Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Стеганографија на слики со OpenCV и Python

- проектна активност -

Студенти:
Ребека Манева, 226133
Ева Кнежевиќ, 236009

Ментор:
Проф. д-р. Ивица Димитровски

Скопје, Јуни, 2025.

Апстракт

Во овој труд е разработена демо-апликација за стеганографија на слики користејќи Python и OpenCV. Апликацијата овозможува криење на една слика во друга преку два методи: **LSB метод** - манипулација со најмалку значајните битови на пикселите и **marker-based метод** - додавање на таен маркер и бајт-стринг на крај на фајлот. Опишани се основните алгоритми, псевдокод, структура на модулите и графички прикази на влезно-излезниот проток на податоци. За секој метод е спроведено тестирање со различни формати и големини на слики, при што е анализиран квалитетот на стего-сликите преку PSNR мерка и визуелна евалуација. Резултатите укажуваат на ниско видливо губење на квалитет и поедноставена имплементација, што го прави предлог-решението погоден инструмент за едукација и лесни апликации за заштита на податоци.

Содржина

Листа на табели

Листа на слики

Вовед

Стеганографијата е вештина и наука за крије на информација во рамки на други носечки медиуми, при што самите медиуми не предизвикуваат сомнеж дека содржат тајни податоци. Во ера на дигитална комуникација, каде размената на слики е сè почеста, потребата за дискретно пренесување на чувствителни податоци расте. Стеганографијата на слики ја користи способноста на човечкото око да не забележи мали промени во пикселските вредности, овозможувајќи пренесување на тајни пораки без привлекување внимание. Целта на оваа демо-апликација е да илустрира два популарни пристапа за стеганографија со слики, имплементирани во Python со помош на OpenCV библиотеката. Првиот метод, познат како LSB (најмалозначаен бит), се темели на манипулација на најмалозначајните битови во секој пиксел на основната („cover“) слика за да се вметнат битови од тајната („secret“) слика. Вториот, marker-based метод, додава посебен бајт-маркер на крајот на фајлот и потоа ги додава и бајтовите од тајниот фајл. Овие два алгоритми овозможуваат едноставна, но ефективна демонстрација на принципите на стеганографија, при што секој од нив има свои предности и ограничувања што ќе бидат детално дискутувани подоцна. Во оваа документација ќе биде опишана теоретската основа на стеганографијата, избраните технологии (Python, OpenCV, NumPy) и архитектурата на системот, вклучувајќи блок-дијаграми и тек на податоци. Потоа, во делот „Опис на алгоритмите“, ќе се презентира псевдокод и објаснување на сложеноста за двата метода. Следува длабинска анализа на имплементацијата во Python, со исечоци од код и објаснување на најважните функции, како и опис на графичкиот кориснички интерфејс. Во делот за тестирање ќе се прикажат резултати од мерки како PSNR и визуелна евалуација на квалитетот на стего-сликите. На крај, ќе се дадат заклучоци за изведбата и предлози за идно унапредување, вклучувајќи идеја за додавање криптографија или напредни трансформациски методи.

Теоретски основи на стеганографијата

Стеганографијата е дисциплина која проучува методи за скриено пренесување на информации, така што постоењето на пораката останува непрепознаено за неовластените набљудувачи. Додека криптографијата ја прави порака нечитлива, стеганографијата ја прави невидлива.

Дефиниција и кратка историја

- **Дефиниција.** Терминот „стеганографија“ доаѓа од грчките зборови *steganos* (покриен) и *graphein* (пишување). Основна цел е тајната порака да се вметне во некој друг медиум без да го наруши неговиот вообичаен изглед или перцепција.
- **Антички примери.** Првите записи датираат уште од античка Грција (скриени пораки на восочни табли), преку средновековните техники на микрографија (многу ситен текст впишан во буквите).
- **Модерна дигитална стеганографија.** Во 1980-тите години се појавуваат првите дигитални пристапи, кога компјутерската обработка стана достапна. Денес, поради широката употреба на слики и видео на интернет, стеганографијата на слики е еден од најраспространетите видови.

Типови стеганографија

LSB (Least Significant Bit) метод

- *Принцип:* Секој пиксел од дигиталната слика се состои од неколку битови. Најмалозначајниот бит (LSB) има најмало влијание врз бојата, па неговата промена е речиси невидлива.
- *Пример:* За 8-битен канал, вредноста 11001010 може да стане 11001011 без забележлива разлика. Секој пиксел може да „скрие“ од 1 до неколку битови тајни податоци.
- *Предности:* Висока капацитетност (многу бајти може да се вметнат) и едноставна имплементација.
- *Недостатоци:* Ниска робусност против компресија или агресивни филтри и лесно откривање со анализа на LSB-статистика.

Marker-based („EOF marker“) метод

- *Принцип:* Текстуален маркер (на пр. STEGANOMARKER) се додава на крајот од носечката слика, по што доаѓа целиот бајт-стрим од тајната слика. При декодирање, системот ги чита податоците по појавата на маркерот.
- *Предности:* Робустен за било какви промени на LSB-то – бидејќи не зависи од корекција на пиксели и поддржува криење на произволни бајтови.
- *Недостатоци:* Голема шанса да ја корумпира сликата ако се користи формат со заглавја (BMP, PNG) и ако не се третира правилно и јасно откривање доколку некој ја отвори датотеката и пребара за текст.

