Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Іван ПАРХОМЕНКО
«____» червня 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)
спеціальність 125 Кібербезпека
(код і назва спеціальності)
освітній ступень бакалавр
освітня програма Кібербезпека
(назва освітньо-професійної програми)
на тему: Програмний модуль для збереження логінів і паролів в гешованому вигляді

Виконавець: студент IV курсу, групи КБ-41

_____ **Яценко ДМИТРО**
(підпис) (ім'я, прізвище)

	Ім'я, прізвище	**Підпис**
Керівник	Яніна ШЕСТАК	

Нормоконтроль	Сергій ДАКОВ	

Київ 2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-41** (група) **Яценку Дмитру Олександровичу** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Програмний модуль для збереження логінів і паролів в гешованому вигляді

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №3 від 20.10.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Концепція геш-функцій, методи стеганографії, генерація надійних паролів

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно ознайомитися з теорією геш-функцій, генераторів паролів, вияснити способи їх застосування
та поєднання з додатковими методами забезпечення безпеки як стеганографія, створити програмний модуль збереження логінів і паролів в
гешованому вигляді, який стане основою для використання цієї інформації в
системі автентифікації.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розроблений програмний модуль для збереження логінів паролів в гешованому вигляді

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 24 жовтня 2023 року

Завдання видала _____ (підпис) Яніна ШЕСТАК (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Яценко ДМИТРО (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	24.10.2022 – 22.01.2023	*виконано*
2	Аналіз літератури	26.01.2023 – 21.02.2023	*виконано*
3	Обґрунтування вибору рішення	29.02.2023 – 7.03.2023	*виконано*
4	Огляд геш-функцій	11.03.2023 – 18.03.2023	*виконано*
5	Опис використаних застосунків	21.03.2023 – 27.03.2023	*виконано*
6	Проектування генератора паролів	2.04.2023 – 08.04.2023	*виконано*
7	Створення програмного модуля для збереження логінів і паролів в гешованому вигляді	09.04.2023 – 12.05.2023	*виконано*
8	Оформлення пояснювальної записки	15.05.2023 – 24.05.2023	*виконано*
9	Підготовка до захисту кваліфікаційної роботи	27.05.2023 – 12.06.2023	*виконано*

Завдання видала _____ (підпис) Яніна ШЕСТАК (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Яценко ДМИТРО (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 12 червня 2023 року

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмний модуль для збереження логінів і паролів в гешованому вигляді» складається зі вступу, основної частини, що містить 3 розділи, висновків і списку літератури та джерел. Загальний обсяг роботи 71 сторінка. Робота містить 62 рисунки, 1 формулу. Список використаних джерел включає 20 джерел.

Мета роботи – Розробка програмного модуля для збереження логінів і паролів в гешованому вигляді, який поєднає в собі три основних компоненти: хеш-функції, генератор паролів та LSB-стеганографію.

Для досягнення мети треба виконати наступі етапи:

- Проаналізувати методи генераторів паролів, стеганографії, геш-функції та способів автентифікації;

- Дослідити інструменти для розробки програмного модуля з метою збереження логінів і паролів в гешованому вигляді

- Розробка системи програмного модуля для збереження логінів і паролів в гешованому вигляді.

Об'єкт дослідження - процес збереження логінів і паролів в гешованому вигляді.

Предмет дослідження – методи та засоби збереження інформації в гешованому вигляді для програмного модуля.

Методи дослідження:

спостереження, порівняння, узагальнення, абстрагування, формалізація, аналіз і синтез.

Практичне значення роботи – полягає у створені програмного модуля для збереження логінів і паролів в гешованому вигляді, що може стати основою для створення системи ідентифікації.

Результати здійснених у дипломній роботі досліджень можуть бути використані спеціалістами із захисту інформації та при подальшому проведені науково-дослідницьких робіт.

Напрямки подальших досліджень – інтегрування програмного модуля в різні сферах, включаючи інформаційні технології, банківську сферу, електронну комерцію та багато інших.

Ключові слова: Програмний модуль, гешований вигляд, геш-функції, автентифікація, база даних.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ПЗ	–	програмне забезпечення
БД	–	База даних
UUID	–	Universal Unique Identifier
IDE	–	integrated development environment
MD5	–	message-digest algorithm
SHA	–	Secure Hash Algorithms
GUI	–	Graphical user interface
PIL	–	Python Imaging Library
RGB	–	red, blue and green LEDs
ASCII	–	American Standard Code for Information Interchange

ЗМІСТ

ВСТУП

У сучасному світі, де цифрові технології проникають в усі сфери життя, безпека особистих даних є надзвичайно важливою. Одним з найпоширеніших методів захисту особистої інформації є збереження логінів і паролів у гешованому вигляді. Використання хеш-функцій, генераторів паролів та стеганографії може забезпечити високий рівень безпеки та надійності систем, де зберігається така інформація.

Ця дипломна робота є актуальною та має велике значення для сфери інформаційних технологій, оскільки безпека особистих даних є однією з найважливіших проблем сучасного цифрового світу. Розробка програмного модуля, який поєднує різні компоненти безпеки, внесе вагомий внесок у підвищення рівня безпеки автентифікації та захисту особистих даних.

Мета роботи – Розробка програмного модуля для збереження логінів і паролів в гешованому вигляді, який поєднає в собі три основних компоненти: хеш-функції, генератор паролів та LSB-стеганографію.

Об'єкт дослідження - процес збереження логінів і паролів в гешованому вигляді.

Предмет дослідження – методи та засоби збереження інформації в гешованому вигляді для програмного модуля.

Методи дослідження:

спостереження, порівняння, узагальнення, абстрагування, формалізація, аналіз і синтез.

Практичне значення роботи – полягає у створені програмного модуля для збереження логінів і паролів в гешованому вигляді, що може стати основою для створення системи ідентифікації.

Результати здійснених у дипломній роботі досліджень можуть бути використані спеціалістами із захисту інформації та при подальшому проведені науково-дослідницьких робіт.

Напрямки подальших досліджень – інтегрування програмного модуля в різні сферах, включаючи інформаційні технології, банківську сферу, електронну комерцію та багато інших.

Для досягнення мети треба виконати наступі етапи:

- Проаналізувати методи генераторів паролів, стеганографії, геш-функції та способів автентифікації;
- Дослідити інструменти для розробки програмного модуля з метою збереження логінів і паролів в гешованому вигляді
- Розробка системи програмного модуля для збереження логінів і паролів в гешованому вигляді.

Очікується, що результати цієї дипломної роботи не лише допоможуть в розробці надійних систем автентифікації, але й сприятимуть покращенню загальної безпеки та захисту особистих даних. Результати досліджень та розробок можуть бути використані в різних сферах, включаючи інформаційні технології, банківську сферу, електронну комерцію та багато інших.

Окрім того, це дослідження може стати основою для подальших наукових робіт та подальшого розвитку методів захисту інформації. Результати цієї дипломної роботи можуть бути використані як основа для подальших досліджень у сфері криптографії, стеганографії та систем автентифікації.

Ключові слова: Програмний модуль, гешований вигляд, геш-функції, автентифікація, база даних.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ГЕНЕРАТОРІВ ПАРОЛІВ, СТЕГАНОГРАФІЇ, ГЕШ-ФУНКЦІЇ ТА СПОСОБІВ АВТЕНТИФІКАЦІЇ

1.1 Огляд геш-функції

Геш-функція - математичний алгоритм, який перетворює довільний масив даних, який складається з цифр та літер будь-якого розміру в дані фіксованого розміру. Його особливість в тому, що при використанні того ж типу хешу, ця довжина залишатиметься незмінною, незалежно від обсягу вступних даних. Геш-функція повинна забезпечувати умовам криптостійкості, яка настає при виконанні двох особливостей:

1.Стійкість до відновлення даних, що хешуються.

2.Стійкість до колізій.

Річ у тім, що жоден з наявних алгоритмів не попадає під ці критерії через те, що знаходження зворотнього хешу - питання лише обчислювальних потужностей, але у випадку з деякими особливо просунутими алгоритмами, цей процес може займати багато часу. Те, що стосується колізії, то жоден алгоритм не може гарантувати її відсутність проте, вони можуть зменшити відсоток його появи при її роботі з різними текстами. (Рисунок. 1.1)

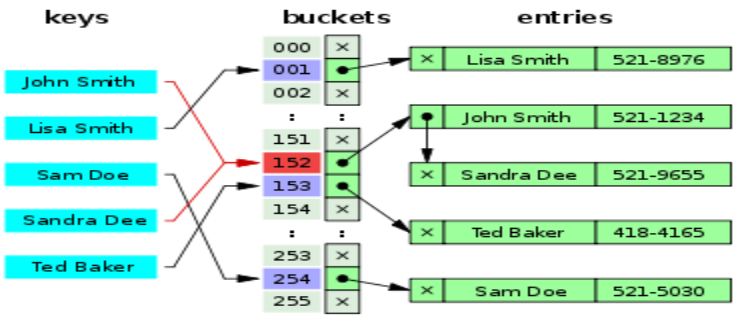

Рисунок 1.1 Приклад колізії при використанні хеш-функції

Хеш працює наступним чином. Якщо взяти слово “діма” користуючись алгоритмом SHA-256, то в результаті ми отримаємо наступний хеш:*a4e82a6821fb5b4db32a09b0053d0e635d339c5ea3d82e2c143c7ee9308f56f*. Проте, якщо взяти слово “Діма”, то ми отримаємо наступне: *74bbc0848483ce6ca6b33ac82d3edaca14a6b68798d5ac949124b60f8d2288a*. З цього можна бачити те, що найменша зміна в текстовому вигляді призводить до значних змін функції хешу.

Існують ще декілька алгоритмів геш-функцій окрім SHA-256:

MD5 - це один з найпоширеніших алгоритмів гешування. Він генерує геш-значення довжиною 128 біт з вхідних даних будь-якої довжини. MD5 використовується для перевірки цілісності даних та як засіб забезпечення безпеки. Проте, станом на 2023 рік, MD5 вважається небезпечним алгоритмом, оскільки на нього можливі атаки з використанням колізій, тобто знаходження двох різних вхідних даних, які дають одне й те ж геш-значення. Тому, використання MD5 для захисту конфіденційних даних не рекомендується, і на його місце рекомендується використовувати більш безпечні алгоритми гешування, такі як:

SHA-2 або SHA-3. SHA-2 і SHA-3 генерують геш-значення довжиною від 224 до 512 біт залежно від використаної версії. Вони є сучасними алгоритмами гешування, які використовуються для забезпечення безпеки та перевірки цілісності даних.

SHA-2 складається з декількох різних алгоритмів гешування, таких як SHA-224, SHA-256, SHA-384 та SHA-512, кожен з яких генерує геш-значення різної довжини. SHA-2 вважається безпечним алгоритмом та досі широко використовується для захисту конфіденційної інформації, такої як паролі, ключі, цифрові підписи та інші.

SHA-3 є останньою версією алгоритму SHA, який був розроблений з метою забезпечення ще більшої безпеки та ефективності. SHA-3 використовує алгоритм “Keccak”, який був вибраний після конкурсу на розробку нового стандарту гешування NIST. SHA-3 забезпечує високий рівень безпеки та ефективності, але на цей момент не є таким поширеним, як SHA-2.

Bcrypt - це алгоритм гешування, спеціально розроблений для зберігання паролів. Він використовує різні техніки гешування та затримки для ускладнення атак методом перебору паролів. Bcrypt генерує геш-значення довжиною 192 біти з вхідних даних будь-якої довжини.

У будь-якому випадку, для захисту конфіденційних даних рекомендується використовувати надійні алгоритми гешування, такі як SHA-2 або SHA-3, а не застарілі алгоритми, такі як MD5.

