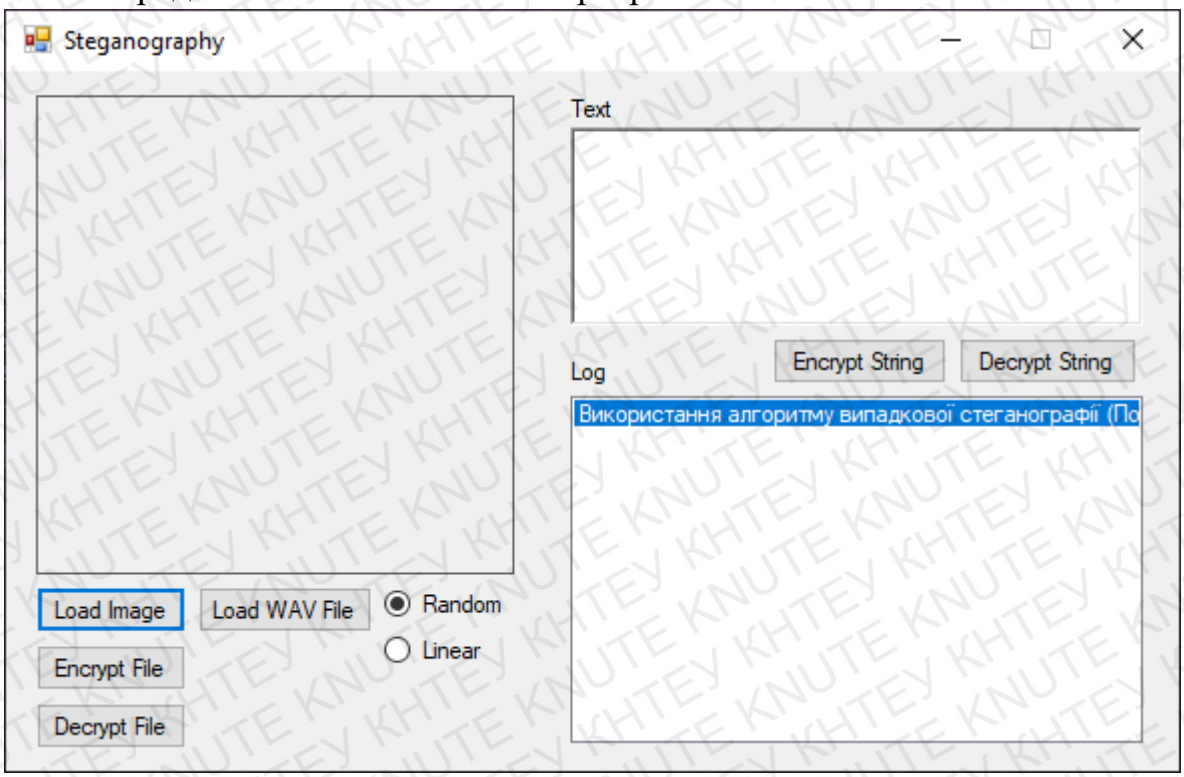


Рис. 3.1. Головна частина програми

Переважну частину програми займає вікно попереднього перегляду зображення, яке буде використовуватись як обкладинка. Щоб обрати зображення, користувач повинен натиснути на кнопку Load Image – після цього програма

відкриє зручне у використанні вікно для вибору зображення(Рис. 3.2.) та покаже що ви обрали на вікні попереднього перегляду(Рис. 3.3.).

Функціонал програми доповнює вибір між випадковим та лінійним алгоритмом стеганографії. За замовчуванням, після відкриття програми обрано алгоритм випадкової стеганографії, але його можна легко змінити просто натиснувши на потрібний вам алгоритм(Рис. 3.4.).

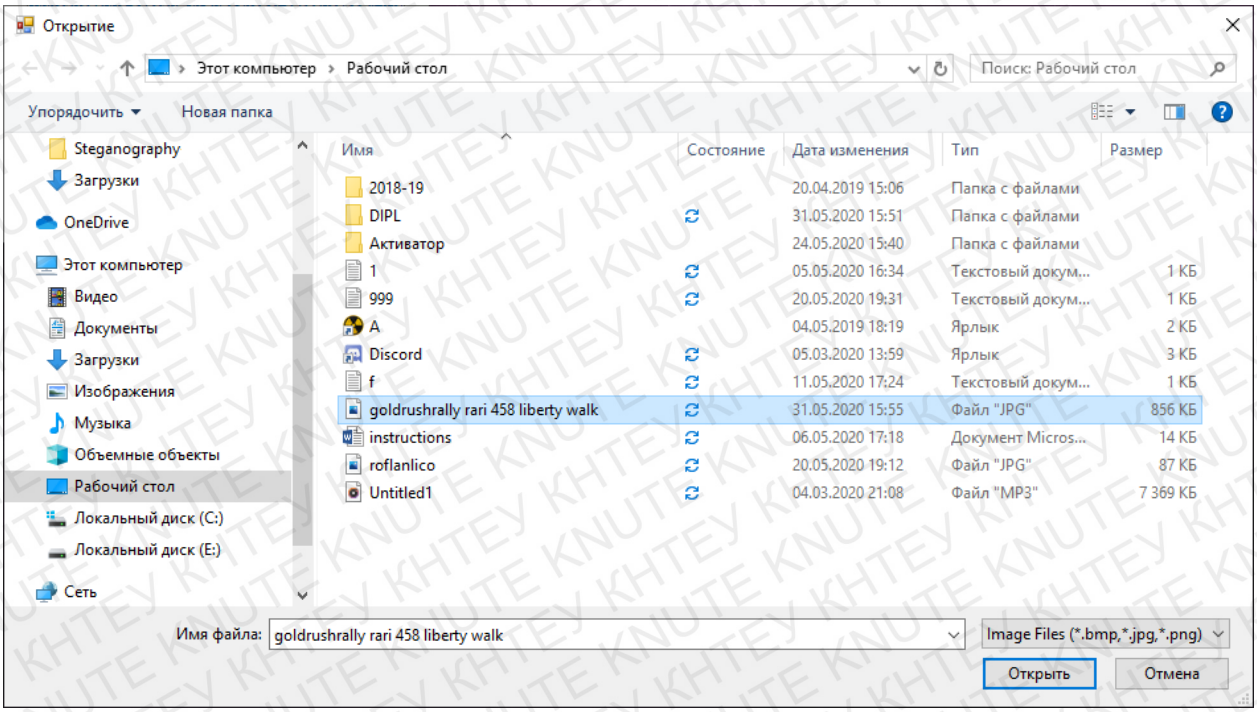


Рис. 3.2. Вікно вибору зображення

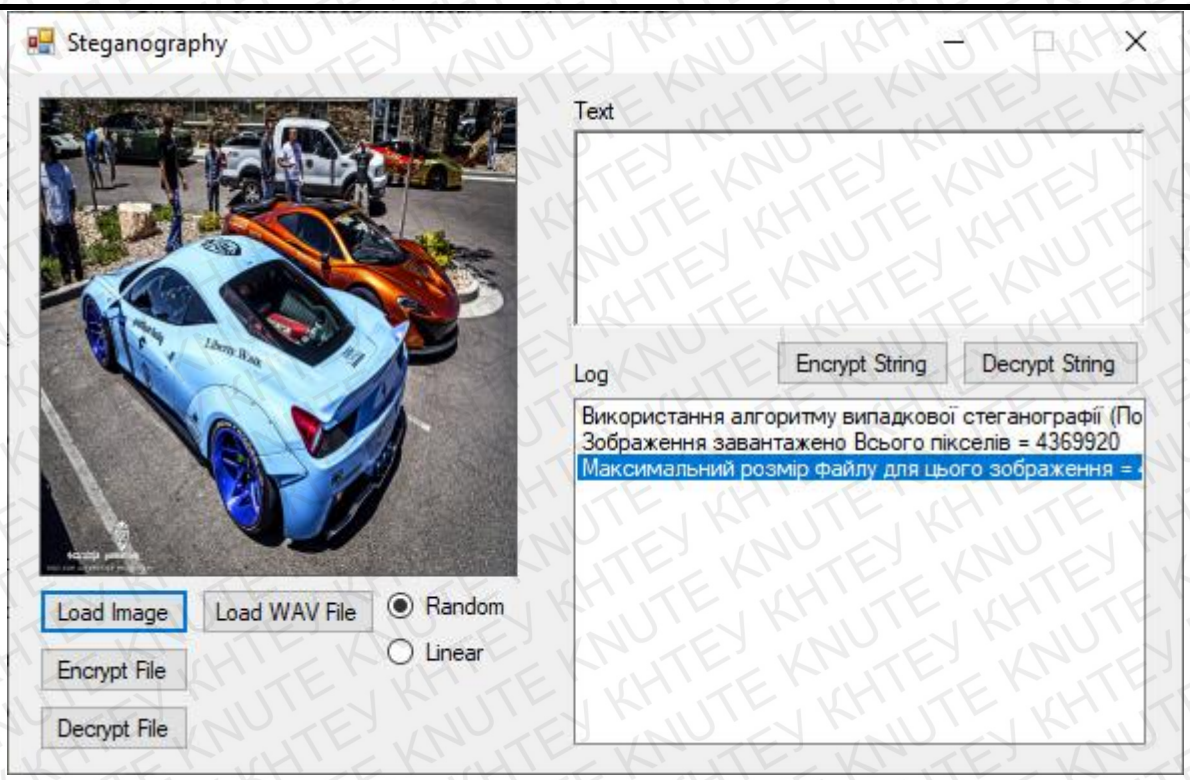


Рис. 3.3. Вікно попереднього перегляду обраного зображення

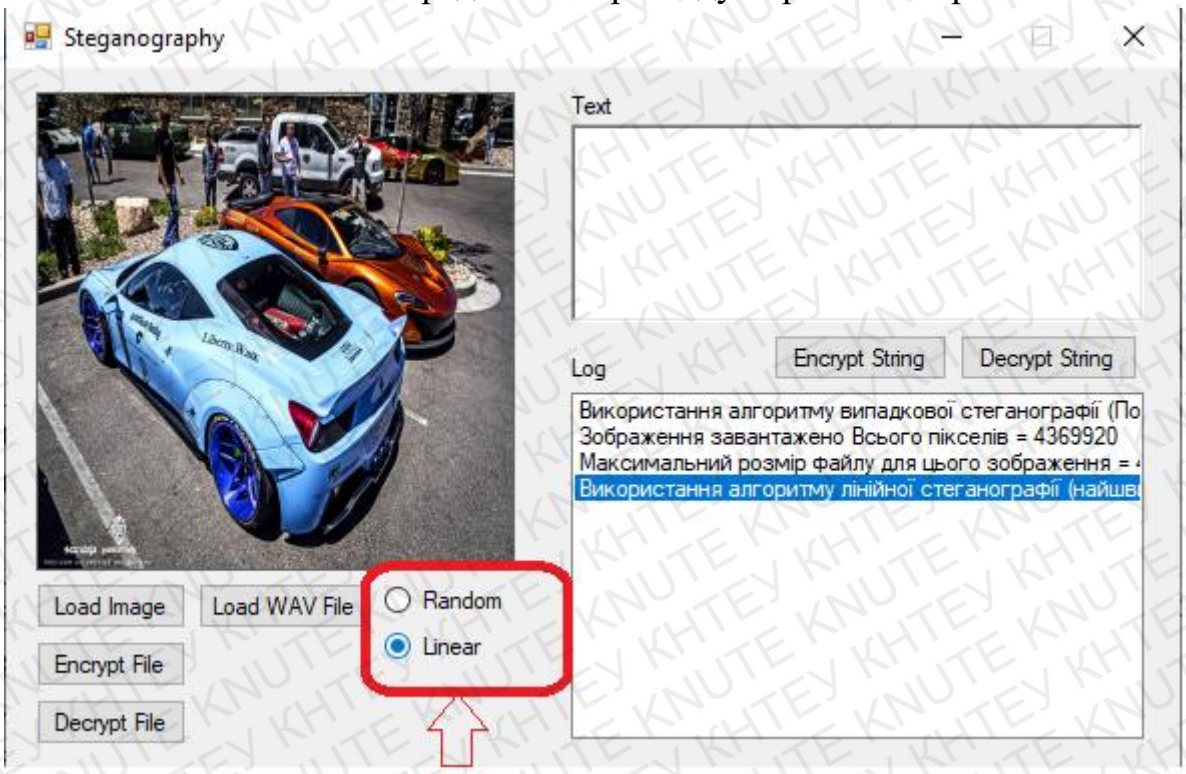


Рис. 3.4. Вибір алгоритму стеганографії

Про усі дії користувача інформує консоль, яка займає найбільшу частину у вікні програми(Рис. 3.5.). Щоб дізнатися більше просто натисніть в консолі два рази на текст, який вас зацікавив – це покаже більше корисної інформації(Рис. 3.6.).