Критериуми за избор на методи во оваа апликација

- Цел на демо-апликацијата: Образовна демонстрација на два базични пристапи со лесно разбирање и видливо споредување на ефектите.
- Капацитет за тајни податоци: LSB нуди поголем капацитет за минимум промена на квалитетот, додека marker-based е погоден за криење на помали фајлови и бајтови.
- Робусност: Во оваа демо-верзија не се користи компресија. Marker-based е поедноставен за реализација, LSB е поинтересен за илустрација на промена на пикселите.
- Комплексност наспроти Читливост: LSB методот е едноставен за разбирање (операции со битови). Marker-based го покажува основниот концепт на апендирање на податоци во датотека. Двата методи се едноставни и кодот е лесно разбирлив.

Технологии и алатки

Во развојот на демо-апликацијата за стеганографија на слики се користеа следниве клучни технологии и практики:

- Python 3.11

Верзија: Апликацијата е тестирана на Python 3.11.9, но е компатибилна со сите верзии ≥ 3.7.
Причини за избор се богатиот систем на библиотеки за обработка на слики и научни пресметки, јасна и читлива синтакса погодна за едукативни демо-примери и брзото прототипирање и лесно споделување преку виртуелни околини.

- OpenCV 4.11

Инсталација: pip install opencv-python (во терминал на PyCharm)
Клучни модули:
  - imread() и imwrite() за читање и запис на слики.
  - Алгоритми за операции на матрици bitwise_or() bitwise_and().
  - Промена на големина на сликите со resize().

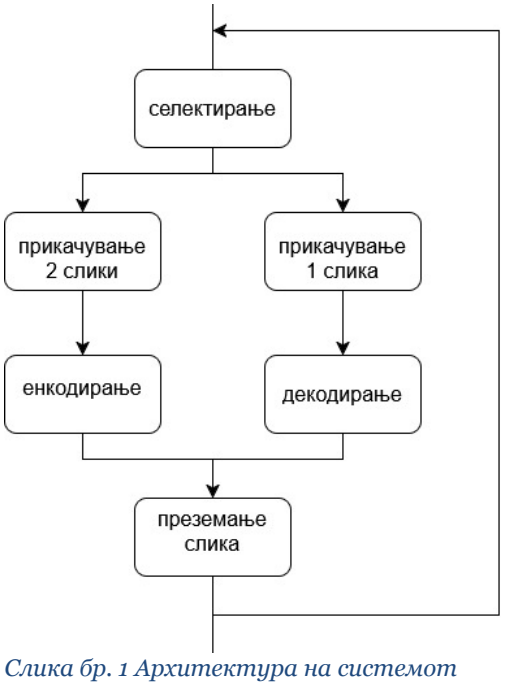
- NumPy 2.2.5

Инсталација: pip install numpy
Употреба во проектот:
  - Користејќи np.full, креирав numpy-низа
  - Брзо пресметување на PSNR

Архитектура на системот

Опис на дијаграмот

1. Селектирање - Корисникот преку GUI ја избира операцијата. За енкодирање: селектира две слики (cover + secret), за декодирање: селектира една слика (stego)
2. Прикачување на слики – Кај енкодирање делот корисникот ги прикачува и ги потврдува cover.jpg и secret.png, кај декодирање делот корисникот го прикачува и го потврдува stego.png
3. Енкодирање / Декодирање - Во encode модул: се вчитуваат прикачените слики, се применува LSB или marker-метод, се произведува stego.png. Во decode модул: се вчитува stego.png, се екстрахираат скриените податоци, се создава recovered_secret.png



Слика бр. 1 Архитектура на системот

4. Преземање на резултат - По успешна операција, GUI го прикажува и нуди копче за преземање на резултирачката слика stego.png или recovered_secret.png
5. Повторна операција - По преземањето или на барање на корисникот, системот се враќа на Селектирање, овозможувајќи нова операција.

Опис на алгоритмите

LSB Bitwise метод

Принцип на работа;

- Секоја пикселска компонента (R, G, B) на 8-битна слика е број од 0 до 255.
- Најмалозначајниот бит има најмало влијание врз перцепцијата на бојата.
- За секоја компонента на секој пиксел:
  - Се „бришат“ n најмалозначајни битови
  - Се вметнуваат n најмногузначајни битови од тајната слика на нивно место.

Резултат: човечко око не ја забележува промена во нијансите.

Анализа на сложеност

- Времена сложеност: O(M × N × 3), каде M×N е бројот на пиксели, а 3 е за RGB канали.
- Мемориска сложеност: O(M×N×C) за вчитаните две слики во меморија.
- Комплексност наспроти читливост – Имаме ниска имплементациска комплексност и висока читливост однсоно секој чекор е јасно видлив во псевдокодот.