1.2 Властивості геш-функцій

Геш-функції мають ряд властивостей, які забезпечують їхню ефективність та захист від різних атак. Основні властивості геш-функцій:

1.Відображення будь-яких даних будь-якої довжини в геш-значення фіксованої довжини. Це забезпечує можливість збереження геш-значень в обмеженому просторі, що спрощує їх збереження та обробку.

2.Одностороння функція, що означає, що від вихідного геш-значення неможливо знайти вхідні дані. Це забезпечує безпеку перетворених даних та можливість збереження паролів без ризику їхнього розголошення.

3.Наявність властивості стійкості до зіткнень - це означає, що ймовірність зіткнення (колізії) двох різних вхідних даних, які генерують однакове геш-значення, дуже мала. Це забезпечує цілісність даних та захист від атак, що ґрунтуються на зіткненні.

4.Наявність детерміністичності, тобто один і той же вхід буде завжди генерувати одне й те ж геш-значення, що дозволяє легко перевіряти цілісність даних.

5.Важливою властивістю геш-функцій є відсутність залежності геш-значення від вхідних даних. Це означає, що навіть найменша зміна вхідних даних призведе до повного змінювання геш-значення. Ця властивість забезпечує надійний захист від зловмисних дій, таких як внесення змін у дані без знання правильного геш-значення.

6.Швидкість роботи. Геш-функції повинні бути достатньо швидкими, щоб їх можна було використовувати в реальному часі. Найбільш поширені алгоритми хешування, такі як MD5 та SHA, є досить швидкими та можуть обробляти великі об'єми даних за короткий час.

1.3 Використання геш-функцій у системах автентифікації

Геш-функції є важливим елементом безпеки в багатьох системах автентифікації. Найбільш поширеним прикладом є зберігання паролів користувачів. Замість того, щоб зберігати паролі у відкритому вигляді, система зберігає їх геш-значення. Коли користувач вводить свій пароль, система хешує його та порівнює отримане геш-значення з тим, що зберігається в базі даних. Якщо геш-значення збігається, то користувачу надається доступ до системи. Збереження паролів з використанням хешу полягає в наступних кроках:

1.Створіть сіль (salt): це випадковий рядок байтів, який додається до пароля перед застосуванням. Сіль використовується для того, щоб ускладнити атаки на пароль за допомогою радужних таблиць (rainbow tables) і завадити використання одного і того ж пароля для різних користувачів.

2.Застосуйте хеш для генерації хешу пароля: використовуйте пароль і сіль, яку ви створили на попередньому кроці, як вхідні дані. В результаті ви отримаєте хеш, який можна зберегти в базу даних.

3.Збережіть хеш та сіль в базу даних: Потрібно зберігати сіль та хеш пароля разом в базі даних.

4.Перевірте пароль: коли користувач вводить пароль для входу, виконайте ті ж самі кроки, що і для генерації хешу пароля, використовуючи збережену сіль. Порівняйте отриманий хеш зі збереженим в базі даних хешем. Якщо хеші збігаються, пароль введено правильно, і користувачу надається доступ до системи.

1.4 Генератори паролів

Генератори паролів - це програми або інструменти, які допомагають створити випадковий пароль для використання в системі або сервісі. Вони можуть генерувати паролі різної довжини та складності, включаючи великі та малі літери, цифри та спеціальні символи.

Використання генераторів паролів у системах автентифікації полягає у тому, що генератор створює випадкові паролі, які є складніші для зламування, ніж прості паролі, створені користувачами. Такі інструменти дуже корисні для захисту ваших особистих даних та запобігання несанкціонованому доступу до вашої інформації. Генератори паролів зазвичай дають можливість налаштувати складність пароля, включаючи довжину та тип символів, які він містить.

1.5 Опис алгоритмів генерації паролів

Випадкова генерація: Цей метод полягає в генерації пароля випадковою послідовністю символів. Для генерації випадкових паролів можна використовувати генератор випадкових чисел в операційній системі або сторонніх бібліотеках.

Комбінування слів: Генератор може комбінувати випадкові слова для створення складніших паролів. Наприклад, злиття двох випадкових слів з дефісом або підкресленням між ними.

Використання фрази: Генератор може створити пароль, використовуючи фразу та перетворивши її в пароль, наприклад, вибравши першу літеру кожного слова в фразі та додавши цифри та спеціальні символи.

Рекомендації: Генератор може запропонувати пароль, який відповідає певним критеріям, таким як мінімальна довжина, наявність великих літер, цифр та спеціальних символів.

Одним зі способів покращити безпеку генерованих паролів є використання додаткових параметрів, таких як довжина пароля та міцність пароля. Наприклад,

міцність пароля може бути визначена за допомогою метрик, таких як ентропія, яка вимірює складність перебору пароля.

1.6 Властивості генераторів паролів

- Випадковість: Генератори паролів мають створювати випадкові та непередбачувані паролі.

- Складність: Генератори повинні створювати паролі, які є достатньо складними для перебору.

- Надійність: Генератори повинні забезпечувати надійність паролів та уникнення повторення.

Важливо враховувати, що надмірна складність паролів може призвести до проблем з запам'ятовуванням користувачами. Тому, в деяких випадках, генератори паролів можуть створювати паролі, які є складнішими за прості паролі, але досить легкі для запам'ятовування.

1.7 Генератори паролів в системах автентифікації

Генератори паролів широко використовуються у системах автентифікації, щоб забезпечити безпеку при вході до облікового запису. Крім того, генератори паролів використовуються в багатьох інших контекстах, наприклад, для генерації паролів до Wi-Fi мереж, SSH ключів, інформаційних запитів та інших випадків, де потрібно забезпечити безпеку даних.

У системах автентифікації генератори паролів зазвичай використовуються для створення випадкових паролів для нових облікових записів або для відновлення забутих паролів. Такі паролі зазвичай містять велику кількість випадкових символів, що робить їх надійними та складними для перебору.

У деяких випадках генератори паролів можуть використовуватися разом з геш-функціями для збереження паролів користувачів. Генератор паролів створює випадковий пароль, який потім хешується та зберігається в базі даних. Коли користувач вводить свій пароль, система хешує його та порівнює зі збереженим хешем для автентифікації.

1.8 Стеганографія

Стеганографія є наукою про приховування інформації, коли одні дані приховуються всередині інших, зазвичай непомітно для спостерігачів. У контексті приховування інформації в картинках, стеганографія дозволяє вбудовувати конфіденційні дані в цифрові зображення, такі як фотографії або графічні файли.

1.9 Опис алгоритмів стеганографії

1.LSB-підстановка: Цей алгоритм використовує найменш значущий біт (LSB) кожного пікселя зображення для зберігання додаткової інформації. Значення LSB замінюється бітами прихованого повідомлення. Цей метод досить простий, але вразливий до атак, оскільки заміна LSB може призвести до помітних змін у зображенні. Вразливості LSB-підстановки:

1.1.Вразливість до статистичного аналізу: Під час LSB-підстановки значення LSB змінюються, що може призвести до нерегулярностей у розподілі значень пікселів. Це може бути помічено при статистичному аналізі зображення. Наприклад, зображення з прихованими даними може мати незвичайно високу або низьку частоту значень LSB, що може викликати підозру.

1.2.Втрата даних при компресії: LSB-підстановка може бути вразлива до втрати даних при компресії зображення. Під час стиснення зображення деякі біти можуть бути видалені або змінені, що може призвести до втрати прихованої інформації. Таким чином, LSB-підстановка не рекомендується для використання з компресованими зображеннями.

1.3.Детектування за допомогою стегоаналізу: Спеціалізовані алгоритми стегоаналізу можуть бути використані для виявлення наявності прихованої інформації за допомогою LSB-підстановки. Ці алгоритми аналізують статистичні

властивості зображення, такі як розподіл пікселів і кореляції між ними, що може допомогти виявити зміни, внесені LSB-підстановкою.

1.4.Якщо зображення обрізається, частина збережених пікселів, де міститься прихована інформація, може бути видалена, що спричинить втрату цих даних. Аналогічно, при масштабуванні зображення можуть змінюватися значення пікселів, що може призвести до некоректної видачі прихованих даних. Крім того, обрізання або масштабування зображення також може призвести до помітних змін у самому зображенні. Це може привернути увагу атакуючих, що можуть намагатися виявити наявність прихованої інформації. 2.Алгоритм диференційного зміщення: Цей алгоритм використовує різницю між пікселями зображення для приховування інформації. Він використовує малі зміщення яскравості або кольору пікселів, які є непомітними для ока, для кодування прихованого повідомлення.

Однак, алгоритм диференційного зміщення має деякі недоліки:

2.1.Вразливість до атак на обробку зображень: Якщо зображення піддається різним операціям обробки, таким як розмиття, зміна розміру або обрізання, то це може призвести до втрати або пошкодження прихованої інформації. Операції обробки можуть змінювати значення пікселів та їх взаємні відношення, що може вплинути на помилкове витягування прихованої інформації.

2.2.Вразливість до стегоаналізу: Стегоаналіз - це процес виявлення та вилучення прихованої інформації зображення. Алгоритм диференційного зміщення може бути вразливим до стегоаналізу, оскільки прихована інформація може бути виявлена або вилучена, якщо атакуючий має знання про використовуваний алгоритм і вміє аналізувати різницю між пікселями зображення.

2.3. Можуть виникнути помітні зміни в зображенні при використанні алгоритму диференційного зміщення в стеганографії. Хоча зміни в яскравості або кольорі пікселів можуть бути малопомітними для більшості спостерігачів, детальний аналіз зображення або спеціалізовані атаки можуть допомогти виявити ці зміни.

3.Алгоритм перетворення фізичних характеристик: Цей алгоритм базується на зміні фізичних характеристик зображення, таких як кольоровий тон, контрастність або яскравість, для приховування інформації. Він може використовувати адаптивні

методи, які залежать від контексту зображення, для покращення стійкості до атак. Основними недоліками цього методу є:

3.1.Вплив на якість зображення: Зміна фізичних характеристик зображення може призводити до помітних змін у візуальному вигляді зображення. Це може зробити стеганографічну інформацію помітною або спричинити втрату якості зображення. Наприклад, зміна кольорового тону може викликати спотворення кольорів або візуальну неприродність.

3.2.Вразливість до атак: Атакуюча сторона може аналізувати фізичні характеристики зображення для виявлення наявності стеганографічної інформації. Візуальний аналіз зображення або використання стегоаналізу можуть допомогти розкрити приховану інформацію.

3.3.Вплив на розмір файлу: Застосування алгоритму перетворення фізичних характеристик може призводити до збільшення розміру файлу зображення. Додаткова інформація потребує додаткового простору для зберігання, що може бути недоцільним або непрактичним у випадку обмежень на розмір файлу.

3.4.Залежність від контексту: Деякі алгоритми перетворення фізичних характеристик можуть бути залежними від контексту зображення, що означає, що їхній успіх і стійкість можуть залежати від типу зображення або його особливостей. Це може ускладнити застосування методу на різних типах зображень або при змінних умовах. Наприклад, алгоритм, який ефективно працює з одним типом зображень, може виявитися менш ефективним для інших типів. Також, зміни в умовах знімання (освітлення, шум, резолюція) можуть впливати на якість приховання інформації або розкриття її при аналізі зображення.

1.10 Основні підходи до забезпечення стійкості алгоритмів стеганографії включають

1.Криптографічну безпеку: Використання криптографічних алгоритмів для захисту стеганографічної інформації. Це може включати застосування шифрування

для приховування та захисту самої інформації та використання цифрових підписів або інших методів аутентифікації для перевірки цілісності та походження даних.

2.Робота з помилками: Врахування можливих помилок, які можуть виникнути при передачі чи зберіганні даних. Застосування методів корекції помилок або детекції помилок може допомогти забезпечити відновлення та цілісність стеганографічної інформації навіть при наявності пошкоджень.

