math_2023_058

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
кафедра математичного моделювання

Використання методів шифрування, для передачі текстової інформації засобами .NET використовуючи мультимедіа

Кваліфікаційна робота
Рівень вищої освіти – перша(бакалаврський)

Виконав:
Студент 4 курсу, 407 групи
Пузенко Євген Володимирович
Керівник:
Доцент, канд. фіз.-мат. наук
Горбатенко Микола Юрійович

До захисту допущено на засіданні кафедри
протокол № 16 від 30 травня 2023 р
Зав. кафедрою_____ проф. Черевко І.М.

Чернівці
2023

Анотація

Метою даної роботи є визначення додаткових шляхів безпечного зберігання інформації для подальшого її транспортування.

Розроблене програмне забезпечення дає змогу додатково посилити можливості захисту передачі конфіденційної інформації каналами зв’язку.

Галузі використання результатів даного дослідження: криптографія, комп’ютерна графіка.

Кількість ілюстрацій – 52.

Кількість використаних джерел – 8.

Кількість додатків – 1.

Обсяг роботи - 47 сторінок.

Ключові слова: Криптографія, WPF, .NET, C#, Комп’ютерна графіка.

Annotation

The goal of this work is to identify additional ways to securely store information for its further transportation.

The developed software makes it possible to further enhance the ability to protect the transmission of confidential information through communication channels.

Areas of application of the results of this research: cryptography, computer graphics.

The number of illustrations - 52.

Number of sources used - 8.

Number of applications - 1.

The volume of work - 47 pages.

Keywords: Cryptography, WPF, .NET, C#, Computer graphics.

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

_____ Є. В. Пузенко

(підпис)

ЗМІСТ

Вступ

Кваліфікаційна робота на тему «Використання методів шифрування для передачі текстової інформації засобами .NET, використовуючи мультимедіа», є для мене цікавим та необхідним проектом, який дозволить мені поглибити знання у галузях криптографії, комп'ютерної графіки та архітектури програмних застосунків.

Від початку масового використання інтернету, користувачі генерують величезну кількість інформації. Спочатку це був тільки текст, потім до тексту додалися фото, відео та інші види даних. Якщо раніше основу контенту в інтернеті складав текст, то сьогодні це відео файли різних розмірів.

У рамках кваліфікаційної роботи було досліджено можливість зберігання текстових даних всередині медіа контейнерів, а саме: зображень та відео файлів. В результаті отримано готовий застосунок, що дозволяє зберігати у фото відео файлах текстову інформацію та, за потреби, отримувати її з файлів. Під час виконання цієї роботи було досліджено наступні теми:

- Опис технологій, та методів створення додатків – розглянуто обрану мову програмування, напрям, шаблони та технології для виконання обраної теми.
- Опис архітектури проекту – розглянуто створення компонентів програми, такі як інтерфейси, сервіси, моделі, представлення.
- Алгоритми роботи з фото відео файлами – розглянуто створені алгоритми для вставлення та отримання даних з медіа контейнерів, таких як зображення та відео.
- Аналіз отриманих результатів – розглянуто отримані результати після створення програмного засобу.

Розділ 1 – Опис технологій та методів створення додатків

1.1 Мова програмування C#

Мова програмування C# з'явилась в 2001 році завдяки роботі Андерса Гейлберга, Скота Вільтамута та Пітера Гольде, що працювали в компанії Microsoft. Від свого початку і до сьогодні має велику та активну спільноту розробників, що підтримують та розвивають мову, та її ресурси. За час свого існування спільнота розробила багато цікавих та унікальних підходів до розробки програмного забезпечення. C# розроблялася як об'єктно орієнтована мова, тому всі програми в цій мові будуються навколо об'єктів, які регулюють їх поведінку та властивості. Дану мову використовують для програмної, веб розробки, а також для створення ігор, синтаксис мови схожий на синтаксис C++, та Java.

Для створення об'єктів в C# використовують класи, які використовуються у якості шаблонів для створення об'єктів з однаковими властивостями і поведінкою.

Для взаємодії між об'єктами використовуються методи, це такий вид повідомлень, які дозволяють корегувати поведінку об'єктів.

Оскільки C# є об'єктно-орієнтованою мовою, то в ній реалізовані принципи цього напрямку програмування, а саме: наслідування, абстракція, поліморфізм та інкапсуляція.

Наслідування – це концепція, що дозволяє створювати ієрархію класів з певними спільними характеристиками та розширювати функціонал класів.

Абстракція – це концепція, що дозволяє виділити спільні характеристики об'єктів та дозволяє ігнорувати їхні незначні деталі фокусуючись тільки на важливих. У мові C# абстракція досягається за допомогою абстрактних класів та спільних для об'єктів інтерфейсів.

Поліморфізм – це здатність різних об'єктів виконувати ті самі дії що і інші об'єкти, але залежно від їхньої реалізації. У C# поліморфізм можна зустріти в віртуальних методах, та під час використання наслідування.

Інкапсуляція – це концепція, що дозволяє об’єднувати різні компоненти, будь то, методи або дані, в один клас і приховувати реалізацію класу від користувачів. В мові C# інкапсуляція реалізована можливістю керувати областю видимості, тобто, public, protected, private та інші.

В основному C# використовується при розробці для платформи .NET. Також, мова C# підтримує асинхронне програмування, для цього в мові існують спеціальні конструкції async та await.

1.2 Платформа користувацького інтерфейсу WPF

WPF (Windows Presentation Foundation) – розроблена Microsoft технологія для створення користувацьких інтерфейсів засобами C#. Перша версія технології була випущена у 2006 році. WPF не залежить від розмірів екрану, оскільки, використовує векторний механізм візуалізації.

В основі WPF лежить мова розмітки XAML, за допомогою якої і відбувається побудова інтерфейсів. Саме завдяки XAML розробники можуть описати структуру та вигляд інтерфейсів. Розмітка XAML схожа до розмітки HTML, тому це спрощує створення та підтримку додатків. Також, через схожість XAML до HTML можна легко створювати комплексні інтерфейси групуючи окремі елементи у групи.

Оскільки XAML це мова розмітки, то вона допомагає розробникам відокремити інтерфейси користувача від бізнес логіки додатку. Також це спрощує повторне використання коду. Розділення логіки та інтерфейсу користувача дозволяє розділити роботу та зосередитись кожному учаснику групи на певній області. Також це розділення дозволяє створювати модульні додатки які легко масштабувати забираючи або додаючи певні елементи, або цілі шари.

Серед доступних елементів керування є тексти, списки, кнопки, таблиці, меню, дерева та багато інших. За допомогою цих вбудованих стандартних елементів можна легко шаблонізувати поведінку, та вигляд додатку. Кожен елемент керування має може включати в себе шрифти, кольори, розміри, стани

видимості та інші властивості. Також кожен елемент керування можна розширювати змінюючи їх функціонал.

WPF підтримує роботу з векторною, та растровою графікою і дозволяє створювати різноманітні візуальні елементи.

Від часу створення технології спільнота створила багато різноманітних стандартів для роботи з WPF, проте найбільш вживаними стали MVVM.

WPF підтримує можливість реагувати на події, такі як використання мишки, натискання кнопок, використання клавіатури, закриття, та відкриття вікон.

1.3 MVVM

MVVM(Model – View - ViewModel)(рис. 1.1) – шаблон для розробки, що масово використовується при роботі з WPF. Даний шаблон використовується для розділення бізнес логіки з дизайном та даними.

Під даними розуміють частину Model де представлені всі дані над якими буде виконуватись якась робота. Компонент модель може включати в себе доступ до бази, обробку даних та інші операції. Тобто Model це дані додатку, які існують незалежно від його візуального представлення.

View це представлення даних. View включає в себе вигляд та розташування елементів, а також стилі та анімацію.

ViewModel – це проміжний шар, що дозволяє взаємодіяти з файлами підготовлювати їх до виводу, тощо. ViewModel існує для того щоб підготувати дані для передачі їх в View.

Завдяки цьому шаблону розробники можуть розділяти елементи розробки, та писати їх незалежно. Цей шаблон спрощує управління проектом, та полегшує тестування.

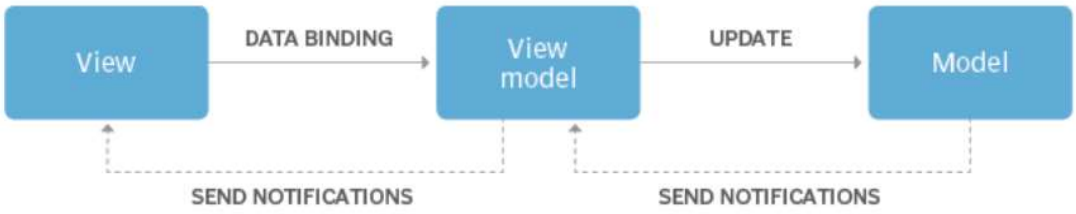

Рисунок 1.1 – загальний вигляд шаблону

1.4 Використання хеш-кодів

Хеш-код – це певне числове значення, яке може отримуватись після використання хеш-алгоритмів що займаються їх генеруванням. Хеш-функція, вона ж хеш-алгоритм, це такий алгоритм що займається перетворенням великого об’єму даних у малий.

Поняття хеш-код у розумінні програмістів з’явилось у 1956 році завдяки Арнольду Думі, він у своїй роботі: «Computers and automation» представив концепцію хешування такою, як якою її сьогодні розуміє більшість програмістів, а саме як залишок від ділення даних на якесь просте число.

Сьогодні хеш-коди використовуються усюди, починаючи від алгоритмів хешування, індексації, пошуку, та не закінчуючи порівнянням даних. Люди почали використовувати хеш-коди для того щоб прискорити дію різних алгоритмів, наприклад, алгоритмів пошуку даних у колекціях. Найбільш вживаними типами колекцій, що використовують хеш-коди є хеш-мапи, хеш-таблиці**.**

Хеш-коди часто використовують для реалізації різних структур даних, таких як множини, хеш-мапи, хеш-таблиці(рис. 1.2).

Оскільки для створення хеш-кодів беруться дані з об’єкту, то можна гарантувати, що 2 однакових об’єкта створюють однакові хеш-коди, проте ми не можемо гарантувати зворотне, що 2 різних об’єкти завжди будуть повертати тільки унікальні хеш-коди. Колізія хеш-кодів це їхній мінус. Колізія виникає тоді коли 2 різних об’єкти утворюють один і той самий хеш-код, а виникає вона

через обмежену розмірність хеш-коду (в C# це 32 біти). Розробники створили багато методів підвищення унікальності, такі як списки, або дерева обробки колізій, створення хеш-кодів, проте жоден з цих методів не гарантує відсутність колізії.

У мові C# хеш-коди реалізовані у вигляді методу GetHashCode(). У якості результату повертає число Int32, що і є хеш-кодом об'єкту. У користувачів є можливість самим реалізувати хеш-кодування використовуючи перезавантаження. В цій мові хеш-коди використовуються для швидкого порівняння об'єктів. Для цього існує спеціальний метод Equals(). В середині даного методу отримуються 2 хеш-коди об'єктів та порівнюються. На основі цих порівнянь повертається результат чи рівні об'єкти чи ні. Метод Equals() також можна перезавантажити.

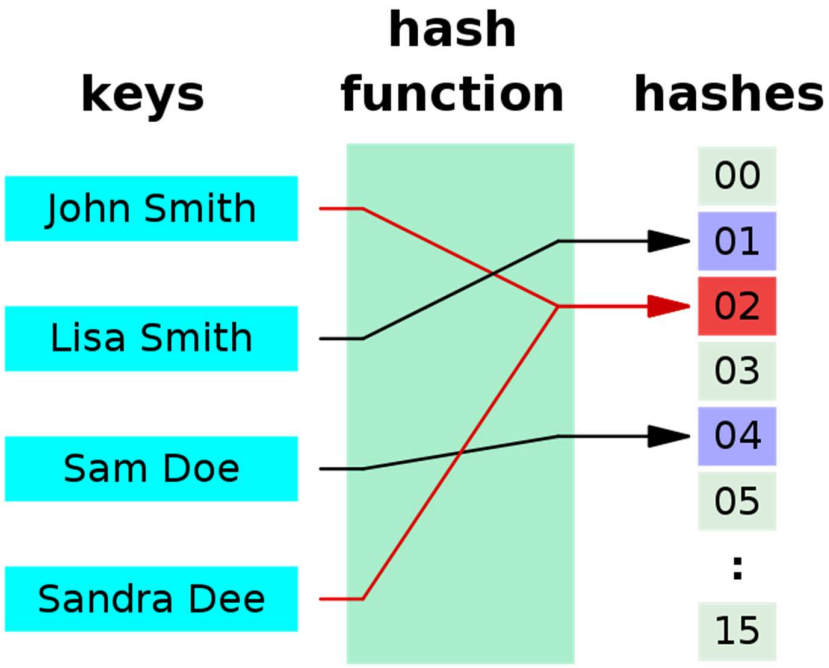

Рисунок 1.2 – приклад використання структури даних що використовує хеш функцію

1.5 Формат JSON

JSON – це формат обміну даних, що спочатку був розроблений для JavaScript, але потім почав використовуватись усюди. Повна назва JavaScript Object Notation. Сама мова з'явилась в 1999 році завдяки Дугласу Крокфорду та його команді. Він же був тою людиною що популяризувала її.

Цей формат є зрозумілим для людей, саме тому JSON використовується для передачі інформації між сервером та клієнтом, а також може використовуватись у конфігураційних файлах. В JSON дані представлені у вигляді пар ключ-значення, подібно до того як представлені об'єкти в JavaScript. В JSON можуть бути представлені дані у вигляді рядків, чисел, масивів, логічних перемінних, та об'єктів. Формат JSON підтримують безліч мов програмування в тому числі і C#. У JSON для оголошення об'єктів використовують {}, а для оголошення масивів []. JSON підтримує вкладення(рис. 1.3), тобто, один об'єкт може мати в собі інший об'єкт, або масив, а той ще інший. Дані в JSON легко зчитувати і для цього створено велику кількість бібліотек, або вбудованих в саму мову функцій. Цей формат підтримує кодування Unicode, що дозволяє використовувати велику кількість символів з різних мов, та загалом підтримує велику кількість мов.

```json
{
    "count": 7,
    "items": ["socks", "pants", "shirts", "hats"],
    "manufacturer": {
        "name": "Molly's Seamstress Shop",
        "id": 39233,
        "location": {
            "address": "123 Pickleton Dr.",
            "city": "Tucson",
            "state": "AZ",
            "zip": 85705
        }
    },
    "total_price": "$393.23",
    "purchase_date": "2022-05-30",
    "country": "USA"
}
```

Рисунок 1.3 – Приклад вигляду даних у форматі json

JSON використовується усюди, в тому числі, і в C#. В C# від використовується для обміну даними між різними компонентами програмного забезпечення або між сервісами. Для роботи з JSON в C# використовують різні

бібліотеки. Хоча існує і вбудована бібліотека System.Text.Json проте, найбільш популярною є Newtonsoft.Json. Дана бібліотека є найбільш популярною, оскільки надає можливості для серіалізації, та десеріалізації. Завдяки цій бібліотеці можна легко перетворювати об’єкти на рядки, та навпаки. У C# можна можливо створювати класи, які відповідають структурі JSON і які можуть використовувати бібліотеки JSON для автоматичного перетворення даних між об’єктами в C# та JSON.