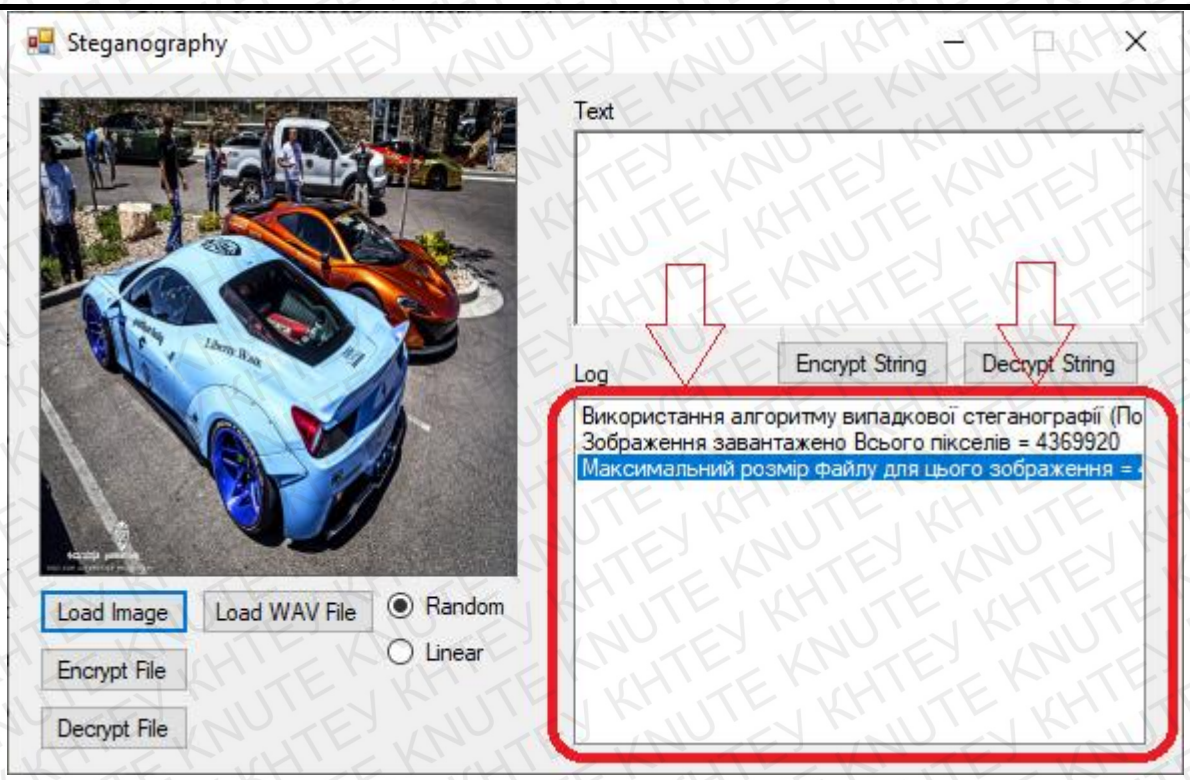


Рис. 3.5. Інформативна консоль

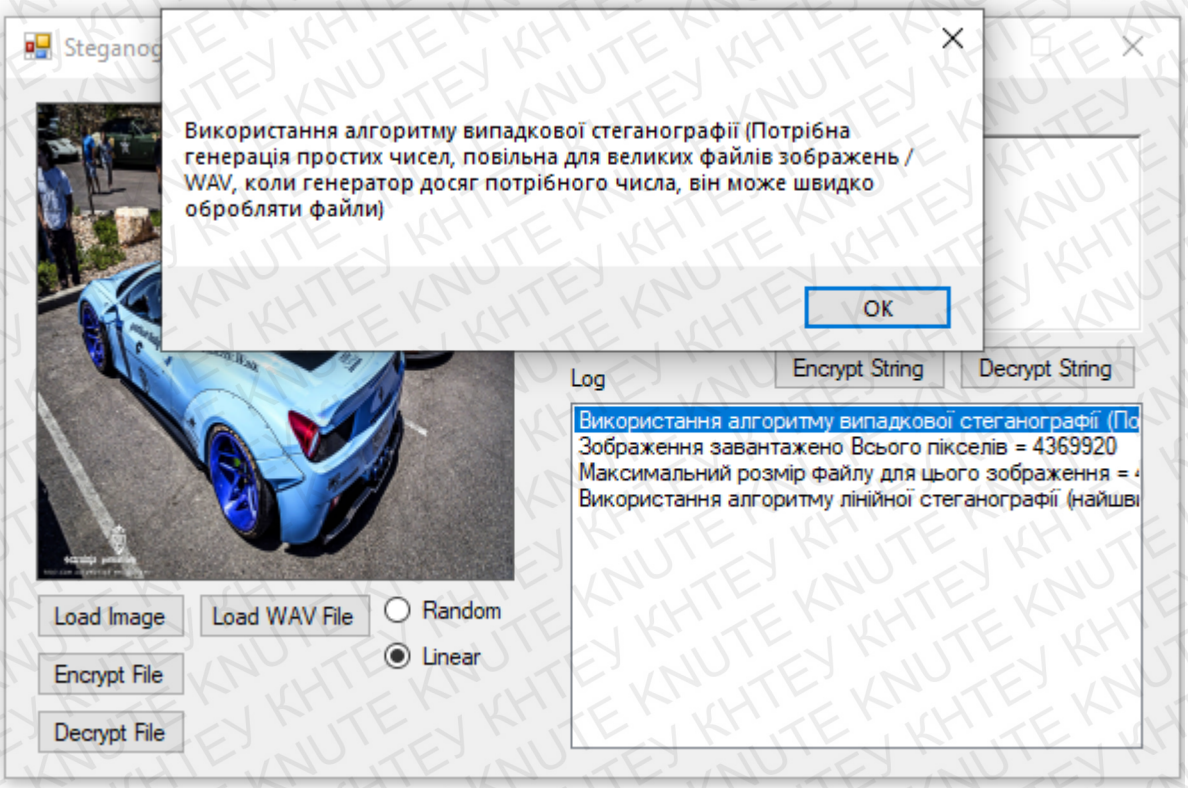


Рис. 3.6. Детальність інформативної консолі

Однією з основних можливостей програми є приховування тексту в обране користувачем зображення або wav файл. Це можна здійснити в вікні Text(Рис. 3.7.),

користуватися яким дуже легко. Просто наберіть або вставте потрібний вам текст в вікно для тексту. Та щоб приховати його, наприклад в обране нами зображення просто напишніть Encrypt String. Після цього програма зашифрує ваш текст в зображення(Рис. 3.7.), все що вам потрібно це тільки обрати куда зберегти уже змінений файл зображення та набрати назву файлу з розширенням, яке вам забажається(Наприклад texttest.png). Якщо вам потрібно розшифрувати текст з зображення потрібно просто обрати файл зображення, в якому було зашифровано текст і натиснути Decrypt String, після цього програма покаже розшифрований текст, в тому ж самому вікні для тексту(Рис. 3.8). Якщо є потреба приховати текст в wav файл – повторіть ті самі дії, але спочатку оберіть не зображення, а аудіофайл, використовуючи кнопку Load WAV File.

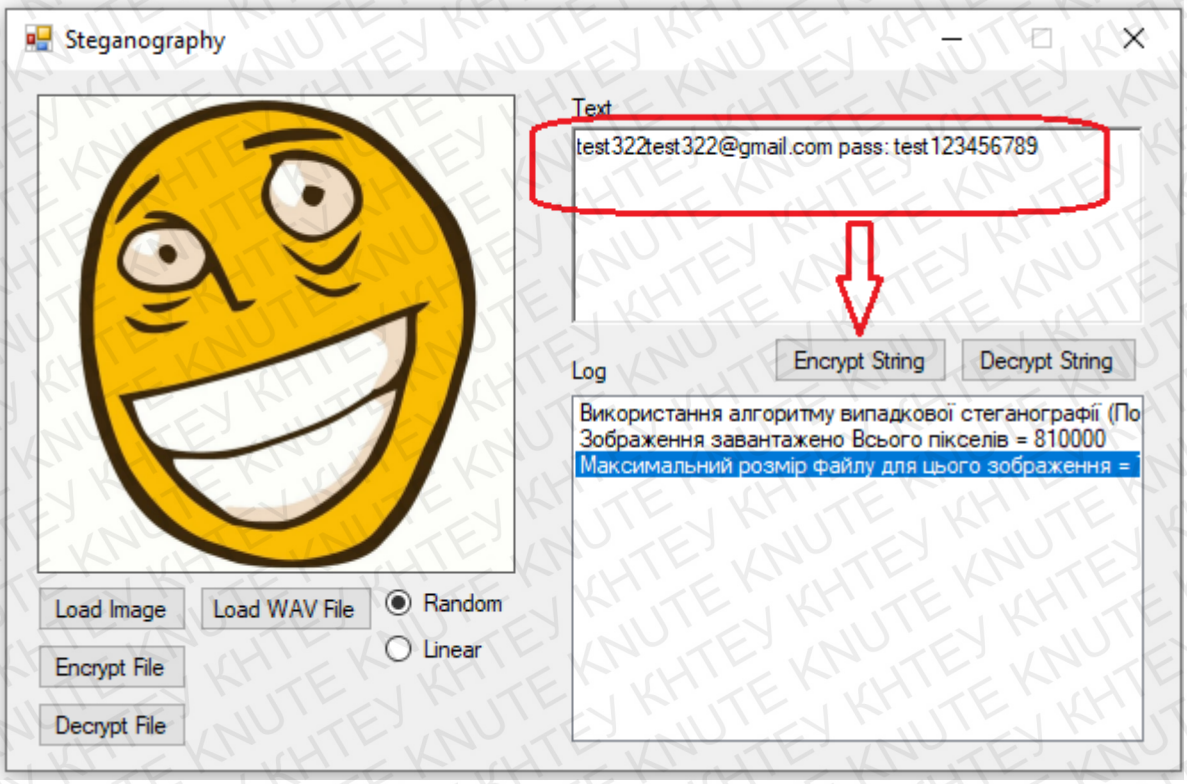


Рис. 3.7. Вікно для шифрування тексту

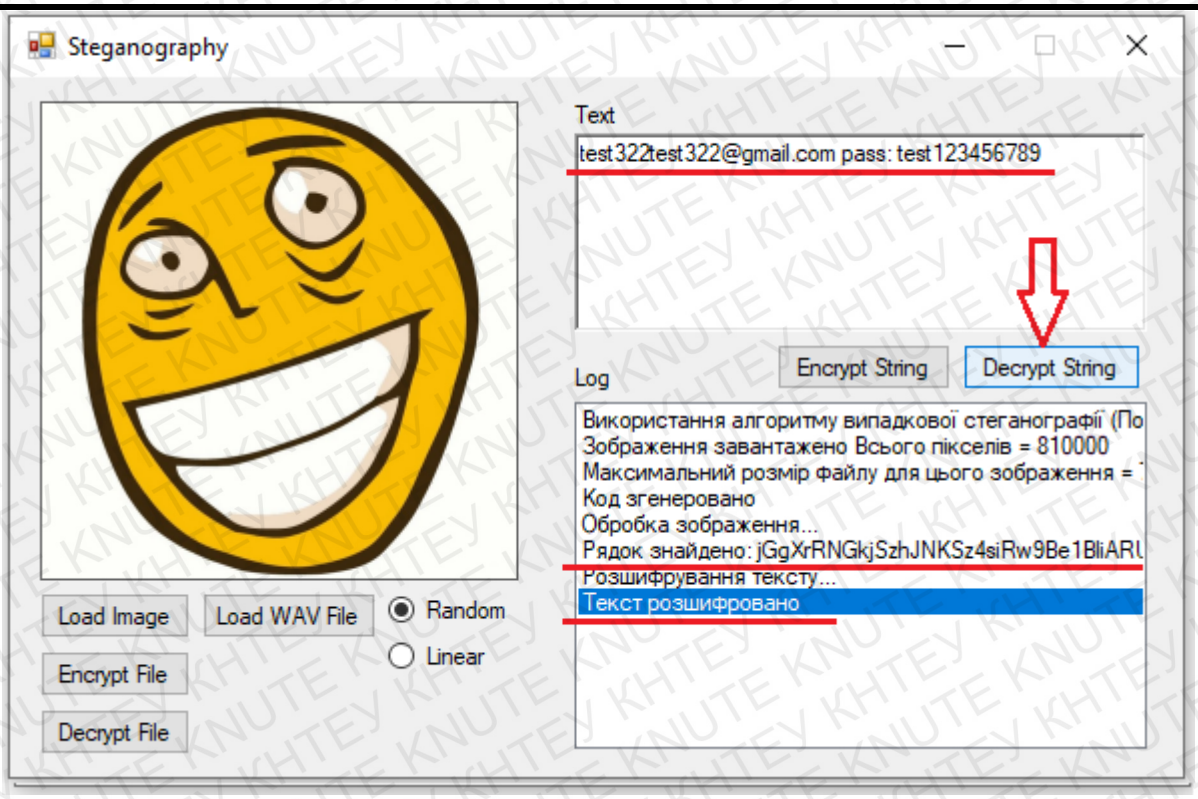


Рис. 3.8. Розшифрування тексту

Наступна основна функція програми – це приховання даних в зображення. Приховати в зображення можна будь-які типи файлів. Зашифрувати файл навіть простіше, ніж приховати текст, усе що потрібно – обрати файл зображення, яке буду слугувати як обкладинка. Розмір файлу який ви хочете приховати залежить від кількості пікселів на обраному вами зображенні, тому якщо є потреба приховати файл не самих малих розмірів – потрібно і зображення обирати побільше. Про усе це Вас інформує консоль, яка показує скільки пікселів в зображенні і який розмір файлу можна в нього приховати.