3.Використання стійких алгоритмів: Вибір стійких алгоритмів стеганографії, які важко розкрити або виявити. Це включає використання складних методів приховування інформації, використання різних шарів захисту та врахування можливих атак на алгоритм.

4.Аналіз стеганографічних методів: Проведення досліджень та аналізу стеганографічних методів для виявлення потенційних вразливостей та розробки проти атак. Це допомагає удосконалювати методи стеганографії та забезпечувати стійкість до розкриття.

1.11 Властивості стеганографії

1.Прихованість: Одна з основних властивостей стеганографії - прихованість інформації. При правильному застосуванні алгоритмів стеганографії, прихована інформація має бути непомітною для неповідомлених осіб або спостерігачів. Зображення, що містить приховані дані, здається незмінним або змінами непомітними для звичайного спостереження.

2.Резистентність до виявлення: Хороші алгоритми стеганографії повинні бути стійкими до розкриття прихованої інформації. Це означає, що стеганографічні методи мають бути відпорні до різних методів аналізу і виявлення прихованої інформації, таких як статистичні аналізи або різноманітні техніки стегоаналізу.

3.Витримка до перетворень: Системи стеганографії повинні бути стійкими до різних перетворень, які можуть вплинути на зображення. Це включає зміни розміру,

компресію, різні фільтри або зміни формату файлу. Стеганографічні алгоритми повинні залишати приховану інформацію незмінною навіть після таких перетворень.

4.Відтворюваність: Приховані дані повинні бути відтворюваними при потребі. Тобто, інформацію, яка була вбудована в зображення, повинна бути можливість вилучити та відновити без помилок або втрати даних.

1.12 Використання стеганографії у системах автентифікації

Стеганографія може бути використана у системах автентифікації для додаткового рівня безпеки та приховання додаткової інформації, яка підтверджує автентичність користувача. Деякі можливі сценарії використання стеганографії в системах автентифікації включають:

1.Приховане зберігання інформації про автентифікацію: Система може використовувати стеганографію для прихованого зберігання додаткових даних про автентифікацію користувача в обраних областях зображення або в інших типах мультимедійних файлів. Це може включати приховане зберігання біометричних даних, таких як відбитки пальців або розпізнавання обличчя, які можуть бути використані для додаткової перевірки автентичності.

2.Використання стеганографії для створення паролів: Система може використовувати стеганографію для генерації унікальних паролів для кожного користувача. Приховані зображення можуть містити додаткові біти, які використовуються для генерації пароля.

3.Використання стеганографії для обміну ключами: Стеганографія може бути використана для прихованого обміну секретними ключами між користувачем і системою автентифікації. Ключі можуть бути вбудовані в зображення або інші мультимедійні файли й передані між сторонами без прямого викриття ключової інформації.

Висновки за розділом 1

У даному розділі було проведено огляд геш-функцій, генераторів паролів і методів стеганографії. Робота включала дослідження властивостей геш-функцій, вибір надійних алгоритмів гешування, рекомендації зі створення міцних паролів та основні методи стеганографії. Ця інформація допоможе у створені програмного модуля для збереження логінів і паролів у гешованому вигляді, що забезпечить безпеку та цілісність даних.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ВИКОРИСТАНИХ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ З МЕТОЮ ЗБЕРЕЖЕННЯ ЛОГІНІВ І ПАРОЛІВ В ГЕШОВАНОМУ ВИГЛЯДІ

У цьому розділі будуть надані більш детальні описи використаних середовищ розробки та їх переваги для проєкту генератора паролів та системи автентифікації. Проєкт був розроблений мовою програмування Python з використанням PyQt5-PyQt6 та бази даних SQLite.

2.1 Опис Python

Python є об'єктно-орієнтованою мовою програмування з високим рівнем абстракції, що пропонує широкі можливості для швидкої розробки додатків. Вона широко використовується для створення сценаріїв і має динамічну семантику. Одна з основних переваг Python у сфері аналізу даних є його відкритий код, що сприяє розширенню мови та наявності великої кількості підтримки. Python також пропонує високу швидкість виконання програм і дозволяє швидко створювати та тестувати додатки. Багато модулів та бібліотек, спеціально призначених для аналізу даних, доступні для використання. Загалом, Python є універсальною мовою програмування, яка дозволяє розробникам зосередитися на аналізі даних та реалізації аналітичних інструментів з високою продуктивністю та гнучкістю.

2.2 Основні властивості мови Python

Простота вивчення: Синтаксис Python легкий для сприйняття і схожий на звичну англійську мову, що робить його легким для вивчення як початківцям, так і досвідченим програмістам.

- Інтерпретованість: Python використовує інтерпретатор, що дозволяє виконувати код без необхідності компіляції. Це спрощує розробку і тестування програм.

- Об'єктно-орієнтоване програмування: Python підтримує об'єктно-орієнтовану парадигму, дозволяючи створювати класи, об'єкти та використовувати принципи спадкування, поліморфізму та інкапсуляції.

- Багата стандартна бібліотека: Python має велику стандартну бібліотеку, яка надає різноманітні інструменти й функціональність для різних завдань, таких як робота з файлами, мережеве програмування, обробка тексту, обчислення, тестування та інше. Це дозволяє програмістам швидко розробляти програми без необхідності писати код з нуля.

- Динамічна типізація: Python є динамічно типізованою мовою, що означає, що змінні не потребують явного оголошення типу. Тип змінної визначається автоматично під час виконання програми. Це дозволяє зручно працювати зі змінними та спрощує процес розробки.

- Розширюваність: Python може бути розширений за допомогою модулів та пакетів. Багато модулів створені спільнотою програмістів і доступні для використання безплатно. Це дозволяє розширювати функціональність мови та використовувати готові рішення для різних завдань.

- Переносимість: Python є переносимою мовою, що означає, що програми, написані на Python, можуть працювати на різних операційних системах, таких як Windows, macOS, Linux та інші. Це забезпечує гнучкість в розгортанні програм і забезпечує доступність на різних платформах.

Широке застосування: Python використовується в різних областях, включаючи веб-розробку, наукові обчислення, аналітику даних, штучний інтелект, машинне навчання, автоматизацію та інше.

2.3 Опис Pycharm

PyCharm є потужним та інтуїтивно зрозумілим інтегрованим середовищем розробки (IDE), спеціально призначеним для розробки мовою програмування Python. Основні переваги використання PyCharm для проєкту включають:

- Підтримка автодоповнення коду: PyCharm надає широкий набір інструментів для швидкого та зручного написання коду. Завдяки автодоповненню, буде доступний список доступних функцій, класів та модулів, що спрощує процес програмування.
- Налагодження: PyCharm має потужний налагоджувач, який дозволяє крок за кроком виконувати код та перевіряти його результати. Це допоможе виявити та виправити помилки та проблеми в процесі розробки.
- Управління проєкт: PyCharm дозволяє зручно керувати вашим проєктом, включаючи створення, перемикання та налагодження віртуальних середовищ, налаштування залежностей та керування версіями.

2.4 Опис PyQt5 та PyQt6

PyQt5 та PyQt6 є інтерфейсами Python для Qt, що дозволяють вам створювати багатофункціональні графічні інтерфейси за допомогою мови програмування Python. Вони надають доступ до розширених можливостей фреймворку Qt, що охоплює широкий набір готових елементів управління (кнопки, поля введення, таблиці тощо), можливості налаштування вигляду та поведінки елементів інтерфейсу, обробку подій, створення власних віджетів та багато іншого.

Використання PyQt5-PyQt6 дозволяє вам ефективно створювати й налаштовувати графічний інтерфейс вашого проєкту, використовуючи можливості, які надає Qt. Ви можете створювати вікна, вкладки, меню, кнопки та інші елементи інтерфейсу, розташовувати їх у відповідних макетах, задавати їхні властивості та відповідні дії, оброблювати події користувача та забезпечувати багатофункціональність вашого додатка.

PyQt5 та PyQt6 є популярними бібліотеками для розробки графічного інтерфейсу користувача (GUI) мовою програмування Python. Використання PyQt5-PyQt6 в проєкт має наступні переваги:

- Розширені можливості GUI: Завдяки бібліотекам PyQt5-PyQt6 ви отримуєте доступ до повноцінного фреймворку Qt, що дозволяє створювати складні та професійні графічні інтерфейси. Ви можете легко створювати вікна, кнопки, поля введення, таблиці, меню та інші елементи інтерфейсу за допомогою розширених функцій та властивостей, що надаються PyQt5-PyQt6.

- Кросплатформеність: PyQt5-PyQt6 підтримує розробку кросплатформових додатків, що дозволяє вам створювати програми, які працюватимуть на різних операційних системах, таких як Windows, macOS і Linux. Це дає можливість розширити вашу аудиторію користувачів і забезпечити їм однаковий функціонал та вигляд незалежно від платформи.

- Багато поточність: PyQt5-PyQt6 надає підтримку багато поточності, що дозволяє розподілити завдання на різні потоки виконання. Це особливо важливо, коли ви маєте справу з операціями, які можуть займати багато часу, наприклад, завантаження великих об'єктів або виконання обчислювально важких задач. Використання багато поточності допоможе забезпечити плавну роботу вашого додатка.

2.5 Опис SQLite

SQLite - це легка вбудовувана реляційна база даних, яка працює без необхідності встановлення окремого сервера баз даних.

Ось кілька додаткових характеристик, які можна сказати про базу даних SQLite:

Простота використання: SQLite надає простий та зрозумілий інтерфейс для створення, зміни та оптимізації таблиць, виконання запитів та отримання результатів. Ви можете використовувати стандартну мову запитів SQL для взаємодії з базою даних.SQLite має наступні властивості:

- Переносимість: База даних SQLite зберігається в одному файлі, що робить її дуже переносною. Ви можете просто копіювати цей файл між різними середовищами або пристроями без необхідності встановлювати окремий сервер бази даних.

- Невеликий розмір: SQLite має маленький розмір, що робить його ідеальним варіантом для проєкт з обмеженими ресурсами, таких як мобільні додатки чи стільникові додатки. Він не потребує великого обсягу дискового простору чи оперативної пам'яті.

- Транзакційна безпека: SQLite підтримує транзакції ACID, що забезпечує надійність та цілісність даних. Ви можете здійснювати операції вставки, оновлення та видалення з гарантією, що дані залишаться у правильному стані, навіть у випадку виникнення помилок або відмов.

- Широкі можливості SQL: SQLite повністю підтримує стандартну мову запитів SQL, що дозволяє вам виконувати складні запити до бази даних. Ви можете створювати таблиці, індекси, обмеження, виконувати операції об'єднання, сортування та інші операції, що сприяють ефективній роботі з даними.

2.6 Використані бібліотеки

Бібліотека string: Ця бібліотека надає набір корисних функцій і констант, пов'язаних з операціями над рядками. Вона містить методи для маніпулювання рядками, такі як обрізання, розбиття на підрядки, форматування, перетворення регістру та багато іншого. Була використана для опрацювання та маніпуляції текстових даних.

Бібліотека secrets: Ця бібліотека надає функції для генерації безпечних випадкових значень, таких як паролі, ключі аутентифікації, токени тощо. Вона використовує криптографічно стійкі методи для генерації випадкових чисел і даних, що робить її корисною для забезпечення безпеки в системах автентифікації та інших аспектах практичної роботи, пов'язаних із генерацією та керуванням конфіденційними даними.

Бібліотека hashlib: Ця бібліотека надає функції для обчислення хеш-сум та криптографічних геш-функцій. Вона підтримує різні алгоритми хешування, такі як MD5, SHA-1, SHA-256 та інші.

Бібліотека uuid: Ця бібліотека дозволяє генерувати унікальні ідентифікатори. UUID є 128-бітним значенням, яке гарантується бути унікальним в межах системи або мережі. Вона була використана для генерації унікальних ідентифікаторів.