1.6 Шифрування

Шифрування – це процес перетворення звичайних даних, зрозумілих для людини, або системи в криптографічно захищені дані. Людство використовує ті чи інші методи шифрування на протязі всієї своєї історії. Одним з найвідоміших серед стародавніх методів шифрування був шифр Цезаря. У основу всіх шифрів полягає сам алгоритм шифрування та ключ, використавши який можна розшифрувати або зашифрувати дані. На протязі всіх часів були ті хто хотів захистити інформацію, та були ті хто хотіли дізнатися те що було зашифровано. Існування таких людей створило науку криптографію що займається і по цей день методами захисту інформації.

Шифрування використовується для забезпечення конфіденційності даних під час передачі або збереження. Шифрування базується на використанні алгоритмів які змінюють структуру або вміст даних таким чином, що вони без ключу стають незрозумілими.

Якщо раніше дані які шифрувались були зображені на папері, то з появою перших радіопередавачів людство адаптувало алгоритми до нових методів передачі даних. Те саме відбулось з алгоритмами при появі комп’ютерів.

Сьогодні як і раніше використовуються різні алгоритми шифрування як для безпечного зберігання, так і для безпечного транспортування. Сьогодні шифрування використовується в основному для захисту особистих даних, банківських транзакцій, паролів та іншої чутливої інформації. Для цього

використовують сильні математичні функції та ключі задля отримання високої надійності і стійкості шифрування.

Шифрування буває як симетричним, так і асиметричним. Симетричне шифрування це таке шифрування де як для процесу шифрування так і для дешифрування використовується один і той самий ключ.

Асиметричне шифрування це таке шифрування де для шифрування використовується один ключ, а для дешифрування інший. Їх ще називають приватний, та публічні ключі. За допомогою приватного відбувається дешифрування, а за допомогою публічного шифрування.

Якщо зрівнювати в загальному ці 2 види шифрування, то симетричне є більше швидким, та ефективнішим для шифрування великих за об'ємом даних. Тоді як асиметричне є більш безпечним, та надає можливість використовувати публічний ключ для безпечного обміну даними.

Хоч зараз і багато існує алгоритмів шифрування, проте найбільш вживаними серед симетричних вважають AES, RC5, HC-256 та RC4. Серед асиметричних: RSA, DSA.

1.6.1 AES

AES (Advanced Encryption Standard) – це симетричний алгоритм шифрування, що був розроблений в 2001 році Д. Деймоном та В. Ріджменом який використовується для захисту даних. Цей алгоритм є одним із найбезпечнішим, що досі є криптографічно стійким. З 2002 року був прийнятий як один із міжнародних стандартів для шифрування даних.

Алгоритм AES базується на перестановці байтів даних з використання ключа шифрування. AES підтримує ключі розміру 128, 196 та 256 байтів. Це розбиття на довжини ключів дозволяє використовувати різні рівні безпеки для шифрування даних.

В мові C# для AES використовується бібліотека System.Security.Cryptography, а саме клас Aes. В даному класі є 2 властивості та

3 методи без використання яких неможливо працювати з класом. Key – ключ який необхідний для шифрування\дешифрування та IV – вектор, випадкове значення яке потім буде використовуватись для шифрування даних. Метод Create() – статичний метод класу, що створює новий екземпляр класу Aes, він потрібний для шифрування та дешифрування. Метод Encrypt() – метод в який в якості параметру передається набір даних, та на виході отримується зашифрований набір даних. Метод Decrypt() – метод що відповідає за дешифрування даних, у якості параметру приймає набір зашифрованих даних, та повертає набір розшифрованих даних.

1.7 Зображення

Зображення – візуальне відображення сцени, об’єкта, або інформації зафіксоване на певному носію даних. В програмуванні зображення представлене у вигляді масиву пікселів де кожна точка зображення містить дані про колір та яскравість; у вигляді набору формул, що потім перетворюються у геометричні фігури.

Зображення сьогодні широко використовуються у всіх напрямках програмування, машинне навчання, веб, ігрова розробка, виведення інформації. Основна ціль зображень сьогодні це візуалізація даних, що допомагають краще розуміти та аналізувати дані. Зображення широко використовують для розробки користувацьких графічних інтерфейсів, які дозволяють взаємодіяти користувачам з інтерфейсом через мишу та клавіатуру. Дуже часто зображення використовують для відображення графіків даних, це спрощує їх аналіз людиною.

Оскільки, в останній час популярність нейромереж зросла, то неможливо не розказати про OCR – візуальне розпізнавання даних у зображені.

Сьогодні зображення представлені у декількох популярних форматах: JPEG, JPG, PNG, SVG, проте всі вони беруть початок від формату BMP. У кожного формату є свої плюси і мінуси, такі як у JPEG та JPG хороша оптимізація розміру, у PNG хороша передача кольорів, у SVG можливість

задати зображення за допомогою математичних формул, тобто малий розмір файлу.

1.8 Відео файли

Відео файл це послідовність зображень, які розташовані один за одним і при відтворені утворюють рухоме зображення. Є формати відео файлів де кожне зображення представляє собою окреме зображення такі як AVI, а є формати відео які зтискають розмір відео використовуючи різні методи, такі як карти кольорів, збереження тільки унікальної інформації, використання алгоритмів стискання, комбіновані методи стискання, які можуть включати в себе як всі так і частину зазначених методів. Всі ці методи використовуються у кодеках. Один кодек може підтримувати декілька форматів відео, проте, як правило, 1 кодек працює лише з 1 розширенням.

Найбільш вживаними форматами відео-файлів є MP4, AVI, MKV. Більшість форматів підтримують контейнеризацію. Контейнеризація це процес об’єднання відео і аудіо доріжок у 1 файл, завдяки цьому відео може містити в собі одночасно субтитри, звук, відео, метадані.

Сьогодні більшість форматів відео підтримує потоки, що дозволяє запускати відео не завантажуючи його повністю, це дозволяє переглядати відео в режимі реального часу з мінімальними затримками.

При роботі з відео часто використовують поняття роздільна здатність. Роздільна здатність – це кількість пікселів в 1 кадрі, якість та чіткість зображення. Загальновизнані роздільні здатності включають в себе HD(1280x720p), Full HD(1920x1080p) та 4K UHD(3840x2160p).

Розділ 2 – Опис програми

2.1 Опис можливостей програми

Дана програма була розроблена для дослідження методів шифрування для передачі текстової інформації засобами .NET використовуючи мультимедіа. Завдяки цій програмі користувач може зашифрувати текстові дані а потім вставити їх у картинку з метою подальшого збереження, та розповсюдження. Користувач може дістати з зображення зашифровані текстові дані та розшифрувавши засобами програми отримати повідомлення. Користувач може попередньо зашифрувавши інформацію помістити її в відео-файл. Після цього передати файл з інформацією іншому користувачу щоб той міг отримати повідомлення після того як дістане, та розшифрує інформацію.

В програмі передбачено різні перевірки, такі як: перевірки вводу, перевірки наявності достатнього місця для запису інформації в контейнер(зображення або відео), перевірка цілісності переданої інформації.

Оскільки, для збереження користувацької інформації використовуються пікселі, то будь яка зміна задіяних пікселів спричинить втрату даних і як результат хеш-коди при порівнянні будуть різні.

2.2 Архітектури додатку

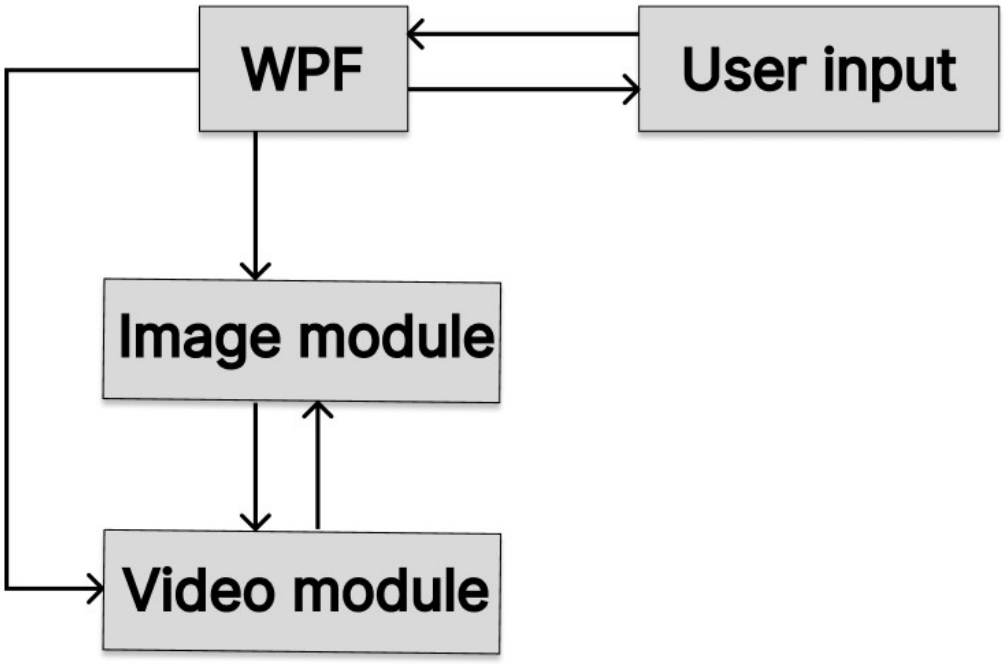

Рисунок 2.1 - Модель архітектури додатку

Під час створення додатку було розроблено її архітектуру(рис. 2.1). В даній архітектурі користувач використовуючи WPF взаємодіє з 2 модулями. Модуль Image при взаємодії з яким користувач може вставити свої дані у зображення, або дістати їх звідти і модуль Video який дозволяє зберігати і отримувати користувацькі дані з відео. У користувача є можливість звернутися до кожного модулю окремо. Між модулями Image і Video є зв'язок, оскільки, під час використання модуля Video використовується модуль Image для зміни пікселів у кадрах відео.

2.3 Опис інтерфейсів

Були створені загальні інтерфейси, що дозволили зробити код більш абстрактним, та незалежним. Були створені наступні інтерфейси:

```
7 references
public interface IARGBImage
{
    3 references
    int Width();
    3 references
    int Height();
    2 references
    Bitmap GetBitmap();
    1 reference
    void SetBitmap(Bitmap bitmap);
    1 reference
    void SetPixel(int x, int y, Color color);
    1 reference
    Color GetPixel(int x, int y);
    2 references
    Bitmap GetCopy();

}
```

Рисунок 2.2 - Вигляд загального інтерфейсу для роботи з зображеннями

Інтерфейс для роботи з зображеннями(рис. 2.2), містить мінімально необхідний набір методів. Метод Height повертає висоту зображення. Метод Width() повертає ширину зображення. Метод GetBitmap повертає збережене в тілі класу зображення. Завдяки методу SetBitmap можна змінити зображення в середині класу. Методи SetPixel та GetPixel необхідні для змінення та отримання даних про колір в зображенні. Метод GetCopy повертає копію розміщеного в середині класу зображення.

```
1 reference
internal interface IImageEmbending
{
    2 references
    Bitmap Embending(IARGBImage image,byte[] data);
    2 references
    byte[] UnEmbending(IARGBImage image);

}
```

Рисунок 2.3 – Вигляд інтерфейсу необхідного для процесу вставлення даних у зображення

Даний інтерфейс(рис. 2.3) потрібен для вставки, та діставання даних в\з зображення. Де метод Embedding відповідає за вставку даних користувача в передане зображення, а метод UnEmbedding відповідає за діставання даних з переданого зображення.

```
1 reference
public interface IEncryptionService
{
    3 references
    byte[] Encrypt(byte[] plainText);
    3 references
    byte[] Decrypt(byte[] cipherText);
    1 reference
    Encoding Encoding { get;}
    3 references
    int GenHashCode(string str);
    4 references
    int GenHashCode(byte[] bytes);
}
```

Рисунок 2.4 - Вигляд інтерфейсу необхідного для процесу шифрування користувацьких даних

Даний інтерфейс(рис. 2.4) необхідний для подальшого використання для шифрування. Даний інтерфейс реалізує мінімальний набір методів та властивостей для шифрування. В метод Encrypt() передається не зашифровані

дані, на виході отримуються зашифровані дані. В метод Decrypt() передаються зашифровані дані де вони розшифровується та повертаються вже в розшифрованому вигляді. Властивість Encoding це властивість в яку передається тип кодування символів. Далі 2 методи генерування хеш кодів. Один для тексту інший для байтів. Хеш код генерується на основі вмісту даних.

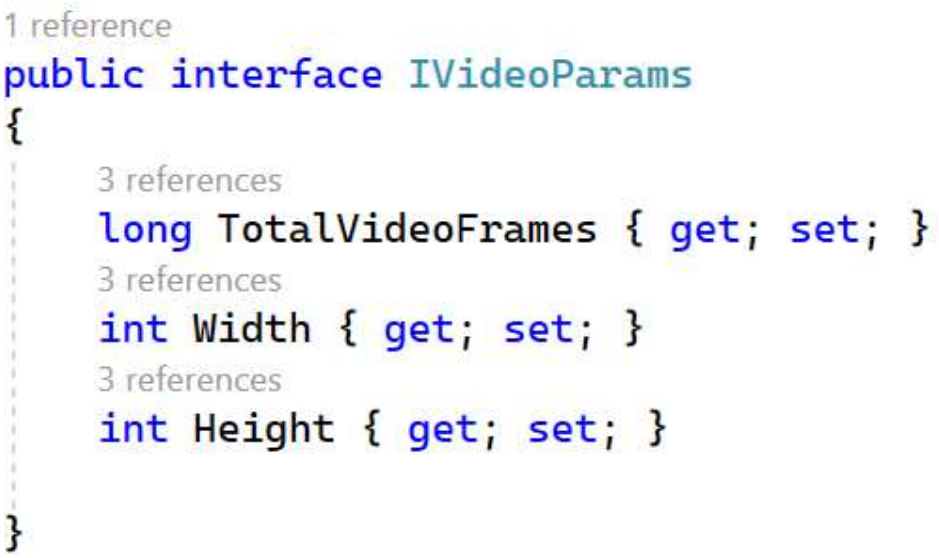

Рисунок 2.5 - Вигляд інтерфейсу необхідного для представлення параметрів відео

Даний інтерфейс(рис. 2.5) необхідний для базового представлення параметрів відео. Складається з 3 властивостей. Загальної кількості кадрів у відео, ширини 1 кадру і висоти 1 кадру.