Marker-based („EOF“) метод

Принцип на работа:

- Носечката (cover) слика се снима нормално
- По записот на крајната слика се отвора истиот фајл во бинарен режим (‘ab’).
- Се додава фиксна низа - „marker“ (на пр. "STEGANOMARKER").
- Потоа се додаваат сите бајтови од тајниот фајл.
- При декодирање, фајлот се вчитува како бајт-стрим, се пребарува marker-от и се чита сè што следи зад него.

Анализа на сложеност
Временска сложеност: O(M × N × 3)
Мемориска сложеност: O(M × N × 3)

Комплексност наспроти читливост – Имаме ниска имплементациска комплексност, само едноставни бит-операции (AND, shift, OR).
Имаме висока читливост, псевдокодот јасно ја прикажува секвенцата на чекори, тој е само 10–15 реда долг и лесно може да се
следи како секој бит од тајната слика се вметнува во најмалозначајниот бит на носечката слика.

Имплементација во Python

Во продолжение следи детално објаснување на функциите и секоја линија од кодот за LSB и за Marker-based (EOF) методите

LSB метод

Енкодирање

Импортирање библиотеки:
- cv2: OpenCV – за читање/запишување и обработка на слики.
- numpy: за создавање и манипулација со n-димензионални низи (мајтрикси).

- Функција за енкодирање со 3 патеки (носечка слика ,тајна слика, стего слика)

Вчитување на сликите:
- **cv2.imread** е функција која вчитува слика, на неа ја предаваме патеката до сликата.
- Доколку фајлот не постои или форматот не се поддржува, резултатот е None.

Чување оригинални димензии:
- Зачувување на оригиналната висина (orig_h) и ширина (orig_w) на тајната слика за да можеме подоцна да ја ресајзираме назад.

Менување големина на слика:
- **cv2.resize**: го прилагодува secret да има иста големина како cover.
- **INTER_LANCZOS4**: високо-квалитетен алгоритам за менување големина на слика.

Исчитување на најмалку значајните 4 бита од носечката слика:
- **np.full(cover.shape, 240, dtype=np.uint8)**: создава маска 240 (0b11110000) со иста големина како cover.
- **cv2.bitwise_and**: врши бит-AND помеѓу cover и маската, така што долните 4 бита од секој пиксел (LSB) се обнуваат.

Намалување на битовите од тајната слика:
- Ова го „поместува“ секој пиксел 4 бита надесно, па неговите горни 4 бита се подготвени да бидат вметнати во долните 4 бита на cover.

Комбинирање на двете слики:

- **cv2.bitwise_or** ги спојува исчистените горни бита од cover со снижените битови од secret, создавајќи ја стего сликата.

Вградба на оригиналните димензии во пикселите:

- ги запишуваме оригиналните ширина и висина на тајната слика во првите два пиксели на стего-сликата (по два бајта за секоја димензија), за при декодирање лесно да ја вратиме сликата на нејзината првобитна големина.

Запис на стего-сликата:

- **cv2.imwrite** ја зачувува готовата стего-слика на диск

Декодирање

- Дефинираме функција за декодирање која ја враќа тајната слика од дадена стего слика.

- Чита стего слика со **cv2.imread**

- ги читаме првите два пиксели (димензиите) на стего-сликата, извлекуваме по два бајта (горен и долен) за ширина и висина, и ги комбинираме (<<8 |) за да ја добиеме оригиналната резолуција на тајната слика.

- Екстракција на долните четири бита:
  0b00001111 (15): маска што го оставува само LSB-то од секој пиксел.
  bitwise_and екстрахира оригиналните битови на тајната слика.

- Поместување на екстрахираните 4 бита четири позиции лево, за да се добијат вистинските пиксел вредности.

- Ја враќаме тајната слика во нејзината првобитна димензија

- Запишувањето на крајната „recovered“ слика на диск.

```python
import cv2
import numpy as np

def encode_image(cover_path, secret_path, output_path):

    cover = cv2.imread(cover_path)
    secret = cv2.imread(secret_path)

    orig_h, orig_w = secret.shape[:2]

    secret = cv2.resize(secret, dsize:(cover.shape[1], cover.shape[0]), interpolation=cv2.INTER_LANCZOS4)

    cover_cleared = cv2.bitwise_and(cover, np.full(cover.shape, fill_value:240, dtype=np.uint8))

    secret_shrunk = secret >> 4

    stego = cv2.bitwise_or(cover_cleared, secret_shrunk)

    stego[0, 0, 0] = (orig_w >> 8) & 0xFF
    stego[0, 0, 1] = orig_w & 0xFF
    stego[0, 1, 0] = (orig_h >> 8) & 0xFF
    stego[0, 1, 1] = orig_h & 0xFF

    cv2.imwrite(output_path, stego)
```

Слика бр. 3 LSB Енкодер

```python
def decode_image(stego_path, output_path):
    stego = cv2.imread(stego_path)

    high_w = int(stego[0, 0, 0]);
    low_w = int(stego[0, 0, 1])
    high_h = int(stego[0, 1, 0]);
    low_h = int(stego[0, 1, 1])
    orig_w = (high_w << 8) | low_w
    orig_h = (high_h << 8) | low_h

    extracted_bits = cv2.bitwise_and(stego, np.full(stego.shape, fill_value:15, dtype=np.uint8))

    secret = extracted_bits << 4

    recovered = cv2.resize(secret, dsize:(orig_w, orig_h), interpolation=cv2.INTER_LANCZOS4)
```

Слика бр. 2 LSB Декодер

Marker-based метод

Енкодирање

Импортирани библиотеки:

- cv2: OpenCV – за читање/запишување и обработка на слики, како и за создавање и манипулирање со дигитални слики (матрици).