Загалом, ці чотири бібліотеки (string, secrets, hashlib і uuid) надають корисні функції для опрацювання рядків, генерації безпечних випадкових значень, обчислення хеш-сум та створення унікальних ідентифікаторів. Вони були використані в практичній роботі, для створення системи програмного модуля для збереження логінів і паролів в гешованому вигляді.

Висновки за розділом 2

За результатами другого розділу були описані та використані ключові інструменти для створення програмного модуля збереження логінів і паролів у гешованому вигляді. Мова програмування Python, середовище розробки PyCharm, база даних SQLite та фреймворк PyQt5-6 були використані для досягнення цілей проєкту. Python забезпечує простий синтаксис і високий рівень абстракції, PyCharm є потужним інтегрованим середовищем розробки, SQLite є легкою вбудованою базою даних, а PyQt5-6 надає зручні засоби для розробки графічного інтерфейсу. Використання цих інструментів сприяло створенню гнучкого та безпечного модуля для збереження логінів і паролів, що є важливим аспектом для забезпечення безпеки даних.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ ПРОГРАМНОГО МОДУЛЯ ДЛЯ ЗБЕРЕЖЕННЯ ЛОГІНІВ І ПАРОЛІВ В ГЕШОВАНОМУ ВИГЛЯДІ

3.1 Генератор паролів у Python

Генератор паролів - це важлива складова безпеки багатьох систем, яка дозволяє створювати надійні та випадкові паролі для захисту від несанкціонованого доступу.

Мій код реалізує алгоритм, який дотримується встановлених вимог до пароля, таких як довжина, використання різних типів символів та уникання простих комбінацій. Він також забезпечує можливість налаштування параметрів генерації паролів, щоб вони відповідали вашим потребам.

Далі я покроково опишу алгоритм та логіку мого коду, додаючи коментарі та пояснення до кожного кроку. Сподіваюся, що цей код стане корисним для вас і допоможе забезпечити безпеку ваших систем.

Для того, щоб конвертувати файл ресурсів від PyQt6, потрібно написати в терміналі: pyside6-rcc, назва файлу ресурсів, прапорець — o і майбутня назва файлу.(Рисунок.3.1)

```
pyside6-rcc .\resources.qrc -o .\resources.py
```

Рисунок 3.1 - Конвертація ресурсів PyQt6 файлу

Файл інтерфейсу конвертується подібним чином, тільки в цьому випадку потрібно використовувати “pyside6-uic”.(Рисунок.3.2)

```
pyside6-uic .\main.ui -o ui_main.py
```

Рисунок 3.2 - Конвертування файлу інтерфейсу.

В стандартній бібліотеці “string” є строкові змінні для пула символів. Обираємо ascii_lowercase, ascii_uppercase, digits й punctuation (Рисунок.3.3).

```
whitespace = ' \t\n\r\v\f'
ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ascii_letters = ascii_lowercase + ascii_uppercase
digits = '0123456789'
hexdigits = digits + 'abcdef' + 'ABCDEF'
octdigits = '01234567'
punctuation = r"""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""
```

Рисунок 3.3 - Обираємо список всіх символів для подальшого використання в генераторі паролів

Створюю перелік який пов’язує кнопки мого графічного інтерфейсу символів до строкових змінних (Рисунок.3.4).

```
from string import ascii_lowercase, ascii_uppercase, digits, punctuatio
from enum import Enum


class Characters(Enum):
    btn_lower = ascii_lowercase
    btn_upper = ascii_uppercase
    btn_digits = digits
    btn_special = punctuation
```

Рисунок 3.4 - Створення логіки графічного інтерфейсу

Створюю словник на кожний тип даних для подальшого обрахування ентропії (Рисунок.3.5).

```
CHARACTER_NUMBER = {
    'btn_lower': len(Characters.btn_lower.value),
    'btn_upper': len(Characters.btn_upper.value),
    'btn_digits': len(Characters.btn_digits.value),
    'btn_special': len(Characters.btn_special.value)
}
```

Рисунок 3.5 - Створення словника

Напишу функцію створення нового пароля. Аргументами будуть довжина пароля і всі можливі символи у вигляді рядка. Для генерації безпечного пароля я буду використовувати бібліотеку secrets. Створюю генератор зі випадково обраних символів (всіх типів символів) і поєднаю їх в єдиний рядок використовуючи метод join (Рисунок.3.6).

```
14  def create_new(length: int, characters: str) -> str:
15      return ''.join(secrets.choice(characters) for _ in range(length))
```

Рисунок 3.6 - Функція створення пароля

Тепер зроблю функцію для вирахування ентропії. В неї потрібно буде передавати довжину пароля і кількість символів. Формула ентропії пароля виглядає наступним чином (Формула 3.1).

$$H = log_2 N^L = L \times log_2 N [3.1]$$

Формула 3.1 - Формула ентропії

Розробляю функцію ентропії мовою python.Рисунок(3.7).

```
def get_entropy(length: int, character_number: int) -> float:
    try:
        entropy = length * log2(character_number)
    except ValueError:
        return 0.0
```

Рисунок 3.7 – Функція ентропії

Функція ентропії буде повертати округлене значення до 2 знаків після коми (Рисунок.3.8).

```
    return round(entropy, 2)
```

Рисунок 3.8 – Повернення округленого результату

Створюю числове перерахування складності пароля відносно його ентропії по нижній границі (Рисунок.3.9).

- Дуже слабкий пароль від 0 до 30 біт
- Слабкий пароль від 30 біт
- Гарний пароль від 50 біт
- Сильний від 70 біт
- Чудовий після 120 біт

```
class StrengthToEntropy(IntEnum):
    Pathetic = 0
    Weak = 30
    Good = 50
    Strong = 70
    Excellent = 120
```

Рисунок 3.9 - Складність пароля

Створюю метод класу додатка, який буде пов’язувати значення слайдера та лічильника. Для цього поєднаю “valueChanged” зі встановленим значенням другого елемента (Рисунок.3.10).

```
def connect_slider_to_spinbox(self) -> None:
    self.ui.slider_length.valueChanged.connect(self.ui.spinbox_length.setValue)
    self.ui.spinbox_length.valueChanged.connect(self.ui.slider_length.setValue)
```

Рисунок 3.10 - Налаштування слайдера та лічильника

Також не забуваємо прописати метод в конструкторі (Рисунок.3.11).

```
        self.connect_slider_to_spinbox()
```

Рисунок 3.11 - Метод слайдера в конструкторі

Створюю новий метод для отримання символів, які були відмічені кнопками. Спочатку створю пустий рядок “chars”. Для кожної кнопки в переліку додам її символи, якщо вона відмічена (Рисунок.3.12).

```
def get_characters(self) -> str:
    chars = ''

    for btn in buttons.Characters:
        if getattr(self.ui, btn.name).isChecked():
            chars += btn.value

    return chars
```

Рисунок 3.12 - Створення методу кнопок

Нарешті створимо метод встановлення пароля. Ставимо елемент “line_password ” новий пароль використовуючи метод “setText”. Довжину отримуємо зі слайдера або лічильника. Опрацюю випадок, якщо не натиснута жодна кнопка символів, просто очищаю поле пароля. Додам ентропію в кінці метода установки пароля. Також додам метод для вирахування сили пароля (Рисунок.3.13).

```
def set_password(self) -> None:
    try:
        self.ui.line_password.setText(
            password.create_new(
                length=self.ui.slider_length.value(),
                characters=self.get_characters())
        )
    except IndexError:
        self.ui.line_password.clear()

    self.set_entropy()
    self.set_strength()
```

Рисунок 3.13 - Логіка генератора пароля

Також не забуваємо додати метод в конструктор.(Рисунок.3.14)

```
    self.set_password()
```

Рисунок 3.14 - Метод генератора паролів

Також потрібно з’єднати зміни слайдера й лічильника з методом для генерації пароля (Рисунок.3.15).

```
self.ui.spinbox_length.valueChanged.connect(self.set_password)
```

Рисунок 3.15 - Поєднання слайдера з методом генерації пароля

Принцип такий же, як і в методі отримання рядка символів, але в цьому випадку ми працюємо зі словником, а не з переліком (Рисунок.3.16).

```
def get_character_number(self) -> int:
    num = 0

    for btn in buttons.CHARACTER_NUMBER.items():
        if getattr(self.ui, btn[0]).isChecked():
            num += btn[1]

    return num
```

Рисунок 3.16 - Отримання кількості спеціальних символів

Створюю метод для ентропії. Використовую f-рядок (Рисунок.3.17).

```
def set_entropy(self) -> None:
    length = len(self.ui.line_password.text())
    char_num = self.get_character_number()

    self.ui.label_entropy.setText(
        f'Entropy: {password.get_entropy(length, char_num)} bit'
    )
```

Рисунок 3.17 - Метод ентропії

Тепер створюю метод для “label” отримання складності пароля. Для кожної наявної складності порівнюю ентропію (Рисунок.3.18).

```
def set_strength(self) -> None:
    length = len(self.ui.line_password.text())
    char_num = self.get_character_number()

    for strength in password.StrengthToEntropy:
        if password.get_entropy(length, char_num) >= strength.value:
            self.ui.label_strength.setText(f'Strength: {strength.name}')
```

Рисунок 3.18 - Отримання складності пароля

Запишу в модуль “buttons” кортеж з іменами кнопок при натисканні на них буде генеруватися новий пароль (Рисунок.3.19).

```
GENERATE_PASSWORD = (
    'btn_refresh', 'btn_lower', 'btn_upper', 'btn_digits', 'btn_special'
)
```

Рисунок 3.19 - Модуль “buttons”

Тепер з’єднаю кнопки з методом в конструкторі класу (Рисунок.3.20).

```
    for btn in buttons.GENERATE_PASSWORD:
        getattr(self.ui, btn).clicked.connect(self.set_password)

    self.ui.btn_visibility.clicked.connect(self.change_password_visibility)
    self.ui.btn_copy.clicked.connect(self.copy_to_clipboard)
```

Рисунок 3.20 - Метод кнопок в конструкторі

Напишу метод для змінення видимості пароля. Якщо кнопка відмічена, ставлю нормальний “echomode”, в іншому випадку ставлю “echomode” пароля (Рисунок.21).

```
def change_password_visibility(self) -> None:
    if self.ui.btn_visibility.isChecked():
        self.ui.line_password.setEchoMode(QLineEdit.Normal)
    else:
        self.ui.line_password.setEchoMode(QLineEdit.Password)
```

Рисунок 3.21 - Метод видимості пароля

Поєдную натискання кнопки з методом в конструкторі класу (Рисунок.3.22).

```
self.ui.btn_visibility.clicked.connect(self.change_password_visibility)
```

Рисунок 3.22 - Процес поєднання

Для того, щоб скопіювати текст в буфер обміну, потрібно викликати метод “clipboard ” класу “QApplication” і поставити в нього текст (Рисунок.3.23).

```
def copy_to_clipboard(self) -> None:
    QApplication.clipboard().setText(self.ui.line_password.text())
```

Рисунок 3.23 - Метод копіювання

Не забуваємо про поєднання методу копіювання в конструкторі (Рисунок.3.24).

```
self.ui.btn_copy.clicked.connect(self.copy_to_clipboard)
```

Рисунок 3.24 - Метод копіювання в конструкторі

Тепер у нас є власний програмний код, який можна використовувати для генерації надійних паролів не боячись використання паролів, які були згенеровані з використанням сторонніх додатків (Рисунок.3.25).