2.4 Опис процесу шифрування користувацьких даних

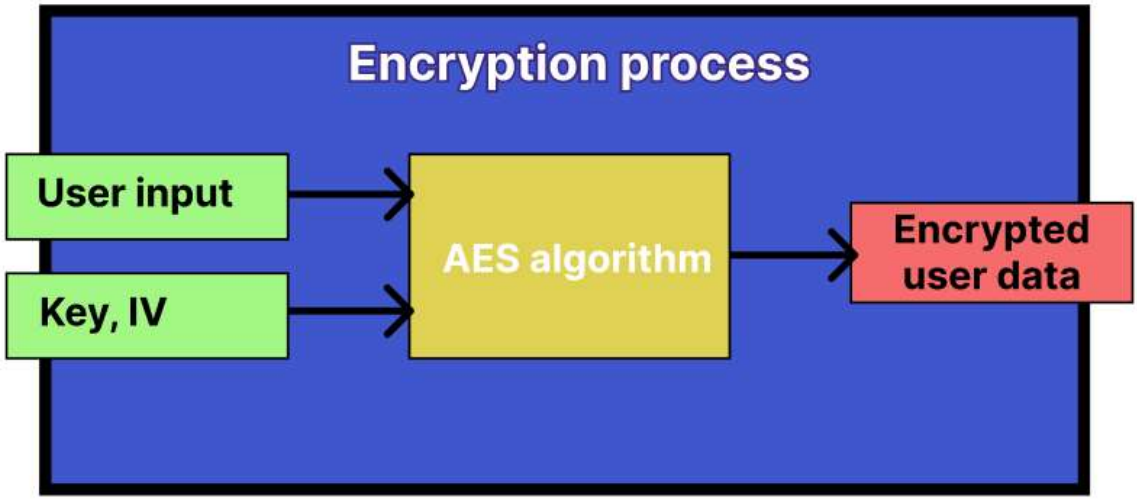

Рис 2.6 - Схема шифрування користувацьких даних

В даній програмі використано алгоритм AES для шифрування користувацьких даних(рис. 2.6). На вхід подаються користувацькі дані та Key і IV, після чого і користувацькі дані і ключі віддаються реалізованому в бібліотеці System.Security.Cryptography класу Aes. Потім, коли процес шифрування закінчено, то повертається зашифровані користувацькі дані.

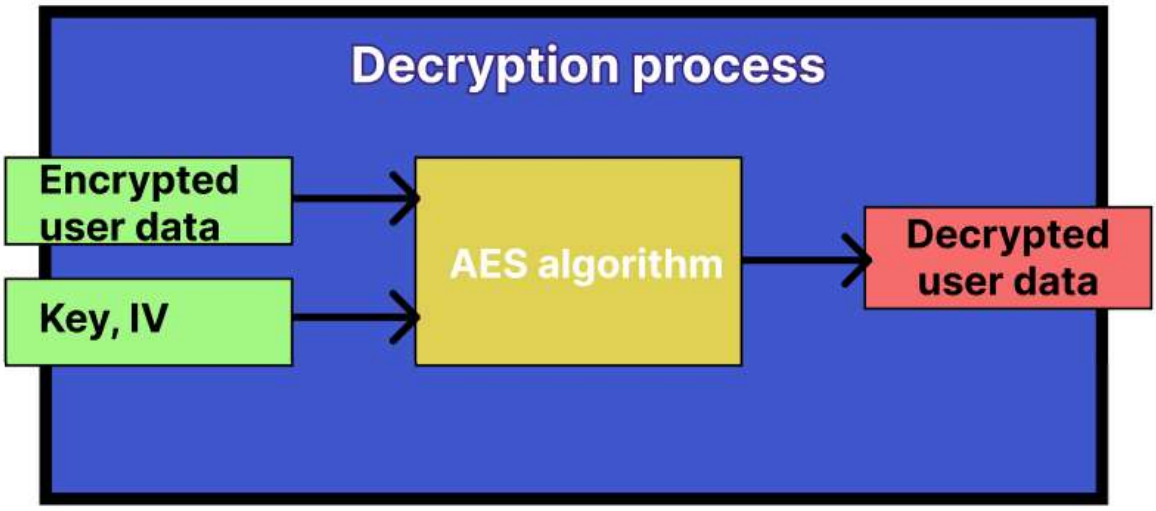

Рисунок 2.7 - Схема дешифрування користувацьких даних

Процес дешифрування аналогічний процесу шифрування. На вхід передаються зашифровані дані, та ключі. Потім використовуючи Aes отримуються користувацькі дешифровані дані(рис. 2.7).

2.5 Опис процесу отримання інформації закодованої в зображенні

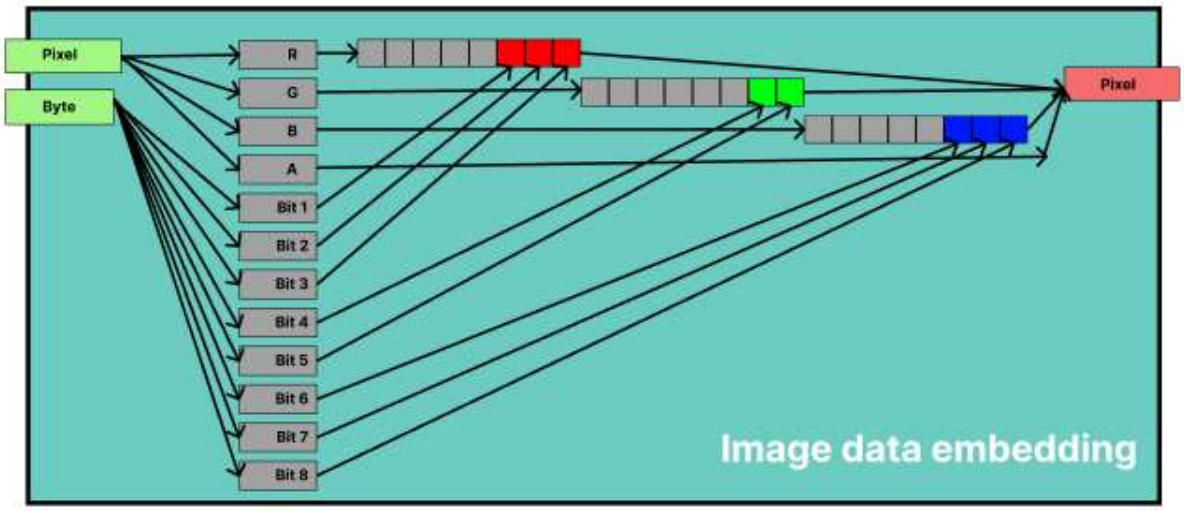

Рисунок 2.8 - Схема процесу вставлення користувацької інформації у піксель зображення

Процес вставлення байту даних в піксель зображення(рис. 2.8). На вході маємо піксель та байт даних. Для процесу вставлення потрібно спочатку розділити піксель на його зіставні частини – канали кольорів. Розділивши піксель на канали потрібно розділити кожен канал, окрім альфа каналу, він не приймає участь у процесі вставлення, тому його одразу записуємо до результуючого пікселю, на біти. Після цього розділяємо байт на біти.

Сам процес вставлення це процес коли у каналі частина бітів замінюється бітами інформації користувача. Замінюються тільки останні біти, оскільки, це майже не впливає на вихідний колір пікселя. В даному випадку у червоному каналі замінюються останні 3 біта на перші 3 біта інформації, у зеленому каналі змінюються останні 2 біти на 4 і 5 біти байта користувацької інформації. У

синьому каналі теж замінюємо 3 останні біти каналу на 3 останніх біти від байту користувацької інформації.

Коли процес зміни завершено, то створюється новий піксель зі зміненими каналами та вставленим користувацьким байтом.

Алгоритм повного процесу запису користувацьких даних у зображення(рис. 2.9):

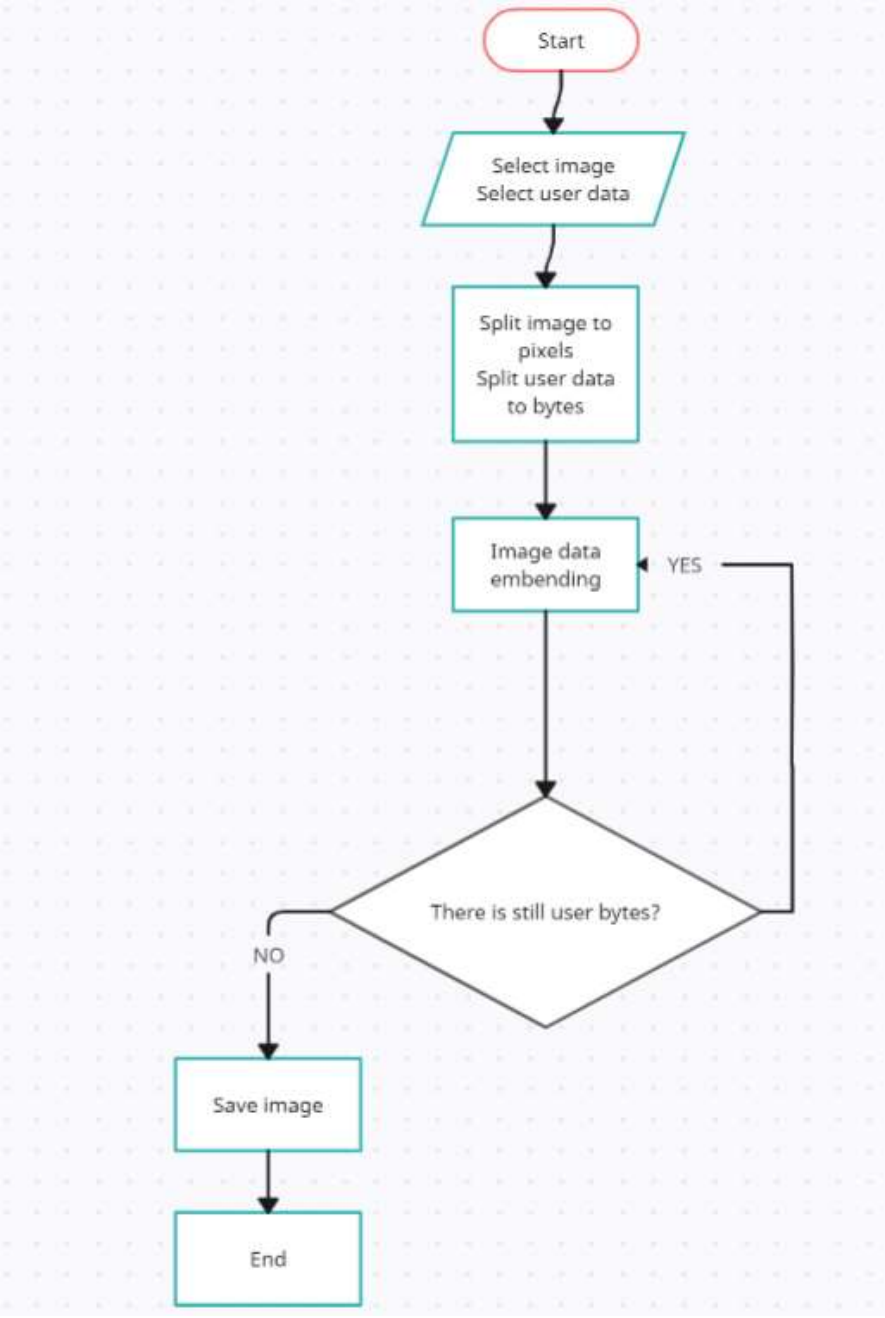

Рисунок 2.9 – Блок схема алгоритму вставлення користувацьких даних у зображення

2.6 Опис процесу діставання інформації з зображення

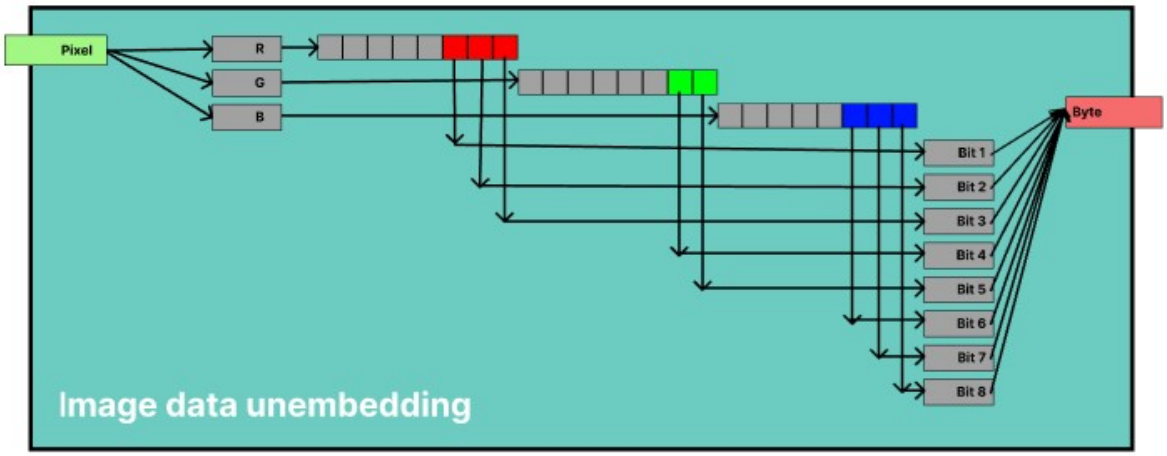

Рисунок 2.10 - Схема де наглядно показано процес отримання користувацької інформації з пікселя зображення

Процес отримання даних з пікселів (рис. 2.10) майже повністю протилежний процесу вставлення.

Спочатку отримуємо піксель зображення та розділяємо його на канали. Червоний, зелений і синій, канал альфа в даному випадку не потрібний, оскільки, ми його не використовували при вставленні. Для отримання користувацького байту ми спочатку беремо 3 останні біти байту червоного кольору та записуємо їх у перші 3 біти вихідного байту. Далі беремо 2 останніх біти байту зеленого кольору і записуємо у 4 і 5 біти користувацького байту. Нарешті беремо останні 3 біти синього каналу та записуємо їх в 3 останні біти користувацького байту.

Після того як ми сформували користувацький байт ми його повертаємо.

