МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження моделей гібридного зберігання зображень для забезпечення безпеки та конфіденційності даних
(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-4
Максим КОЗИНЕЦЬ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення.
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Ірина КИРИЧЕНКО
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Козинця Максима Сергійовича
(Прізвище, ім’я, по батькові)

1. Тема роботи: «Дослідження моделей гібридного зберігання зображень для забезпечення безпеки та конфіденційності даних»
Затверджена наказом по університету №290 Ст від 15.04.2025
2. Термін подання студентом роботи до екзаменаційної комісії 14.06.2025
3. Вихідні дані до роботи *Науково-технічні публікації, дані Інтернет-джерел та друкованих видань щодо гібридних систем зберігання даних, сучасних методів шифрування і забезпечення безпеки, дослідження та розробки хмарних платформ, технічна документація для AWS, Microsoft Azure, Pos*.
4. Перелік питань, які потрібно опрацювати в роботі:
Аналіз предметної галузі. Аналіз літературних та наукових джерел. Постановка задачі дослідження. Аналіз методів вирішення поставленої задачі.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	19.05.2025	*виконано*
2	Аналіз предметної галузі і постановка задачі	20.10.2024	*виконано*
3	Огляд й аналіз літературних, наукових джерел	01.11.2024	*виконано*
4	Постановка задачі	15.11.2024	*виконано*
5	Теоретичне дослідження	14.12.2024	*виконано*
6	Дизайн архітектур гібридних систем	26.02.2025	*виконано*
7	Розробка архітектур гібридних сиситем	21.04.2025	*виконано*
9	Проведення експериментального дослідження	23.04.2025	*виконано*
10	Підготовка до апробації результатів дослідження. Публікація матеріалів	24.04.2025	*виконано*
11	Програмна реалізація	27.05.2025	*виконано*
12	Підготовка пояснювальної записки	07.06.2025	*виконано*
13	Підготовка презентації та доповіді	08.06.2025	*виконано*
14	Перевірка на плагіат	09.06.2025	*виконано*
15	Нормоконтроль	11.06.2025	*виконано*
16	Рецензування	13.06.2025	*виконано*
17	Попередній захист	14.06.2025	*виконано*
18	Занесення диплома в електронний архів	15.06.2025	*виконано*
19	Допуск до захисту у зав. кафедри	15.06.2025	*виконано*

Дата видачі завдання 19.05.2025р.

Студентка _____ Максим Козинець
(підпис)

Керівник роботи _____ доц. Ірина КИРИЧЕНКО
(підпис) (посада, Власне ім’я, прізвище)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 87 с., 9 рис., 7 табл., 16 джерел.

ГІБРИДНЕ ЗБЕРІГАННЯ ЗОБРАЖЕНЬ, АНАЛІЗ, БАЗА ДАНИХ, ВІЗУАЛЬНІ ДАНІ, ЗОБРАЖЕННЯ, ГІБРИДНЕ СХОВИЩЕ, БЕЗПЕКА ДАНИХ, МАСШТАБОВАНІСТЬ, ПРОДУКТИВНІСТЬ, ХМАРНІ СЕРВІСИ, AWS, AZURE

Об’єктом дослідження є гібридні системи зберігання зображень, які поєднують локальні та хмарні ресурси та способи для забезпечення безпеки, конфіденційності та доступності даних для цих систем.

Метою роботи є проектування декількох систем гібридного сховища на основі існуючих архітектур, аналіз та вибір декількох методів шифрування, які ці системи будуть використовувати для порівняння ефективності конкретного методу шифрування для конкретної архітектури.

Методи дослідження включають аналіз наукових публікацій та різних моделей гібридного зберігання, методів забезпечення безпеки, проектування прототипів системи та експерименте тестування цих систем з різніми тестовими даними.

У результаті роботи були проведені дослідження методів сегментації та класифікації медичних зображень, запропоновано порівняння результатів для комерційних і відкритих систем.

Практичне значення роботи полягає у можливості застосування отриманих результатів у різних галузях, таких як медицина, електронна комерція та безпекові платформи.

HYBRID IMAGE STORAGE, ANALYSIS, DATABASE, VISUAL DATA, IMAGES, HYBRID STORAGE, DATA SECURITY, SCALABILITY, PERFORMANCE, CLOUD SERVICES, AWS, AZURE.

The object of the research is hybrid image storage systems that combine local and cloud resources and methods for ensuring security, confidentiality and availability of data for these systems.

The purpose of the work is to analyze modern models of hybrid image storage, identify the advantages and disadvantages of different approaches and study the optimal combination of security and efficiency of such systems.

The research methods include the analysis of scientific publications, comparison of different hybrid storage models, security methods, as well as the development and testing of system prototypes.

The results of the work include the analysis of methods and metrics for comparing the security and performance of hybrid systems for further research.

The practical significance of the work lies in the possibility of applying the obtained results in various industries, such as medicine, e-commerce and security platforms.

Завідувачу кафедри
ПІ_____
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ
(вчене звання, власне ім'я, прізвище)

ЗАЯВА

щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Козинець Максим Сергійович, здобувач вищої освіти на другому (магістерському) рівні вищої освіти академічної групи ІПЗм-23-4
кафедра _____програмної інженерії_____
(повна назва кафедри)
заявляю: моя кваліфікаційна робота на тему
Дослідження моделей гібридного зберігання зображень для забезпечення безпеки та конфіденційності даних_____,
(назва роботи)
що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в репозиторії "ElArKhNURE". Погоджуюся з авторським договором, відповідно до Положення про репозиторій ХНУРЕ "ElArKhNURE". Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з вимогами академічної доброчесності, згідно з якими виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

11.06.2025 Максим КОЗІНЕЦЬ

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Amazon Web Services (AWS) – комерційна публічна хмара, що підтримується і розвивається компанією Amazon з 2006 року.

Google Cloud Platform (укр. «Хмарна платформа Google», скор. GCP) – наданий компанією Google набір хмарних служб, які виконуються на тій самій інфраструктурі, яку Google використовує для своїх продуктів, призначених для кінцевих споживачів, таких як Google Search та YouTube.

AES (Advanced Encryption Standard) – алгоритм симетричного шифрування (тобто ключі шифрування та дешифрування збігаються).

RSA (абревіатура від прізвищ Rivest, Shamir та Adleman) – криптографічний алгоритм з відкритим ключем, що ґрунтується на обчислювальній складності завдання факторизації великих напівпростих чисел.

ВСТУП

Сучасний розвиток інформаційних технологій вимагає зростаючої уваги до забезпечення безпеки та конфіденційності даних, особливо в умовах збільшення обсягів цифрової інформації та поширення кіберзагроз. Однією з ключових областей, яка потребує постійного вдосконалення, є зберігання зображень. Це стосується як зберігання особистих фото, медичних даних, так і критичної інформації для бізнесу. Традиційні методи зберігання, такі як локальні сховища чи централізовані хмарні системи, не завжди відповідають сучасним вимогам до захисту від несанкціонованого доступу, крадіжок або втрати даних.