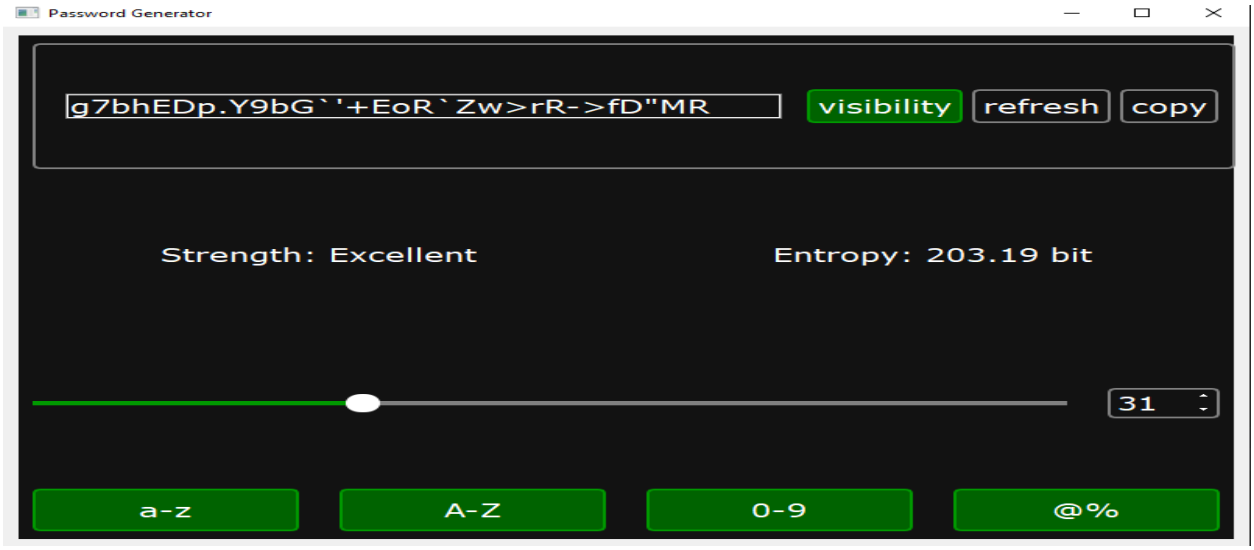

Рисунок 3.25 - Генератор паролів

3.2 Розробка функції гешування

Цей код відображає набір функцій для гешування тексту з використанням бібліотеки “hashlib”. Функція hashText (salt, text): Приймає два аргументи: salt (сіль) і text (текст, який потрібно гешувати). Функція використовує SHA256 для гешування тексту з унікальною сіллю. Сіль додається до тексту перед гешуванням. В кінці повертаємо хеш у вигляді рядка (Рисунок.3.26).

```
import hashlib


def hashText(salt, text):
    """
        Basic hashing function for a text using random unique salt.
    """
    return hashlib.sha256(salt.encode() + text.encode()).hexdigest()
```

Рисунок 3.26 - Функція гешування

Функція matchHashedText (hashedText, providedText) приймає два аргументи: hashedText (гешований текст у форматі хеш:сіль і providedText (текст, який потрібно перевірити). В результаті, ми повертаємо True, якщо геш збігається, False в протилежному разі (Рисунок.3.27).

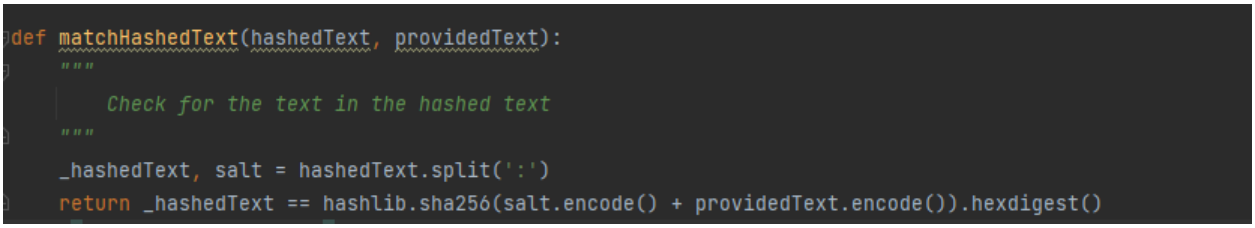

Рисунок 3.27 - Функція перевірки гешу

Наступний приклад буде трошки не коректний до моєї функції, яку я буду використовувати надалі для гешування і перевірки моїх збережених хеш-значень в базі даних. Для більшого я трошки модифікував код таким чином, щоб можна було зрозуміти, що він повертає після завершення і як його використовувати. В представленому коді використовується модуль uuid для генерації випадкового унікального ідентифікатора. В цьому випадку це наша сіль. Hex – метод, який перетворює UUID в його шістнадцятирічне представлення у вигляді рядка символів:(0-9, a-f) (Рисунок.3.28).

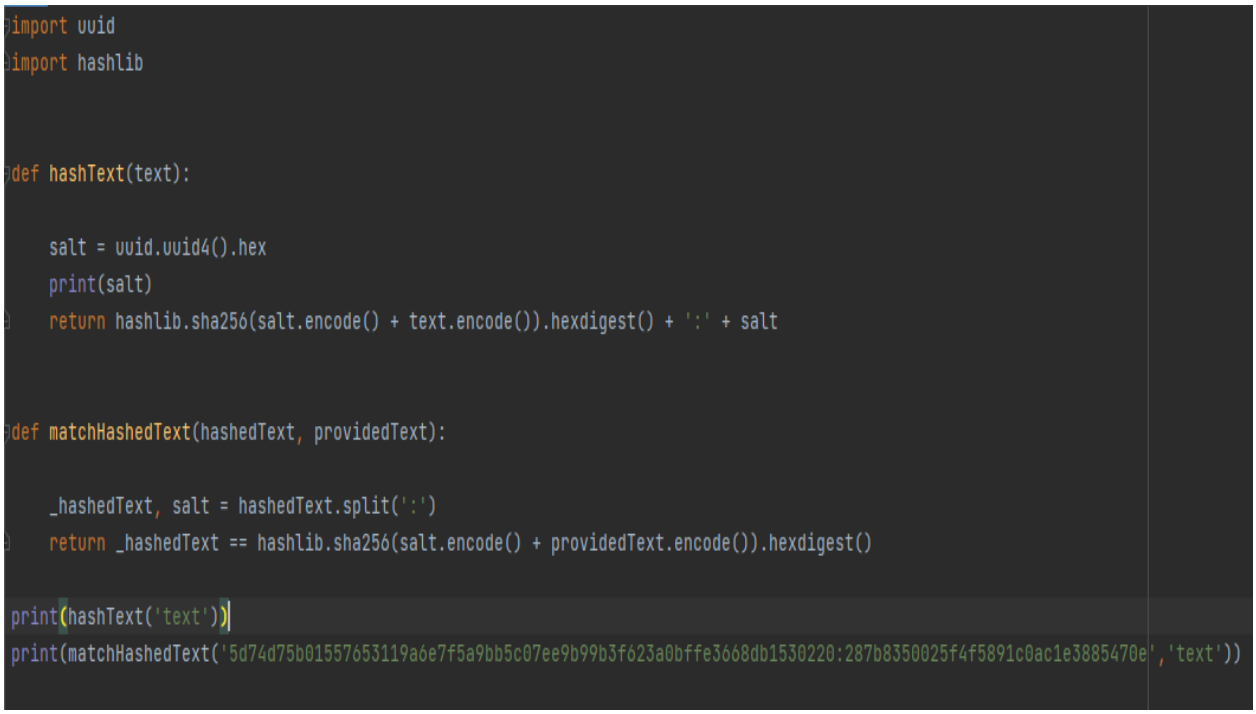

Рисунок 3.28 - Функція гешування та перевірки на правдивість

Перший рядок - це випадкова згенерована сіль з використанням методу uuid.uuid4().hex. Другий рядок це гешований текст з використанням солі й алгоритму Sha-256. Третій рядок це перевірка на правдивість (Рисунок.3.29).

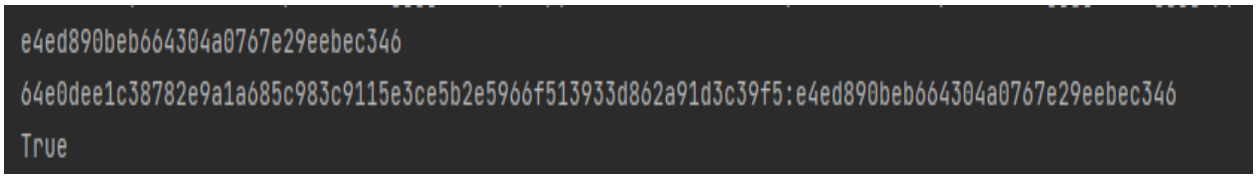

Рисунок 3.29 - Процес використання геш функції в системах автентифікації

3.3.Створення програми яка реазуліє LSB стеганографію

Щоб краще розуміти як працює LSB-стеганографія, розглянемо цифрове зображення як двовимірний масив пікселів. Кожен піксель містить значення залежно від його типу та глибини. Ми розглянемо найбільш поширені режими — RGB (3x8-

бітних пікселів, справжній колір) і RGBA (4x8-бітних пікселів, справжній колір з маскою прозорості). Ці значення варіюються від 0 до 255 (8-бітні значення)(Рисунок.3.30).

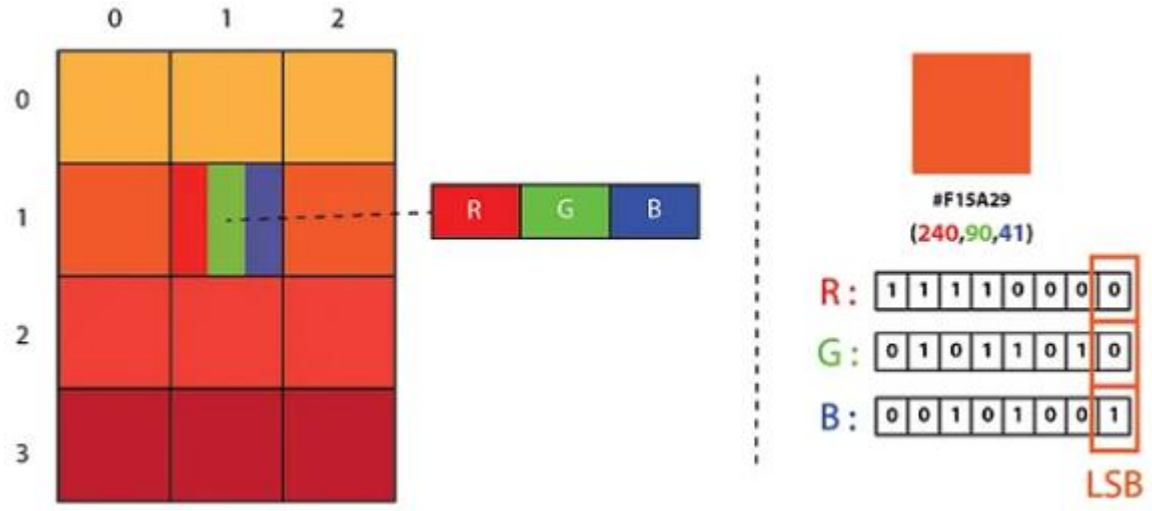

Рисунок 3.30 - Модель RGB-бітних пікселів

Ми можемо перетворити повідомлення в десяткові значення, а потім у двійкові за допомогою таблиці ASCII. Потім ми повторюємо значення пікселів одне за одним, після перетворення їх у двійкові ми замінюємо кожен найменш значущий біт бітами цього повідомлення в послідовності.

Щоб декодувати закодоване зображення, ми просто повертаємо процес у зворотному напрямку. Зберіть і збережіть останні біти кожного пікселя, потім розділіть їх на групи по 8 і конвертуйте їх назад у символи ASCII, щоб отримати приховане повідомлення.