Функција за енкодирање со три параметри (патека до слика носител, патека до слика која ја криеме, патека до добиената слика)

Читање на сликата:

- **cv2.imread(cover_path)** ја чита сликата од дадената патека и ја складира во cover_img.
- Ако сликата не може да се отвори, се враќа None.

Проверува дали сликата е успешно прочитана. Ако не е, функцијата се прекинува.

Запишување на излезната слика:

- **cv2.imwrite(output_path, cover_img)** запишува cover_img во новата слика на output_path, што ќе биде основната слика која ќе ја содржи тајната слика.

Додавање на тајната слика во носителот:

- **open(output_path, 'ab')** отвора нова слика за додавање бинарни податоци на крајот (ab – append binary).
- **open(secret_path, 'rb')** отвора тајната слика за читање во бинарен формат.
- **f_out.write(b'STEGANOMARKER')** додава еден маркер (бинарни податоци) во носителот за идентификација.
- **f_out.write(f_secret.read())** додава содржината на тајната слика во носителот.

Декодирање

Функција за декодирање:

- decode е функција која прима 2 параметри:

stego_path: патека до сликата која содржи тајната информација.
recovered_secret_path: патека каде ќе се зачува откриената тајна слика.

Читање на сликата:

- **open(stego_path, 'rb')** отвора слика за читање во бинарен формат.
- **f.read()** чита целосната содржина на сликата која потоа се сместува во „data“.

Пребарување на маркерот:

- **marker**: Дефинира маркерот за тајната слика.
- **data.find(marker)**: Пребарува дали маркерот се наоѓа во **data** и го враќа неговиот индекс.

Проверка за валидноста на маркерот така што ако тој не е пронајден функцијата ќе се прекине.

Извлекување на тајната слика:

- **secret_data** = data[marker_index + len(marker):]
  извлекува податоците кои следат по маркерот, што го содржат секретот.

Запишување на тајната слика:

- **open(recovered_secret_path, 'wb')** отвора патека за запишување на тајната слика.
- **f.write(secret_data)** запишува податоци од тајната слика во новиот фајл.

```python
import cv2

def encode(cover_path, secret_path, output_path):  1 usage  RebekaManeva
    cover_img = cv2.imread(cover_path)
    if cover_img is None:
        return

    cv2.imwrite(output_path, cover_img)

    with open(output_path, 'ab') as f_out, open(secret_path, 'rb') as f_secret:
        f_out.write(b'STEGANOMARKER')
        f_out.write(f_secret.read())

def decode(stego_path, recovered_secret_path):  new *
    with open(stego_path, 'rb') as f:
        data = f.read()

    marker = b'STEGANOMARKER'
    marker_index = data.find(marker)

    if marker_index == -1:
        return

    secret_data = data[marker_index + len(marker):]

    with open(recovered_secret_path, 'wb') as f:
        f.write(secret_data)
```

Слика бр. 4 EOF код

Кориснички интерфејс (GUI)

Библиотека

Оваа апликација е развиена со помош на PyQt5 — една од најпопуларните библиотеки за креирање графички кориснички интерфејси (GUI) во Python.

PyQt5 нуди голем број на готови контроли (widgets), лесно поврзување на настани (events) со функции и овозможува брзо развивање на модерни, стабилни и професионални GUI апликации. Библиотеката базира на Qt framework, кој е широко користен и за развој на мултиплатформски апликации.

Работата со PyQt5 во оваа апликација овозможува едноставно уредување на распоредот на елементите, прецизна контрола на стиловите и динамично управување со состојбата на екранот без потреба од комплексна манипулација со координати или рачно цртање.

Типови на користени Widgets

- Qwidget

Основен контејнер кој ги држи сите други елементи во апликацијата.
Тој служи како главен прозорец (App) во кој се вметнати сите останати визуелни компоненти. Секој елемент е поставен како под-widget на главниот QWidget.

- QpushButton

Користени за интеракција со корисникот — копчињата „Encrypt“, „Decrypt“, „Hide (LSB)“, „Hide (EOF)“, „Expose (LSB)“, „Expose (EOF)“ и „Download Result“. Во оваа апликација, копчињата служат за:

- Иницирање на операции за криење или откривање податоци во слика;
- Визуелно се менуваат (боја) за да покажат која опција е активна;
- Се ресетираат во нормална состојба при промена на режимот;

- Qlabel

Овој widget е користен за:

- Приказ на слики (резултатите од енкрипција или декрипција);
- Приказ на икони и инструкции во upload полињата;

Labels се поставени така што имаат фиксна големина, за да не се менува распоредот при вчитување нови слики. Празно Label поле се користи и за "резервирање простор" каде подоцна ќе се прикажат резултатите.