Щоб приховати файл в зображення потрібно обрати зображення і натиснути кнопку Encrypt File, після чого знову відкриється вікно вибору файлу, в якому і треба обрати потрібний вам файл, що буде вбудовано в зображення. Після цих дій консоль проінформує Вас про розмір обраного файлу і якщо розмір файлу не перевищує потрібну кількість пікселів, то почнеться шифрування(Рис 3.9.).

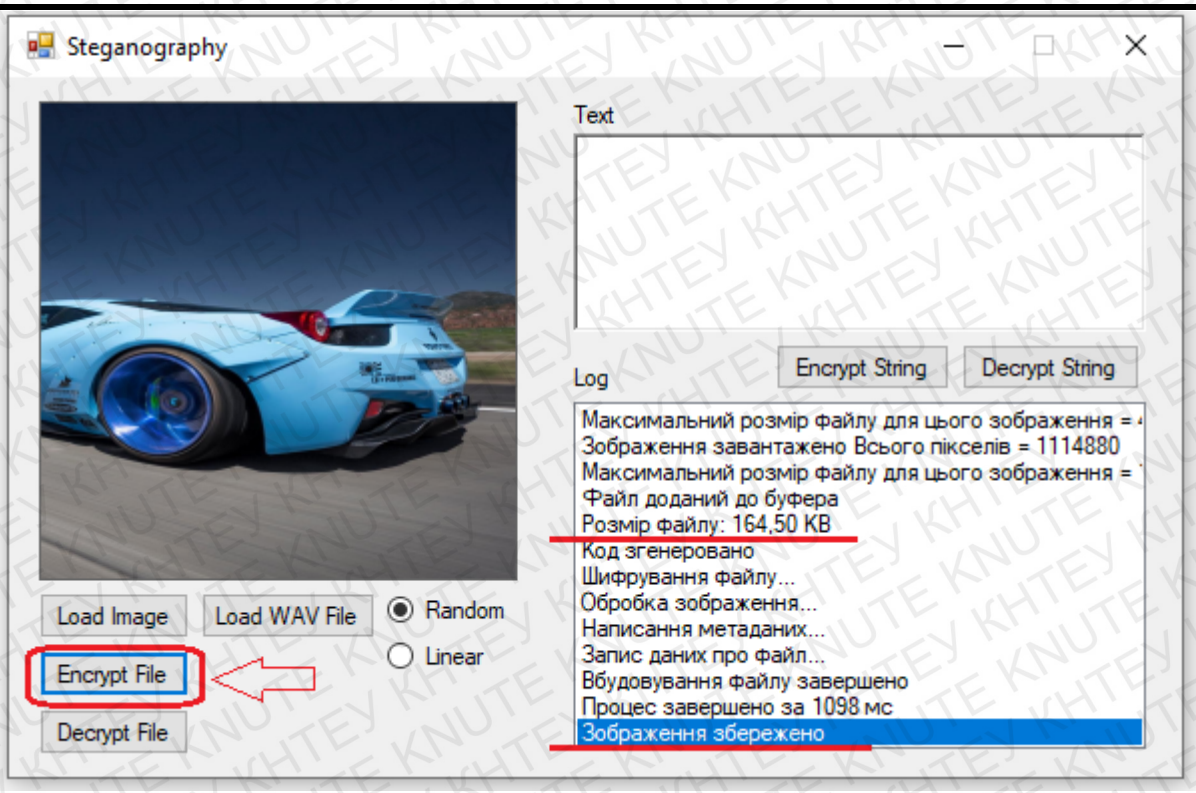


Рис. 3.9. Шифрування файлу в зображення

Щоб дістати зашифрований файл з зображення потрібно обрати зображення, в якому знаходиться прихована інформація кнопкою Load Image і після цього – натиснути Decrypt File, тоді програма почне обробку зображення і почне розшифровку файлу. Коли розшифровка закінчиться консоль проінформує Вас, який файл було вилучено і якіого розміру він(Рис. 3.10.). Після цього лише потрібно обрати куди буде збережено розшифрований файл в вікні вибору. Вміст та якість інформації вилученого файлу не порушується після розшифровки.

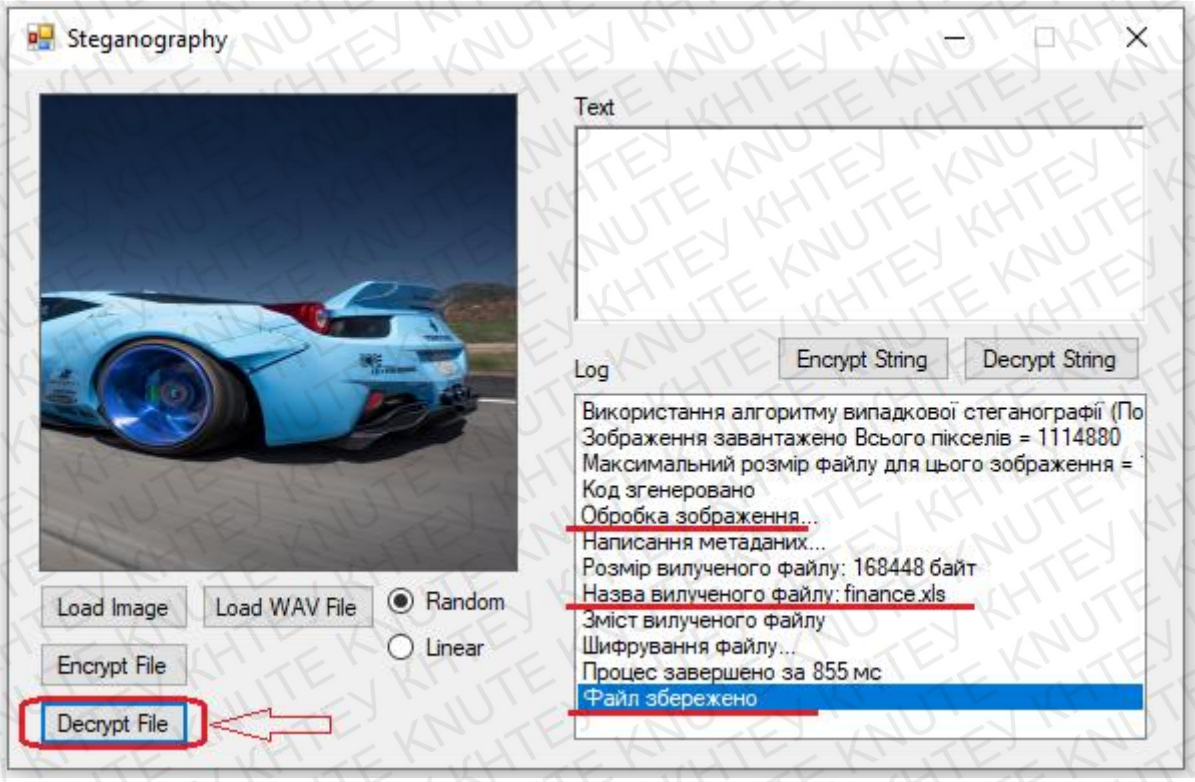


Рис. 3.10. Розшифрування файлу з зображення

Остання, але не по важливості, функція програми – приховування інформації у wav файл. Аудіо файл, який буде обкладинкою повинен мати обов'язково 8, 16, 24 або 32 біт, щоб зашифрована інформація не була помітною і не порушувала якість файла обкладинки.

Щоб приховати інформацію у wav файл потрібно натисути на кнопку Load WAV File, після чого на вікні попереднього перегляду буде виведено назву обраного файлу. Консоль проінформує вас, про кількість знайдених аудіо шаблонів, від яких залежить який по розміру файл можна зашифрувати в нього. Якщо ліміт розміру Вас влаштовує – потрібно обрати файл для приховування кнопкой Encrypt File. Програма обробить wav файл і залишиться тільки зберегти змінений файл з розширенням wav(Рис. 3.11.). Розшифровка аудіо файлу не дуже відрізняється. Треба натиснути Load WAV File і обрати аудіо файл, що містить приховану інформацію, після цього просто натиснути на кнопку Decrypt File і після обробки аудіо файлу потрібно буде зберегти розшифрований файл(Рис. 3.12.).