Тепер ми спробуємо реалізувати вищезазначену концепцію крок за кроком за допомогою бібліотек Python — PIL і NumPy (Рисунок.3.31).

```
import numpy as np
from PIL import Image
```

Рисунок 3.31 - Бібліотеки PIL і NumPy

По-перше, ми пишемо код для перетворення вихідного зображення в масив пікселів NumPy і збереження розміру зображення. Ми перевіряємо, чи є режим зображення RGB або RGBA, отже, встановлюємо значення n. Також обчислюємо загальну кількість пікселів (Рисунок.3.32).

```
def Encode(src, message, dest):

    img = Image.open(src, 'r')
    width, height = img.size
    array = np.array(list(img.getdata()))
    password = 'abobus'

    if img.mode == 'RGB':
        n = 3
    elif img.mode == 'RGBA':
        n = 4

    total_pixels = array.size//n
```

Рисунок 3.32 - Процес опрацювання зображення

По-друге, ми додаємо роздільник ("password") у кінці секретного повідомлення, щоб програма, коли декодує, знала, коли зупинитися. Ми перетворюємо це оновлене повідомлення у двійкову форму та обчислюємо необхідні пікселі. (Рисунок.3.33).

```
message += password
b_message = ''.join([format(ord(i), "08b") for i in message])
req_pixels = len(b_message)
```

Рисунок 3.33 - Обчислення пікселів

По-третє, ми перевіряємо, чи достатньо доступних пікселів для секретного повідомлення чи ні. Якщо так, ми переходимо до повторення пікселів один за одним і змінюємо їхні молодші значущі біти на біти секретного повідомлення, поки не буде приховано повне повідомлення, включаючи розділювач. (Рисунок.3.34).

```
if req_pixels > (total_pixels * 3):
    print("ERROR: Need larger file size")

else:
    index=0
    for p in range(total_pixels):
        for q in range(0, 3):
            if index < req_pixels:
                array[p][q] = int(bin(array[p][q])[2:9] + b_message[index], 2)
                index += 1
```

Рисунок 3.34 - Процес кодування

Нарешті, у нас є оновлений масив пікселів, і ми можемо використовувати його для створення та збереження його як кінцевого вихідного зображення (Рисунок.3.35).

```
    array=array.reshape(height, width, n)
    enc_img = Image.fromarray(array.astype('uint8'), img.mode)
    enc_img.save(dest)
    print("Image Encoded Successfully")
```

Рисунок 3.35 - Кодування зображення

Щоб створити функцію декодера, по-перше, ми повторюємо аналогічну процедуру збереження пікселів вихідного зображення у вигляді масиву, визначення режиму та обчислення загальної кількості пікселів (Рисунок.3.36).

```
def Decode(src):

    img = Image.open(src, 'r')
    array = np.array(list(img.getdata()))
    password='abobus'

    if img.mode == 'RGB':
        n = 3
    elif img.mode == 'RGBA':
        n = 4

    total_pixels = array.size//n
```

Рисунок 3.36 - Процес опрацювання зображення

По-друге, нам потрібно витягти молодші біти з кожного пікселя, починаючи з верхнього лівого кута зображення, і зберегти його в групах по 8. Далі ми перетворюємо ці групи в символи ASCII, щоб знайти приховане повідомлення, поки не прочитаємо роздільник, вставлений раніше повністю (Рисунок.3.37).

```
hidden_bits = ""
for p in range(total_pixels):
    for q in range(0, 3):
        hidden_bits += (bin(array[p][q])[2:][-1])

hidden_bits = [hidden_bits[i:i+8] for i in range(0, len(hidden_bits), 8)]

message = ""
hiddenmessage = ""
for i in range(len(hidden_bits)):
    x = len(password)
    if message[-x:] == password:
        break
    else:
        message += chr(int(hidden_bits[i], 2))
        message = f'{message}'
        hiddenmessage = message
```

Рисунок 3.37 - Процес витягування молодших бітів

Нарешті ми перевіряємо, знайдено роздільник чи ні. Якщо ні, це означає, що в зображенні не було прихованого повідомлення (Рисунок.3.38).

```
if password in message:
    return hiddenmessage[:-x]
else:
    print("You entered the wrong password: Please Try Again")
```

Рисунок 3.38 - Перевірка роздільника

- Для основної функції ми запитуємо користувача, яку функцію він хоче виконати — кодувати чи декодувати.

- Для “Encode” ми просимо користувача ввести наступні дані — ім’я вихідного зображення з розширенням, секретне повідомлення та ім’я цільового зображення з розширенням.

- Для декодування ми запитуємо у користувача вихідне зображення, яке має приховане повідомлення (Рисунок.3.39).

-

```
#main function
def Stego():
    print("--Welcome to $t3g0--")
    print("1: Encode")
    print("2: Decode")

    func = input()

    if func == '1':
        print("Enter Source Image Path")
        src = input()
        print("Enter Message to Hide")
        message = uuid.uuid4().hex
        print("Enter Destination Image Path")
        dest = input()
        print("Encoding...")
        Encode(src, message, dest)

    elif func == '2':
        print("Enter Source Image Path")
        src = input()
        print("Decoding...")
        Decode(src)

    else:
        print("ERROR: Invalid option chosen")

Stego()
```

Рисунок 3.39 - Логіка коду

Ось момент, де ми використовуємо раніше описану функцію “uuid” для генерації випадкової солі й подальшого приховування (Рисунок.3.40).

```
print("Enter Message to Hide")
message = uuid.uuid4().hex
```

Рисунок 3.40 - Функція “uuid” в стеганографіїНаочний приклад кодування повідомлення і його подальше декодування (Рисунок.3.41-3.42).

```
--Welcome to $t3g0--
1: Encode
2: Decode
1
Enter Source Image Path
Login.png
Enter Message to Hide
Enter Destination Image Path
Login1.png
Encoding...
Image Encoded Successfully

Process finished with exit code 0
```

Рисунок 3.41 - Процес кодування

```
--Welcome to $t3g0--
1: Encode
2: Decode
2
Enter Source Image Path
Login1.png
Decoding...
b3b593ae376c4bcb89ce36f2db5cef72
```

Рисунок 3.42 - Процес декодування

3.4 Створення системи авторизації

Додаємо 2 кнопки до конструктора (Рисунок.3.43).

```
class Interface(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.ui = Ui_Form()
        self.ui.setupUi(self)

        self.ui.pushButton.clicked.connect(self.reg)
        self.ui.pushButton_2.clicked.connect(self.auth)
```

Рисунок 3.43 - Поєднання графічного інтерфейсу і коду

Створюю рядок де є наш логін і пароль.(Рисунок.3.44).

```
self.base_line_edit = [self.ui.lineEdit, self.ui.lineEdit_2]
```

Рисунок 3.44 - Створення рядка логіну і пароля

Створюю функцію, яка перевіряє правильність вводу даних з наявною інформацією в базі даних.(Рисунок.3.45).

```
def check_input(funct):
    def wrapper(self):
        for line_edit in self.base_line_edit:
            if len(line_edit.text()) == 0:
                return
        funct(self)
    return wrapper
```

Рисунок 3.45 - Перевірка даних

Використовуючи декоратор, який був створений раніше, ми отримуємо логін і пароль. Потім визиваємо метод, який знаходиться в чек бд. (Рисунок.3.46).

```
@check_input
def auth(self):
    name = hashText(Decode(self.fname11),self.ui.lineEdit.text())
    passw = hashText(Decode(self.fname12),self.ui.lineEdit_2.text())
    self.check_db.thr_login(name, passw)
```

Рисунок 3.46 - Перевірка логіну і паролю

Створюю екземпляр класа цього модуля і обробник сигнала. (Рисунок.3.47).

```
self.check_db = CheckThread()
self.check_db.mysignal.connect(self.signal_handler)
```

Рисунок 3.47 - Опрацьовуємо сигнал

Пишу метод де буду опрацьовувати сигнали з вище створеного модуля. (Рисунок.3.48).

```
def signal_handler(self, value):
    QtWidgets.QMessageBox.about(self, 'Сповіщення', value)
```

Рисунок 3.48 - Опрацьовувач сигналів

Підключаємо всі потрібні класи й підключаємо опрацьовувач, який знаходиться в “db.handler.py”. Він буде опрацьовувати запити до бази даних (Рисунок.3.49).

```
from PyQt5 import QtCore, QtGui, QtWidgets
from handler.db_handler import *
```

Рисунок 3.49 - Імпорт потрібних елементів

Тут ми використовуємо сигнал, який передає рядок і 2 методи. Вони використовують файл “db.handler” для поєднання з базою даних. Для перевірки використовується пароль і сигнал для того, щоб з цього сигналу можна було передати дані в основний файл інтерфейсу і підключитися до опрацьовувача (Рисунок.3.50).

```
class CheckThread(QtCore.QThread):
    mysignal = QtCore.pyqtSignal(str)

    def thr_login(self, name, passw):
        login(name, passw, self.mysignal)

    def thr_register(self, name, passw):
        register(name, passw, self.mysignal)
```

Рисунок 3.50 - Методи опрацювання інформації в базі даних.

Створюю функцію “login”, яка приймає логін, пароль і сигнал, вона підключається до бази даних (Users). Створюю курсор для виконання дій в цій базі даних. Ми отримуємо всі значення, якщо ім’я користувача відповідає логіну. Якщо ця умова виконується в “value” запишеться результат. Потім іде перевірка на правильність введення пароля, якщо це не пустий кортеж і 2 значення дорівнює паролю, то ми отримаємо успішну авторизацію. Після того ми підключаємося до (Рисунок.3.48), який буде виводити цю інформацію (Рисунок.3.51).

```
def login(login, passw, signal):
    con = sqlite3.connect('handler/users')
    cur = con.cursor()

    # Проверяем есть ли такой пользователь
    cur.execute(f'SELECT * FROM users WHERE name="{login}";')
    value = cur.fetchall()

    if value != [] and value[0][2] == passw:
        signal.emit('Успішна авторизація!')
    else:
        signal.emit('Перевірте правильність введених даних!')

    cur.close()
    con.close()
```

Рисунок 3.51 - Перевірка авторизації

Ця функція приймає ті ж значення. Підключається до бази даних. Створюємо курсор для взаємодії, також шукаємо наш логін тільки в цьому випадку для перевірки, чи вже існує такий логін. Якщо ми знаходимо такий нікнейм. Виводимо вже використовується. Якщо це значення пусте ми вставляємо в таблицю “Users” в вказані стовпці “name”, ”password” додаємо values і передаємо логін і пароль (Рисунок.3.52).

```
def register(login, passw, signal):
    con = sqlite3.connect('handler/users')
    cur = con.cursor()

    cur.execute(f'SELECT * FROM users WHERE name="{login}";')
    value = cur.fetchall()

    if value != []:
        signal.emit('Такий логін вже використовується!')

    elif value == []:
        cur.execute(f"INSERT INTO users (name, password) VALUES ('{login}', '{passw}')")
        signal.emit('Ви успішно зареєстровані!')
        con.commit()

    cur.close()
    con.close()
```

Рисунок 3.52 - Процес реєстрації користувача

Нарешті створюю функцію “Clicker” для вибору фотографії, в якому є захована сіль, щоб надалі використовуючи функцію розшифрування, яка була створена раніше з метою отримання солі (Рисунок.3.53).

```
def clicker(self):
    fname = QFileDialog.getOpenFileName(self, "Open File")
    #open the image
    self.pixmap = QPixmap(fname[0])
    #add the picture to label
    self.label100.setPixmap(self.pixmap)

    fname1 = QFileDialog.getOpenFileName(self, "Open File")

    self.pixmap1 = QPixmap(fname1[0])
    self.label101.setPixmap(self.pixmap1)
    #отримуємо назву файлів
    self.fname11 = fname[0].split('/')[-1]
    self.fname12 = fname1[0].split('/')[-1]
```

Рисунок 3.53 - Вибір фотографії для автентифікації/реєстрації

Нарешті я можу показати фінальну частину свого програмного модуля для збереження логінів і паролів в гешованому вигляді на практиці. Зараз я покроково поясню як працювати з цим додатком.