Алгоритм діставання та формування користувацьких даних з зображення(рис. 2.11):

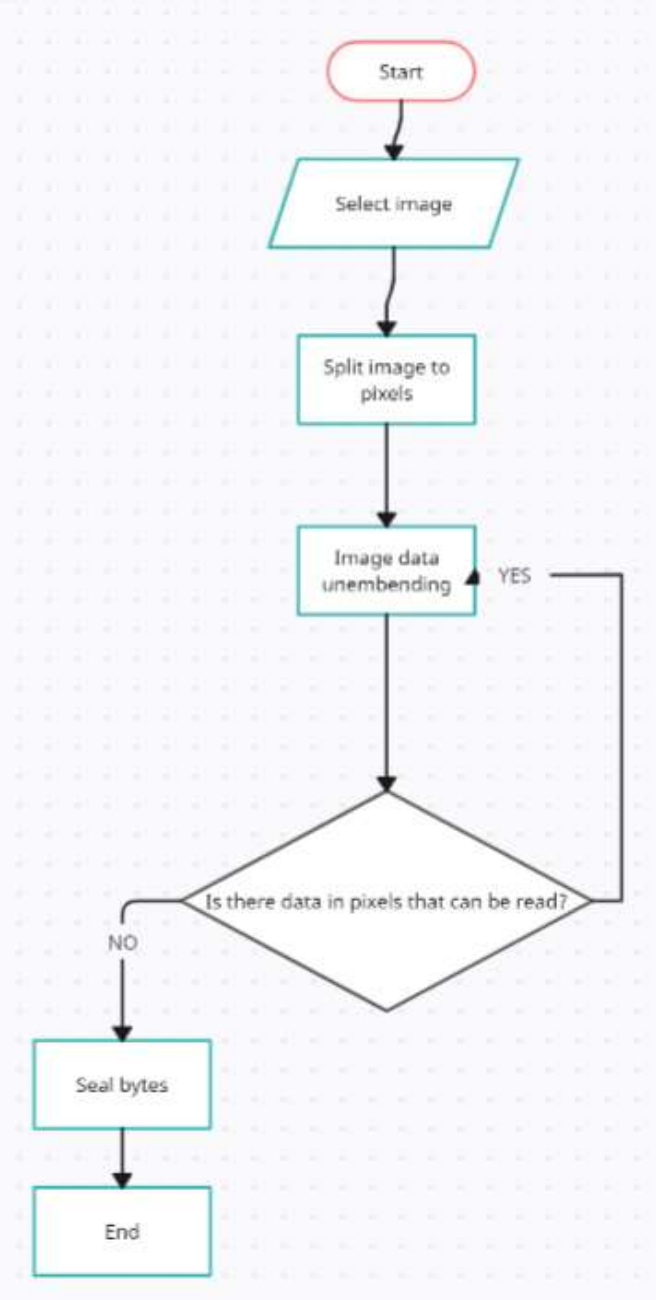

Рисунок 2.11 - Блок схема алгоритму діставання користувацької інформації з зображення

2.7 Опис процесу заміни кадрів у відео

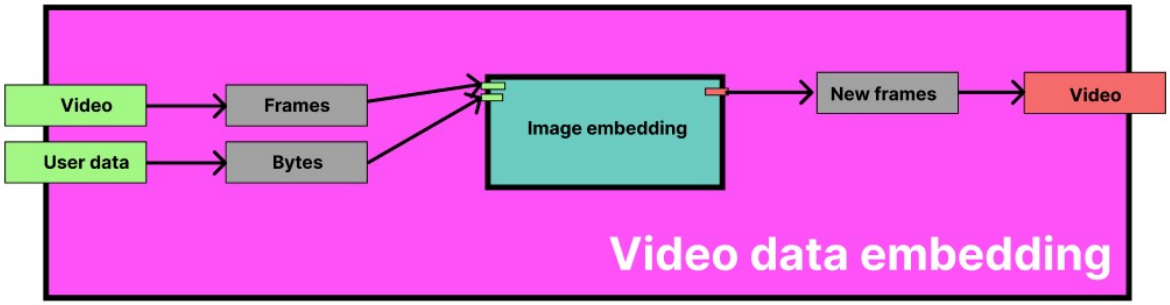

Рисунок 2.12 - Схема де показано процес вставлення користувацької інформації у відео файл

Процес запису користувацьких даних у відео(рис. 2.12) складається з декількох етапів. Спочатку отримуємо відео та користувацькі дані. Розділяємо відео на кадри, а користувацькі дані на байти. Після чого використовуючи вже створений модуль вставлення даних у зображення беремо кожен кадр та записуємо в кожен піксель кадру користувацький байт. І продовжуємо записувати поки не закінчиться користувацькі дані. Коли всі дані записані у кадри, то створюємо нове відео в яке спочатку додаються змінені кадри, а потім додаються залишкові не використані у вставленні даних кадри. Після чого повертаємо нове відео зі зміненими кадрами та вставленими у ці кадри користувацькі дані.

Алгоритм запису у відео користувацької інформації(рис. 2.13):

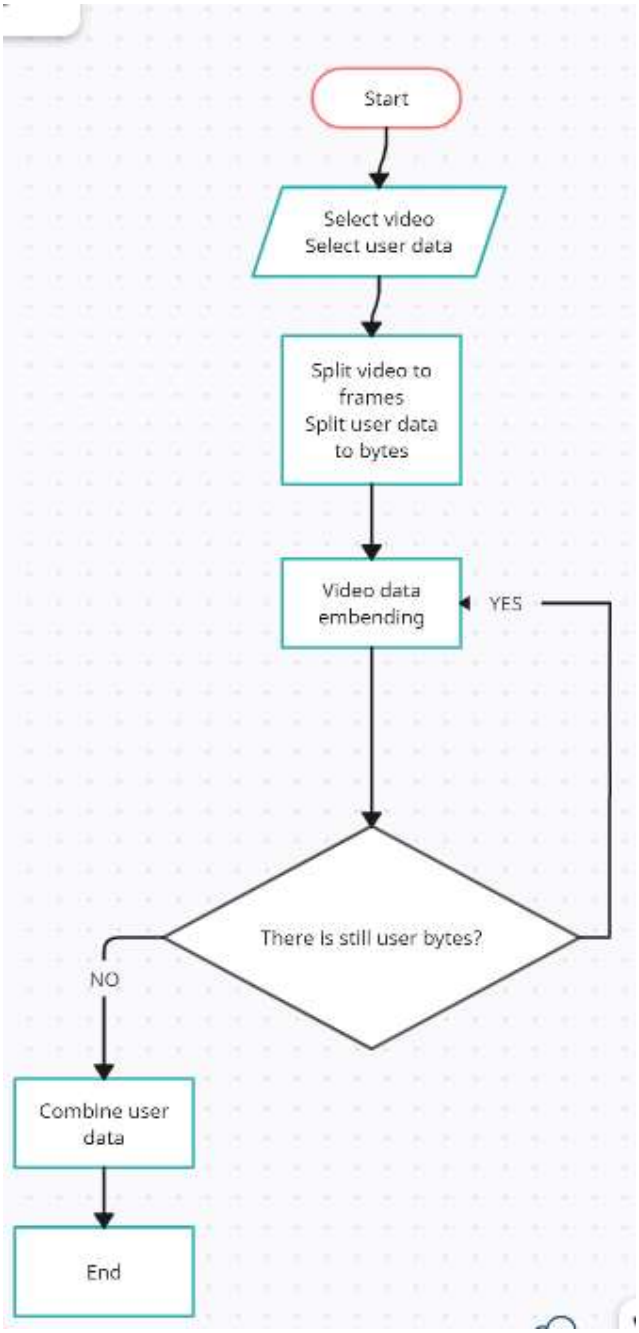

Рисунок 2.13 - Блок схема алгоритму вставлення даних у відео-файл

2.8 Опис процесу діставання користувацьких даних з відео

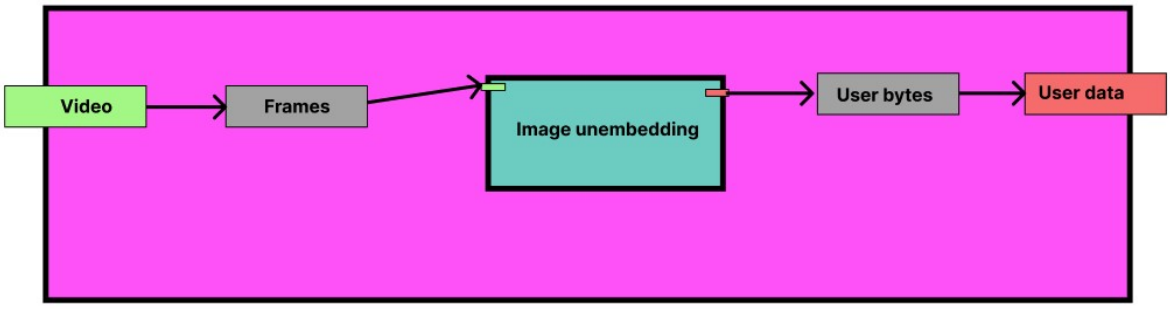

Рисунок 2.14 - Схема діставання користувацьких даних з відео

Процес отримання даних з користувацького відео(рис. 2.14) дуже схожий на процес вставлення. Спочатку ми отримуємо відео з даними в середині, потім розділяємо відео на зображення і пропускаємо кожне зображення до поки не дістанемо всі дані з відео. Після чого об’єднуємо дані та повертаємо.

Алгоритм діставання даних з відео(рис. 2.15):

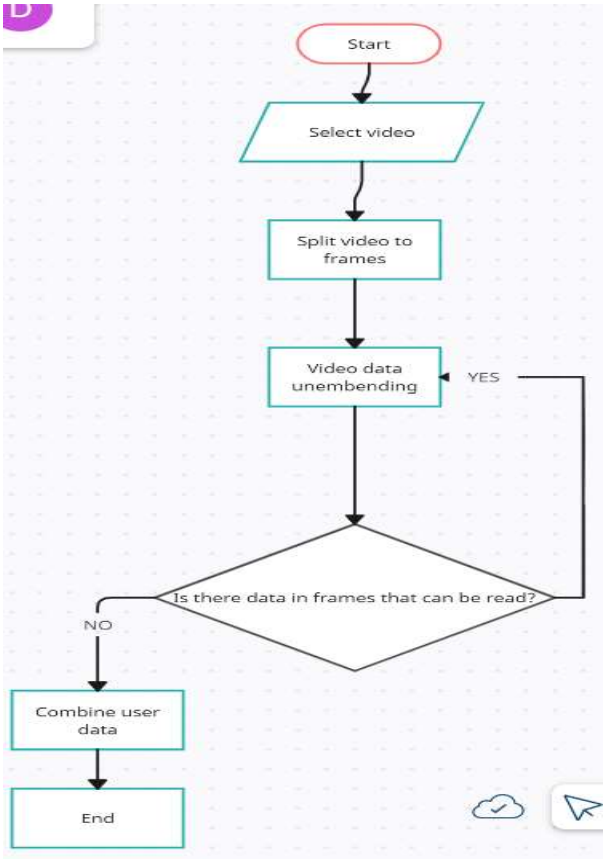

Рисунок 2.15 - Блок схема діставання користувацьких даних з відео

2.9 Використання бібліотеки Accord

Це велика бібліотека для різних задач. Починаючи від роботи з зображеннями і відео та закінчуючи машинним навчанням. В даному проекті був використаний модуль Accord.Video.FFMPEG для роботи з відео, а саме для зміни кадрів відео, та створення нових відео.

Були використані класи VideoFileReader і VideoFileWriter для редагування відео.

Клас VideoFileReader був використаний для отримання кадрів з відео. Для цього був використаний вбудований метод ReadVideoFrame. Завдяки цьому методу можливо прочитати відео покадрово та отримати кожен кадр окремо від інших.

Клас VideoFileWriter був використаний для створення нового відео з наявних, змінених, кадрів. Для цього потрібно виростати метод Open та

передати необхідні параметри, а саме назву файлу, розміри 1 кадру, кількість кадрів на секунду відео, відеокодек. В деяких версіях бібліотеки метод Open з параметром відеокодеку недоступний, проте для цього проекту метод з цим параметром мусить бути присутній. У версії 3.8.0 він наявний.

Для даного проекту був використаний VideoCodec.Raw, оскільки саме цей кодек не змінює пікселі при запіканні відео. Всі решта кодеків, завдяки алгоритмам стискання даних знищують користувацьку інформацію в пікселях, оскільки в процесі оптимізації змінюють кольори пікселів.

Дана бібліотека підтримує різні формати відео, та підтримує конвертацію одного типу відео в інший тип, завдяки цьому користувач може використовувати у додатку відео з різними розширеннями.

Після того як кадри були змінені, то запис нових кадрів у відео відбувається за використання методу WriteVideoFrame, який на вхід приймає кадр.

Коли нове відео сформовано, то потрібно використати метод Close який зліпить кадри у відео та збереже відео там де було вказано.

Розділ 3 – Опис технічної реалізації, та особливості

3.1 Опис моделей

```
public class ARGBImageModel : IARGBImage
{
    private Bitmap _bitmap;
    0 references
    public ARGBImageModel(int width, int height)...
    2 references
    public ARGBImageModel(Bitmap bitmap) ...
    2 references
    public Bitmap GetBitmap()...
    1 reference
    public Color GetPixel(int x, int y)...
    3 references
    public int Height()...
    1 reference
    public void SetPixel(int x, int y, Color color)...
    3 references
    public int Width()...
    1 reference
    public void SetBitmap(Bitmap bitmap)...
    3 references
    private void CheckData(int width, int height)...
    2 references
    private void CheckData(Bitmap bitmap)...
    2 references
    public Bitmap GetCopy()...
}
```

Рисунок 3.1 - Модель ARGBImageModel що реалізує загальний інтерфейс для зображень

Клас модель ARGBImageModel (рис. 3.1) імплементує інтерфейс IARGBImage та реалізує його. Для реалізації цього інтерфейсу створено приватну перемінну зображення _bitmap та створено конструктори та методи для перевірки даних. Є можливість створити новий бітмап передавши параметри ширини та висоти і є можливість використовувати раніше створений бітмап.

```
public class Aes256Model
{
    [JsonProperty("key")]
    2 references
    public byte[] Key { get; private set; }

    [JsonProperty("iv")]
    2 references
    public byte[] IV { get; private set; }

    [NonSerialized]
    public Encoding Encoding;

    4 references
    public static Aes256Model Load(string configFilePath)
    {
        using (StreamReader reader = new StreamReader(configFilePath))
        {
            string json = reader.ReadToEnd();
            Aes256Model aes = JsonConvert.DeserializeObject<Aes256Model>(json);
            if(aes == null)
            {
                throw new Exception("appconfig is empty!");
            }
            return aes;
        }
    }
}
```

Рисунок 3.2 - Клас модель Aes256Model необхідний для отримання включів для шифрування з файлу

Клас модель Aes256Model (рис. 3.2) реалізує в собі зчитування з конфігу даних для подальшого використання у кодуванні інформації, де Key це ключ необхідний для Aes та IV це випадкове значення що теж потрібне для підвищення стійкості алгоритму шифрування. Створення моделі відбувається завдяки статичному методу Load який приймає у якості параметру шлях до файлу конфігурацій, який за допомогою бібліотеки Newtonsoft.Json здійснює перетворення даних з файлу у модель класу.

```
3 references
public class FFmpegVideoModel : IVideoParams
{
    3 references
    public long TotalVideoFrames { get; set; }
    3 references
    public int Width { get; set; }
    3 references
    public int Height { get; set; }
}
```

Рисунок 3.3 - Клас модель що реалізує загальний інтерфейс з параметрами відео

Даний клас модель (рис. 3.3) реалізує інтерфейс IVideoParams та потрібний виключно для збереження параметрів даних відео. А саме розмір одного кадру, та загальна кількість кадрів у відео.

```
internal class ImageEncWindowModel
{
    public string InPath;
    public string OutPath;
    public Encoding Encoding;
    public string Data;
}
```

Рисунок 3.4 - Модель що зберігає дані для подальшого їх використання для вставлення користувацьких даних у зображення

Даний клас модель (рис. 3.4) зберігає в собі дані потрібні для роботи з view для вікна шифрування для зображень.