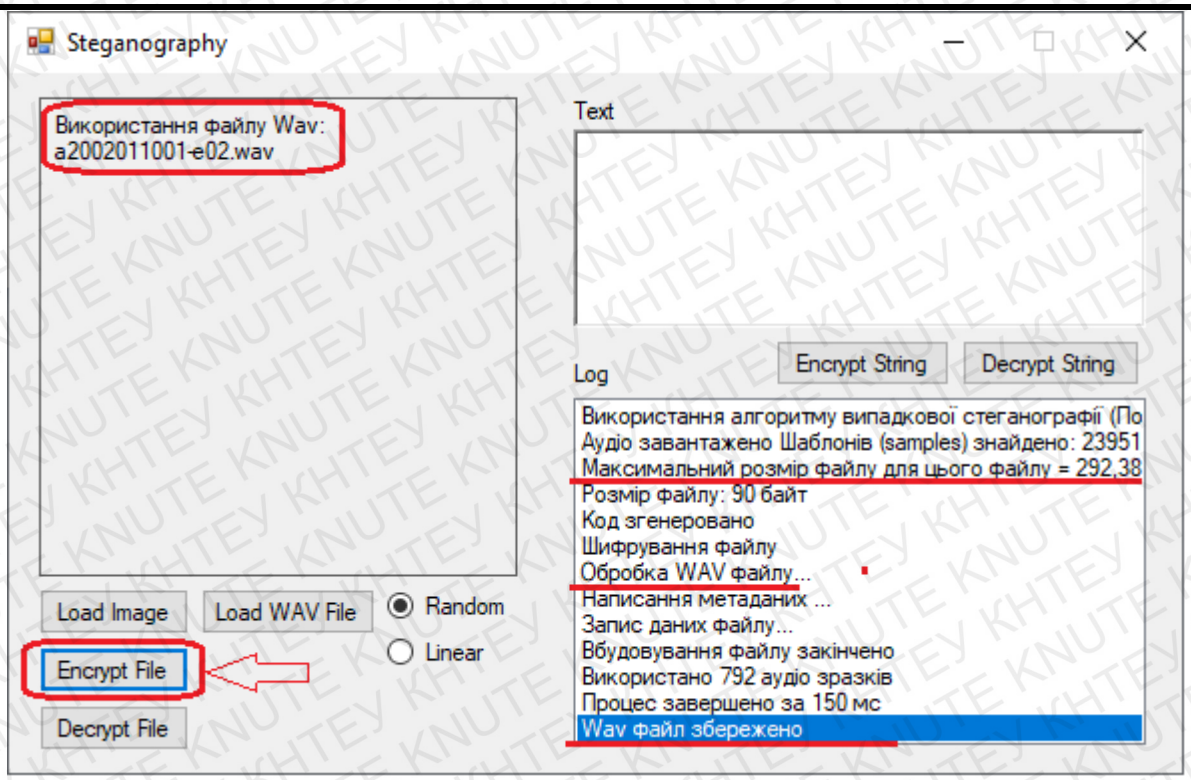


Рис. 3.11. Шифрування файлу у wav файл

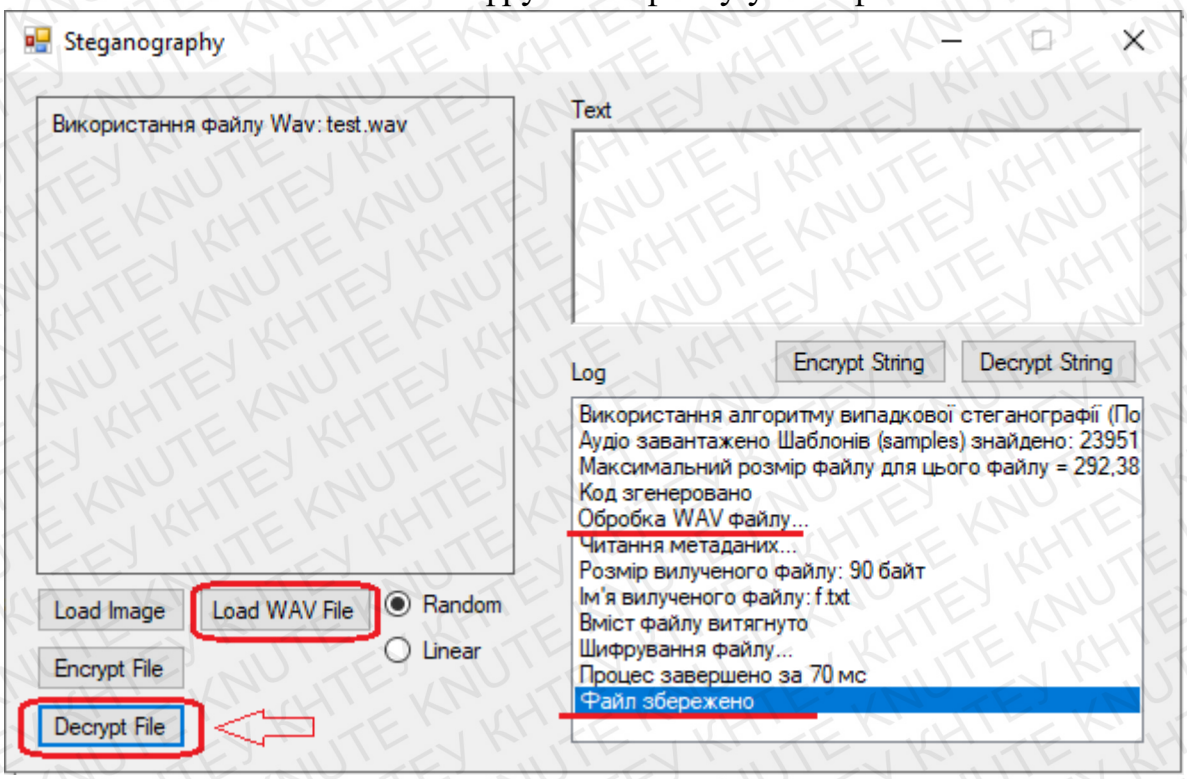


Рис. 3.12. Розшифрування файлу з wav файлу

3.2. *Тестування програми*

Після створення програми для приховування інформації важливим кроком було тестування, тому що вона має незвичайний функціонал, який повинен

працювати коректно і без помилок. Програма повинна приховувати інформацію і тим самим покращувати безпеку даних користувача, тож її працездатність має бути досконалою. Щоб уникнути проблем з програмою було протестовано усі її функції окремо, разом та в роботі з інтерфейсом. Це було дуже важливо, щоб уникнути пошкоджень даних або файлів програмою, через їх стискання та шифрування. Виконано було:

1) Перевірка створеного програмного коду та усунення проблем;
2) Тестування методу стиснення файлів;
3) Тестування функції приховування та розшифрування тексту в зображення та wav файл;
4) Тестування функції приховування та розшифрування файлів в зображення ;
5) Тестування функції приховування та розшифрування файлів у wav файл;
6) Перевірка кнопок так працездатності інтерфейсу;

3.3. Висновки до розділу 3

У третьому розділі було описано користувацький інтерфейс програми для приховування інформації та її можливості.

Оглянуто усі функції програми, її методи та можливості.

Було проведено послідовне тестування розробленого програмного коду та програми в цілому, щоб забезпечити коректність роботи.

Детально розписано про кожну кнопку інтерфейсу та її призначення і продемонстровано її працездатність.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У ході виконаного дослідження було визначено важливість безпеки інформації і було розроблено вирішення проблеми безпеки персональних даних.

Досліджено методи та способи приховування інформації різним шляхом.

Визначено послідовність стадій процесу створення програми для приховування інформації та детально було розглянуто кожен з них.

Створено технічне завдання на розробку програми для приховування інформації, де було розписано основні етапи створення та написання програмного коду.

Розроблена програма для приховування інформації. У розробці були використані:

- Інтегроване середовище розробки - Visual Studio;
- Мова програмування – C# з використанням .NET Framework;
- Спосіб приховування інформації – Стеганографія тексту, зображення та аудіо;

Перевірено працездатність розробленого програмного коду, проведено тестування інтерфейсу та функціоналу.

Доведено, що програма виконує свою задачу коректно та відповідає усім встановленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. “Hiding data in images by simple LSB substitution”. Chi-Kwong Chan, L.M. Cheng. Department of Computer Engineering and Information Technology, City University of Hong Kong, переглянута форма 11 липня 2003 року;
2. “An Effective Technique for Hiding Image in Audio”. Najiya Thasneem. E, Renjith V Ravi. International Journal of Science and Research (IJSR) , 2013;
3. “A Steganography Technique for Hiding Image in an Image using LSB Method for 24 Bit Color Image” Deepesh Rawat, Vijaya Bhandari. International Journal of Computer Applications(0975 – 8887) Volume 64– No.20, Лютий 2013 року;
4. “LSB Modification and Phase Encoding Technique of Audio Steganography Revisited”. Prof. Samir Kumar and Bandyopadhyay Barnali and Gupta Banik. International Journal of Advanced Research in Computer and Communication Engineering Vol. 1, Issue 4, Червень 2012 року;
5. “A Review of Methods and Approach for Secure Stegnography”. Shaveta Mahajan and Arpinder Singh. International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 10, Жовтень 2012 року;
6. “An Overview of Different Type of Data Hiding Scheme in Image using Steganographic Techniques”. Mukesh Garg and A.P. Gurudev Jangra. International Journal of Advanced Research in Computer Science and Software Engineering, Січень 2014 року;
7. “Advanced Digital Image Steganography Using LSB, PVD, and EMD: Emerging Research and Opportunities”. Gandharba Swain, Січень 2019 року;

8. THE IMPORTANCE OF INFORMATION SECURITY NOWADAYS [Електронний ресурс]. – Режим доступу: https://pecb.com/pdf/articles/27-pecb_the-importance-of-information-security-nowadays.pdf
9. Text Hiding in Multimedia by Huffman Encoding Algorithm Using Steganography [Електронний ресурс]. – Режим доступу: https://www.irjet.net/archives/V3/i7/IRJET-V3I7480.pdf
10. Image Steganography [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/314116270_Image_Steganography
11. Implementation of Huffman encoding [Електронний ресурс]. – Режим доступу: https://medium.com/@hemalatha.psna/implementation-of-huffman-encoding-b19aa344cb65
12. Audio Steganography [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/320124080_Audio_Steganography_Techniques_A_Survey
13. Steganography An Art of Hiding Data [Електронний ресурс]. – Режим доступу: https://arxiv.org/ftp/arxiv/papers/0912/0912.2319.pdf
14. Hybrid Approach to Text & Image Steganography using AES and LSB Technique [Електронний ресурс]. – Режим доступу: https://www.irjet.net/archives/V5/i4/IRJET-V5I4337.pdf

ДОДАТКИ

Додаток А

Steganography.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Drawing;

namespace Steganography
{
  public static class Steganography
  {

    public static long GetImageMaxLinear(Bitmap img)
    {
      return img.Width * img.Height;
    }

    private static Point LinearIndexToPoint(int index, int width, int height)
    {
      if (index < 0)
      {
        index *= -1;
      }
      return new Point(index % width, (int)Math.Floor((double)(index / width)));
    }

    public static Bitmap InsertEncryptedTextToImage(Image image, string text)
    {
      Bitmap img = new Bitmap(image);
      EncodeMessage(ref img, text);
      return img;
    }

    public static Bitmap InsertEncryptedTextToImageLinear(Image image, string text)
    {
      Bitmap img = new Bitmap(image);
      EncodeMessageLinear(ref img, text);
```

```
    return img;
}

private static void EncodeMessage(ref Bitmap img, string text)
{
    int maxLinear = img.Width * img.Height;
    SeedRNG generator = new SeedRNG((maxLinear + img.Width).GetHashCode(), maxLinear);
    OutputConsole.Write("Код згенеровано");
    string pass = string.Format("{0}x{1}={2}", img.Width, img.Height, maxLinear);
    AESEncrypt encrypt = new AESEncrypt();
    string encrypted = encrypt.EncryptString(text, pass);
    OutputConsole.Write(string.Format("Текст зашифрований \n{0}", encrypted));
    OutputConsole.Write("Обробка зображення...");
    if (encrypted.Length < maxLinear)
    {
        for (int i = 0; i < encrypted.Length; i++)
        {
            Point point = LinearIndexToPoint(generator.Next, img.Width, img.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            char letter = encrypted[i];
            int value = Convert.ToInt32(letter);
            Color n = EncodePixel(pixel,value);
            img.SetPixel(point.X, point.Y, n);
        }
        //Нульове знач
        Point pointEnd = LinearIndexToPoint(generator.Next, img.Width, img.Height);
        Color pixelEnd = img.GetPixel(pointEnd.X, pointEnd.Y);
        img.SetPixel(pointEnd.X, pointEnd.Y, EncodePixel(pixelEnd, 0));
        OutputConsole.Write("Вбудовування зашифрованого тексту завершено");
    }
    else
    {
        OutputConsole.Write("Помилка: Розмір зображення не підтримує зашифрований розмір тексту");
        img = null;
    }
}

private static void EncodeMessageLinear(ref Bitmap img, string text)
{
    int maxLinear = img.Width * img.Height;
    string pass = string.Format("{0}x{1}={2}", img.Width, img.Height, maxLinear);
    AESEncrypt encrypt = new AESEncrypt();
```

```
        int c = 0;
        string encrypted = encrypt.EncryptString(text, pass);
        OutputConsole.Write(string.Format("Текст зашифровано \n{0}", encrypted));
        OutputConsole.Write("Обробка зображення...");
        if (encrypted.Length < maxLinear)
        {
            for (int i = 0; i < encrypted.Length; i++)
            {
                Point point = LinearIndexToPoint(i, img.Width, img.Height);
                Color pixel = img.GetPixel(point.X, point.Y);
                char letter = encrypted[i];
                int value = Convert.ToInt32(letter);
                Color n = EncodePixel(pixel, value);
                img.SetPixel(point.X, point.Y, n);
                c = i;
            }
            //Нульове значення розміщене в кінці для позначення кінця тексту - або використати 255 для кращого
приховування
            Point pointEnd = LinearIndexToPoint(c, img.Width, img.Height);
            Color pixelEnd = img.GetPixel(pointEnd.X, pointEnd.Y);
            img.SetPixel(pointEnd.X, pointEnd.Y, EncodePixel(pixelEnd, 0));
            OutputConsole.Write("Вбудовування зашифрованого тексту завершено");
        }
        else
        {
            OutputConsole.Write("Помилка: Розмір зображення не підтримує зашифрований розмір тексту");
            img = null;
        }
    }

    public static string GetDecryptedTextFromImage(Image image)
    {
        Bitmap img = new Bitmap(image);
        int maxLinear = image.Width * image.Height;
        SeedRNG generator = new SeedRNG((maxLinear + image.Width).GetHashCode(), maxLinear);
        OutputConsole.Write("Код згенеровано");
        string pass = string.Format("{0}x{1}={2}", image.Width, image.Height, maxLinear);
        AESEncrypt encrypt = new AESEncrypt();
        string text = string.Empty;
        int value=0;
        OutputConsole.Write("Обробка зображення...");
        do
```