Створюємо сіль і використовуємо стеганографію для приховування цієї інформації в нашій фотографії. (Рисунок.3.54).

```
--Welcome to $t3g0--
1: Encode
2: Decode
1
Enter Source Image Path
Login.png
Enter Message to Hide
Enter Destination Image Path
Login1.png
Encoding...
Image Encoded Successfully
```

Рисунок 3.54 - Процес кодування

Далі нам потрібно ввести логін і пароль, щоб зареєструватися (Рисунок.3.55).

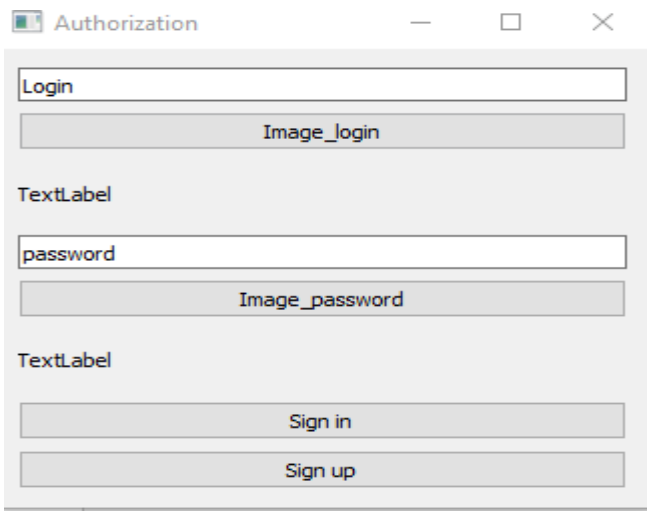

Рисунок 3.55 - Введення логіну/паролю

Знаходимо наші раніше створені фотографії де є сіль, щоб використати функцію гешування (Рисунок.3.56).

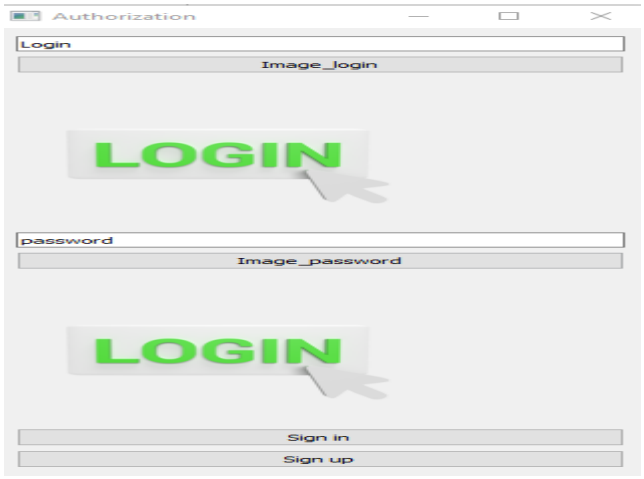

Рисунок 3.56 - Процес обрання фотографії

Для завершення процесу реєстрації, ми повинні натиснути на кнопку “Sign up” (Рисунок.3.57).

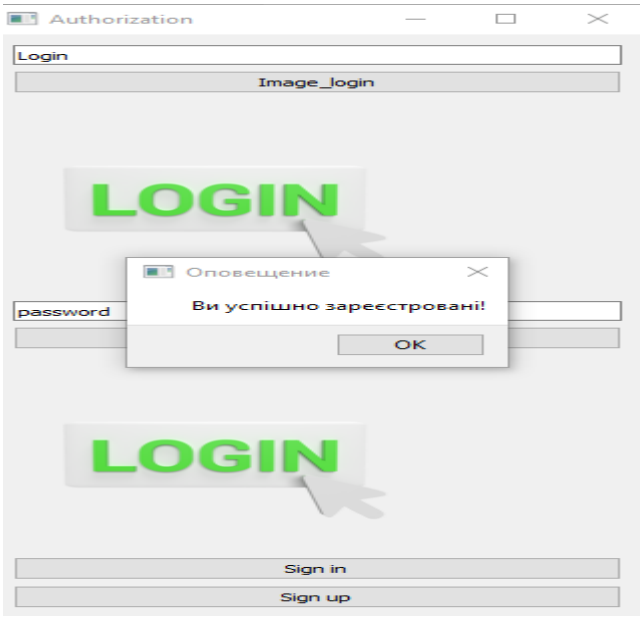

Рисунок 3.57 - Завершення процесу реєстрації користувача

Наш новий створений користувач (Рисунок.3.58).

15	24	8eda175d85db02fa16c72c3ce38da1eb7ef06fe7f2...	738ead210907d0714eac7be371cbd17f0f06c4462...

Рисунок 3.58 - База даних “SQLite”

Тепер для авторизації нам потрібно виконати всі ті самі кроки, що й раніше тільки в цьому випадку потрібно натиснути кнопку ”Sign in” (Рисунок.3.59).

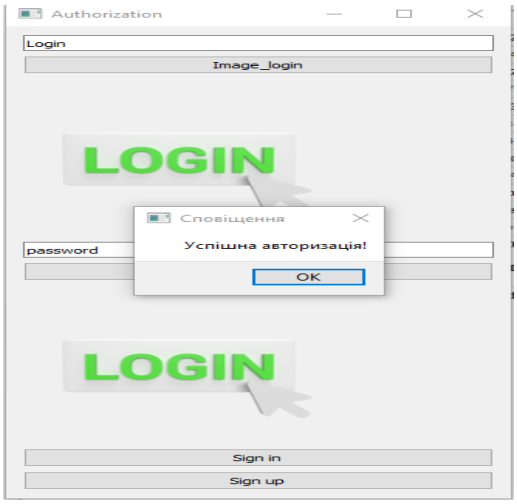

Рисунок 3.59 - Результат успішної авторизації

Для правдивості, що система працює належним чином я виконаю спробу авторизації використовуючи неправдиві дані (Рисунок.3.60).

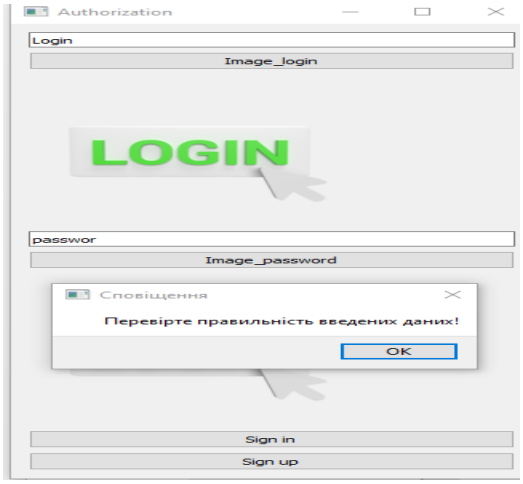

Рисунок 3.60 - Невдала спроба авторизації

Також виконую аналогічне дослідження, тільки цього разу, намагаюся зареєструвати вже наявного користувача (Рисунок.3.61).

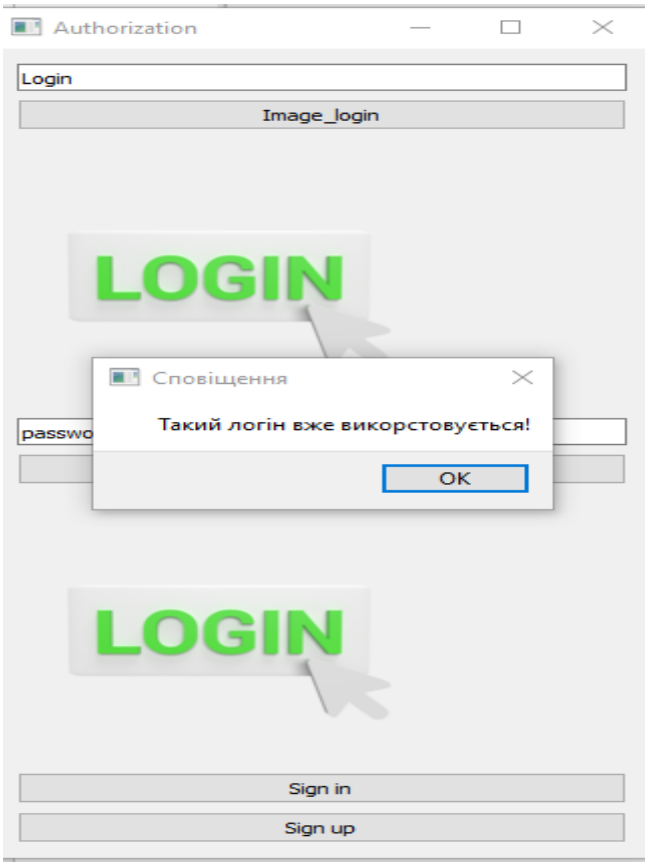

Рисунок 3.61 - Невдала спроба реєстрації

Висновки за розділом 3

У третьому розділі були практично реалізовані три важливі завдання: розробка геш-функцій, розробка генератора паролів і застосування методів стеганографії. Використання потужних інструментів, таких як мова програмування Python, інтегроване середовище розробки PyCharm, вбудована база даних SQLite і фреймворк PyQt5-6 для розробки графічного інтерфейсу, дозволило розробити програмний модуль, який забезпечує збереження логінів і паролів у гешованому вигляді. Загалом, результати цього розділу дають нам потужний і гнучкий інструмент для збереження і захисту конфіденційної інформації.

ВИСНОВКИ

У першому розділі було проведено огляд геш-функцій, генераторів паролів і методів стеганографії. Робота включала дослідження властивостей геш-функцій, вибір надійних алгоритмів гешування, рекомендації зі створення міцних паролів та основні методи стеганографії. Ця інформація допоможе у створені програмного модуля для збереження логінів і паролів у гешованому вигляді, що забезпечить безпеку та цілісність даних.

За результатами другого розділу були описані та використані ключові інструменти для створення програмного модуля збереження логінів і паролів у гешованому вигляді. Мова програмування Python, середовище розробки PyCharm, база даних SQLite та фреймворк PyQt5-6 були використані для досягнення цілей проєкту. Python забезпечує простий синтаксис і високий рівень абстракції, PyCharm є потужним інтегрованим середовищем розробки, SQLite є легкою вбудованою базою даних, а PyQt5-6 надає зручні засоби для розробки графічного інтерфейсу. Використання цих інструментів сприяло створенню гнучкого та безпечного модуля для збереження логінів і паролів, що є важливим аспектом для забезпечення безпеки даних.

У третьому розділі були практично реалізовані три важливі завдання: розробка геш-функцій, розробка генератора паролів і застосування методів стеганографії. Використання потужних інструментів, таких як мова програмування Python, інтегроване середовище розробки PyCharm, вбудована база даних SQLite і фреймворк PyQt5-6 для розробки графічного інтерфейсу, дозволило розробити програмний модуль, який забезпечує збереження логінів і паролів у гешованому вигляді. Загалом, результати цього розділу дають нам потужний і гнучкий інструмент для збереження і захисту конфіденційної інформації.

Результатом роботи є розроблена система автентифікації, яка базується на використанні геш-функції, генератора паролів та стеганографії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Emam, M. M., Aly, A. A., &Omara, F. A. An Improved Image Steganography Method Based on LSB Technique with Random Pixel Selection. International Journal of Advanced Computer Science & Applications, 1(7), pp. 361-366, (2016).

2. Mutaz Rasmi Abu Sara Rashad J. Rasras, Ziad A. AlQadi, Engineering, A Methodology Based on Steganography and Cryptography to Protect Highly Secure Messages Technology & Applied Science Research, Vol.9 Issue 1, Pages 3681-3684, 2019.

3. Gupta, Sunny Gupta, Anuradha Signals, Importance and Techniques of Information Hiding: A Review, International Journal of Computer Trends and Technology (IJCTT) –volume 9number5–Mar 2014.