У відповідь на ці виклики з'являються моделі гібридного зберігання. Вони поєднують переваги різних підходів, таких як хмарні сховища, локальні носії. Гібридні моделі дозволяють не лише забезпечити високу доступність та масштабованість даних, але й підвищити їх захищеність. Однак існує така проблема: чим складніші методи шифрування використовуються, тим більші ресурси потрібні для їхньої обробки, що може негативно вплинути на продуктивність системи. Тому треба знаходити компроміс та баланс під час розробки системи між рівнем безпеки та швидкістю доступу до даних.

Актуальність теми дослідження обумовлена необхідністю розробки та вдосконалення підходів до зберігання зображень, які забезпечують конфіденційність, цілісність і доступність даних навіть за умов потенційних загроз.

Об'єктом дослідження є процеси захисту та гібридного зберігання зображень.

Головною метою цієї роботи є дослідження оптимальної комбінації моделі гібридного сховища даних із методами захисту даних для різних задач. Важливо визначити, які методи найкраще підходять для зберігання та обробки різних типів зображень, таких як особисті фото, високоякісні знімки, медичні зображення тощо.

Для досягнення цієї мети у роботі ставляться наступні задачі:

- проаналізувати та обрати декілька моделей гібридних систем, на архітектурі яких буде подальшому реалізовані свої моделі;
- проаналізувати методи забезпечення безпеки для таких систем та обрати декілька для подальшого застосування у своїх моделях;
- спроектувати загальну архітектуру для обраних гібридних систем з різними модулями безпеки, для подальшого зрівняння безпеки та продуктивності;
- проаналізувати та вивести метрики для урахування безпеки та продуктивності системи;
- провести практичний експеримент на спроектованих гібридних системах системах викоритовуючи тестові дані;
- проаналізувати отримані результати та зробити висновки.

Методологія дослідження включає декілька етапів. Аналіз існуючих рішень гібридних систем зберігання даних та методів їх захисту. Дослідження їхнього використання у різних ситуаціях, таких як шифрування приватних фото, безпечне збереження знімків високої роздільної здатності, обробка медичних зображень із підвищеним рівнем конфіденційності. Порівняння рівня безпеки та продуктивності різних комбінацій систем зберігання та методів шифрування.

У результаті роботи буде проведено комплексний аналіз, який дозволить знайти оптимальний баланс між безпекою та швидкістю доступу до даних. Це дозволить створити рекомендації для побудови ефективної системи гібридного зберігання зображень, яка відповідатиме вимогам конфіденційності та продуктивності.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі дослідження

У сучасному цифровому світі обсяг та якість зображень, що зберігаються в інформаційних системах, стрімко зростають. Це зумовлено розвитком технологій, доступністю високоякісних камер у смартфонах, цифрових фотоапаратів, дронів, а також зростанням використання візуального контенту у соціальних мережах, медицині, промисловості та інших сферах. Сьогодні фотографії та відео є основними форматами даних в інтернеті. За даними компанії Google, у її сервісі «Photo» у 2020 році зберігалося понад 4 трильйони фотографій, а щотижня додавалося ще 28 мільярдів нових зображень. Очікується, що до 2025 року загальний обсяг даних у світі перевищить 175 зетабайт, значну частину з яких становитимуть зображення та відео [1].

Зі збільшенням обсягів даних виникає потреба в ефективних та гнучких системах зберігання. Гібридний хмарний підхід до керування сховищем даних використовує як хмарні, так і локальні ресурси, поєднуючи можливості приватних і загальнодоступних хмар для формування інтегрованої архітектури зберігання (див. рис. 1.1).

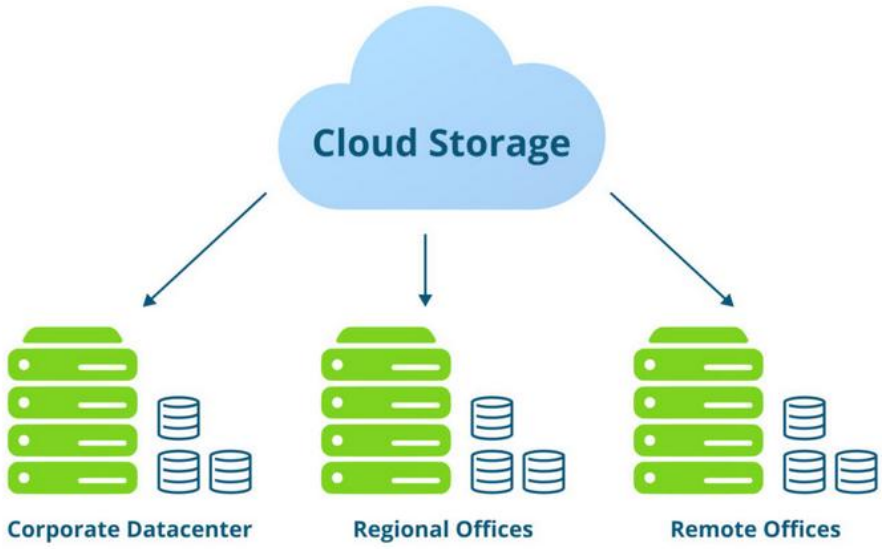

Рисунок 1.1 – Концепція гібридного хмарного сховища [2]

Гібридне хмарне сховище файлів пропонує можливості, які можуть принести користь компаніям будь-якого розміру, включаючи гнучкість, масштабованість і економію. Гібридна хмарна архітектура – це тип інфраструктури центру обробки даних, яка поєднує приватні та загальнодоступні хмари разом із локальними ресурсами та може обмінюватися даними в усіх середовищах. Гібридні хмарні системи дозволяють зберігати критично важливі дані на власних серверах, а менш чутливу інформацію – у публічних хмарах.

Існує велика кількість хмарних сервісів, серед яких найбільш популярні AWS, Microsoft Azure та GCP (рис. 1.2) [3].

#	Cloud Service Provider	Regions	Availability Zones
1	Amazon Web Services (AWS)	33	105
2	Microsoft Azure	64	126
3	Google Cloud Platform (GCP)	40	121
4	Alibaba Cloud	30	89
5	Oracle Cloud	48	58
6	IBM Cloud	10	30
7	Tencent Cloud	21	65
8	OVHcloud	17	37
9	DigitalOcean	9	15
10	Linode (Akamai)	20	20

Рисунок 1.2 – Список найпопулярніших хмарних сервісів [3]

Кожен із цих сервісів має свої особливості та переваги. AWS відомий своєю широкою інфраструктурою та багатим набором послуг, що робить його привабливим для великих підприємств з високими вимогами до масштабованості. Microsoft Azure інтегрується з продуктами Microsoft, такими як Windows Server, Active Directory та SQL Server, що є зручним для організацій, які вже використовують ці технології. Google Cloud Platform виділяється своїми можливостями в області обробки великих даних та машинного навчання, пропонуючи такі інструменти, як BigQuery та TensorFlow. Alibaba Cloud є

провідним хмарним провайдером у Китаї та Азіатсько-Тихоокеанському регіоні. Oracle Cloud спеціалізується на корпоративних рішеннях і забезпечує високу продуктивність баз даних.

За прогнозами компанії Gartner [4], у 2021 році 75% великих та середніх бізнесів використовуватимуть саме гібридний або мультихмарний підхід. У корпоративному секторі такі системи дозволяють оптимізувати витрати, підвищити продуктивність та забезпечити резервне копіювання даних, що особливо важливо для компаній, які працюють з великими обсягами графічної інформації.