```
    {
        Point point = LinearIndexToPoint(generator.Next, image.Width, image.Height);
        Color pixel = img.GetPixel(point.X, point.Y);
        value = DecodePixel(pixel);

        if (value != 0)
            text += Convert.ToChar(value);
    } while (value != 0);
    try
    {
        OutputConsole.Write(string.Format("Рядок знайдено: \n{0}", text));
        OutputConsole.Write("Розшифрування тексту...");
        return encrypt.DecryptString(text, pass);
    }
    catch (Exception e)
    {
        OutputConsole.Write("Помилка: Текст не знайдено");
        Console.WriteLine(e.Message);
        return null;
    }
}

public static string GetDecryptedTextFromImageLinear(Image image)
{
    Bitmap img = new Bitmap(image);
    int maxLinear = image.Width * image.Height;
    string pass = string.Format("{0}x{1}={2}", image.Width, image.Height, maxLinear);
    AESEncrypt encrypt = new AESEncrypt();
    string text = string.Empty;
    int value = 0;
    int i = 0;
    OutputConsole.Write("Обробка зображення...");
    do
    {
        Point point = LinearIndexToPoint(i, image.Width, image.Height);
        Color pixel = img.GetPixel(point.X, point.Y);
        value = DecodePixel(pixel);
        i++;
        if (value != 255)
            text += Convert.ToChar(value);
    } while (value != 255);
    try
```

```
    {
        OutputConsole.Write(string.Format("Рядок знайдено: \n{0}", text));
        OutputConsole.Write("Розшифрування тексту...");
        return encrypt.DecryptString(text, pass);
    }
    catch (Exception e)
    {
        OutputConsole.Write("Помилка: Текст не знайдено");
        Console.WriteLine(e.Message);
        return null;
    }
}

public static Bitmap InsertFileToImage(Image image, byte[] file, string filename)
{
    Bitmap img = new Bitmap(image);
    EncodeFileWithColor(ref img, file, filename);
    return img;
}

public static Bitmap InsertFileToImageLinear(Image image, byte[] file, string filename)
{
    Bitmap img = new Bitmap(image);
    EncodeFileWithColorLinear(ref img, file, filename);
    return img;
}

public static Bitmap InsertFileToImage2(Image image, byte[] file, string filename)
{
    Bitmap img = new Bitmap(image);
    EncodeFileWithColor2(ref img, file, filename);
    return img;
}

private static void EncodeFileWithColor(ref Bitmap img, byte[] file, string filename)
{
    Bitmap backup = img.Clone() as Bitmap;
    int maxLinear = img.Width * img.Height;
    OutputConsole.Write(string.Format("Розмір файлу: {0}", FileSizeFormatProvider.GetFileSize(file.Length)));
    SeedRNG generator = new SeedRNG((maxLinear + img.Width).GetHashCode(), maxLinear);
    OutputConsole.Write("Код згенеровано");
    int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
```

```
HiddenFile f = new HiddenFile(file, filename);
f.cipherFile((maxLinear + img.Width).GetHashCode());
OutputConsole.Write("Шифрування файлу...");
if (file.Length < maxLinear - extraBytes)
{
  string fileLength = file.Length.ToString();
  OutputConsole.Write("Обробка зображення...");
  OutputConsole.Write("Написання метаданих...");
  for (int i = 0; i < fileLength.Length; i++)
  {
    Point point = LinearIndexToPoint(generator.Next, img.Width, img.Height);
    Color pixel = img.GetPixel(point.X, point.Y);
    char letter = fileLength[i];
    int value = Convert.ToInt32(letter);
    img.SetPixel(point.X, point.Y, EncodePixel(pixel,value));
  }
  //Вставте #, щоб розділити довжину файлу від імені файлу
  Point point1 = LinearIndexToPoint(generator.Next, img.Width, img.Height);
  Color pixel1 = img.GetPixel(point1.X, point1.Y);
  int value1 = Convert.ToInt32('#');
  img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1, value1));

  //Вставте ім'я файлу та закінчіть нульовою графою
  for (int i = 0; i < filename.Length; i++)
  {
    Point point = LinearIndexToPoint(generator.Next, img.Width, img.Height);
    Color pixel = img.GetPixel(point.X, point.Y);
    char letter = filename[i];
    int value = Convert.ToInt32(letter);
    img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
  }
  point1 = LinearIndexToPoint(generator.Next, img.Width, img.Height);
  pixel1 = img.GetPixel(point1.X, point1.Y);
  value1 = 0;
  img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1,value1));
  OutputConsole.Write("Запис даних про файл...");
  //Написати файл
  for (int i = 0; i < file.Length; i++)
  {
    Point point = LinearIndexToPoint(generator.Next, img.Width, img.Height);
    Color pixel = img.GetPixel(point.X, point.Y);
    int value = f.file[i];
```

```
            img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
        }
        OutputConsole.Write("Вбудовування файлу завершено");
    }
    else
    {
        OutputConsole.Write("Розмір файлу перевищує загальний розмір пікселів у зображенні з додатковими даними, змініть розмір або використовуйте інше зображення для шифрування цього файлу");
        img = null;
    }
}

private static void EncodeFileWithColorLinear(ref Bitmap img, byte[] file, string filename)
{
    Bitmap backup = img.Clone() as Bitmap;
    int maxLinear = img.Width * img.Height;
    int c = 0;
    OutputConsole.Write(string.Format("Розмір файлу: {0}", FileSizeFormatProvider.GetFileSize(file.Length)));
    int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
    HiddenFile f = new HiddenFile(file, filename);
    f.cipherFile((maxLinear + img.Width).GetHashCode());
    OutputConsole.Write("Шифрування файлу...");
    if (file.Length < maxLinear - extraBytes)
    {
        string fileLength = file.Length.ToString();
        OutputConsole.Write("Обробка зображення...");
        OutputConsole.Write("Написання метаданих...");
        for (int i = 0; i < fileLength.Length; i++)
        {
            Point point = LinearIndexToPoint(c, img.Width, img.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            char letter = fileLength[i];
            int value = Convert.ToInt32(letter);
            img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
            c++;
        }
        //Вставте #, щоб розділити довжину файлу від імені файлу
        Point point1 = LinearIndexToPoint(c, img.Width, img.Height);
        Color pixel1 = img.GetPixel(point1.X, point1.Y);
        int value1 = Convert.ToInt32('#');
        img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1, value1));
        c++;
```

```
            //Вставте ім'я файлу та закінчіть нульовою графою
            for (int i = 0; i < filename.Length; i++)
            {
                Point point = LinearIndexToPoint(c, img.Width, img.Height);
                Color pixel = img.GetPixel(point.X, point.Y);
                char letter = filename[i];
                int value = Convert.ToInt32(letter);
                img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
                c++;
            }
            point1 = LinearIndexToPoint(c, img.Width, img.Height);
            pixel1 = img.GetPixel(point1.X, point1.Y);
            value1 = 0;
            img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1, value1));
            c++;
            //Написати файл
            OutputConsole.Write("Запис даних про файл...");
            for (int i = 0; i < file.Length; i++)
            {
                Point point = LinearIndexToPoint(c, img.Width, img.Height);
                Color pixel = img.GetPixel(point.X, point.Y);
                int value = f.file[i];
                img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
                c++;
            }
            OutputConsole.Write("Вбудовування файлу завершено");
        }
        else
        {
            OutputConsole.Write("Розмір файлу перевищує загальний розмір пікселів у зображенні з додатковими
даними, змініть розмір або використовуйте інше зображення для шифрування цього файлу");
            img = null;
        }
    }

    private static void EncodeFileWithColor2(ref Bitmap img, byte[] file, string filename)
    {
        Bitmap backup = img.Clone() as Bitmap;
        int maxLinear = img.Width * img.Height;
        OutputConsole.Write(string.Format("Розмір файлу: {0}", FileSizeFormatProvider.GetFileSize(file.Length)));
        SeedRNG generator = new SeedRNG((maxLinear + img.Width).GetHashCode(), maxLinear, true);
        OutputConsole.Write("Код згенеровано");
```

```
int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
HiddenFile f = new HiddenFile(file, filename);
f.cipherFile((maxLinear + img.Width).GetHashCode());
OutputConsole.Write("Шифрування файлу...");
if (file.Length < maxLinear - extraBytes)
{
   string fileLength = file.Length.ToString();
   OutputConsole.Write("Обробка зображення...");
   OutputConsole.Write("Написання метаданих...");
   for (int i = 0; i < fileLength.Length; i++)
   {
      Point point = LinearIndexToPoint(generator.NextN, img.Width, img.Height);
      Color pixel = img.GetPixel(point.X, point.Y);
      char letter = fileLength[i];
      int value = Convert.ToInt32(letter);
      img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
   }
   //Вставте #, щоб розділити довжину файлу від імені файлу
   Point point1 = LinearIndexToPoint(generator.NextN, img.Width, img.Height);
   Color pixel1 = img.GetPixel(point1.X, point1.Y);
   int value1 = Convert.ToInt32('#');
   img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1, value1));

   //Вставте ім'я файлу та закінчіть нульовою графою
   for (int i = 0; i < filename.Length; i++)
   {
      Point point = LinearIndexToPoint(generator.NextN, img.Width, img.Height);
      Color pixel = img.GetPixel(point.X, point.Y);
      char letter = filename[i];
      int value = Convert.ToInt32(letter);
      img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
   }
   point1 = LinearIndexToPoint(generator.NextN, img.Width, img.Height);
   pixel1 = img.GetPixel(point1.X, point1.Y);
   value1 = 0;
   img.SetPixel(point1.X, point1.Y, EncodePixel(pixel1, value1));
   OutputConsole.Write("Запис даних про файл...");
   //Написати файл
   for (int i = 0; i < file.Length; i++)
   {
      Point point = LinearIndexToPoint(generator.NextN, img.Width, img.Height);
      Color pixel = img.GetPixel(point.X, point.Y);
```

```
            int value = f.file[i];
            img.SetPixel(point.X, point.Y, EncodePixel(pixel, value));
        }
        OutputConsole.Write("Вбудовування файлу завершено");
    }
    else
    {
        OutputConsole.Write("Розмір файлу перевищує загальний розмір пікселів у зображенні з додатковими даними, змініть розмір або використовуйте інше зображення для шифрування цього файлу");
        img = null;
    }
}

public static HiddenFile GetFileFromImage(Image image)
{
    try
    {
        Bitmap img = new Bitmap(image);
        int maxLinear = image.Width * image.Height;
        SeedRNG generator = new SeedRNG((maxLinear + image.Width).GetHashCode(), maxLinear);
        OutputConsole.Write("Код згенеровано");
        string text = string.Empty;
        int value = 0;
        //Прочитати довжину файлу
        OutputConsole.Write("Обробка зображення...");
        OutputConsole.Write("Написання метаданих...");
        do
        {
            Point point = LinearIndexToPoint(generator.Next, image.Width, image.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            value = DecodePixel(pixel);

            if (value != '#')
                text += Convert.ToChar(value);
        } while (value != '#' && char.IsNumber((char)value));
        int filelength = int.Parse(text);
        OutputConsole.Write(string.Format("Розмір вилученого файлу: {0} байт", filelength));
        text = string.Empty;
        value = 0;
        //Прочитати назву файлу
        do
        {
```

```
            Point point = LinearIndexToPoint(generator.Next, image.Width, image.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            value = DecodePixel(pixel);

            if (value != 0)
              text += Convert.ToChar(value);
        } while (value != 0);
        string filename = text;
        OutputConsole.Write(string.Format("Назва вилученого файлу: {0}", filename));
        byte[] file = new byte[filelength];
        for (int i = 0; i < filelength; i++)
        {
            Point point = LinearIndexToPoint(generator.Next, image.Width, image.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            value = DecodePixel(pixel);
            file[i] = (byte)value;
        }
        OutputConsole.Write(string.Format("Зміст вилученого файлу"));
        HiddenFile f = new HiddenFile(file, filename);
        f.cipherFile((maxLinear + img.Width).GetHashCode());
        OutputConsole.Write("Шифрування файлу...");
        return f;
    }
    catch (Exception e)
    {
        OutputConsole.Write("Помилка: Файл не знайдено");
        Console.WriteLine(e.Message);
        return null;
    }
}

public static HiddenFile GetFileFromImageLinear(Image image)
{
    try
    {
        Bitmap img = new Bitmap(image);
        int maxLinear = image.Width * image.Height;
        int c = 0;
        string text = string.Empty;
        int value = 0;
        OutputConsole.Write("Обробка зображення...");
        OutputConsole.Write("Написання метаданих...");
```