- QVBoxLayout и QHBoxLayout

Layout-ите се користат за автоматско организирање на елементите така што: QVBoxLayout за вертикално наредување на делови (горно мени, upload полиња, акциски копчиња, резултат, download копче). QHBoxLayout за

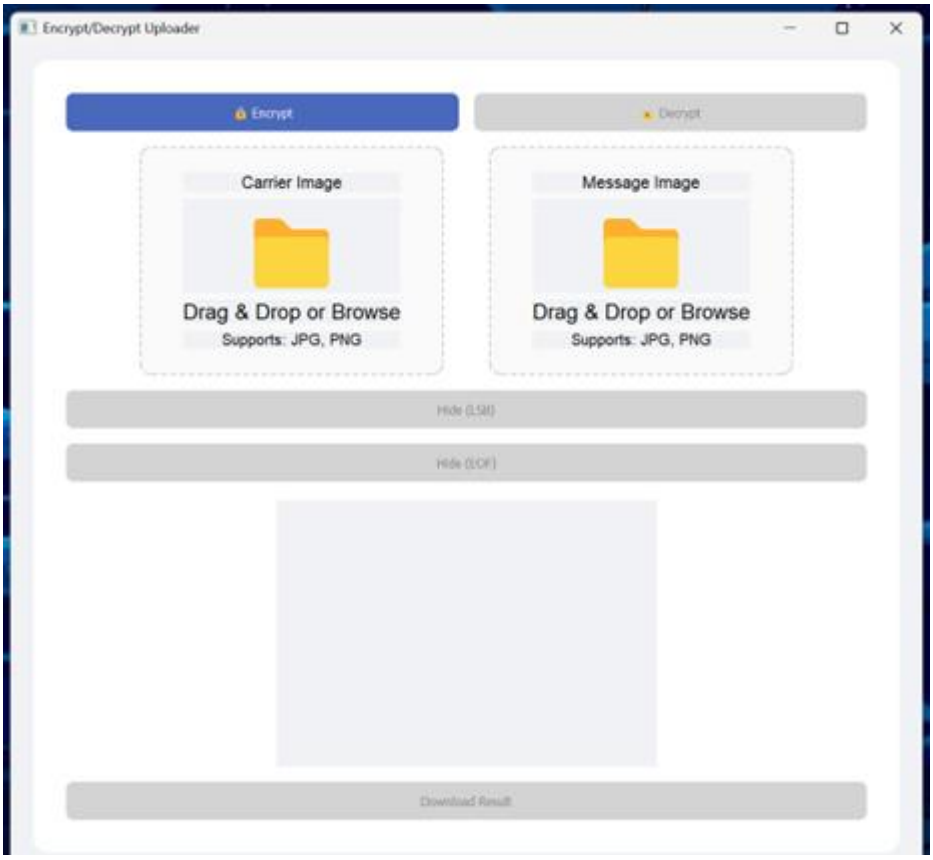
поставување на upload полињата едно до друго со центрирање.
Stretch елементи се користени во layout-ите за да се постигне урамнотежено центрирање на елементите.

- QFrame

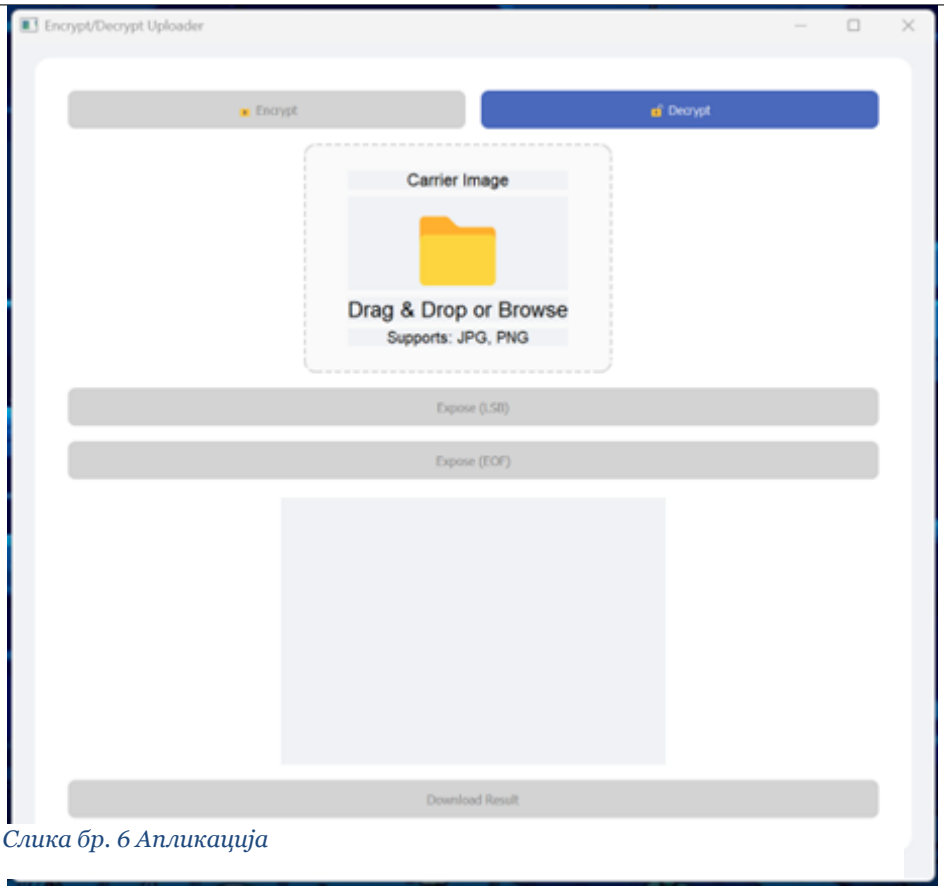
Користено како бела позадина (container) која го држи целиот GUI внатре во прозорецот. QFrame служи за визуелно одвојување на апликацијата од сивата позадина на прозорецот и овозможува контролирање на внатрешниот распоред со padding и radius стилови.