Однак, разом із перевагами гібридних систем виникають і виклики, зокрема щодо безпеки збережених зображень. Незахищені або недостатньо захищені дані можуть стати об'єктом кібератак, що призводить до витоку конфіденційної інформації. Згідно зі звітом IBM X-Force Threat Intelligence [5], атаки на дані становлять понад 19% всіх кіберзагроз у світі, причому витік конфіденційних файлів через несанкціонований доступ або хакерські атаки є однією з основних загроз для підприємств та користувачів.

Стаття Veam «Data Protection Opportunities in the Hybrid Cloud» [6] визначає наступний список основних аспектів захисту даних:

- шифрування даних як у стані спокою, так і під час передачі;
- реалізація суворих політик управління ідентифікацією та доступом (IAM) з використанням багатофакторної автентифікації;
- регулярне резервне копіювання даних та розробка планів відновлення після збоїв;
- постійний моніторинг та аналіз журналів для виявлення аномалій та потенційних загроз.

У гібридних системах також дуже важлива розробка багаторівневої архітектури безпеки, що включає декілька різних видів захисту даних.

Шифрування є одним із ключових методів забезпечення безпеки даних, оскільки воно ускладнює несанкціонований доступ до інформації. Але вибір

потрібного методу є складним завданням через велику кількість доступних алгоритмів. Кожен з них має свої переваги, недоліки та сферу застосування. Наприклад, потрібно зашифрувати велику колекцію медичних знімків (МРТ-скани), які мають високу роздільну здатність та повинні бути збережені у довготривалому архіві. AES добре підходить для шифрування окремих файлів, але не є оптимальним для зображень, які зберігаються у стислих форматах, таких як JPEG або PNG. Це пов’язано з тим, що AES працює з блоками фіксованого розміру (наприклад, 128 біт), і якщо застосувати його без адаптації до стислих форматів, це може призвести до збільшення розміру файлу або порушення структури стиснення.

Аналіз предметної області показує, що існує велика різноманітність типів зображень, кожен з яких має свої особливості. Наприклад, медичні знімки вимагають високого рівня конфіденційності, тоді як зображення в соціальних мережах більше орієнтовані на швидкість доступу та обробки. Водночас моделі гібридних систем також відрізняються між собою. Більш того використовуються різні варіанти хмарних платформ, таких як AWS, Azure або Google Cloud, тощо.

У таких системах безпека відіграє ключову роль. Більшість сучасних платформ використовують стандартні модулі безпеки, такі як багатофакторна автентифікація, контроль доступу та механізми журналювання. Однак шифрування є унікальним компонентом для кожної конкретної системи, оскільки саме воно визначає рівень захисту даних від несанкціонованого доступу. Існує велика кількість методів шифрування, кожен із яких має свої переваги та недоліки.

Тому необхідно провести детальне дослідження з метою визначення найбільш ефективних методів для різних випадків використання гібридних систем зберігання.

1.2 Постановка задачі

Дана робота спрямована на вивчення найефективнішої комбінації моделі гібридного сховища даних із методами їхнього захисту для різних сценаріїв

використання. Ключовим завданням є визначення оптимальних підходів до зберігання та обробки різних категорій зображень, включаючи особисті фотографії, високоякісні знімки, медичні зображення тощо.

У ході виконання роботи треба виконати наступні завдання:

- обрати декілька існуючих архітектур моделей гібридних систем, для подальшої реалізації;
- вирішити задачу багатокритеріального вибору для хмарного сервісу;
- дослідити та вибрати декілька сучасних методів шифрування для зображень, які будуть використовуватись у досліджуваних гібридних системах;
- визначити формати тестових даних (форматів зображення);
- спланувати архітектури гібридних моделей на базі обраних структур;
- проаналізувати методи та метрики порівняння безпеки та продуктивності гібридних систем;
- програмно реалізувати обрані системи і методи шифрування;
- провести практичний експеримент над тестовими даними та отримати результати;
- проаналізувати результати та зробити на їх основі висновки.

2 ТЕОРИТИЧНЕ ДОСЛІДЖЕННЯ

2.1 Використані сучасні архітектури гібридних систем

У цьому розділі розглянуто сучасні підходи до побудови гібридних систем зберігання зображень. Оскільки основною метою роботи є визначення оптимального поєднання моделі гібридного сховища та алгоритму захисту, важливо проаналізувати існуючі рішення. У цьому контексті розглянуті кілька актуальних архітектур, які в подальшому будуть використані у дослідженні.

Архітектура програмного забезпечення для систем гібридного зберігання зображень є ключовим елементом, який забезпечує належну функціональність, масштабованість і захист даних. Основна мета архітектури – створити гнучку й ефективну систему, яка об’єднує можливості локальних і хмарних ресурсів. Існують декілька підходів для зберігання зображень у гібридних системах. У наукових публікаціях та аналітичних звітах представлено різноманітні підходи до побудови гібридних систем, які відрізняються за типом інтеграції, рівнем автоматизації та застосованими технологіями. Дослідження таких архітектур дозволяє визначити оптимальні рішення для конкретних сфер використання, зокрема для збереження та обробки зображень, що потребують високого рівня безпеки.

У дослідженні [7] розглядається архітектура, яка поєднує блокчейн-технології та традиційні бази даних для ефективного зберігання зображень. Основна ідея полягає у використанні блокчейну для забезпечення незмінності та прозорості даних, тоді як традиційні бази даних відповідають за швидкий доступ та масштабованість. У цій архітектурі метадані зображень, такі як хеші та інформація про права доступу, зберігаються в блокчейні, що гарантує їхню цілісність та захист від несанкціонованих змін. Самі ж зображення зберігаються у традиційних базах даних, що забезпечує ефективне управління великими обсягами даних та швидкий доступ до них.

Архітектура даної системи має декілька блоків (див. рис 2.1).

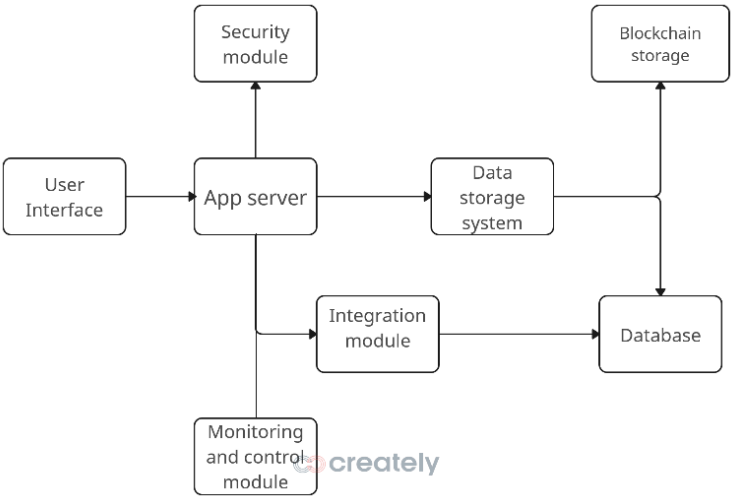

Рисунок 2.1 – Схема архітектури гібридної системи з використанням блокчейн та традиційних баз даних (виконано самостійно)

Користувацький інтерфейс дозволяє завантажувати, переглядати та керувати зображеннями. Взаємодія із системою здійснюється через веб-застосунок або мобільний додаток. Сервер застосунків обробляє запити від користувацького інтерфейсу, керує логікою застосунку, включаючи обробку зображень та роботу з системами зберігання даних. Система зберігання даних складається з блокчейн-сховища та традиційної бази даних. Блокчейн використовується для збереження хешів зображень і метаданих, що гарантує незмінність та перевіреність інформації, а також забезпечує децентралізоване зберігання, підвищуючи безпеку та надійність. Традиційна база даних містить самі зображення або посилання на них, що забезпечує швидкий доступ і ефективне управління великими обсягами даних. Модуль безпеки шифрує дані під час зберігання та передачі, а також керує автентифікацією та авторизацією користувачів. Модуль інтеграції синхронізує дані між блокчейн-мережею та традиційною базою даних, забезпечуючи їх узгоджене використання. Модуль моніторингу та управління відстежує стан системи, контролює її продуктивність та надає інструменти для адміністрування.