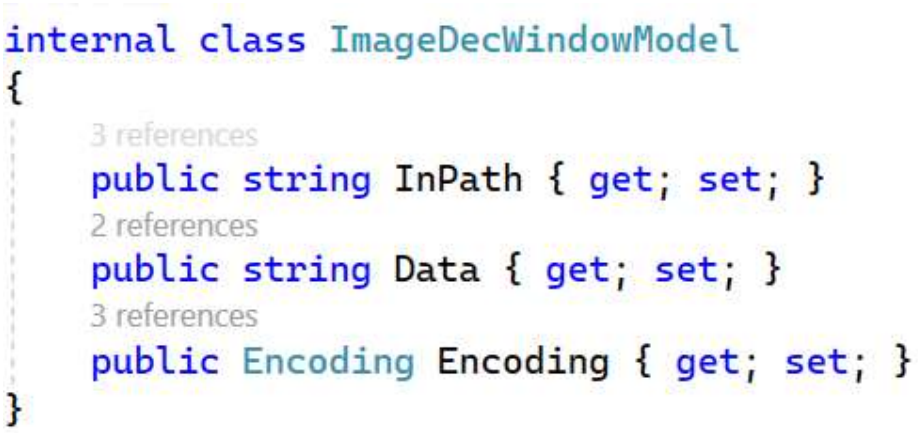

Рисунок 3.5 - Модель що зберігає дані для подальшого їх використання для отримання користувацьких даних з зображення

Даний клас модель (рис. 3.5) зберігає в собі дані потрібні для роботи з view для вікна дешифрування для зображень.

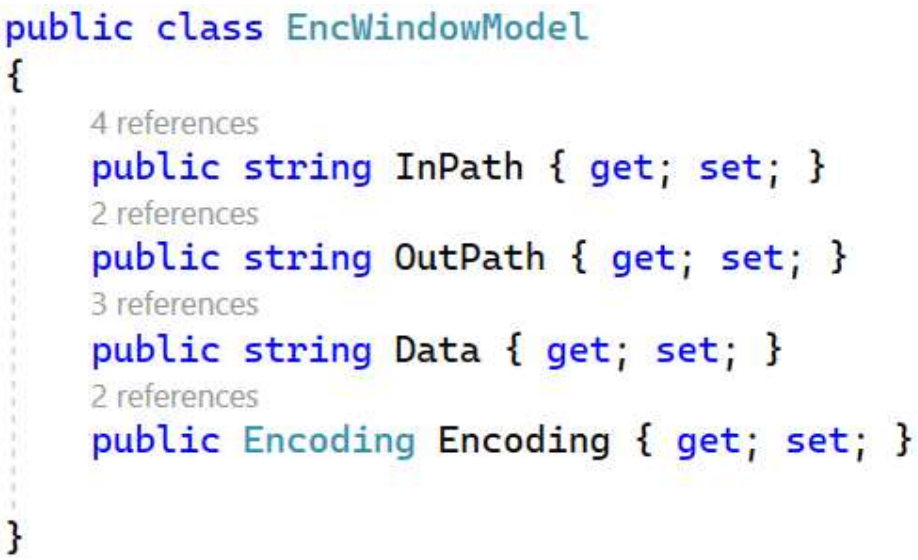

Рисунок 3.6 - Клас модель для відео необхідна для роботи з вікном вставлення даних у відео

Даний клас модель (рис. 3.6) зберігає в собі дані потрібні для роботи з view для вікна шифрування для відео.

```
internal class DecWindowModel
{

    public string InPath { get; set; }
    2 references
    public string Data { get; set; }
    3 references
    public Encoding Encoding { get; set; }
}
```

Рисунок 3.7 - Клас модель для відео необхідна для роботи з вікном отримання даних з відео

Даний клас модель (рис. 3.7) зберігає в собі дані потрібні для роботи з view для вікна дешифрування для відео.

```
public class MenuWindowModel
{
    28 references
    public Window MenuWindow { get; set; }
}
```

Рисунок 3.8 - Модель клас вікна меню

Даний клас модель (рис. 3.8) зберігає в собі дані потрібні для роботи з view для меню програми.

3.2 Опис сервісів

```
9 references
internal class ARGBImageEmbending : IImageEmbending
{
    static readonly byte[] stopBytes = { 0, 0 };// same to '\0'

    2 references
    public virtual Bitmap Embending(IARGBImage image, byte[] data)...

    2 references
    public virtual byte[] UnEmbending(IARGBImage image)...
```

Рисунок 3.9 - клас що відповідає за процес ставлення і дістування даних з фото

Даний сервіс(рис. 3.9) був створений для вставлення користувацьких даних у зображення. Він імплементує інтерфейс IImageEmbedding та реалізує 2

його методи. Embedding та UnEmbedding. Перший метод приймає у якості параметрів зображення та користувацькі дані та повертає змінене зображення з вставленою користувацькою інформацією. Другий метод займається екстракцією даних з зображення. На вхід приймає зображення з даними, на виході повертає зчитані дані з пікселів зображення. Також тут наявний статичний масив з байтами який вказує на те що далі зчитувати інформацію не потрібно, оскільки це кінець повідомлення. Детальна реалізація цих методів буде розглянута пізніше. Також в класі наявні методи, це допоміжні методи, та методи що дозволяють розбити алгоритм на більш малі частини для того щоб підвищити читабельність та зрозумілість алгоритму.

```
public class Aes256 : IEncryptionService
{
    private Aes256Model _model;

    4 references
    public Aes256(Aes256Model model)...

    1 reference
    public Encoding Encoding => _model.Encoding;

    3 references
    public byte[] Decrypt(byte[] cipherText)...

    3 references
    public byte[] Encrypt(byte[] plainText)...

    4 references
    public int GenHashCode(byte[] bytes)...

    3 references
    public int GenHashCode(string str)...
}
```

Рисунок 3.10 - клас що реалізує інтерфейс для шифрування даних користувача

Даний сервіс (рис. 3.10) реалізує інтерфейс IEncryptionService. Даний клас реалізує шифрування даних використовуючи метод шифрування Aes. В собі цей клас має модель Aes звідки бере ключі для шифрування та дешифрування. Деталі реалізації шифрування та дешифрування будуть написані далі.

```
3 references
public class FFmpegVideo_v2
{
    static readonly byte[] stopBytes = { 0, 0 };// same to '\0'

    1 reference
    public static void Embending(string inPath, string outPath, byte[] data)...

    0 references
    private static byte[] AddStopBytes(byte[] data)...

    1 reference
    public static byte[] UnEmbending(string path)...

    1 reference
    public static FFmpegVideoModel GetVideoParams(string inPath)...
}
```

Рисунок 3.11 - Клас що відповідає за вставлення і діставання даних для відео

Даний клас сервіс (рис. 3.11) був розроблений для вставки користувацьких даних у відео файли. 2 основних його методи це Embedding з параметрами шляхів та користувацьких даних для вставки користувацьких даних у відео, та UnEmbedding з параметрами шляху відео з якого потрібно дістати дані. У якості результату повертає отримані дані з відео. Також тут є статичний масив дані якого використовуються у якості байтів які означають кінець даних для читання. Реалізація цих методів буде описана в подальшому.

3.3 Опис представлень

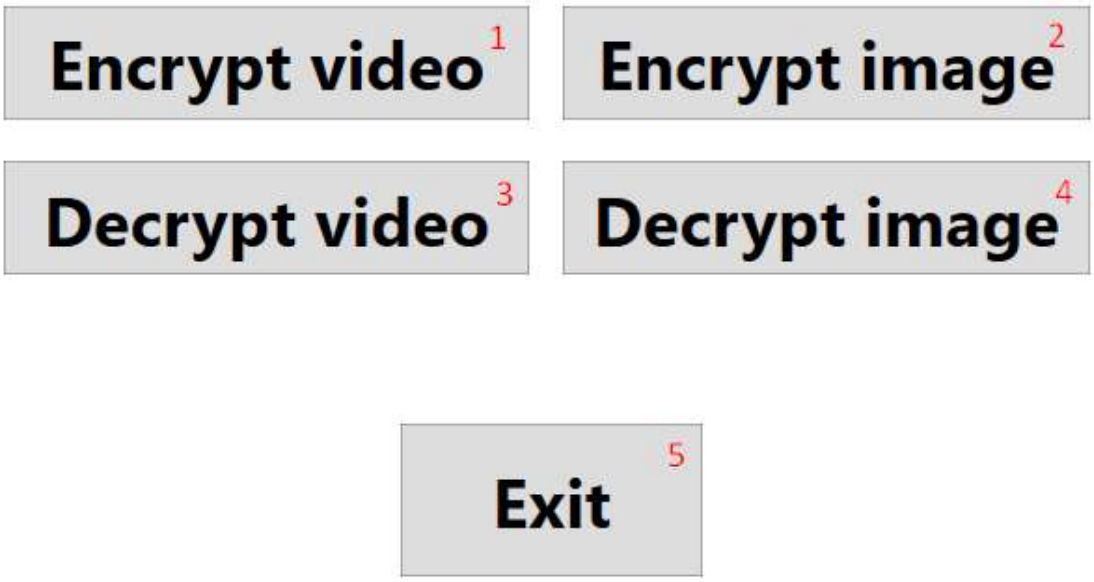

Рисунок 3.12 - Представлення меню програми

Перше представлення (рис. 3.12) це меню програми. Де відображені кнопки програми. Кнопки під номером 1 і 3 відповідають за роботу з відео.

Кнопка під номером 1 відповідає за перехід на вікно де користувач може вставити дані у відео.

Кнопка під номером 3 відповідає за перехід на вікно де користувач може дістати дані з відео.

Кнопки 2 і 4 відповідають за роботу з зображеннями.

Кнопка 2 відповідає за перехід на вікно де користувач може вставити дані у зображення.

Кнопка 4 відповідає за перехід на вікно де користувач може дістати дані з зображення.

Кнопка 5 відповідає за вихід з програми.

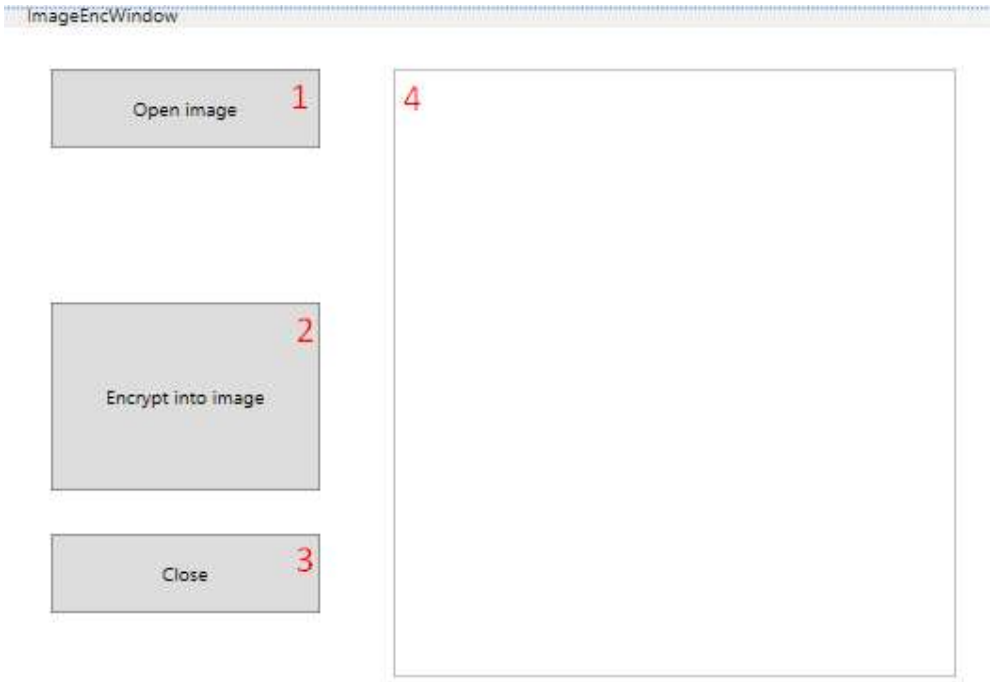

Рисунок 1.13 - Представлення вікна для вставлення даних у зображення

Наступне представлення (рис. 3.13) це вікно де користувач може вставити інформацію в зображення:

кнопка 1 – це кнопка через яку можливо вибрати зображення;

кнопка 2 – це кнопка через яку можна зашифрувати і вставити у зображення користувацькі дані;

кнопка 3 – це кнопка що дозволяє повернутися до меню;

текстове поле(цифра 4) - це поле де користувач може вставити свої дані(текст).

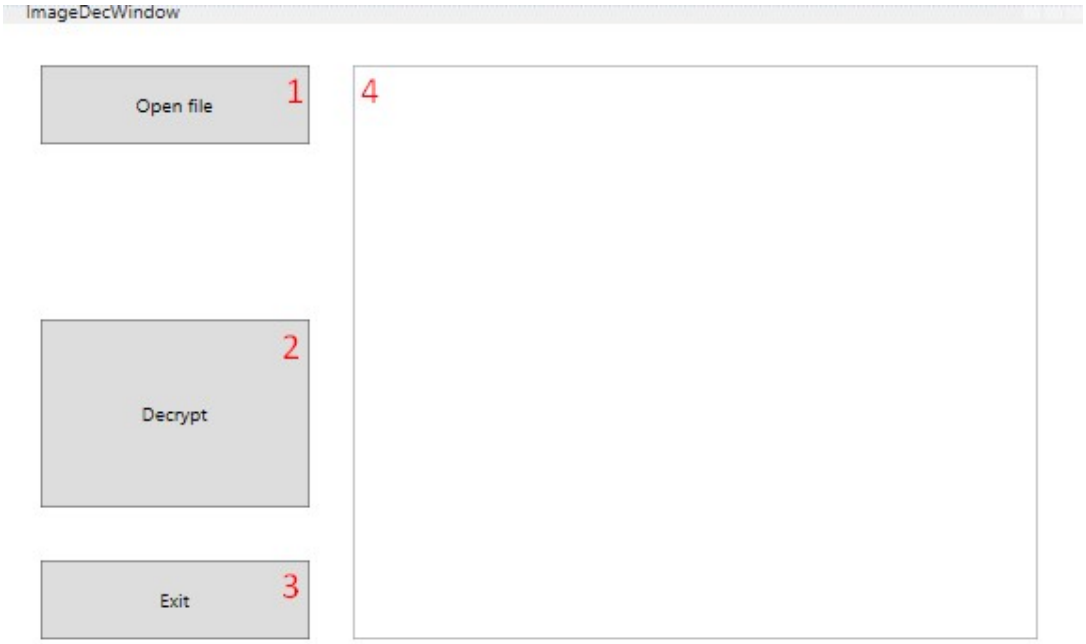

Рисунок 3.14 - Представлення вікна для отримання інформації з зображення

Наступне представлення (рис. 3.14) це вікно де користувач може дістати інформацію з зображення:

кнопка 1 – це кнопка через яку можливо вибрати зображення;

кнопка 2 – це кнопка через яку можна дешифрувати та дістати користувацькі дані;

кнопка 3 – це кнопка що дозволяє повернутися до меню;

текстове поле(цифра 4) - це поле де користувач може побачити після дешифрування та діставання свої дані(текст).