```
    //Прочитати довжину файлу
    do
    {
        Point point = LinearIndexToPoint(c, image.Width, image.Height);
        Color pixel = img.GetPixel(point.X, point.Y);
        value = DecodePixel(pixel);
        c++;
        if (value != '#')
            text += Convert.ToChar(value);
    } while (value != '#' && char.IsNumber((char)value));
    int filelength = int.Parse(text);
    OutputConsole.Write(string.Format("Розмір вилученого файлу: {0} байт", filelength));
    text = string.Empty;
    value = 0;
    //Прочитати назву файлу
    do
    {
        Point point = LinearIndexToPoint(c, image.Width, image.Height);
        Color pixel = img.GetPixel(point.X, point.Y);
        value = DecodePixel(pixel);
        c++;
        if (value != 0)
            text += Convert.ToChar(value);
    } while (value != 0);
    string filename = text;
    OutputConsole.Write(string.Format("Назва вилученого файлу: {0}", filename));
    byte[] file = new byte[filelength];
    for (int i = 0; i < filelength; i++)
    {
        Point point = LinearIndexToPoint(c, image.Width, image.Height);
        Color pixel = img.GetPixel(point.X, point.Y);
        value = DecodePixel(pixel);
        file[i] = (byte)value;
        c++;
    }
    OutputConsole.Write(string.Format("Зміст вилученого файлу"));
    HiddenFile f = new HiddenFile(file, filename);
    f.cipherFile((maxLinear + img.Width).GetHashCode());
    OutputConsole.Write("Шифрування файлу...");
    return f;
}
catch (Exception e)
```

```
    {
        OutputConsole.Write("Помилка: Файл не знайдено");
        Console.WriteLine(e.Message);
        return null;
    }
}

public static HiddenFile GetFileFromImage2(Image image)
{
    try
    {
        Bitmap img = new Bitmap(image);
        int maxLinear = image.Width * image.Height;
        SeedRNG generator = new SeedRNG((maxLinear + image.Width).GetHashCode(), maxLinear, true);
        OutputConsole.Write("Код згенеровано");
        string text = string.Empty;
        int value = 0;
        //Прочитати довжину файлу
        OutputConsole.Write("Обробка зображення...");
        OutputConsole.Write("Читання метаданих...");
        do
        {
            Point point = LinearIndexToPoint(generator.NextN, image.Width, image.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            value = DecodePixel(pixel);

            if (value != '#')
                text += Convert.ToChar(value);
        } while (value != '#' && char.IsNumber((char)value));
        int filelength = int.Parse(text);
        OutputConsole.Write(string.Format("Розмір вилученого файлу: {0} байт", filelength));
        text = string.Empty;
        value = 0;
        //Прочитати назву файлу
        do
        {
            Point point = LinearIndexToPoint(generator.NextN, image.Width, image.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            value = DecodePixel(pixel);

            if (value != 0)
                text += Convert.ToChar(value);
```

```
        } while (value != 0);
        string filename = text;
        OutputConsole.Write(string.Format("Назва вилученого файлу: {0}", filename));
        byte[] file = new byte[filelength];
        for (int i = 0; i < filelength; i++)
        {
            Point point = LinearIndexToPoint(generator.NextN, image.Width, image.Height);
            Color pixel = img.GetPixel(point.X, point.Y);
            value = DecodePixel(pixel);
            file[i] = (byte)value;
        }
        OutputConsole.Write(string.Format("Зміст вилученого файлу"));
        HiddenFile f = new HiddenFile(file, filename);
        f.cipherFile((maxLinear + img.Width).GetHashCode());
        OutputConsole.Write("Шифрування файлу...");
        return f;
    }
    catch (Exception e)
    {
        OutputConsole.Write("Помилка: Файл не знайдено");
        Console.WriteLine(e.Message);
        return null;
    }
}

public static Color EncodePixel(Color pixel, int value)
{
    int blueValue = value & 7;
    int greenValue = (value >> 3) & 7;
    int redValue = (value >> 6) & 3;

    int red = (pixel.R & 0xFC) | redValue;
    int green = (pixel.G & 0xF8) | greenValue;
    int blue = (pixel.B & 0xF8) | blueValue;

    return Color.FromArgb(red, green, blue);
}

public static int DecodePixel(Color pixel)
{
    int red = (pixel.R & 3);
    int green = (pixel.G & 7);
```

```
            int blue = (pixel.B & 7);
            int value = blue | (green << 3) | (red << 6);
            return value;
        }
    }
}
```

AESEncrypt.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Security.Cryptography;
using System.IO;
namespace Steganography
{
    public class AESEncrypt
    {
        static AesCryptoServiceProvider CreateAES(string key)
        {
            byte[] bytes = Encoding.ASCII.GetBytes(key);
            AesCryptoServiceProvider a = new AesCryptoServiceProvider();
            Rfc2898DeriveBytes r = new Rfc2898DeriveBytes(key, new byte[] { 1,2,3,4,5,6,7,8});
            a.Key = r.GetBytes(a.BlockSize / 8);
            a.IV = new byte[a.BlockSize / 8];
            a.Padding = PaddingMode.PKCS7;
            a.Mode = CipherMode.CBC;
            return a;
        }
        public string EncryptString(string text, string password)
        {
            try {
                byte[] textBytes = Encoding.Unicode.GetBytes(text);
                MemoryStream stream = new MemoryStream();
                AesCryptoServiceProvider aes = CreateAES(password);
                CryptoStream crypt = new CryptoStream(stream, aes.CreateEncryptor(), CryptoStreamMode.Write);
                crypt.Write(textBytes, 0, textBytes.Length);
                crypt.FlushFinalBlock();
                return Convert.ToBase64String(stream.ToArray());
            }
            catch
            {

            }
            return null;
        }

        public string DecryptString(string text, string password)
        {
            try {
                byte[] textBytes = Convert.FromBase64String(text);
                MemoryStream stream = new MemoryStream();
                AesCryptoServiceProvider aes = CreateAES(password);
                CryptoStream crypt = new CryptoStream(stream, aes.CreateDecryptor(), CryptoStreamMode.Write);
                crypt.Write(textBytes, 0, textBytes.Length);
                crypt.FlushFinalBlock();
                return Encoding.Unicode.GetString(stream.ToArray());
            }
            catch
            {

            }
            return null;
        }
    }
}
```

CipherFile.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Steganography
{
   public static class CipherFile //шифрування файлів
   {
      private static byte[] pad = new byte[0];
      private static int seed;
      private static int x;
      private static int previous;

      private static void preparePad(int l)
      {
         if (l > pad.Length)
         {
            Array.Resize(ref pad, l);
            //складання
            for (int i = previous; i < l; i++)
            {
               do
               {
                  x = (int)((0x13793A1F2 + (x >> 5) * 0xFF7AB) & 0xFFFFFFFF); //RNG формула
               } while ((x & 0xFF) != 0); //Уникайте складання хог з 0
               pad[i] = (byte)(x & 0xFF);
            }
            previous = l;
         }
      }

      public static byte[] cipherFile(byte[] file, int key)
      {
         byte[] newFile = new byte[file.Length];
         seed = key;
         previous = 0;
         preparePad(file.Length);
         for (int i = 0; i < file.Length; i++)
         {
            newFile[i] = (byte)(file[i] ^ pad[i]);
         }
         return newFile;
      }
   }
}
```

LCG.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Numerics;

namespace Steganography
{
   public class LCG
   {
      private BigInteger m;
      private BigInteger a;
      private BigInteger c;
      private BigInteger x0;
      private int index = 0;
      private int seed = 0;

      public List<BigInteger> x { get; private set; }

      public LCG(BigInteger m, BigInteger x0)
      {
         this.m = m;
         this.x0 = x0 % m;
         Initialize();
      }

      public LCG(BigInteger m, BigInteger x0, int seed)
      {
         this.m = m;
         this.x0 = x0 % m;
         this.seed = seed;
         Initialize();
      }

      private void Initialize()
      {
         x = new List<BigInteger>();
         List<BigInteger> p = primes();
         List<BigInteger> f = factors(p);
         c = calcC(p, f);
         a = f.Aggregate((x, y) => x * y);
         if (m % 4 == 0)
         {
            if (a % 4 == 0)
            {
               a++;
            }
            else
            {
               a = (a * 4) + 1;
            }
         }
         else
         {
            a++;
         }

         index = 0;
```

```
    x.Add(x0);
    for (BigInteger i = 1; i < m; i++)
    {
        x.Add(((a * x[(int)i - 1]) + c) % m);
    }
}

public void setIndex(int i)
{
    index = i;
}

public BigInteger get(int i)
{
    return x[i];
}

public BigInteger next()
{
    BigInteger r = x[index];
    index++;
    if (index > x.Count)
    {
        index = 0;
    }
    return r;
}

public List<double> getList()
{
    List<double> list = new List<double>();
    for (BigInteger i = 0; i < m; i++)
    {
        list.Add((double)x[(int)i]);
    }
    return list;
}

private BigInteger calcC(List<BigInteger> prime, List<BigInteger> factor)
{

    List<BigInteger> l = new List<BigInteger>();

    l.AddRange(prime);
    for(int i = 0; i < factor.Count; i++)
    {
        l.Remove(factor[i]);
    }

    if(l.Count > 0)
    {
        return l[new Random(seed).Next(0, l.Count)];
    }

    return 1;
}

private List<BigInteger> factors(List<BigInteger> prime)
{
    List<BigInteger> f = new List<BigInteger>();
    foreach(BigInteger p in prime)
    {
```

```
            if(m % p == 0)
            {
                f.Add(p);
            }
        }
        return f;
    }

    private List<BigInteger> primes()
    {
        List<BigInteger> p = new List<BigInteger>();
        p = Prime.getPrimes(m);
        p.Add(1); //додати 1, потрібне якщо m == prime

        return p;

    }


}
}
```

Додаток Д

SeedRNG.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Numerics;

namespace Steganography
{
    public class SeedRNG
    {
        private int seed { get; set; }
        private List<int> obtained;
        private int limit { get; set; }
        private Random r;
        private List<int> exp = new List<int>();
        private LCG lcg;
        public SeedRNG(int seed, int limit, bool b = false)
        {
            this.seed = seed;
            this.limit = limit;
            lcg = new LCG(limit, new Random(seed).Next(0, limit),seed);
            obtained = new List<int>();
            r = new Random(seed);
            if (b)
            {
                exp = Enumerable.Range(0, limit).ToList();
            }
        }
```

```
    public int Next
    {
        get
        {
            return (int)lcg.next();
        }
    }

    public int NextN
    {
        get
        {
            int x = 0;
            int n = r.Next(0, exp.Count);
            x = exp[n];
            exp.RemoveAt(n);
            return x;
        }
    }
}
public class SeedURNG
{
    private uint seed { get; set; }
    private List<uint> obtained;
    private uint limit { get; set; }
    private Random r;
    private List<uint> exp = new List<uint>();
    public SeedURNG(uint seed, uint limit, bool b = false)
    {
        this.seed = seed;
        this.limit = limit;
        obtained = new List<uint>();
        r = new Random((int)seed);
        if (b)
        {
            for (uint i = 0; i < limit; i++)
            {
                exp.Add(i);
            }
        }
    }

    public uint Next
    {
        get
        {
            uint x = 0;
            do
            {
                x = (uint)r.Next(0, (int)limit);
            } while (obtained.Contains(x));
            obtained.Add(x);
            return x;
        }
    }

    public uint NextN
    {
        get
        {
            uint x = 0;
            int n = r.Next(0, exp.Count);
```

```
            x = exp[n];
            exp.RemoveAt(n);
            return x;
        }
    }
}