Подібний підхід до інтеграції блокчейн-рішень для управління даними розглядається і в роботі [8], де запропонована концепція використання

децентралізованих технологій для підвищення безпеки зберігання візуальних даних. У дослідженні аналізуються різні аспекти розподіленого управління метаданими та запропонована модель, яка дозволяє ефективно синхронізувати збережені дані між різними компонентами гібридної архітектури. Додатково автори розглядають можливість використання криптографічних підходів для збереження конфіденційності даних, що є актуальним для безпечного зберігання та обробки зображень.

Основні елементи цієї архітектури включають:

- блокчейн-компонент. Усі транзакції, пов'язані зі зберіганням зображень, записуються в блокчейн. Це забезпечує аудит, захист від несанкціонованих змін та можливість відстеження доступу до файлів;
- традиційні бази даних. Для зберігання великих обсягів зображень використовується реляційна база даних, така як PostgreSQL. Метадані про файли (розташування, власник, дата завантаження) зберігаються в таблицях бази даних;
- гібридна інтеграція. Механізми взаємодії між блокчейн та базою даних дозволяють синхронізувати записи, забезпечуючи цілісність даних у системі.

Особливістю цього підходу є високий рівень безпеки та прозорості. Проте він вимагає значних ресурсів для підтримки блокчейн-мережі, що може ускладнювати масштабування.

Інший підхід представлений у [9], де запропонована архітектура багатовимірного сховища даних, що дозволяє ефективно виконувати запити як до агрегованих, так і до детальних даних.

Сховище, побудоване за цим підходом, містить список компонентів (див. рис. 2.2) такі як операційні бази даних (OSS), перехідна область (DSS), вітрини даних (DPA), засоби доступу до даних (DAT).

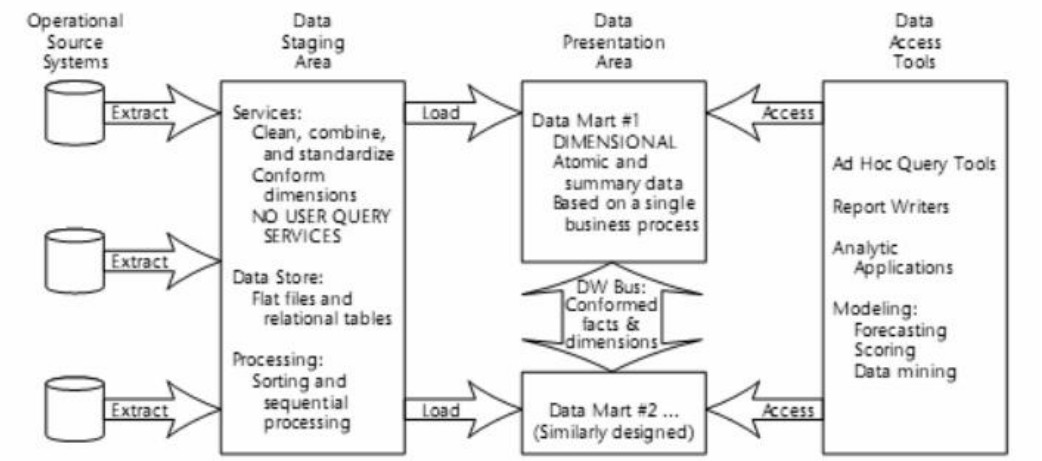

Рисунок 2.2 – Схема архітектури багатовимірного сховища даних [9]

У цьому сховищі операційні бази даних (OSS) зберігають первинну інформацію. Перехідна область (DSA) слугує для перенесення даних з OSS до вітрин даних (DPA). Вітрини даних (DPA) є компонентом, у якому містяться як детальні, так і агреговані дані. Способом організації області представлення даних слугує шина, для якої вводиться матриця шини вимірів. Ця матриця описує наявність вимірів для кожної вітрини, застосовуючи поняття узгодженості вимірів.

Основні характеристики цієї архітектури:

- використовується для зберігання метаданих, наприклад, таблиці бази даних можуть містити інформацію про категорії зображень, авторів або часові мітки;
- зображення та великі мультимедійні файли зберігаються у форматі об'єктів, що дозволяє оптимізувати зберігання та доступ до них;
- для прискорення доступу до часто використовуваних файлів застосовується кешування, яке зберігає результати найпопулярніших запитів;
- система забезпечує можливість отримання агрегованих даних, що є корисним для аналітики та бізнес-рішень.

Ця архітектура відрізняється балансом між продуктивністю та витратами на інфраструктуру. Вона особливо підходить для систем, які працюють з великим обсягом даних, але не вимагають постійного доступу до всіх файлів.

У статті [10] описано підхід, при якому локальні ресурси поєднуються з хмарними сервісами. Основна мета такої системи – забезпечити безперервність бізнес-процесів і швидкий доступ до критично важливих даних. Гібридне хмарне сховище є архітектурним підходом, що поєднує можливості локального зберігання даних та хмарних сервісів. Така модель забезпечує гнучкість, масштабованість і надійність, роблячи її оптимальним вибором для зберігання великих обсягів даних, таких як зображення.

Додатково, у дослідженні [11] розглядається модель побудови гібридного середовища, що одночасно враховує питання безпеки, ефективності та дотримання політик відповідності. Запропонована архітектура у поєднанні зі стратегіями шифрування та керування потоками даних демонструє ефективність при роботі в багатохмарних середовищах, таких як AWS та Azure, що підтверджує релевантність обраного підходу.

Гібридне сховище будується на основі двох основних компонентів – локального сервера і хмарного провайдера. Ключовим елементом архітектури є модуль маршрутизації, який відповідає за визначення, куди будуть спрямовані дані. Локальне сховище використовується для даних, до яких потрібен швидкий доступ (наприклад, останні завантаження). Це зменшує затримки, спричинені мережею. Хмарне сховище AWS S3, або аналогічні сервіси, забезпечують зберігання великих обсягів даних, що не часто використовуються. Модуль маршрутизації відповідає за визначення, які дані слід залишити локально, а які передати до хмари. Це дозволяє оптимізувати витрати на зберігання.