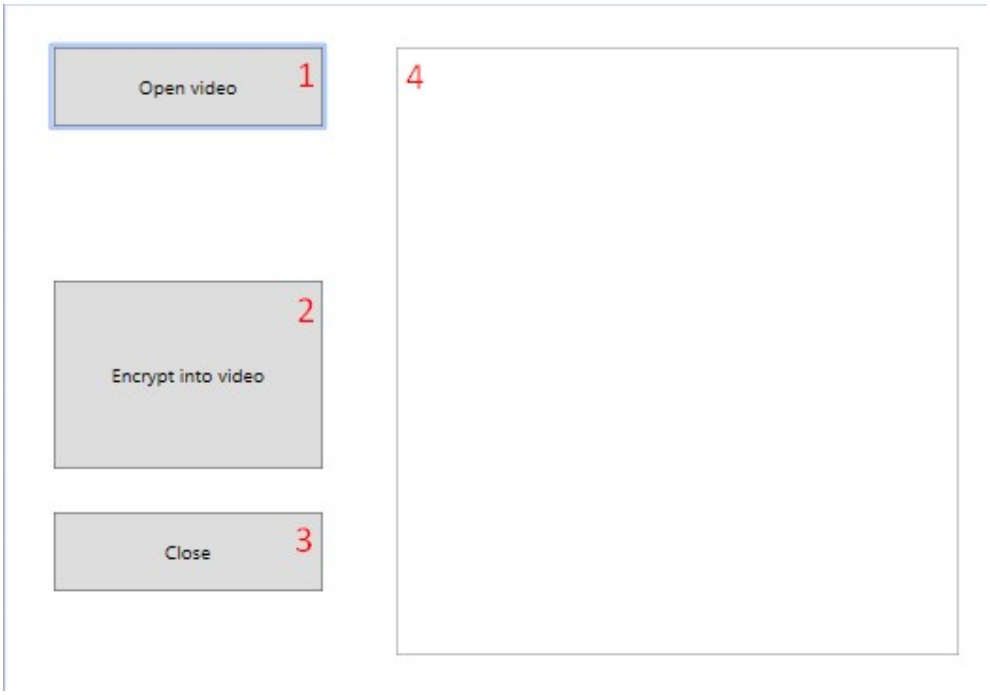

Рисунок 3.15 - Представлення вікна для вставлення інформації у відео

Наступне представлення (рис. 3.15)це вікно де користувач може дістати інформацію з зображення:

кнопка 1 – це кнопка через яку можливо вибрати відео;

кнопка 2 – це кнопка через яку можна зашифрувати та вставити користувацькі дані у відео;

кнопка 3 – це кнопка що дозволяє повернутися до меню;

текстове поле(цифра 4) - це поле де користувач може ввести свої дані, які потім будуть зашифровані.

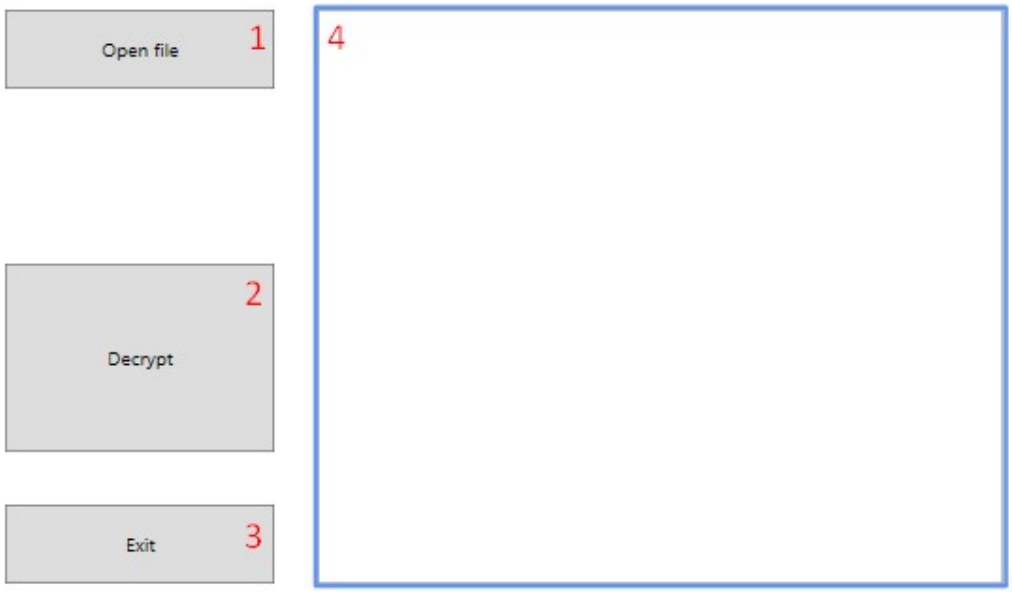

Рисунок 3.16 - Представлення вікна для отримання даних з відео

Наступне представлення (рис. 3.16) це вікно де користувач може дістати інформацію з зображення:

кнопка 1 – це кнопка через яку можливо вибрати відео;

кнопка 2 – це кнопка через яку можна дешифрувати та дістати користувацькі дані;

кнопка 3 – це кнопка що дозволяє повернутися до меню;

текстове поле(цифра 4) - це поле де користувач може побачити після дешифрування та діставання свої дані(текст).

3.4 Реалізація шифрування інформації введеної користувачем

Реалізація шифрування у проекті відбувається за допомогою класу Aes256, що використовує клас Aes з бібліотеки Cryptography.Шифрування даних відбувається у методі Encrypt. Шифрування починається з того що створюється новий екземпляр класу Aes, та передаються у цей клас ключі(рис. 3.17).

```
Aes aesAlg = Aes.Create();

aesAlg.Key = _model.Key;
aesAlg.IV = _model.IV;
```

Рисунок 3.17 - Створення нового екземпляру Aes

Після чого створюються 2 потоки, перший потік це потік в якому будуть зберігатись дані які потрібні під час роботи алгоритму, другий потік це спеціальний потік з Cryptography який дозволяє обгорнути перший потік, та застосувати до першого потоку криптографічні дії, в проекті використано Aes, що дозволяють потоково зашифрувати та розшифрувати користувацькі дані. У якості параметрів для 2го потоку передається перший потік, енкриптор, та режим роботи(рис. 3.18).

```
using (MemoryStream msEncrypt = new MemoryStream())
{
    using (CryptoStream csEncrypt =
        new CryptoStream(msEncrypt, aesAlg.CreateEncryptor(), CryptoStreamMode.Write))
```

Рисунок 3.18 - Створення streams

Коли потоки створені, то передаємо у CryptoStream користувацькі дані, та параметри того з якого елементу починати та скільки елементів шифрувати(рис. 3.19).

```
csEncrypt.Write(plainText, 0, plainText.Length);
```

Рисунок 3.19 - Шифрування тексту

Коли процес шифрування закінчено отримуємо зашифровані дані з потоку, та повертаємо їх(рис. 3.20).

```
byte[] encrBytes = msEncrypt.ToArray();
return encrBytes;
```

Рисунок 3.20 - Повернення зашифрованих даних

3.5 Реалізація дешифрування інформації введеної користувачем

Реалізація дешифрування у проекті відбувається за допомогою класу Aes256, що використовує клас Aes з бібліотеки Cryptography. Дешифрування даних відбувається у методі Decrypt. Дешифрування починається з того що створюється новий екземпляр класу Aes, та передаються у цей клас ключі(рис. 3.21).

```
Aes aesAlg = Aes.Create();
aesAlg.Key = _model.Key;
aesAlg.IV = _model.IV;
```

Рисунок 3.21 - Створення нового екземпляру Aes

Після чого створюються 2 потоки, такі самі як і при шифруванні, єдина відмінність це створення декриптора, а не енкриптора(рис. 3.22).

```
using (MemoryStream msDecr = new MemoryStream())
{
    using (CryptoStream csDecr =
        new CryptoStream(msDecr, aesAlg.CreateDecryptor(), CryptoStreamMode.Write))
```

Рисунок 3.22 - Створення streams

Коли потоки створено, то передаємо дані які потрібно розшифрувати(рис. 3.23).

```
csDecr.Write(cipherText, 0, cipherText.Length);
```

Рисунок 3.23 - Дешифрування даних

Коли користувацькі дані розшифровані, то отримуємо їх, та повертаємо.

```
byte[] decryptedBytes = msDecr.ToArray();
return decryptedBytes;
```

Рисунок 3.24 - Повернення даних

3.6 Реалізація вставлення інформації у зображення

Вставлення інформації реалізовано в класі ARGBImageEmbedding у методі Embedding який на вхід приймає інтерфейс для роботи з зображенням та користувацькі дані.

Коли метод визваний, то передані дані піддаються перевірці(рис. 3.25):

```
CheckData(image, data);

CheckData(image);
if (data == null) throw new ArgumentNullException("data is null");
if (image.Width() * image.Height() < data.Length)
    throw new ArgumentException("Not enough place for embending");

if (image == null) throw new ArgumentNullException("Image is null");
```

Рисунок 3.25 - Перевірки даних

Тут робляться перевірки на достатність місця для користувацьких даних, та наявність самого зображення.

Далі, створюємо копію зображення, та об’являємо 2 індексатора для зручного доступу до даних(рис. 3.26).

```
var copy = image.GetCopy();
var pixelIndex = 0;
var dataIndex = 0;
```

Рисунок 3.26 - Отримання зображення та ініціалізація індексів

Далі завдяки 2 циклам (рис. 3.27) крок за кроком перебираємо пікселі зображення. У кожний отриманий піксель вставляємо користувацьку інформацію, до поки користувацька інформація не закінчиться, після чого повертається змінене зображення.

```
for (int y = 0; y < copy.Height; y++)
{
    for (int x = 0; x < copy.Width; x++)
    {
        var pixel = copy.GetPixel(x, y);

        if (pixelIndex < data.Length)
        {
            var databyte = data[dataIndex++];
            Color newPixel = InsertDataToPixel(pixel, databyte);
            copy.SetPixel(x, y, newPixel);
        }

        pixelIndex++;
    }
}

return copy;
```

Рисунок 3.27 - Перебір пікселів і вставлення даних

Вставлення 1 байту користувацьких даних у піксель зображення(рис. 3.28):

```
int r, g, b;
r = (pixel.R & 0b11111000) | (databyte >> 5 & 0b111);
g = ((pixel.G >> 2) << 2) | (databyte >> 3 & 0b11);
b = ((pixel.B >> 3) << 3) | (databyte & 0b111);
var newPixel = Color.FromArgb(pixel.A, r, g, b);
return newPixel;
```

Рисунок 3.28 - Вставлення байту в піксель

Вставлення даних відбувається у окремому методі InsertDataToPixel, на вхід передаються піксель, та байт. Вставлення бітів байту відбувається завдяки бітовим операціям.

Для формування нового червоного каналу, ми застосовуємо маску до старого значення каналу, яка очищає останні 3 біти, після чого за допомогою побітового or записуємо останні 3 біти користувацького байту у канал.

Для формування нового зеленого каналу ми використовуючи побітовий зсув теж очищаємо останні 2 біти каналу. Потім записуємо туди 2 середніх біти користувацького байту.

Для формування нового зеленого каналу очищаємо 3 останніх біти каналу і записуємо туди 3 перші біти користувацької інформації.

Після чого створюємо новий піксель з новими кольорами та повертаємо його.

3.7 Реалізація зчитування інформації з зображення

Зчитування інформації з зображення починається з перевірок та створення списку для збереження байтів з зображення. Тут використовується список, оскільки, достовірно кількість інформації для зчитування не відомо(рис. 3.29).

```
CheckData(image);

var data = new List<byte>();
```

Рисунок 3.29 - Перевірки даних та створення списку для користувацьких даних

Потім створюється булева зміна, яка потрібна для того щоб вказати чи дійшов процес екстракції інформації до кінця чи ні (рис. 3.30).

```
bool stopBytesFound = false;
```

Рисунок 3.30 – перемінна в який вказується чи було знайдено кінець повідомлення

Далі отримується бітпам зображення і перебирається попіксельно до поки не будуть знайдені стоп пікселі(рис. 3.31). . Отримані байти записуються у список.

```
var bitmap = image.GetBitmap();
for (int y = 0; y < bitmap.Height; y++)
{
    for (int x = 0; x < bitmap.Width; x++)
    {
        var extractedByte = SelectByteFromPixel(bitmap, x, y);
        data.Add(extractedByte);

        CheckIfStopBytesAreFound(data, ref stopBytesFound);
        if (stopBytesFound)
        {
            break;
        }
    }

    if (stopBytesFound)
    {
        break;
    }
}
```

Рисунок 3.31 – Отримання даних з зображення

Екстракція даних відбувається у методі SelectByteFromPixel, куди передаються бітмап, та координати. На виході отримується байт інформації(рис. 3.32).

```
int r = (bitmap.GetPixel(x, y).R & 0b111) << 5;
int g = (bitmap.GetPixel(x, y).G & 0b11) << 3;
int b = bitmap.GetPixel(x, y).B & 0b111;

int dataByte = r | g | b;
return (byte)dataByte;
```

Рисунок 3.32 - Отримання байту даних з пікселю

Спочатку з кожного каналу отримуємо біти інформації після чого об’єднуємо їх та повертаємо у вигляді цілого байту.

Визначення того чи були знайдені стоп байти робиться у методі CheckIfStopBytesAreFound, в який передається список і вказівник на stopBytesFound. Де порівнюються 2 стоп байти «0» «0» з 2 останніми байтами дістаних даних. Байти «0» «0» - це «\0» тобто значення що вказує що це кінець даних. Оскільки цей символ «\0» користувач ніяк не може ввести, то він був обраний як маркер кінця даних (рис. 3.33)..

```
if (data.Count >= stopBytes.Length &&
            data[data.Count - stopBytes.Length] == stopBytes[0] &&
            data[data.Count - 1] == stopBytes[1] &&
            data.Count % 2 == 0)
{
    stopBytesFound = true;
}
```

Рисунок 3.33 – Перевірка на те чи були отримані стоп байти

3.8 Реалізація заміни кадрів у відео

Заміна кадрів у відео файлі відбувається у класі FFmpegVideo_v2 у методі Embedding, який у якості параметрів приймає шлях до відео, шлях на місце де зберігти нове відео треба та зашифровану користувацьку інформацію.

Далі створюємо об’єкт VideoFileWriter для створення нового відео. Далі використовуючи reader отримуємо кадри відео і попіксельно змінюємо у них колір додаючи у пікселі користувацьку інформацію. Коли нове зображення створено то додаємо його у writer(рис. 3.34)..

```
using (var writer = new VideoFileWriter())
{
    writer.Open(outPath, width, height, (int)reader.FrameRate, VideoCodec.Raw);

    int pixelIndex = 0;
    int dataIndex = 0;
    for (int i = 0; i < reader.FrameCount; i++)
    {
        var bitmap = reader.ReadVideoFrame();
        if (data.Length > dataIndex)
        {
            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    if (pixelIndex < data.Length)
                    {
                        ARGBImageEmbending emb = new ARGBImageEmbending();
                        var embedPixel = emb.InsertDataToPixel(bitmap.GetPixel(x, y), data[dataIndex++]);
                        bitmap.SetPixel(x, y, embedPixel);
                    }
                    pixelIndex++;

                }
            }
        }

        writer.WriteVideoFrame(bitmap);
```

Рисунок 3.34 – Процес запису даних у відео

Потім закриваєм та звільнюєм reader і writer. При закритті writer він автоматично зберігає зміни що були внесені.

3.9 Реалізація зчитування кадрів з відео

Реалізація зчитування даних з відео подібна записуванню та тому як це відбувається у модулі з зображеннями. Спочатку відкривається відео для читання вихідного відео, потім по-кадрово і по-піксельно читається до поки не будуть знайдені стоп байти (рис. 3.35)..