Нашата апликација

На почеток, се поставуваат главните toggle-копчиња ("Encrypt" и "Decrypt") на врвот на прозорецот. Под нив, преку хоризонтален layout, се распоредуваат upload полињата за слики. Во зависност од избраниот режим едно upload полињата се сокрива. Под upload полињата, се поставуваат акциските копчиња за избор на методот при енкрипција или декрипција. Резултатот од акцијата се прикажува во голем фиксно-центриран Label со поддршка за прикажување на слики. Постои изборот за преземање на сликата на нашиот уред со копчето “Download”. Овој кориснички интерфејс е интуитивен, со тоа што копчињата се во состојба disabled доколку некоја операција е задолжително да се направи пред да се кликне тоа копче. Пример за ова се “Hide” копчињата, тие не се активни доколку не се прикачат две слики.

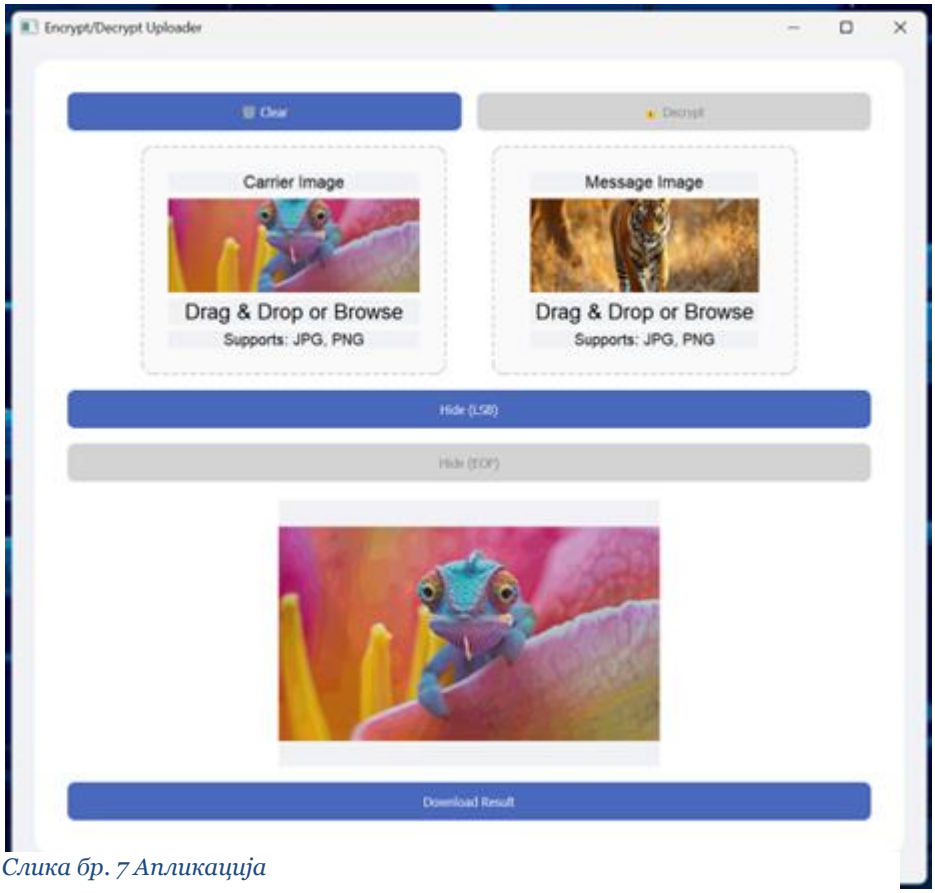


Слика бр. 5 Апликација



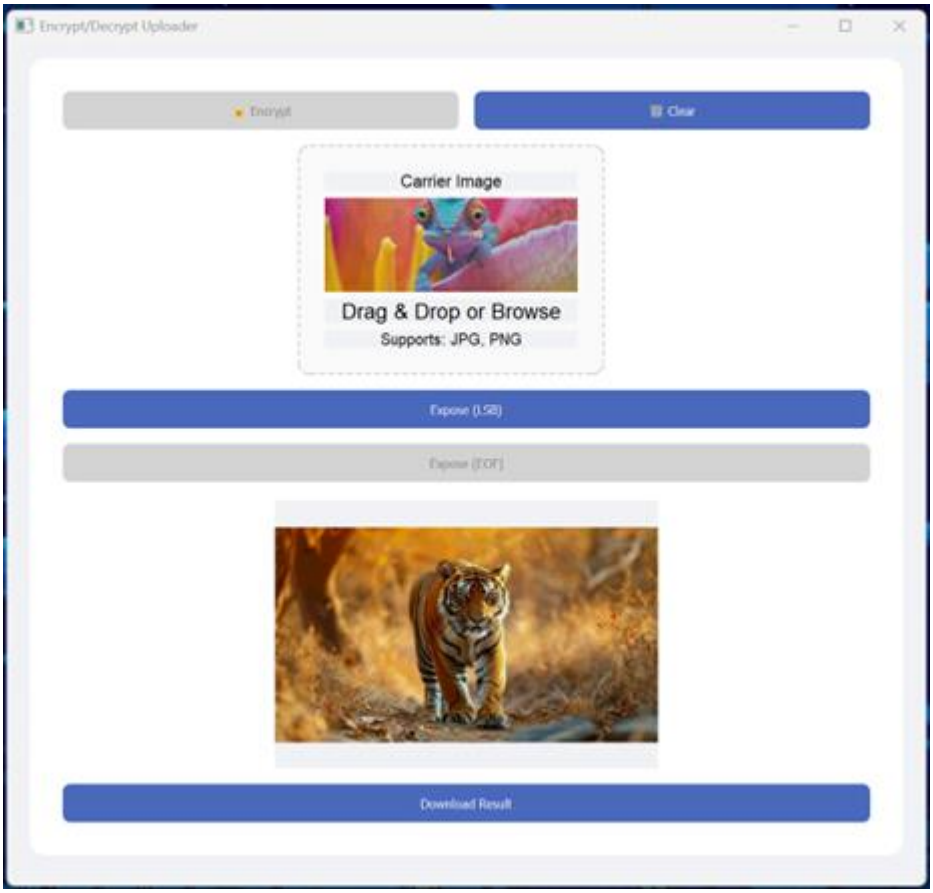
Слика бр. 6 Апликација

На Слика бр.1 и Слика бр.2 се прикажани двата режима на апликацијата, односно Encrypt и Decrypt делот соодветно.



Слика бр. 7 Апликација

На Слика бр.3 е прикажана состојбата каде имаме режим на енкрипција, се прикачени две слики преку Drag-and-Drop опцијата и е кликнато копчето “Hide (LSB)”. Во самиот прозорец ни се прикажува добиената слика со скриената порака. Дозволена ни е опцијата да ја преземеме сликата. Копчето “Clear” служи за отстранување на сите досегашни потези, односно не враќа повторно на состојбата од Слика бр.1.



Слика бр. 8 Апликација