Як виглядає архітектура гібридного хмарного сховища (див. рис. 2.3):

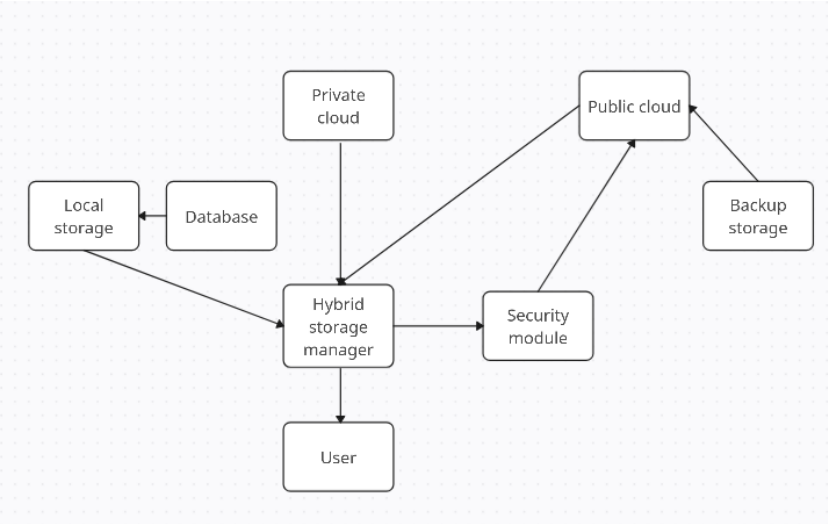

Рисунок 2.3 – Архітектура гібридного хмарного сховища (виконано самостійно)

Розглянуті архітектури гібридних систем мають як відмінності, так і спільні риси. Наприклад, система, що використовує блокчейн-технології, забезпечує високий рівень безпеки та прозорості, проте вимагає значних ресурсів для підтримки. У той же час, архітектура багатовимірного сховища даних орієнтована на ефективне управління великими обсягами даних і оптимізацію продуктивності. Спільною рисою для всіх моделей є поєднання локальних і хмарних ресурсів для забезпечення балансу між швидкістю доступу та масштабованістю. Це підтверджує необхідність подальшого дослідження оптимальних підходів до гібридного зберігання, з урахуванням специфіки конкретних завдань та вимог безпеки.

2.2 Огляд методів шифрування зображень

У цьому розділі розглядаються сучасні алгоритми захисту зображень, що забезпечують високий рівень безпеки та оптимальну продуктивність. Це дозволить оцінити, наскільки ефективно кожна гібридна система працює із певним алгоритмом шифрування, що є ключовим для визначення оптимального поєднання технологій у подальшому дослідженні.

Алгоритми шифрування даних є ключовим елементом забезпечення безпеки в гібридних системах зберігання, особливо коли йдеться про чутливі дані, такі як

зображення. Використання сучасних криптографічних алгоритмів дозволяє забезпечити цілісність, конфіденційність і захищеність даних як під час їхнього зберігання, так і під час передачі.

Як зазначено у статті [12], використання передових алгоритмів шифрування, зокрема RSA та AES-256, може значно підвищити стійкість систем до кібератак.

AES-256 є симетричним блоковим алгоритмом шифрування, що працює з блоками даних розміром 128 бітів. Його стійкість базується на використанні 256-бітного ключа, що робить алгоритм надзвичайно захищеним від сучасних атак, включаючи брутфорс.

Формула основного етапу AES включає наступні кроки.

Крок 1. SubBytes – кожен байт замінюється за допомогою таблиці замін (S-Box). У контексті AES, S-Box (Substitution Box) – це таблиця замін, яка використовується для нелінійної заміни байтів у процесі шифрування. S- Box забезпечує криптографічну стійкість, роблячи алгоритм нечутливим до лінійного криптоаналізу. Таблиця S-Box створена на основі зворотних елементів у скінченному полі $GF(2^8)$ із додатковими афінними перетвореннями, що збільшує рівень дифузії та заплутаності даних.

Крок 2. MixColumns – дані змішуються за допомогою лінійної трансформації;

Крок 3. ShiftRows – байти рядків блоку зміщуються на певну кількість позицій;

Крок 4. MixColumns – дані змішуються за допомогою лінійної трансформації;

Крок 5. AddRoundKey – до кожного блоку додається раундовий ключ.

Цикли повторюються кілька разів залежно від довжини ключа (14 раундів для 256-бітного ключа).

RSA є асиметричним алгоритмом шифрування, що базується на складності розкладу великих чисел на прості множники. Для шифрування даних використовується публічний ключ, а для розшифрування – приватний.

Основні формули.

Генерація ключів:

$$n = p \cdot q, \tag{2.1}$$

де p і q – великі прості числа,

n – модуль.

Шифрування:

$$c = m^e mod(n), \tag{2.2}$$

де m – повідомлення,

e – експонента.

Розшифрування:

$$m = c^d mod(n), \tag{2.3}$$

де d – приватний ключ, обчислений як $d \cdot e \equiv 1\ mod\ \phi(n)$,

$\phi(n)$ – функція Ейлера.

У дослідженні [13] «Про один алгоритм шифрування-дешифрування зображень з використанням порозрядних операцій» запропоновано стійкий алгоритм шифрування та дешифрування зображень, який забезпечує широкий діапазон ключа та захист від основних методів криптоаналізу. Особливістю цього підходу є використання порозрядних операцій, що унеможливлюють візуальне розпізнавання деталей зображення. Це досягається шляхом застосування тригонометричних функцій, які чутливі до змін аргументу, забезпечуючи нелінійність перетворень інтенсивностей пікселів.

Алгоритм працює наступним чином.

Крок 1. Зображення розглядається як послідовність дискретних сигналів з кількістю пікселів $w \times h$, де w – ширина, а h – висота зображення. Воно представляється у вигляді вектора послідовних пікселів:

$$a = \{a_1, a_2, \dots, a_{w \times h}\}, \tag{2.4}$$

де a_i – значення інтенсивності i-го пікселя.

Крок 2. Ключем є символьний рядок довільної довжини mmm, представлений як вектор:

$$F = \{F_0, F_1, \dots, F_m - 1\}, \tag{2.5}$$

Початкове значення ключа F_0 може бути, наприклад, F_0=*"23f3cba216"*.

Крок 3. Визначається рекурсивна функція перетворення ключа:

$$F_i + 1 = f(F_i), \tag{2.6}$$

де f – задана функція,

F_i – попереднє значення функції.

При i=1 у функцію передається F_0.

Крок 4. Алгоритм шифрування:

Обчислюється значення функції від *F0*

$$\mathrm{F_1} = \mathrm{f(F_0)}, \tag{2.7}$$

Отримане значення додається за модулем 2 до поточного значення *ai*:

$$a_i' = a_i \oplus F_1 \tag{2.8}$$

де $\oplus$ – операція XOR.

Обчислюється наступне значення функції:

$$Fi + 2 = f(Fi + 1) \tag{2.9}$$

Кроки повторюються для всіх елементів вектора а $w \times h$ разів.

Крок 5. Завдяки симетрії операції додавання за модулем 2, для дешифрування до зашифрованого зображення застосовуються ті ж самі кроки алгоритму шифрування.