Коли стоп байти знайдені то повертається список з дістаними байтами.

```
using (var reader = new VideoFileReader())
{
    reader.Open(path);
    var data = new List<byte>();
    bool stopBytesFound = false;
    for (int i = 0; i < reader.FrameCount; i++)
    {
        var frame = reader.ReadVideoFrame();
        for (int y = 0; y < frame.Height; y++)
        {
            for (int x = 0; x < frame.Width; x++)
            {
                var emb = new ARGBImageEmbending();
                int dataByte = emb.SelectByteFromPixel(frame, x, y);
                data.Add((byte)dataByte);

                if (data.Count >= stopBytes.Length &&
                    data[data.Count - stopBytes.Length] == stopBytes[0] &&
                    data[data.Count - 1] == stopBytes[1] &&
                    data.Count % 2 == 0)
                {
                    stopBytesFound = true;
                    break;
                }
            }

            if (stopBytesFound) break;
        }

        if (stopBytesFound) break;
```

Рисунок 3.35 – Процес отримання даних з відео

Висновки

Під час виконання кваліфікаційної роботи було використано Aes як метод шифрування для забезпечення конфіденційності та безпеки текстової інформації засобами мультимедіа. Цей алгоритм є універсальними і забезпечує високий рівень захисту. Використання цього методу шифрування дозволяє зберігати цілісність текстових даних, гарантуючи їх приватність та неможливість несанкціонованого доступу сторонніх осіб під час передачі в мультимедіа.

Досліджено можливість збереження, редагування і забезпечення безпеки даних, які знаходяться у мультимедійних контейнерах, таких як зображення та відео різних форматів.

Завдяки кодуванню Unicode було додано підтримку збереження тексту різними мовами та можливість їх комбінувати для подальшого збереження у фото та відео контейнерах.

Реалізовано алгоритм зі збереження даних у фотографіях, використовуючи алгоритм зміни останніх бітів у каналах кольорів зображення.

Успішно реалізовано алгоритм зі збереження даних у кадрах відео, використовуючи відповідний модифікований алгоритм для зображень.