4. Morton, B. Why we Need to Move to SHA-2. Retrieved January 2014, from https://casecurity.org/2014/01/30/why-we-need-to-move-to-sha-2/

5. Tiwari, H., & Asawa, K. (2012). A secure and efficient cryptographic hash function based on NewFORK-256. Egyptian Informatics Journal, 13, 199–208.

6. Yantao, L. (2016). Collision analysis and improvement of a hash function based on chaotic tent map. Optik, 127, 4484–4489.

7. Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu. Finding Collisions in the Full SHA-1. CRYPTO’05, LNCS 3621, pp17–36, Springer-Verlag, 2005.

8. Biryukov, A., Lamberger, M., Mendel, F., and Nikolic, I. SecondOrder Differential Collisions for Reduced SHA-256. In Lee and Wang [128], pp. 270–287.

9. . Sabzevar, A. P., & Stavrou, A. Universal multi-factor authentication using graphical passwords. In Signal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference on(pp. 625-632). IEEE.

10. Sabzevar, A. P., & Stavrou, A. (2008, November). Universal multi-factor authentication using graphical passwords. InSignal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference on(pp. 625-632). IEEE.

11. K Matrouk, A Al-Hasanat, H Alasha'ary, Z. Al-Qadi, H Al-Shalabi, “Speech fingerprint to identify isolated word person”, World Applied Sciences Journal, Vol. 31, No. 10, pp. 1767-1771, 2014.

12. J. Al-Azzeh, B. Zahran, Z. Alqadi, B. Ayyoub, M. Abu-Zaher, “A Novel zero-error method to create a secret tag for an image”, Journal of Theoretical and Applied Information Technology, Vol. 96. No. 13, pp. 4081-4091, 2018.

13. M. Jose, “Hiding Image in Image Using LSB Insertion Method with ImprovedSecurity and Quality”, International Journal of Science and Research, Vol. 3, No. 9, pp. 2281-2284, 2014.

14. R. M. Patel, D. J. Shah, “Conceal gram :Digital image in image using LSB insertion method”, International Journal of Electronics and Communication Engineering & Technology, Vol. 4, No. 1, pp. 230- 2035, 2013.

15. M. Juneja, P. S. Sandhu, “An improved LSB based Steganography with enhanced Security and Embedding/Extraction”, 3rd International Conference on Intelligent Computational Systems, Hong Kong China, January 26-27, 2013.

16. J. Nadir, Z. Alqadi, A. Abu Ein, “Classification of Matrix Multiplication, Methods Used to Encrypt-decrypt Color Image”, International Journal of Computer and Information Technology, Vol. 5, No. 5, pp. 459-464, 2016.

17. J. N. Abdel-Jalil, “Performance analysis of color image encryption/decryption techniques”, International Journal of Advanced Computer Technology, Vol. 5, No. 4, pp. 13-17, 2016.

18. T. Sivakumar, R. Venkatesan, “A Novel Image Encryption Approach using Matrix Reordering”, WSEAS Transactions on Computers, Vol. 12, No. 11, pp. 407-418, 2013.

19. H. Gao, Y. Zhang, S. Liang, D. Li, “A New Chaotic Algorithm for Image Encryption”, Chaos, Solitons & Fractals, Vol. 29, No. 2, pp. 393- 399, 2006.

20. K. Loukhaoukha, J. Y. Chouinard, A. Berdai, “A Secure Image Encryption Algorithm Based on Rubik's Cube Principle”, Journal of Electrical and Computer Engineering, Vol. 2012, ArticleID 173931, 2011

ДОДАТОК А
ЛІСТИНГ ПРОГРАМИ ГЕНЕРАТОРА ПАРОЛІВ

```
import sys
from PySide6.QtWidgets import QApplication, QMainWindow,QLineEdit
from PySide6.QtGui import QCloseEvent
import buttons
import password
from ui_main import Ui_MainWindow

class PasswordGenerator(QMainWindow):
    def __init__(self):
        super(PasswordGenerator, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.connect_slider_to_spinbox()
        self.set_password()
        self.do_when_password_edit()

        for btn in buttons.GENERATE_PASSWORD:
            getattr(self.ui, btn).clicked.connect(self.set_password)

        self.ui.btn_visibility.clicked.connect(self.change_password_visibility)
        self.ui.btn_copy.clicked.connect(self.copy_to_clipboard)

    def connect_slider_to_spinbox(self) -> None:
     self.ui.slider_length.valueChanged.connect(self.ui.spinbox_length.setValue)
self.ui.spinbox_length.valueChanged.connect(self.ui.slider_length.setValue)
        self.ui.spinbox_length.valueChanged.connect(self.set_password)
```

```
def do_when_password_edit(self) -> None:
    self.ui.line_password.textEdited.connect(self.set_entropy)
    self.ui.line_password.textEdited.connect(self.set_strength)

def get_characters(self) -> str:
    chars = ''

    for btn in buttons.Characters:
        if getattr(self.ui, btn.name).isChecked():
            chars += btn.value

    return chars

def set_password(self) -> None:
    try:
        self.ui.line_password.setText(
            password.create_new(
                length=self.ui.slider_length.value(),
                characters=self.get_characters())
        )
    except IndexError:
        self.ui.line_password.clear()

    self.set_entropy()
    self.set_strength()

def get_character_number(self) -> int:
    num = 0
```

```
    for btn in buttons.CHARACTER_NUMBER.items():
        if getattr(self.ui, btn[0]).isChecked():
            num += btn[1]

    return num

def set_entropy(self) -> None:
    length = len(self.ui.line_password.text())
    char_num = self.get_character_number()

    self.ui.label_entropy.setText(
        f'Entropy: {password.get_entropy(length, char_num)} bit'
    )

def set_strength(self) -> None:
    length = len(self.ui.line_password.text())
    char_num = self.get_character_number()

    for strength in password.StrengthToEntropy:
        if password.get_entropy(length, char_num) >= strength.value:
            self.ui.label_strength.setText(f'Strength: {strength.name}')

def change_password_visibility(self) -> None:
    if self.ui.btn_visibility.isChecked():
        self.ui.line_password.setEchoMode(QLineEdit.Normal)
    else:
        self.ui.line_password.setEchoMode(QLineEdit.Password)

def copy_to_clipboard(self) -> None:
```

```
        QApplication.clipboard().setText(self.ui.line_password.text())

    def closeEvent(self, event: QCloseEvent) -> None:
        QApplication.clipboard().clear()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = PasswordGenerator()
    window.show()
    sys.exit(app.exec())
```

ДОДАТОК Б

ЛІСТИНГ ПРОГРАМИ ДЛЯ ЗАСТОСУВАННЯ СТЕГАНОГРАФІЇ

```
import sys
import numpy as np
from PIL import Image
np.set_printoptions(threshold=sys.maxsize)
import uuid
import hashlib

#encoding function
def Encode(src, message, dest):

    img = Image.open(src, 'r')
    width, height = img.size
    array = np.array(list(img.getdata()))
    password = 'abobus'

    if img.mode == 'RGB':
        n = 3
    elif img.mode == 'RGBA':
        n = 4

    total_pixels = array.size//n

    message += password
    b_message = ''.join([format(ord(i), "08b") for i in message])
    req_pixels = len(b_message)
```

```
    if req_pixels > (total_pixels * 3):
        print("ERROR: Need larger file size")

    else:
        index=0
        for p in range(total_pixels):
            for q in range(0, 3):
                if index < req_pixels:
                    array[p][q] = int(bin(array[p][q])[2:9] + b_message[index], 2)
                    index += 1

        array=array.reshape(height, width, n)
        enc_img = Image.fromarray(array.astype('uint8'), img.mode)
        enc_img.save(dest)
        print("Image Encoded Successfully")


#decoding function
def Decode(src):

    img = Image.open(src, 'r')
    array = np.array(list(img.getdata()))
    password='abobus'

    if img.mode == 'RGB':
        n = 3
    elif img.mode == 'RGBA':
        n = 4
```

```
    total_pixels = array.size//n

    hidden_bits = ""
    for p in range(total_pixels):
        for q in range(0, 3):
            hidden_bits += (bin(array[p][q])[2:][-1])

    hidden_bits = [hidden_bits[i:i+8] for i in range(0, len(hidden_bits), 8)]

    message = ""
    hiddenmessage = ""
    for i in range(len(hidden_bits)):
        x = len(password)
        if message[-x:] == password:
            break
        else:
            message += chr(int(hidden_bits[i], 2))
            message = f'{message}'
            hiddenmessage = message
    #verifying the password
    if password in message:
        print("Hidden Message:", hiddenmessage[:-x])
    else:
        print("You entered the wrong password: Please Try Again")

#main function
def Stego():
    print("--Welcome to $t3g0--")
    print("1: Encode")
    print("2: Decode")
```

```
    func = input()

    if func == '1':
        print("Enter Source Image Path")
        src = input()
        print("Enter Message to Hide")
        message = uuid.uuid4().hex
        print("Enter Destination Image Path")
        dest = input()
        print("Encoding...")
        Encode(src, message, dest)

    elif func == '2':
        print("Enter Source Image Path")
        src = input()
        print("Decoding...")
        Decode(src)

    else:
        print("ERROR: Invalid option chosen")

Stego()
```

ДОДАТОК В

ЛІСТИНГ ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ ЗБЕРЕЖЕННЯ ЛОГІНІВ І ПАРОЛІВ В ГЕШОВАНОМУ ВИГЛЯДІ

```
import sys
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QPushButton,QLabel,QFileDialog
from PyQt5.QtGui import QPixmap
from check_db import *
from des import *
from SHA256 import *
from Pictures import *

class Interface(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.ui = Ui_Form()
        self.ui.setupUi(self)

        self.ui.pushButton.clicked.connect(self.reg)
        self.ui.pushButton_2.clicked.connect(self.auth)
        self.base_line_edit = [self.ui.lineEdit, self.ui.lineEdit_2]

        self.check_db = CheckThread()
        self.check_db.mysignal.connect(self.signal_handler)

        self.button100 =self.findChild(QPushButton, "Image_login")
        self.label100 = self.findChild(QLabel, "png_login")
```

```
    self.button101 = self.findChild(QPushButton, "Image_password")
    self.label101 = self.findChild(QLabel, "png_password")

    self.button100.clicked.connect(self.clicker)
    self.button101.clicked.connect(self.clicker)

def clicker(self):
    fname = QFileDialog.getOpenFileName(self, "Open File")
    #open the image
    self.pixmap = QPixmap(fname[0])
    #add the picture to label
    self.label100.setPixmap(self.pixmap)

    fname1 = QFileDialog.getOpenFileName(self, "Open File")

    self.pixmap1 = QPixmap(fname1[0])
    self.label101.setPixmap(self.pixmap1)
    #отримуємо назву файлів
    self.fname11 = fname[0].split('/')[-1]
    self.fname12 = fname1[0].split('/')[-1]

# Перевірка правильного вводу
def check_input(funct):
    def wrapper(self):
        for line_edit in self.base_line_edit:
            if len(line_edit.text()) == 0:
                return
        funct(self)
    return wrapper
```

```
    # Обробник сигналів
    def signal_handler(self, value):
        QtWidgets.QMessageBox.about(self, 'Сповіщення', value)

    @check_input
    def auth(self):
        name = hashText(Decode(self.fname11),self.ui.lineEdit.text())
        passw = hashText(Decode(self.fname12),self.ui.lineEdit_2.text())
        self.check_db.thr_login(name, passw)

    @check_input
    def reg(self):
        name = hashText(Decode(self.fname11),self.ui.lineEdit.text())
        passw = hashText(Decode(self.fname12),self.ui.lineEdit_2.text())
        self.check_db.thr_register(name, passw)

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    mywin = Interface()
    mywin.show()
    sys.exit(app.exec_())
```