У дослідженні [14] автори запропонували метод шифрування зображень у вейвлет-області, який поєднує хаотичне перемішування та модуляцію. Спочатку зображення перетворюється з просторової області у вейвлет-область за допомогою перетворення Хаара. Потім застосовується хаотичне перемішування Fisher-Yates для заплутування зв'язку між оригінальним та зашифрованим зображеннями. Далі результуючі коефіцієнти модулюються за допомогою хаотичної карти. Цей підхід забезпечує високий рівень безпеки та ефективності при шифруванні зображень.

Основні етапи алгоритму.

Крок 1. Перетворення зображення у вейвлет-область: зображення перетворюється з просторової області у вейвлет-область за допомогою перетворення Хаара. Це дозволяє розділити зображення на апроксимуючі та детальні коефіцієнти, що спрощує подальше оброблення.

Крок 2. Хаотичне перемішування: для перемішування коефіцієнтів використовується алгоритм Фішера-Йетса, модифікований хаотичною картою. Це знижує кореляцію між сусідніми пікселями та ускладнює відновлення початкового зображення без ключа.

Крок 3. Модуляція та дифузія: після перемішування апроксимуючі коефіцієнти модулюються за допомогою хаотичної послідовності, що забезпечує додаткову дифузію та підвищує стійкість до криптоаналізу.

Алгоритм має наступні основні математичні формули:

– Вейвлет-перетворення Хаара: апроксимуючі (*A*) та детальні (*D*) коефіцієнти обчислюються як:

$$A = \frac{a+b}{\sqrt{2}}, \tag{2.10}$$

$$D = \frac{a-b}{\sqrt{2}}, \tag{2.11}$$

де *a* та *b* – сусідні пікселі.

– Хаотична карта: використовується кусочно-лінійна хаотична карта для генерації псевдовипадкових чисел:

$$x_{n+1} = \begin{cases} \frac{x_n}{p}, 0 \le x_n < p \\ \frac{(1-x_n)}{1-p}, p \le x_n \le 1 \end{cases}, \tag{2.12}$$

де x_n – значення на поточному кроці,

x_n+1 – значення на наступному кроці,

p – параметр хаотичної карти, що визначає поведінку генерації.

У дослідженні «Image Encryption Decryption Using Chaotic Logistic Mapping and DNA Encoding» [15] автори запропонували метод шифрування зображень, який поєднує хаотичне логістичне відображення та ДНК-кодування. Спочатку зображення перетворюється за допомогою логістичного відображення для створення хаотичної послідовності, яка забезпечує дифузію пікселів. Потім застосовується ДНК-кодування, де пікселі зображення кодуються у послідовності нуклеотидів (A, T, G, C), що підвищує рівень заплутаності та безпеки. Цей підхід забезпечує кращі результати порівняно з методами, які використовують лише хаотичне відображення, завдяки підвищеній складності та стійкості до атак.

Основні етапи алгоритму.

Крок 1. Генерація хаотичної послідовності: використовується логістичне відображення, яке визначається рівнянням:

$$x_n + 1 = r \times x_n \times (1 - x_n), \quad (2.13)$$

де r – параметр системи,

x_n – значення на n-му кроці.

Крок 2. Кодування ДНК: піксельні значення зображення перетворюються у ДНК-послідовності за допомогою правил кодування, де кожному двійковому значенню відповідає одна з нуклеотидних пар:

– 00: A (Аденін);
– 01: C (Цитозин);
– 10: G (Гуанін);
– 11: T (Тимін).

Крок 3. Диффузія та перемішування: застосовується хаотична послідовність для перемішування та модифікації ДНК-кодів пікселів, забезпечуючи дифузію та заплутування інформації.

Крок 4. Декодування ДНК: після обробки ДНК-послідовності перетворюються назад у двійкові значення, а потім у піксельні значення для отримання зашифрованого зображення.

Проведений аналіз дозволив розглянути 5 алгоритмів шифрування зображень, кожен із яких має свої особливості та сфери застосування. Досліджені методи відрізняються один від одного як підходами до шифрування, так і рівнем безпеки. Наприклад, метод, що використовує хаотичне логістичне відображення та ДНК-кодування, забезпечує високу стійкість до атак, тоді як комбінований метод шифрування та стеганографії може бути корисним для прихованого зберігання даних. Ці алгоритми будуть використані для подальшого дослідження.

3 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Використані технології

Для здійснення дослідження треба реалізовані гібридні системи на основі 3 оглянутих архітектур. Кожна система має модуль безпеки, який усередині буде мати один з алгоритмів шифрування. Алгоритм у цьому модулі можна буде налаштовувати, тобто для першого тесту один алгоритм, для другого – інший і так далі. Таким чином, можна дослідити роботу конкретної системи з конкретним методом шифрування.

Усі компоненти системи реалізовано мовою програмування Java із використанням стандартної бібліотеки Java SE, що забезпечує повну автономність від зовнішніх фреймворків. Для створення графічного інтерфейсу застосовано бібліотеку Swing, яка входить до складу Java та дозволяє створювати кросплатформені GUI-додатки з підтримкою користувацької взаємодії.

Серверна логіка та алгоритми шифрування реалізовані у вигляді локальних класів без застосування REST-сервісів або сторонніх кешуючих платформ. Для зберігання зашифрованих файлів застосовано гібридну файлову систему, яка імітує поділ на локальне та хмарне сховище. Залежно від розміру об’єкта, дані зберігаються у відповідних каталогах (storage/local/ або storage/cloud/), що дозволяє емулювати поведінку реальних розподілених систем зберігання.

3.2 Аналіз вхідних даних

У процесі розробки гібридних систем зберігання зображень треба визначити, які формати та типи вхідних даних будуть використовуватися. Існує велика кількість форматів зображень, кожен з яких має свої особливості та сферу застосування. У дослідженні будуть використані наступні формати:

– JPEG (Joint Photographic Experts Group) – один із найпоширеніших форматів для зберігання цифрових фотографій. Використовує стиснення з втратою якості, що дозволяє зменшити розмір файлу при збереженні візуального вигляду;

- PNG (Portable Network Graphics) – формат із підтримкою прозорості, що забезпечує високу якість зображення без втрат. Часто використовується для графічного контенту, що потребує збереження деталей;
- TIFF (Tagged Image File Format) – формат із мінімальним стисненням, який забезпечує високу якість та використовується в професійній фотографії та друку;
- BMP (Bitmap Image File) – нерідко застосовується для збереження зображень без втрат, але має великий розмір файлу;
- GIF (Graphics Interchange Format) – використовується для анімаційних зображень, обмежений палітрою з 256 кольорів;
- WEBP – сучасний формат, розроблений Google, що підтримує як стиснення з втратами, так і без втрат, зберігаючи високу якість при меншому розмірі файлу;
- DICOM (Digital Imaging and Communications in Medicine) – стандартний формат для зберігання та передачі медичних зображень, таких як рентгенівські знімки, МРТ, КТ. Він містить не лише графічні дані, але й метаінформацію про пацієнта та параметри знімка.

Зображення буде проходити декілька етапів. При завантаженні, воно буде у стані файлу певного формату. Потім файл конвертується у формат масиву байтів для подальшого шифрування. Після шифрування створюється бінарний файл з метаданими о шифруванні, для подальшого дешифрування та повернення користувачу.