На Слика бр.4 ни е прикажана состојбата каде имаме режим на декрипција, ни е прикачена сликата добиена како резултат на Слика бр.3 и е притиснато копчето “Expose (LSB)”. На прозорецот ни се прикажува екстрахираната скриена слика со понудената опција за преземање на истата. Со копчето “Clear” повторно се враќаме на состојбата прикажана на Слика бр.2.

Тестирање и евалуација

Во рамки на тестирањето беше спроведена серија експерименти со цел да се евалуира прецизноста, робусноста и визуелниот квалитет на добиените стего-слики, користејќи ги двата имплементирани метода: LSB (Least Significant Bit) и Marker-based (EOF marker).

Примери на тест-случаи

Тестирањето опфати:

- Слики од различни димензации: 128×128, 256×256, 512×512 и 1024×1024.
- Формати: PNG (без загуба на податоци), JPG (со компресија).
- Типови на слики: еднобојни (бела/црна подлога), фотографии со природни сцени, цртежи, шарени слики со висока комплексност.

Кај LSB методот беше утврдено дека најдобри резултати се постигнуваат кога се крие податок во шарени слики со многу детали и нијанси. Во такви случаи, дури и со внесување на 4 битови во секој пиксел, немаше никаква видлива деградација на квалитетот.

Кај Marker-based методот, успешноста не зависеше од содржината на сликата, туку од тоа дали користениот формат поддржува апендирање на податоци без нарушување. PNG работеше стабилно, а JPG повремено даваше грешки поради тоа што со загуба и понекогаш EOF бајтите се третираа како невалидни .

Мерки на визуелен квалитет

За мерење на квалитетот на стего-сликите беше користена метриката PSNR (Peak Signal-to-Noise Ratio**)**, која го споредува оригиналот и резултатот. Се применуваше само на LSB методот бидејќи кај marker-based методот сликата останува непроменета на пикселско ниво.