}
```

Prime.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Numerics;
using System.IO;

namespace Steganography
{
   public static class Prime
   {
      private static List<BigInteger> primeList = new List<BigInteger>();
      private static BigInteger index = 0;
      private static Thread thread;
      private static Thread loadThread;
      private static readonly string file = "primes.bin";
      private static StringBuilder sbuilder = new StringBuilder();
      private static int intent = 0;
      private static readonly int maxIntent = 300;
      private static BigInteger max = 1000000;


      public static void Initialize()
      {
         primeList = new List<BigInteger>();

         loadThread = new Thread(InitializeUsingFiles);
         loadThread.Start();
      }

      private static void startGenerator()
      {
         thread = new Thread(generate);
         thread.Start();
      }

      private static void InitializeUsingFiles()
      {
         if (!File.Exists(file))
         {
            primeList.Add(2);
            primeList.Add(3);
            index = 5;
            File.WriteAllText(file, "2\r\n3\r\n");
            startGenerator();
         }
         else
         {
            foreach (string s in File.ReadAllLines(file))
            {
               BigInteger b = 0;
               if (BigInteger.TryParse(s.TrimEnd('\r','\n'), out b))
               {
                  primeList.Add(b);
               }else
               {
                  primeList.Clear();
```

```
            primeList.Add(2);
            primeList.Add(3);
            index = 5;
            startGenerator();
            return;
        }
    }
    index = primeList[primeList.Count - 1];
    startGenerator();
  }
}

private static void generate()
{
  for(BigInteger i = index; i < max; i+=2)
  {
    bool e = true;
    if (i % 2 == 0 || i % 3 == 0)
    {
      e = false;
    }
    for (BigInteger n = 5; n*n <= i;n+=6)
    {
      if(i % n == 0 || i % (n+2) == 0)
      {
        e = false;
      }
    }
    if (e)
    {
      lock (primeList)
      {
        primeList.Add(i);
        sbuilder.Append(i.ToString() + "\r\n");
        intent++;
        if (intent > maxIntent)
        {
          try
          {
            File.AppendAllText(file, sbuilder.ToString());
            intent = 0;
            sbuilder.Clear();
            sbuilder = new StringBuilder();
          }
          catch (Exception ex)
          {
            Console.WriteLine(ex.ToString());
          }
        }
      }
    }
    index+=2;
  }
}

public static List<BigInteger> getPrimes(BigInteger m)
{
  if (max < m)
  {
    IncreaseMax(m);
  }
  while (index < m)
```

```
        {
            OutputConsole.Write($"Поточний індекс:{index}, потрібний:{m}");
            Thread.Sleep(1000);
        }
        List<BigInteger> list = new List<BigInteger>();
        lock (primeList)
        {
            list = primeList.Where(n => n <= m).ToList();
        }

        return list;
    }

    public static void IncreaseMax(BigInteger nmax)
    {
        max = nmax;
        if (thread != null)
        {
            if (!thread.IsAlive)
            {
                thread = new Thread(generate);
                thread.Start();
            }
        }
    }

    public static void Finish()
    {
        try
        {
            if (thread != null)
            {
                thread.Abort();
            }
        }
        catch (Exception e)
        {

        }
        try
        {
            if (loadThread != null)
            {
                loadThread.Abort();
            }
        }
        catch (Exception e)
        {

        }
        try
        {
            if (!string.IsNullOrWhiteSpace(sbuilder.ToString()))
            {
                File.AppendAllText(file, sbuilder.ToString());
            }
        }catch(Exception e)
        {

        }
    }
}
```

```
}
```

Додаток Є

HiddenFile.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Steganography
{
   public class HiddenFile
   {
      public string filename { get; set; }
      public byte[] file { get; set; }
      public int size { get; set; }

      public HiddenFile(byte[] file, string filename)
      {
         this.file = file;
         this.filename = filename;
         this.size = file.Length;
      }

      public void cipherFile(int seed)
      {
         byte[] newFile = CipherFile.cipherFile(file, seed);
         file = newFile;
      }
   }
}
```

Додаток Ж

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Steganography
{
   static class Program
   {
      /// <summary>
      /// Основна точка вступу для програми.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Prime.Initialize();
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
         Prime.Finish();
      }
   }
}
```

OutputConsole.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;

namespace Steganography
{
   public static class OutputConsole
   {
      private static ListBox list;

      public static void Bind(ListBox listbox)
      {
         list = listbox;
      }

      public static void Show()
      {
         if (list != null)
         {
            list.Visible = true;
         }
      }

      public static void Hide()
      {
         if (list != null)
         {
            list.Visible = false;
         }
      }

      public static void Write(string text)
      {
         if (list != null)
         {
            list.Items.Add(text);
            if (list.Items.Count > (list.Height / list.ItemHeight))
            {
               list.Items.RemoveAt(0);
            }
            list.SelectedIndex = list.Items.Count - 1;
         }
      }

      public static void Clear()
      {
         if (list != null)
         {
            list.Items.Clear();
         }
      }

   }
```

```
}
```

Додаток 3

FileSizeFormatProvider.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Steganography
{
   public class FileSizeFormatProvider : IFormatProvider, ICustomFormatter
   {
      public static string GetFileSize(long bytes)
      {
         return String.Format(new FileSizeFormatProvider(), "{0:fs}", bytes);
      }

      public object GetFormat(Type formatType)
      {
         if (formatType == typeof(ICustomFormatter)) return this;
         return null;
      }

      private const string fileSizeFormat = "fs";
      private const Decimal OneKiloByte = 1024M;
      private const Decimal OneMegaByte = OneKiloByte * 1024M;
      private const Decimal OneGigaByte = OneMegaByte * 1024M;

      public string Format(string format, object arg, IFormatProvider formatProvider)
      {
         if (format == null || !format.StartsWith(fileSizeFormat))
         {
            return defaultFormat(format, arg, formatProvider);
         }

         if (arg is string)
         {
            return defaultFormat(format, arg, formatProvider);
         }

         Decimal size;

         try
         {
            size = Convert.ToDecimal(arg);
         }
         catch (InvalidCastException)
         {
            return defaultFormat(format, arg, formatProvider);
         }

         string suffix;
         if (size > OneGigaByte)
         {
            size /= OneGigaByte;
```

```
            suffix = "GB";
        }
        else if (size > OneMegaByte)
        {
            size /= OneMegaByte;
            suffix = "MB";
        }
        else if (size > OneKiloByte)
        {
            size /= OneKiloByte;
            suffix = "KB";
        }
        else
        {
            suffix = "B";
        }

        string precision = format.Substring(2);
        if (String.IsNullOrEmpty(precision)) precision = "2";
        return String.Format("{0:N" + precision + "} {1}", size, suffix);

    }

    private static string defaultFormat(string format, object arg, IFormatProvider formatProvider)
    {
        IFormattable formattableArg = arg as IFormattable;
        if (formattableArg != null)
        {
            return formattableArg.ToString(format, formatProvider);
        }
        return arg.ToString();
    }

  }
}
```

Додаток И

AudioSteganography.cs

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;


namespace Steganography

{

  public static class AudioSteganography

  {

    public static byte[] EncryptText(byte[] wav, string text)

    {

      WavAudio audio = new WavAudio(wav);
```

```
        uint value = 0;
        string pass = string.Format(audio.bitsPerSample.ToString());
        AESEncrypt encrypt = new AESEncrypt();
        string encrypted = encrypt.EncryptString(text, pass);
        OutputConsole.Write(string.Format("Текст зашифровано \n{0}", encrypted));
        if (encrypted.Length <= Math.Floor((double)(audio.totalSamples / 8)))
        {
            SeedURNG generator = new SeedURNG(audio.totalSamples, audio.totalSamples);
            OutputConsole.Write("Код згенеровано");
            OutputConsole.Write("Обробка WAV файлу...");
            for (int i = 0; i < encrypted.Length; i++)
            {
                value = encrypted[i];
                for (int x = 0; x < 8; x++)
                {
                    uint sample = generator.Next;
                    uint sampleValue = audio.samples[sample];
                    sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
                    audio.samples[sample] = sampleValue;
                }

            }
            value = 0;
            for (int x = 0; x < 8; x++)
            {
                uint sample = generator.Next;
                uint sampleValue = audio.samples[sample];
                sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
                audio.samples[sample] = sampleValue;
            }
            audio.Save();
            OutputConsole.Write(string.Format("Текст зашифровано... використано {0} аудіо зразків", encrypted.Length
* 8));
            OutputConsole.Write("Збереження WAV файлу");
            return audio.data;
        }
```

```
    else
    {
        return null;
    }

}
public static byte[] EncryptTextLinear(byte[] wav, string text)
{
    WavAudio audio = new WavAudio(wav);
    uint value = 0;
    string pass = string.Format(audio.bitsPerSample.ToString());
    AESEncrypt encrypt = new AESEncrypt();
    string encrypted = encrypt.EncryptString(text, pass);
    OutputConsole.Write(string.Format("Текст зашифровано \n{0}", encrypted));
    if (encrypted.Length <= Math.Floor((double)(audio.totalSamples / 8)))
    {
        uint n = 0;
        OutputConsole.Write("Код згенеровано");
        OutputConsole.Write("Обробка WAV файлу...");
        for (int i = 0; i < encrypted.Length; i++)
        {
            value = encrypted[i];
            for (int x = 0; x < 8; x++)
            {
                uint sample = n;
                uint sampleValue = audio.samples[sample];
                sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
                audio.samples[sample] = sampleValue;
                n++;
            }

        }
        value = 0;
        for (int x = 0; x < 8; x++)
        {
```

```
            uint sample = n;
            uint sampleValue = audio.samples[sample];
            sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
            audio.samples[sample] = sampleValue;
            n++;
          }
          audio.Save();
          OutputConsole.Write(string.Format("Текст зашифровано... використано {0} аудіо зразків", encrypted.Length
* 8));
          OutputConsole.Write("Збереження WAV файлу");
          return audio.data;
        }
        else
        {
          return null;
        }

      }

      public static string DecryptText(byte[] wav)
      {
        WavAudio audio = new WavAudio(wav);
        string text = string.Empty;
        SeedURNG generator = new SeedURNG(audio.totalSamples, audio.totalSamples);
        uint value = 0;
        string pass = string.Format(audio.bitsPerSample.ToString());
        AESEncrypt encrypt = new AESEncrypt();
        OutputConsole.Write("Обробка WAV файлу...");
        do
        {
          value = 0;
          for (int x = 0; x < 8; x++)
          {
            uint sample = generator.Next;
            uint sampleValue = audio.samples[sample];
            value = value | ((sampleValue & 1) << x);
```

```
        }
        if (value != 0)
            text += Convert.ToChar(value);
    } while (value != 0);
    OutputConsole.Write("Розшифрування тексту...");
    try
    {
        return encrypt.DecryptString(text, pass);
    }
    catch (Exception e)
    {
        OutputConsole.Write("Помилка: текст не знайдено");
        Console.WriteLine(e.Message);
        return null;
    }
}

public static string DecryptTextLinear(byte[] wav)
{
    WavAudio audio = new WavAudio(wav);
    string text = string.Empty;
    uint n = 0;
    uint value = 0;
    string pass = string.Format(audio.bitsPerSample.ToString());
    AESEncrypt encrypt = new AESEncrypt();
    OutputConsole.Write("Обробка WAV файлу...");
    do
    {
        value = 0;
        for (int x = 0; x < 8; x++)
        {
            uint sample = n;
            uint sampleValue = audio.samples[sample];
            value = value | ((sampleValue & 1) << x);
            n++;
```

```
        }
        if (value != 0)
            text += Convert.ToChar(value);
    } while (value != 0);
    OutputConsole.Write("Розшифрування тексту...");
    try
    {
        return encrypt.DecryptString(text, pass);
    }
    catch (Exception e)
    {
        OutputConsole.Write("Помилка: текст не знайдено");
        Console.WriteLine(e.Message);
        return null;
    }
}

public static byte[] EncryptFile(byte[] wav, byte[] file, string filename)
{
    WavAudio audio = new WavAudio(wav);
    uint value = 0;
    int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
    HiddenFile f = new HiddenFile(file, filename);
    OutputConsole.Write(string.Format("Розмір файлу: {0} байт", file.Length));
    f.cipherFile((int)audio.totalSamples);
    if (file.Length <= Math.Floor((double)(audio.totalSamples / 8)) - extraBytes)
    {
        SeedURNG generator = new SeedURNG(audio.totalSamples, audio.totalSamples);
        OutputConsole.Write("Код згенеровано");
        OutputConsole.Write("Шифрування файлу");
        OutputConsole.Write("Обробка WAV файлу...");
        OutputConsole.Write("Написання метаданих ...");
        //Написати розмір файлу
        for (int i = 0; i < file.Length.ToString().Length; i++)
        {
```