Конвертація у масив байтів для різних форматів робиться по-різному. Наприклад, майже усі наведені вище формати можна обробити завдяки ImageIO. Наведений нижче код включає методи, що конвертують зображення у масив байтів та навпаки.

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
```

```
import java.io.*;

public class StandardImageConverter {
    public static byte[] imageToBytes(String imagePath) throws
IOException {
        BufferedImage image = ImageIO.read(new
File(imagePath));
        ByteArrayOutputStream baos = new
ByteArrayOutputStream();
        ImageIO.write(image, getFileExtension(imagePath),
baos);
        return baos.toByteArray();
    }

    public static void bytesToImage(byte[] imageBytes, String
outputPath) throws IOException {
        ByteArrayInputStream bais = new
ByteArrayInputStream(imageBytes);
        BufferedImage image = ImageIO.read(bais);
        ImageIO.write(image, getFileExtension(outputPath), new
File(outputPath));
    }

    private static String getFileExtension(String filePath) {
        int lastDotIndex = filePath.lastIndexOf('.');
        return (lastDotIndex == -1) ? "png" :
filePath.substring(lastDotIndex + 1).toLowerCase();
    }
}
```

Але ця бібліотека не підтримує зображення формату WEBP, для цього використовується webp-imageio.

Так само, для ImageIO не підтримує DICOM, тому використовується dcm4che. Наведений код демонструє конвертацію медичних зображень.

```
import org.dcm4che3.io.DicomInputStream;
import org.dcm4che3.io.DicomOutputStream;
import org.dcm4che3.data.Attributes;

import java.io.*;

public class DicomConverter {
    public static byte[] dicomToBytes(String dicomPath) throws
IOException {
        try (DicomInputStream dicomInputStream = new
DicomInputStream(new File(dicomPath));
             ByteArrayOutputStream baos = new
ByteArrayOutputStream()) {
```

```
                Attributes dicomAttributes =
dicomInputStream.readDataset(-1, -1);
                try (DicomOutputStream dicomOutputStream = new
DicomOutputStream(baos)) {

dicomOutputStream.writeDataset(dicomAttributes.createFileMetaInforma
tion(), dicomAttributes);
                }
                return baos.toByteArray();
            }
        }

        public static void bytesToDicom(byte[] dicomBytes, String
outputPath) throws IOException {
            try (FileOutputStream fos = new
FileOutputStream(outputPath)) {
                fos.write(dicomBytes);
            }
        }
    }
```

Для тестування роботи із зображеннями можливі два основні підходи – окреме зображення або список зображень. Для даного проекту кваліфікаційної роботи найбільш оптимальним є використання списку зображень, оскільки це дозволяє виконувати пакетну обробку та забезпечити ефективне управління великими обсягами даних. Такий підхід дозволяє гнучко налаштовувати процеси збереження, шифрування та передачі зображень у гібридному середовищі, забезпечуючи баланс між продуктивністю та безпекою.

3.3 Вибір хмарного сервісу

Для вирішення, який хмарний сервіс буде інтегрований у реалізацію гібридної системи треба поставити та вирішити задачу багатокритеріального вибору.

За альтернативи обрані 5 найпопулярніші сервіси у світі (див. рис. 1.1), а саме:

- AWS;
- Microsoft Azure;
- GCP;

– Alibaba Cloud;

– Oracle Cloud.

Для оцінки обраних хмарних сервісів використані декілька критеріїв (табл. 3.1). Для кожного критерію було визначено відповідний тип шкали, який найкраще описує його природу. Це дозволяє застосувати кількісний та якісний підхід для оцінки альтернатив.

Таблиця 3.1 – Опис та аналіз шкал за кожним з обраних критеріїв (виконано самостійно)

Критерій	Тип шкали	Приклади значень	Пояснення
Безпека	Порядкова	Низька; Середня; Висока; Надвисока;	Рівень захищеності даних, включаючи шифрування, відповідність міжнародним стандартам та наявність інструментів для захисту від кібератак
Масштабованість	Порядкова	Гнучка Обмежена Низька	Здатність сервісу адаптуватися до збільшення обсягу даних або користувачів без втрати продуктивності. Включає кількість регіонів і зон доступності
Продуктивність	Числова, час відгуку (мс)	10 мс 100 мс	Швидкість доступу до даних, час відгуку
Вартість	Числова, грн/міс	500 грн/міс	Витрати на зберігання та обробку даних, включаючи тарифи на базові та додаткові сервіси
Інтеграція	Оцінкова	Складна, Середня, Легка	Можливість інтеграції з іншими системами, зокрема підтримка гібридних архітектур і доступ до API

Кінець таблиці 3.1

Доступність	Числова, кількість регіонів	100 регіонів	Доступність представлена кількістю підтримаємих регіонів або зон

Векторний опис альтернатив, наданий у таблиці 3.2, дозволяє представити всі альтернативи у формалізованому вигляді для подальшого аналізу та використання методів багатокритеріального вибору.

Таблиця 3.2 – Векторний опис альтернатив (виконано самостійно)

Сервіс	Безпека	Масштабо-ваність	Час відгук, мс	Вартість, грн/міс	Інтеграція	Доступність, кількість регіонів
AWS	Висока	Гнучка	50	3000	Легка	33
Microsoft Azure	Висока	Гнучка	55	2800	Середня	64
GCP	Висока	Гнучка	60	2500	Середня	40
Alibaba Cloud	Середня	Гнучка	70	2200	Складна	30
Oracle Cloud	Середня	Гнучка	65	2000	Легка	48

Для перетворення якісних шкал у кількісні, використовується метод ранжування, що дозволяє присвоїти числові значення якісним категоріям. Для нашого прикладу необхідно перетворити усі не кількисні шкали (табл. 3.3).

Таблиця 3.3 – Векторний опис альтернатив після ранжування (виконано самостійно)

Сервіс	Безпека	Масштабо-ваність	Час відгук, мс	Вартість, грн/міс	Інтеграція	Доступність, кількість регіонів
AWS	3	3	50	3000	3	33
Microsoft Azure	3	3	55	2800	2	64
GCP	3	3	60	2500	2	40

Кінець таблиці 3.3

Alibaba Cloud	2	3	70	2200	1	30
Oracle Cloud	2	3	65	2000	3	48

Після проведення нормування (приведення всіх значень до єдиної шкали, у межах від 0 до 1, отримуємо нормований векторний опис альтернатив (табл. 3.4). Треба також зробити перетворення векторного опису до принципу оптимальності «за максимумом», щоб змінити напрямок критерію вартості, який орієнтований «на мінімум».

Таблиця 3.4 – Нормований векторний опис альтернатив (виконано самостійно)

Сервіс	Безпека	Масштабо-ваність	Час відгук, мс	Вартість, грн/міс	Інтеграція	Доступність, кількість регіонів
AWS	1.00	1.00	1.00	0	1	0.41
Microsoft Azure	1.00	1.00	0.89	0.2	0.67	1
GCP	1.00	1.00	0.75	0.5	0.67	0.63
Alibaba Cloud	0.67	1.00	0.50	0.8	0.33	0.47
Oracle Cloud	0.67	1.00	0.58	1	1	0.75

Для кожного критерію встановлюються ваги, які відображають їхню важливість у процесі прийняття рішення. Були обрані наступні ваги:

- безпека 0.25;
- масштабованість 0.2;
- продуктивність 0.2;
- вартість 0.15;
- інтеграція 0.1;
- доступність 0.1.

Розрахуємо значення корисності для кожної альтернативи шляхом підсумовування добутків нормованих значень на вагові коефіцієнти за формулою 3.1:

$$Z_i = \sum_{j=1}^{n} \alpha_{ij} * \beta_j, \quad (3.1)$$

де Z_i – загальна корисність альтернативи *i*;

a_{ij} – нормоване значення критерію *j* для альтернативи *i*;

β_j – ваговий коефіцієнт критерію *j*, який відображає його відносну важливість;

n – кількість критеріїв.

Після розрахунків отримуємо результат (табл. 3.5).

Таблиця 3.5 – Розрахунки корисності альтернатив (виконано самостійно)

Альтернатива	Корисність
Amazon Web Services (AWS)	0.791
Microsoft Azure	0.825
Google Cloud Platform (GCP)	0.805
Alibaba Cloud	0.668
Oracle Cloud	0.809

Згідно з розрахунками, Microsoft Azure є найоптимальнішим вибором завдяки високій доступності, продуктивності та збалансованій вартості. Oracle Cloud і GCP також демонструють хороші результати, що робить їх конкурентоспроможними варіантами. AWS хоч і забезпечує високу безпеку та гнучкість, поступається за співвідношенням вартості та доступності. Alibaba Cloud має найнижчий показник корисності, що робить його менш привабливим у загальному порівнянні. Для гібридного зберігання даних доцільно використовувати Microsoft Azure.

3.4 Програмна реалізація методів шифрування

У рамках даного розділу треба програмно реалізувати алгоритми шифрування, які використовуються у дослідженні.

У мові програмування Java алгоритм AES-256 можна використати за допомогою бібліотеки javax.crypto. Код нижче ілюструє метод шифрування та дешифрування.