- Во сите тестирани примери, PSNR беше поголем од 30 dB, што укажува на висок квалитет без видливо губење.
- Во повеќето случаи со 4-битно LSB енкодирање, PSNR беше меѓу 30 и 40 dB, со просечна вредност од околу 36 dB.
- Визуелната проверка покажа дека човечкото око не може да забележи разлика дури ни при директна споредба.

Тип на слика	Формат	Метод	PSNR	Видливо губење
Портрет во боја	PNG	LSB	38.2 dB	Нема
QR-код во боја	JPG	LSB	35.8 dB	Минимално
Шарена сцена	PNG	LSB	39.6 dB	Нема
Црно-бела слика	JPG	LSB	30.1 dB	Лесно видливо
Шарена сцена	PNG	Marker-based	N/A	Нема

Табела 1 PNSR вредности

Време на извршување

Покрај визуелниот квалитет, беше измерено и времето потребно за енкодирање и декодирање.

Метод	Енкодирање (сек.)	Декодирање (сек.)
LSB	<0.5	околу 2
EOF	<0.1	<1.5

Табела 2 Време на извршување на програмата

Временската сложеност е ниска, а апликацијата реагира брзо и без забележливи доцнења, дури и на стандардни лаптопи без GPU забрзување. Ова го прави решението многу практично и употребливо во реални услови.

Заклучоци од тестирањето

- LSB методот работи најдобро со слики што имаат многу детали и бои — на пример, фотографии од природа, уметнички слики и портрети.
- Кај еднобојни слики, промените во битовите се лесно забележливи, па тие не се препорачуваат како cover слики.
- Marker-based методот е стабилен и едноставен, но не е погоден за формати со заглавја (JPG), каде може да се изгуби додадениот бајт-стрим.

Заклучок и идни правци

Заклучок

Во рамки на овој проект беше развиена едукативна демо-апликација за стеганографија на слики, користејќи два различни пристапи: LSB (Least Significant Bit) и Marker-based метод. Преку имплементација во Python со библиотеките OpenCV и NumPy, успешно се демонстрираше како тајна слика може да се вметне и подоцна да се извлече од cover слика, без значително нарушување на визуелниот квалитет.
Спроведеното тестирање покажа дека:

- LSB методот овозможува висок капацитет ирелативно висок PSNR (> 35 dB) кога се користат шарени, детални слики.
- Marker-based методот е поедноставен, но бара внимателен избор на формат, бидејќи компресирачки формати како JPG можат да ја уништат скриената порака.
- Обата методи имаат едноставна имплементација, што ги прави соодветни за образовни цели и демонстрации на основни принципи на стеганографија.

Целите на проектот беа постигнати успешно, а добиеното решение е стабилно, читливо и лесно за користење преку GUI изграден со PyQt5.

Идни правци

Иако тековната верзија претставува функционален и едукативен прототип, постојат неколку можности за надградба и унапредување:

- Криптографска заштита: Пред да се вметне тајната слика, може да се додаде чекор за енкрипција што значително ќе ја зголеми безбедноста.
- Transform-based стеганографија: Наместо директна манипулација со пиксели, може да се истражат DCT (како кај JPEG) или FFT базирани пристапи за поголема робустност.
- Поддршка за видео: Истите принципи може да се прошират на видеа — на пр. вметнување на слики во поединечни фрејмови.
- Мобилна апликација: Со помош на PyQt6 или Kotlin/Flutter, може да се развие верзија за Android или iOS, достапна за поширока употреба.

Со имплементација на овие надградби, апликацијата би добила нова димензија и би можела да се користи не само во едукација, туку и во реални сценарија каде е потребна сигурна и невидлива размена на информации.

Користена литература

1. Provos, N., & Honeyman, P. (2003). *Hide and seek: An introduction to steganography*. IEEE Security & Privacy, 1(3), 32–44. https://doi.org/10.1109/MSECP.2003.1203220
2. Petitcolas, F. A. P., Anderson, R. J., & Kuhn, M. G. (1999). *Information hiding—A survey*. Proceedings of the IEEE, 87(7), 1062–1078. https://doi.org/10.1109/5.771065
3. OpenCV (n.d.). *OpenCV documentation*. Retrieved from https://docs.opencv.org/
4. NumPy Developers. (n.d.). *NumPy documentation*. Retrieved from https://numpy.org/doc/
5. PyQt5 Documentation. (n.d.). *PyQt5 Reference Guide*. Retrieved from https://www.riverbankcomputing.com/static/Docs/PyQt5/
6. Python Software Foundation. (n.d.). *Python 3.11 documentation*. Retrieved from https://docs.python.org/3/
7. Bender, W., Gruhl, D., Morimoto, N., & Lu, A. (1996). *Techniques for data hiding*. IBM Systems Journal, 35(3.4), 313–336.
8. Khan, M., & Ahmad, R. (2020). *A Comparative Study of Image Steganography in Frequency and Spatial Domain*. International Journal of Computer Applications, 975(8887).