```
    value = file.Length.ToString()[i];
    for (int x = 0; x < 8; x++)
    {
        uint sample = generator.Next;
        uint sampleValue = audio.samples[sample];
        sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
        audio.samples[sample] = sampleValue;
    }

}
value = '#';
for (int x = 0; x < 8; x++)
{
    uint sample = generator.Next;
    uint sampleValue = audio.samples[sample];
    sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
    audio.samples[sample] = sampleValue;
}
//Написати ім'я файлу
for (int i = 0; i < filename.Length; i++)
{
    value = filename[i];
    for (int x = 0; x < 8; x++)
    {
        uint sample = generator.Next;
        uint sampleValue = audio.samples[sample];
        sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
        audio.samples[sample] = sampleValue;
    }

}
value = 0;
for (int x = 0; x < 8; x++)
{
    uint sample = generator.Next;
```

```
            uint sampleValue = audio.samples[sample];
            sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
            audio.samples[sample] = sampleValue;
        }
        //Написати вміст файлу
        OutputConsole.Write("Запис даних файлу...");
        for (int i = 0; i < file.Length; i++)
        {
            value = f.file[i];
            for (int x = 0; x < 8; x++)
            {
                uint sample = generator.Next;
                uint sampleValue = audio.samples[sample];
                sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
                audio.samples[sample] = sampleValue;
            }

        }

    }
    else
    {
        OutputConsole.Write("Помилка");
        return null;
    }
    OutputConsole.Write("Вбудовування файлу закінчено");
    OutputConsole.Write(string.Format("Використано {0} аудіо зразків", (file.Length + extraBytes) * 8));
    audio.Save();
    return audio.data;
}

public static byte[] EncryptFileLinear(byte[] wav, byte[] file, string filename)
{
    WavAudio audio = new WavAudio(wav);
    uint value = 0;
```

```
int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
HiddenFile f = new HiddenFile(file, filename);
OutputConsole.Write(string.Format("Розмір файлу: {0} байт", file.Length));
f.cipherFile((int)audio.totalSamples);
if (file.Length <= Math.Floor((double)(audio.totalSamples / 8)) - extraBytes)
{
   uint n = 0;
   OutputConsole.Write("Шифрування файлу");
   OutputConsole.Write("Обробка WAV файлу...");
   OutputConsole.Write("Написання метаданих ...");
   //Написати розмір файлу
   for (int i = 0; i < file.Length.ToString().Length; i++)
   {
      value = file.Length.ToString()[i];
      for (int x = 0; x < 8; x++)
      {
         uint sample = n;
         uint sampleValue = audio.samples[sample];
         sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
         audio.samples[sample] = sampleValue;
         n++;
      }

   }
   value = '#';
   for (int x = 0; x < 8; x++)
   {
      uint sample = n;
      uint sampleValue = audio.samples[sample];
      sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
      audio.samples[sample] = sampleValue;
      n++;
   }
   //Написати ім'я файлу
   for (int i = 0; i < filename.Length; i++)
```

```
{
    value = filename[i];
    for (int x = 0; x < 8; x++)
    {
        uint sample = n;
        uint sampleValue = audio.samples[sample];
        sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
        audio.samples[sample] = sampleValue;
        n++;
    }

}
value = 0;
for (int x = 0; x < 8; x++)
{
    uint sample = n;
    uint sampleValue = audio.samples[sample];
    sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
    audio.samples[sample] = sampleValue;
    n++;
}
//Написати вміст файлу
OutputConsole.Write("Запис даних файлу...");
for (int i = 0; i < file.Length; i++)
{
    value = f.file[i];
    for (int x = 0; x < 8; x++)
    {
        uint sample = n;
        uint sampleValue = audio.samples[sample];
        sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
        audio.samples[sample] = sampleValue;
        n++;
    }
```

```
        }
    }
    else
    {
        OutputConsole.Write("Помилка");
        return null;
    }
    OutputConsole.Write("Вбудовування файлу завершено");
    OutputConsole.Write(string.Format("Використано {0} аудіо зразків", (file.Length + extraBytes) * 8));
    audio.Save();
    return audio.data;
}

public static byte[] EncryptFile2(byte[] wav, byte[] file, string filename)
{
    WavAudio audio = new WavAudio(wav);
    uint value = 0;
    int extraBytes = 2 + filename.Length + file.Length.ToString().Length;
    HiddenFile f = new HiddenFile(file, filename);
    OutputConsole.Write(string.Format("Розмір файлу: {0} байт", file.Length));
    f.cipherFile((int)audio.totalSamples);
    if (file.Length <= Math.Floor((double)(audio.totalSamples / 8)) - extraBytes)
    {
        SeedURNG generator = new SeedURNG(audio.totalSamples, audio.totalSamples, true);
        OutputConsole.Write("Код згенеровано");
        OutputConsole.Write("Шифрування файлу");
        OutputConsole.Write("Обробка WAV файлу...");
        OutputConsole.Write("Написання метаданих...");
        //Write file size
        for (int i = 0; i < file.Length.ToString().Length; i++)
        {
            value = file.Length.ToString()[i];
            for (int x = 0; x < 8; x++)
            {
                uint sample = generator.NextN;
```

```
        uint sampleValue = audio.samples[sample];
        sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
        audio.samples[sample] = sampleValue;
    }

}
value = '#';
for (int x = 0; x < 8; x++)
{
    uint sample = generator.NextN;
    uint sampleValue = audio.samples[sample];
    sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
    audio.samples[sample] = sampleValue;
}
//Написати ім'я файлу
for (int i = 0; i < filename.Length; i++)
{
    value = filename[i];
    for (int x = 0; x < 8; x++)
    {
        uint sample = generator.NextN;
        uint sampleValue = audio.samples[sample];
        sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
        audio.samples[sample] = sampleValue;
    }

}
value = 0;
for (int x = 0; x < 8; x++)
{
    uint sample = generator.NextN;
    uint sampleValue = audio.samples[sample];
    sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
    audio.samples[sample] = sampleValue;
}
```

```
        //Написати вміст файлу
        OutputConsole.Write("Запис даних файлу...");
        for (int i = 0; i < file.Length; i++)
        {
          value = f.file[i];
          for (int x = 0; x < 8; x++)
          {
            uint sample = generator.NextN;
            uint sampleValue = audio.samples[sample];
            sampleValue = (sampleValue & 0xFFFFFFFE) | ((value >> x) & 1);
            audio.samples[sample] = sampleValue;
          }

        }

      }
      else
      {
        OutputConsole.Write("Помилка");
        return null;
      }
      OutputConsole.Write("Вбудовування файлу завершено");
      OutputConsole.Write(string.Format("Використано {0} аудіо зразків", (file.Length + extraBytes) * 8));
      audio.Save();
      return audio.data;
    }

    public static HiddenFile DecryptFile(byte[] wav)
    {
      try
      {
        WavAudio audio = new WavAudio(wav);
        string text = string.Empty;
        SeedURNG generator = new SeedURNG(audio.totalSamples, audio.totalSamples);
        OutputConsole.Write("Код згенеровано");
```

```
OutputConsole.Write("Обробка WAV файлу...");
OutputConsole.Write("Читання метаданих...");
uint value = 0;
do
{
   value = 0;
   for (int x = 0; x < 8; x++)
   {
      uint sample = generator.Next;
      uint sampleValue = audio.samples[sample];
      value = value | ((sampleValue & 1) << x);
   }
   if (value != '#')
      text += Convert.ToChar(value);
} while (value != '#' && char.IsNumber((char)value));
int filesize = int.Parse(text);
OutputConsole.Write(string.Format("Розмір вилученого файлу: {0} байт", filesize));
text = string.Empty;
do
{
   value = 0;
   for (int x = 0; x < 8; x++)
   {
      uint sample = generator.Next;
      uint sampleValue = audio.samples[sample];
      value = value | ((sampleValue & 1) << x);
   }
   if (value != 0)
      text += Convert.ToChar(value);
} while (value != 0);
string filename = text;
OutputConsole.Write(string.Format("Ім'я вилученого файлу: {0}", filename));
byte[] file = new byte[filesize];
for (int i = 0; i < filesize; i++)
{
```

```
            value = 0;
            for (int x = 0; x < 8; x++)
            {
               uint sample = generator.Next;
               uint sampleValue = audio.samples[sample];
               value = value | ((sampleValue & 1) << x);
            }
            file[i] = (byte)value;
         }
         OutputConsole.Write(string.Format("Вміст файлу витягнуто"));
         HiddenFile f = new HiddenFile(file, filename);
         OutputConsole.Write("Шифрування файлу...");
         f.cipherFile((int)audio.totalSamples);
         return f;
      }
      catch (Exception e)
      {
         return null;
      }
   }

   public static HiddenFile DecryptFileLinear(byte[] wav)
   {
      try
      {
         WavAudio audio = new WavAudio(wav);
         string text = string.Empty;
         uint n = 0;
         OutputConsole.Write("Обробка WAV файлу...");
         OutputConsole.Write("Читання метаданих...");
         uint value = 0;
         do
         {
            value = 0;
            for (int x = 0; x < 8; x++)
```

```
    {
        uint sample = n;
        uint sampleValue = audio.samples[sample];
        value = value | ((sampleValue & 1) << x);
        n++;
    }
    if (value != '#')
        text += Convert.ToChar(value);
} while (value != '#' && char.IsNumber((char)value));
int filesize = int.Parse(text);
OutputConsole.Write(string.Format("Розмір вилученого файлу: {0} байт", filesize));
text = string.Empty;
do
{
    value = 0;
    for (int x = 0; x < 8; x++)
    {
        uint sample = n;
        uint sampleValue = audio.samples[sample];
        value = value | ((sampleValue & 1) << x);
        n++;
    }
    if (value != 0)
        text += Convert.ToChar(value);
} while (value != 0);
string filename = text;
OutputConsole.Write(string.Format("Ім'я вилученого файлу: {0}", filename));
byte[] file = new byte[filesize];
for (int i = 0; i < filesize; i++)
{
    value = 0;
    for (int x = 0; x < 8; x++)
    {
        uint sample = n;
        uint sampleValue = audio.samples[sample];
```

```
            value = value | ((sampleValue & 1) << x);
            n++;
          }
          file[i] = (byte)value;
        }
        OutputConsole.Write(string.Format("Вміст файлу витягнуто"));
        HiddenFile f = new HiddenFile(file, filename);
        OutputConsole.Write("Шифрування файлу...");
        f.cipherFile((int)audio.totalSamples);
        return f;
      }
      catch (Exception e)
      {
        return null;
      }
    }

    public static HiddenFile DecryptFile2(byte[] wav)
    {
      try
      {
        WavAudio audio = new WavAudio(wav);
        string text = string.Empty;
        SeedURNG generator = new SeedURNG(audio.totalSamples, audio.totalSamples, true);
        OutputConsole.Write("Код згенеровано");
        OutputConsole.Write("Обробка WAV файлу...");
        OutputConsole.Write("Читання метаданих...");
        uint value = 0;
        do
        {
          value = 0;
          for (int x = 0; x < 8; x++)
          {
            uint sample = generator.NextN;
            uint sampleValue = audio.samples[sample];
```

```
        value = value | ((sampleValue & 1) << x);
    }
    if (value != '#')
        text += Convert.ToChar(value);
} while (value != '#' && char.IsNumber((char)value));
int filesize = int.Parse(text);
OutputConsole.Write(string.Format("Розмір вилученого файлу: {0} байт", filesize));
text = string.Empty;
do
{
    value = 0;
    for (int x = 0; x < 8; x++)
    {
        uint sample = generator.NextN;
        uint sampleValue = audio.samples[sample];
        value = value | ((sampleValue & 1) << x);
    }
    if (value != 0)
        text += Convert.ToChar(value);
} while (value != 0);
string filename = text;
OutputConsole.Write(string.Format("Ім'я вилученого файлу: {0}", filename));
byte[] file = new byte[filesize];
for (int i = 0; i < filesize; i++)
{
    value = 0;
    for (int x = 0; x < 8; x++)
    {
        uint sample = generator.NextN;
        uint sampleValue = audio.samples[sample];
        value = value | ((sampleValue & 1) << x);
    }
    file[i] = (byte)value;
}
OutputConsole.Write(string.Format("Вміст файлу витягнуто"));
```

```
            HiddenFile f = new HiddenFile(file, filename);
            OutputConsole.Write("Шифрування файлу...");
            f.cipherFile((int)audio.totalSamples);
            return f;
        }
        catch (Exception e)
        {
            return null;
        }
    }
}
}
```