Список використаних джерел

1. JSON .NET [Електронний ресурс] - https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview
2. Accord .NET [Електронний ресурс] - http://accord-framework.net/
3. Accord.FFMPEG .NET [Електронний ресурс] - http://accord-framework.net/docs/html/N_Accord_Video_FFMPEG.htm
4. WPF [Електронний ресурс] - https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-wpf?view=vs-2022
5. Мова програмування C# [Електронний ресурс] - https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/?view=vs-2022
6. Image WPF [Електронний ресурс] - https://learn.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/how-to-create-a-bitmap-from-a-visual?view=netframeworkdesktop-4.8
7. MVVM [Електронний ресурс] - https://metanit.com/sharp/wpf/22.1.php
8. UNICODE [Електронний ресурс] - https://learn.microsoft.com/en-us/dotnet/standard/base-types/character-encoding-introduction

```
namespace WPF_9.image.model
{
   public class ARGBImageModel : IARGBImage
   {
      private Bitmap _bitmap;
      public ARGBImageModel(int width, int height)
      {
         CheckData(width, height);
         _bitmap = new Bitmap(width, height);
      }
      public ARGBImageModel(Bitmap bitmap)
      {
         CheckData(bitmap);
         _bitmap = bitmap;
      }
      public Bitmap GetBitmap()
      {
         return _bitmap;
      }
      public Color GetPixel(int x, int y)
      {
         CheckData(x,y);
         return _bitmap.GetPixel(x, y);
      }
      public int Height()
      {
         return _bitmap.Height;
      }
      public void SetPixel(int x, int y, Color color)
```

```
        {
            CheckData(x,y);
            _bitmap.SetPixel(x, y, color);
        }
        public int Width()
        {
            return _bitmap.Width;
        }
        public void SetBitmap(Bitmap bitmap)
        {
            CheckData(bitmap);
            _bitmap = bitmap;
        }
        private void CheckData(int width, int height)
        {
            if (width < 0 || height < 0)
            {
                throw new ArgumentException($"Values should be more than 0. Width:
{width}. Height: {height}");
            }
        }
        private void CheckData(Bitmap bitmap)
        {
            if (bitmap == null)
            {
                throw new ArgumentException("The bitmap is null");
            }
        }
        public Bitmap GetCopy()
        {
```

```
            return new Bitmap( _bitmap );
        }
    }
}

namespace WPF_9.image.service
{
    internal class ARGBImageEmbedding : IImageEmbedding
    {
        static readonly byte[] stopBytes = { 0, 0 };// same to '\0'

        public virtual Bitmap Embedding(IARGBImage image, byte[] data)
        {
            CheckData(image, data);

            var copy = image.GetCopy();
            var pixelIndex = 0;
            var dataIndex = 0;

            for (int y = 0; y < copy.Height; y++)
            {
                for (int x = 0; x < copy.Width; x++)
                {
                    var pixel = copy.GetPixel(x, y);

                    if (pixelIndex < data.Length)
                    {
                        var databyte = data[dataIndex++];
                        Color newPixel = InsertDataToPixel(pixel, databyte);
                        copy.SetPixel(x, y, newPixel);
```

```
            }

            pixelIndex++;
        }
    }

    return copy;
}

public virtual byte[] UnEmbedding(IARGBImage image)
{
    CheckData(image);

    var data = new List<byte>();
    bool stopBytesFound = false;

    var bitmap = image.GetBitmap();
    for (int y = 0; y < bitmap.Height; y++)
    {
        for (int x = 0; x < bitmap.Width; x++)
        {
            var extractedByte = SelectByteFromPixel(bitmap, x, y);
            data.Add(extractedByte);

            CheckIfStopBytesAreFound(data, ref stopBytesFound);
            if (stopBytesFound)
            {
                break;
            }
        }
```

```
            if (stopBytesFound)
            {
                break;
            }
        }

        if (stopBytesFound)
        {
            //RemoveStopBytes(data);
        }

        return data.ToArray();
    }

    public byte[] RemoveStopBytes(byte[] data)
    {
        Array.Resize(ref data, data.Length - stopBytes.Length);
        return data;
    }

    private void CheckIfStopBytesAreFound(List<byte> data,ref bool
stopBytesFound)
    {
        if (data.Count >= stopBytes.Length &&
                data[data.Count - stopBytes.Length] == stopBytes[0] &&
                data[data.Count - 1] == stopBytes[1] &&
                data.Count % 2 == 0)
        {
            stopBytesFound = true;
```

```
    }
}

public virtual byte SelectByteFromPixel( Bitmap bitmap, int x, int y)
{
    int r = (bitmap.GetPixel(x, y).R & 0b111) << 5;
    int g = (bitmap.GetPixel(x, y).G & 0b11) << 3;
    int b = bitmap.GetPixel(x, y).B & 0b111;

    int dataByte = r | g | b;
    return (byte)dataByte;
}

public byte[] AddStopBytes(byte[] data)
{
    Array.Resize(ref data, data.Length + 2);
    data[data.Length - 2] = stopBytes[0];
    data[data.Length - 1] = stopBytes[1];
    return data;
}

public virtual Color InsertDataToPixel(Color pixel, byte databyte)
{
    int r, g, b;
    r = (pixel.R & 0b11111000) | (databyte >> 5 & 0b111);
    g = ((pixel.G >> 2) << 2) | (databyte >> 3 & 0b11);
    b = ((pixel.B >> 3) << 3) | (databyte & 0b111);
    var newPixel = Color.FromArgb(pixel.A, r, g, b);
    return newPixel;
}
```

```
        private static void CheckData(IARGBImage image, byte[] data)
        {
            CheckData(image);
            if (data == null) throw new ArgumentNullException("data is null");
            if (image.Width() * image.Height() < data.Length)
                throw new ArgumentException("Not enough place for embedding");
        }

        private static void CheckData(IARGBImage image)
        {
            if (image == null) throw new ArgumentNullException("Image is null");
        }

    }
}

namespace WPF_9.image.views.models
{
    internal class ImageDecWindowModel
    {
        public string InPath { get; set; }
        public string Data { get; set; }
        public Encoding Encoding { get; set; }
    }
}

internal class ImageEncWindowModel
    {
        public string InPath;
        public string OutPath;
```

```
        public Encoding Encoding;
        public string Data;
    }
namespace WPF_9.image.views
{
    /// <summary>
    /// Interaction logic for ImageDecWindow.xaml
    /// </summary>
    public partial class ImageDecWindow : Window
    {
        MenuWindowModel _menu = new MenuWindowModel();
        ImageDecWindowModel _dec = new ImageDecWindowModel();
        public ImageDecWindow(MenuWindowModel menu)
        {
            _menu = menu;
            _dec.Encoding = Encoding.Unicode;
            this.Closing += Grid_Closing;
            this.Loaded += Grid_Loaded;
            InitializeComponent();
        }

        private void Grid_Loaded(object sender, RoutedEventArgs e)
        {
            this.Left = _menu.MenuWindow.Left;
            this.Top = _menu.MenuWindow.Top;
            _menu.MenuWindow.Hide();
        }
```

```
        private void Grid_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            _menu.MenuWindow.Left = this.Left;
            _menu.MenuWindow.Top = this.Top;
            _menu.MenuWindow.Show();
        }

        private void OpenFileButton_Click(object sender, RoutedEventArgs e)
        {
            var openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Image Files (*.jpg;*.png)|*.jpg;*.png|All Files
(*.*)|*.*";

            if ((bool)openFileDialog.ShowDialog())
            {
                _dec.InPath = openFileDialog.FileName;
            }
        }

        private void DecryptButton_Click(object sender, RoutedEventArgs e)
        {
            if (_dec.InPath == null)
            {
                MessageBox.Show("Choose a video to decode");
                return;
            }
            var model = new
ARGBImageModel((Bitmap)Bitmap.FromFile(_dec.InPath));
            Aes256 aes;
```

```
            byte[] dataBytes;
            byte[] encDataBytes = UnEmbing(model);

            Decrypting(encDataBytes, out aes, out dataBytes);
            int hashFromFile = SelectHashBytes(ref dataBytes);

            var dataHash = aes.GenHashCode(dataBytes);
            if (hashFromFile != dataHash)
            {
                MessageBox.Show("The file is corrupted. Hashes do not match!");
                return;
            }

            _dec.Data = _dec.Encoding.GetString(dataBytes);

            DecryptedTextBox.Text = _dec.Data;
        }

        private static int SelectHashBytes(ref byte[] dataBytes)
        {
            var oneNumberSizeInBytes = new byte[4];
            Array.Copy(dataBytes, dataBytes.Length - 4, oneNumberSizeInBytes, 0, 4);
            var hashFromFile = BitConverter.ToInt32(oneNumberSizeInBytes, 0);

            Array.Resize(ref dataBytes, dataBytes.Length - 4);
            return hashFromFile;
        }

        private void Decrypting(byte[] encDataBytes, out Aes256 aes, out byte[]
dataBytes)
```

```
        {
            var aesModel = Aes256Model.Load("configs\\appconfig.json");
            aesModel.Encoding = _dec.Encoding;
            aes = new Aes256(aesModel);
            dataBytes = aes.Decrypt(encDataBytes);
        }

        private static byte[] UnEmbing(ARGBImageModel model)
        {
            var embedding = new ARGBImageEmbedding();
            var encDataBytes = embedding.UnEmbedding(model);
            encDataBytes = embedding.RemoveStopBytes(encDataBytes);
            return encDataBytes;
        }

        private void ExitButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
    }
}

namespace WPF_9.image.views
{
    /// <summary>
    /// Interaction logic for ImageEncWindow.xaml
    /// </summary>
    public partial class ImageEncWindow : Window
    {
        MenuWindowModel _menu = new MenuWindowModel();
```

```
        ImageEncWindowModel _enc = new ImageEncWindowModel();
        public ImageEncWindow(MenuWindowModel menu)
        {
            _menu= menu;
            _enc.Encoding = Encoding.Unicode;

            this.Closing += Grid_Closing;
            this.Loaded += Grid_Loaded;
            InitializeComponent();
        }
        private void Grid_Loaded(object sender, RoutedEventArgs e)
        {
            this.Left = _menu.MenuWindow.Left;
            this.Top = _menu.MenuWindow.Top;
            _menu.MenuWindow.Hide();
        }

        private void Grid_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            _menu.MenuWindow.Left = this.Left;
            _menu.MenuWindow.Top = this.Top;
            _menu.MenuWindow.Show();
        }
        private void FindImageFile_Click(object sender, RoutedEventArgs e)
        {
            var openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Image Files (*.jpg;*.png)|*.jpg;*.png|All Files
(*.*)|*.*";
```

```
            if ((bool)openFileDialog.ShowDialog())
            {
                _enc.InPath = openFileDialog.FileName;
                var fullOutPath =
System.IO.Path.GetDirectoryName(openFileDialog.FileName) +
                    $"\\out - {System.IO.Path.GetFileName(openFileDialog.FileName)}";
                _enc.OutPath = fullOutPath;
            }
        }

        private void EncryptButton_Click(object sender, RoutedEventArgs e)
        {
            if (UserDataInputTextBox.Text == "")
            {
                MessageBox.Show("Enter the text to start the encryption");
                return;
            }
            if (_enc.InPath == null)
            {
                MessageBox.Show("Select the image file where the text will be
encrypted");
                return;
            }

            Bitmap bitmap = (Bitmap)Bitmap.FromFile(_enc.InPath);
            ARGBImageModel model = new ARGBImageModel(bitmap);

            var totalPlaceInBytes = model.Width() * model.Height();
            var oneNumberSizeInBytes = 4;
```

```
            var dataSizeInBytes =
Encoding.Unicode.GetBytes(UserDataInputTextBox.Text).Length +
oneNumberSizeInBytes;
            if (totalPlaceInBytes < dataSizeInBytes)
            {
                MessageBox.Show("There is not enough space in the image file for your
text " +
                    "Make your message shorter, or choose another image");
                return;
            }
            _enc.Data = UserDataInputTextBox.Text;
            Embedding(model);

            MessageBox.Show("Your text has been successfully inserted into the
image");
        }

        private void Embedding(ARGBImageModel model)
        {
            byte[] encryptedText = Encrypting();
            var embedding = new ARGBImageEmbedding();
            encryptedText = embedding.AddStopBytes(encryptedText);
            var embedBitmap = embedding.Embedding(model, encryptedText);
            embedBitmap.Save(_enc.OutPath);
        }

        private byte[] Encrypting()
        {
            var aesModel = Aes256Model.Load("configs\\appconfig.json");
            aesModel.Encoding = _enc.Encoding;
```

```
            var aes = new Aes256(aesModel);

            var hash = aes.GenHashCode(_enc.Data);
            var hashInBytes = BitConverter.GetBytes(hash);

            var fullData = Encoding.Unicode.GetBytes(_enc.Data);

            Array.Resize(ref fullData, fullData.Length + 4);
            fullData[fullData.Length - 4] = hashInBytes[0];
            fullData[fullData.Length - 3] = hashInBytes[1];
            fullData[fullData.Length - 2] = hashInBytes[2];
            fullData[fullData.Length - 1] = hashInBytes[3];
            byte[] encryptedText = aes.Encrypt(fullData);
            return encryptedText;
        }

        private void ExitButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
    }
}

namespace WPF_9.image
{
    public interface IARGBImage
    {
        int Width();
        int Height();
        Bitmap GetBitmap();
```

```
        void SetBitmap(Bitmap bitmap);
        void SetPixel(int x, int y, Color color);
        Color GetPixel(int x, int y);
        Bitmap GetCopy();

    }
}
namespace WPF_9.image
{
    internal interface IImageEmbedding
    {
        Bitmap Embedding(IARGBImage image,byte[] data);
        byte[] UnEmbedding(IARGBImage image);

    }
}
namespace WPF_9.security.models
{
    public class Aes256Model
    {
        [JsonProperty("key")]
        public byte[] Key { get; private set; }

        [JsonProperty("iv")]
        public byte[] IV { get; private set; }

        [NonSerialized]
        public Encoding Encoding;
```

```
        public static Aes256Model Load(string configFilePath)
        {
            using (StreamReader reader = new StreamReader(configFilePath))
            {
                string json = reader.ReadToEnd();
                Aes256Model aes =
JsonConvert.DeserializeObject<Aes256Model>(json);
                if(aes == null)
                {
                    throw new Exception("appconfig is empty!");
                }
                return aes;
            }
        }
    }
}

namespace WPF_9.security.service
{
    public class Aes256 : IEncryptionService
    {
        private Aes256Model _model;
        public Encoding Encoding => _model.Encoding;
        public Aes256(Aes256Model model) => _model = model;

        public byte[] Decrypt(byte[] cipherText)
        {
            Aes aesAlg = Aes.Create();
            aesAlg.Key = _model.Key;
            aesAlg.IV = _model.IV;
```

```
            using (MemoryStream msDecr = new MemoryStream())
            {
                using (CryptoStream csDecr =
                    new CryptoStream(msDecr, aesAlg.CreateDecryptor(),
CryptoStreamMode.Write))
                {
                    csDecr.Write(cipherText, 0, cipherText.Length);
                }

                byte[] decryptedBytes = msDecr.ToArray();
                return decryptedBytes;
            }
        }

        public byte[] Encrypt(byte[] plainText)
        {
            Aes aesAlg = Aes.Create();

            aesAlg.Key = _model.Key;
            aesAlg.IV = _model.IV;

            using (MemoryStream msEncrypt = new MemoryStream())
            {
                using (CryptoStream csEncrypt =
                    new CryptoStream(msEncrypt, aesAlg.CreateEncryptor(),
CryptoStreamMode.Write))
                {
                    csEncrypt.Write(plainText, 0, plainText.Length);
                }
```

```
            byte[] encrBytes = msEncrypt.ToArray();
            return encrBytes;
        }
    }

    public int GenHashCode(byte[] bytes)
    {
        int hash = 23;
        for (int i = 0; i < bytes.Length; i++)
        {
            hash = hash * 31 + bytes[i];
        }
        return hash;
    }

    public int GenHashCode(string str)
    {
        return GenHashCode(_model.Encoding.GetBytes(str));
    }
  }
}

namespace WPF_9.security
{
  public interface IEncryptionService
  {
    byte[] Encrypt(byte[] plainText);
    byte[] Decrypt(byte[] cipherText);
    Encoding Encoding { get;}
    int GenHashCode(string str);
```

```
        int GenHashCode(byte[] bytes);
    }
}

namespace WPF_9.video.models
{
    public class FFmpegVideoModel : IVideoParams
    {
        public long TotalVideoFrames { get; set; }
        public int Width { get; set; }
        public int Height { get; set; }
    }
}

namespace WPF_9.video.service
{
    public class FFmpegVideo_v2
    {
        static readonly byte[] stopBytes = { 0, 0 };// same to '\0'

        public static void Embedding(string inPath, string outPath, byte[] data)
        {
            //data = AddStopBytes(data);

            using (var reader = new VideoFileReader())
            {
                reader.Open(inPath);

                int width = reader.Width;
                int height = reader.Height;
```

```
            using (var writer = new VideoFileWriter())
            {
                writer.Open(outPath, width, height, (int)reader.FrameRate,
VideoCodec.Raw);

                int pixelIndex = 0;
                int dataIndex = 0;
                for (int i = 0; i < reader.FrameCount; i++)
                {
                    var bitmap = reader.ReadVideoFrame();
                    if (data.Length > dataIndex)
                    {
                        for (int y = 0; y < height; y++)
                        {
                            for (int x = 0; x < width; x++)
                            {
                                if (pixelIndex < data.Length)
                                {
                                    ARGBImageEmbedding emb = new
ARGBImageEmbedding();
                                    var embedPixel =
emb.InsertDataToPixel(bitmap.GetPixel(x, y), data[dataIndex++]);
                                    bitmap.SetPixel(x, y, embedPixel);
                                }
                                pixelIndex++;

                            }
                        }
                    }
```

```
                writer.WriteVideoFrame(bitmap);
            }

            writer.Close();
        }

        reader.Close();
    }

}

private static byte[] AddStopBytes(byte[] data)
{
    Array.Resize(ref data, data.Length + 2);
    data[data.Length - 2] = stopBytes[0];
    data[data.Length - 1] = stopBytes[1];
    return data;
}

public static byte[] UnEmbedding(string path)
{
    using (var reader = new VideoFileReader())
    {
        reader.Open(path);
        var data = new List<byte>();
        bool stopBytesFound = false;
        for (int i = 0; i < reader.FrameCount; i++)
        {
            var frame = reader.ReadVideoFrame();
            for (int y = 0; y < frame.Height; y++)
```

```
            {
                for (int x = 0; x < frame.Width; x++)
                {
                    var emb = new ARGBImageEmbedding();
                    int dataByte = emb.SelectByteFromPixel(frame, x, y);
                    data.Add((byte)dataByte);

                    if (data.Count >= stopBytes.Length &&
                        data[data.Count - stopBytes.Length] == stopBytes[0] &&
                        data[data.Count - 1] == stopBytes[1] &&
                        data.Count % 2 == 0)
                    {
                        stopBytesFound = true;
                        break;
                    }
                }

                if (stopBytesFound) break;
            }

            if (stopBytesFound) break;
        }
        reader.Close();
        return data.ToArray();
    }
}

public static FFmpegVideoModel GetVideoParams(string inPath)
{
    FFmpegVideoModel model = new FFmpegVideoModel();
```

```
            using (VideoFileReader reader = new VideoFileReader())
            {
                reader.Open(inPath);
                model.TotalVideoFrames = reader.FrameCount;
                model.Width = reader.Width;
                model.Height = reader.Height;
                reader.Close();
            }
            return model;
        }
    }
}
namespace WPF_9.video.views.models
{
    internal class DecWindowModel
    {
        public string InPath { get; set; }
        public string Data { get; set; }
        public Encoding Encoding { get; set; }
    }
}

namespace WPF_9.video.views.models
{
    public class EncWindowModel
    {
        public string InPath { get; set; }
        public string OutPath { get; set; }
        public string Data { get; set; }
        public Encoding Encoding { get; set; }
```

```
    }
}

namespace WPF_9.video.views
{
    /// <summary>
    /// Interaction logic for DecWindow.xaml
    /// </summary>
    public partial class DecWindow : Window
    {
        MenuWindowModel _menu = new MenuWindowModel();
        DecWindowModel _dec = new DecWindowModel();
        public DecWindow(MenuWindowModel menu)
        {
            _menu= menu;
            _dec.Encoding = Encoding.Unicode;
            this.Closing += Grid_Closing;
            this.Loaded += Grid_Loaded;
            InitializeComponent();
        }

        private void Grid_Loaded(object sender, RoutedEventArgs e)
        {
            this.Left = _menu.MenuWindow.Left;
            this.Top = _menu.MenuWindow.Top;
            _menu.MenuWindow.Hide();
        }
```

```
        private void Grid_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            _menu.MenuWindow.Left = this.Left;
            _menu.MenuWindow.Top = this.Top;
            _menu.MenuWindow.Show();
        }

        private void OpenFileButton_Click(object sender, RoutedEventArgs e)
        {
            var openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Video Files (*.avi;*.mp4)|*.avi;*.mp4|All Files
(*.*)|*.*";

            if ((bool)openFileDialog.ShowDialog())
            {
                _dec.InPath = openFileDialog.FileName;
            }
        }

        private void ExitButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }

        private void DecryptButton_Click(object sender, RoutedEventArgs e)
        {
            if (_dec.InPath == null)
            {
                MessageBox.Show("Choose a video to decode");
```

```
            return;
        }
        ARGBImageEmbedding emb;
        byte[] encDataBytes;
        Unembedding(out emb, out encDataBytes);
        encDataBytes = emb.RemoveStopBytes(encDataBytes);
        Aes256 aes;
        byte[] dataBytes;
        Decryption(encDataBytes, out aes, out dataBytes);
        var oneNumberSizeInBytes = new byte[4];
        Array.Copy(dataBytes, dataBytes.Length - 4, oneNumberSizeInBytes, 0, 4);
        var hashFromFile = BitConverter.ToInt32(oneNumberSizeInBytes, 0);

        Array.Resize(ref dataBytes, dataBytes.Length - 4);

        var dataHash = aes.GenHashCode(dataBytes);
        if (hashFromFile != dataHash)
        {
            MessageBox.Show("The file is corrupted. Hashes do not match!");
            return;
        }

        _dec.Data = _dec.Encoding.GetString(dataBytes);

        DecryptedTextBox.Text = _dec.Data;
    }

    private void Decryption(byte[] encDataBytes, out Aes256 aes, out byte[]
dataBytes)
    {
```

```
            var aesModel = Aes256Model.Load("configs\\appconfig.json");
            aesModel.Encoding = _dec.Encoding;
            aes = new Aes256(aesModel);
            dataBytes = aes.Decrypt(encDataBytes);
        }

        private void Unembedding(out ARGBImageEmbedding emb, out byte[]
encDataBytes)
        {
            emb = new ARGBImageEmbedding();
            encDataBytes = FFmpegVideo_v2.UnEmbedding(_dec.InPath);
        }
    }

}

namespace WPF_9.video.views
{
    /// <summary>
    /// Interaction logic for EncWindow.xaml
    /// </summary>
    public partial class EncWindow : Window
    {
        private MenuWindowModel _menu = new MenuWindowModel();
        public EncWindowModel _enc = new EncWindowModel();
        public EncWindow(MenuWindowModel menu)
        {
            _menu = menu;
            _enc.Encoding = Encoding.Unicode;
            this.Closing += Grid_Closing;
```

```
            this.Loaded += Grid_Loaded;
            InitializeComponent();

        }

        private void Grid_Loaded(object sender, RoutedEventArgs e)
        {
            this.Left = _menu.MenuWindow.Left;
            this.Top = _menu.MenuWindow.Top;
            _menu.MenuWindow.Hide();
        }

        private void Grid_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            _menu.MenuWindow.Left = this.Left;
            _menu.MenuWindow.Top = this.Top;
            _menu.MenuWindow.Show();
        }

        private void FindVideoFile_Click(object sender, RoutedEventArgs e)
        {
            var openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Video Files (*.avi;*.mp4)|*.avi;*.mp4|All Files
(*.*)|*.*";

            if ((bool)openFileDialog.ShowDialog())
            {
                _enc.InPath = openFileDialog.FileName;
```

```
            var fullOutPath =
System.IO.Path.GetDirectoryName(openFileDialog.FileName) +
                $"\\out - {System.IO.Path.GetFileName(openFileDialog.FileName)}";
            _enc.OutPath = fullOutPath;
        }
    }

    private void EncryptButton_Click(object sender, RoutedEventArgs e)
    {
        if (UserDataInputTextBox.Text == "")
        {
            MessageBox.Show("Enter the text to start the encryption");
            return;
        }
        if (_enc.InPath == null)
        {
            MessageBox.Show("Select the video file where the text will be
encrypted");
            return;
        }

        var videoParams = FFmpegVideo_v2.GetVideoParams(_enc.InPath);
        var totalPlaceInBytes = videoParams.TotalVideoFrames *
videoParams.Width * videoParams.Height;
        var oneNumberSizeInBytes = 4;
        var dataSizeInBytes =
Encoding.Unicode.GetBytes(UserDataInputTextBox.Text).Length +
oneNumberSizeInBytes;
        if (totalPlaceInBytes < dataSizeInBytes)
        {
```

```
            MessageBox.Show("There is not enough space in the video file for your
text " +
                "Make your message shorter, or choose another video");
            return;
        }

        _enc.Data = UserDataInputTextBox.Text;
        byte[] encryptedText = Encryption();
        encryptedText = AddStopBytes(encryptedText);
        FFmpegVideo_v2.Embedding(_enc.InPath, _enc.OutPath, encryptedText);

        MessageBox.Show("Your text has been successfully inserted into the
video");
    }

    private static byte[] AddStopBytes(byte[] encryptedText)
    {
        var emb = new ARGBImageEmbedding();
        encryptedText = emb.AddStopBytes(encryptedText);
        return encryptedText;
    }

    private byte[] Encryption()
    {
        var aesModel = Aes256Model.Load("configs\\appconfig.json");
        aesModel.Encoding = _enc.Encoding;
        var aes = new Aes256(aesModel);

        var hash = aes.GenHashCode(_enc.Data);
        var hashInBytes = BitConverter.GetBytes(hash);
```

```
            var fullData = Encoding.Unicode.GetBytes(_enc.Data);
            Array.Resize(ref fullData, fullData.Length + 4);
            fullData[fullData.Length - 4] = hashInBytes[0];
            fullData[fullData.Length - 3] = hashInBytes[1];
            fullData[fullData.Length - 2] = hashInBytes[2];
            fullData[fullData.Length - 1] = hashInBytes[3];
            byte[] encryptedText = aes.Encrypt(fullData);
            return encryptedText;
        }

        private void ExitButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
    }
}
namespace WPF_9.video
{
    public interface IVideoParams
    {
        long TotalVideoFrames { get; set; }
        int Width { get; set; }
        int Height { get; set; }

    }
}
namespace WpfApp7.views
{
```

```
    /// <summary>
    /// Interaction logic for MenuViewWindow.xaml
    /// </summary>
    public partial class MenuViewWindow : Window
    {
        public MenuViewWindow()
        {
            InitializeComponent();
        }

        private void EncButton_Click(object sender, RoutedEventArgs e)
        {
            var encrypt = new EncWindow(new MenuWindowModel() { MenuWindow
= this});
            encrypt.Show();
        }

        private void DecButton_Click(object sender, RoutedEventArgs e)
        {
            var decrypt = new DecWindow(new MenuWindowModel() { MenuWindow
= this });
            decrypt.Show();
        }

        private void ExitButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }

        private void DecImageButton_Click(object sender, RoutedEventArgs e)
```

```
        {
            var decrypt = new ImageDecWindow(new MenuWindowModel() {
MenuWindow = this });
            decrypt.Show();
        }

        private void EncImageButton_Click(object sender, RoutedEventArgs e)
        {
            var encrypt = new ImageEncWindow(new MenuWindowModel() {
MenuWindow = this });
            encrypt.Show();
        }
    }
}

namespace WPF_9.views.viewsmodels
{
    public class MenuWindowModel
    {
        public Window MenuWindow { get; set; }
    }
}
```