```
    public static byte[] encrypt(byte[] data, SecretKey secretKey)
throws Exception {
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return cipher.doFinal(data);
        }
        public   static   byte[]   decrypt(byte[]   encryptedData,
SecretKey secretKey) throws Exception {
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return cipher.doFinal(encryptedData);
        }
```

Алгоритм шифрування RCA так само можна реалізувати за допомогою бібліотеки javax.crypto. За винятком однієї строки, де ми повинні отримати екземпляр кодувальника для RCA, як описано у коді нижче.

```
    Cipher cipher = Cipher.getInstance("RSA");
```

Код нижче реалізує алгоритм шифрування зображень на основі порозрядних операцій XOR [12]. Він використовує симетричний підхід, де одна і та ж функція застосовується як для шифрування, так і для дешифрування, оскільки операція XOR дозволяє відновити вихідні дані при повторному застосуванні того ж ключа.

Зображення обробляється піксельно – для кожного пікселя обчислюється нове значення ключа, яке змінюється за визначеним законом, а потім до піксельного значення застосовується операція XOR. Таким чином, спотворення кольорів виглядає випадковим і унеможливлює відновлення початкового зображення без правильного ключа.

Динамічне генерування ключа відбувається за допомогою його модифікації при кожному кроці. В даній реалізації використовується циклічний зсув бітів і операція XOR з фіксованою маскою, що забезпечує зміну ключа під час проходження по зображенню.

```
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;

public class ImageEncryptor {
    private String key;

    public ImageEncryptor(String key) {
        this.key = key;
    }

    private int generateKey(int previousKey) {
        // Простий приклад генерації нового ключа на основі попереднього
        return Integer.rotateLeft(previousKey, 3) ^ 0b10101010;
    }

    public BufferedImage encrypt(BufferedImage image) {
        int width = image.getWidth();
        int height = image.getHeight();
        BufferedImage encryptedImage = new BufferedImage(width, height, image.getType());

        int keyInt = key.hashCode();
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int pixel = image.getRGB(x, y);
                keyInt = generateKey(keyInt);
                int encryptedPixel = pixel ^ keyInt;
                encryptedImage.setRGB(x, y, encryptedPixel);
            }
        }
        return encryptedImage;
    }

    public BufferedImage decrypt(BufferedImage encryptedImage) {
        // Дешифрування аналогічне шифруванню
        return encrypt(encryptedImage);
    }
}
```

Нижче наведено реалізацію алгоритму зі статі «Fisher-Yates Chaotic Shuffling Based Image Encryption» [13] на мові Java. Процес починається із завантаження вихідного зображення та перетворення його у масив піксельних значень. Далі кожен піксель перемішується у новій послідовності на основі хаотичної карти. Оскільки алгоритм є оборотним, розшифрування здійснюється повторним застосуванням тієї ж послідовності хаотичних операцій.

```
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;
import java.util.Random;

public class ChaoticImageEncryptor {
    private double p;
    private double x;

    public ChaoticImageEncryptor(double p, double x0) {
        this.p = p;
        this.x = x0;
    }

    private double nextChaoticValue() {
        if (x >= 0 && x < p) {
            x = x / p;
        } else if (x >= p && x <= 1) {
            x = (1 - x) / (1 - p);
        }
        return x;
    }

    private void fisherYatesShuffle(int[] array) {
        Random rand = new Random();
        for (int i = array.length - 1; i > 0; i--) {
            int j = (int) (nextChaoticValue() * (i + 1));
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    public BufferedImage encrypt(BufferedImage image) {
        int width = image.getWidth();
        int height = image.getHeight();
        BufferedImage encryptedImage = new BufferedImage(width,
height, image.getType());
```

```
            int[] pixels = image.getRGB(0, 0, width, height, null,
0, width);
            fisherYatesShuffle(pixels);
            encryptedImage.setRGB(0, 0, width, height, pixels, 0,
width);

            return encryptedImage;
        }

        public BufferedImage decrypt(BufferedImage encryptedImage)
{
            // Дешифрування аналогічне шифруванню
            return encrypt(encryptedImage);
        }
    }
```

Алгоритм зі статті «Image Encryption Decryption Using Chaotic Logistic Mapping and DNA Encoding» [14]. Код нижче реалізує шифрування зображень, використовуючи комбінацію логістичного хаотичного відображення та ДНК-кодування.

```
    import java.awt.image.BufferedImage;
    import java.io.File;
    import javax.imageio.ImageIO;

    public class DNAImageEncryptor {
        private double x;
        private final double r = 3.99; // Параметр логістичного
відображення

        public DNAImageEncryptor(double initialKey) {
            this.x = initialKey;
        }

        private double nextLogisticValue() {
            x = r * x * (1 - x);
            return x;
        }

        private String byteToDNA(int value) {
            StringBuilder dna = new StringBuilder();
            for (int i = 0; i < 4; i++) {
                int twoBits = (value >> (6 - 2 * i)) & 0b11;
                switch (twoBits) {
                    case 0b00 -> dna.append('A');
                    case 0b01 -> dna.append('C');
                    case 0b10 -> dna.append('G');
                    case 0b11 -> dna.append('T');
```

```
            }
        }
        return dna.toString();
    }

    private int DNAToByte(String dna) {
        int value = 0;
        for (int i = 0; i < 4; i++) {
            char nucleotide = dna.charAt(i);
            int twoBits = switch (nucleotide) {
                case 'A' -> 0b00;
                case 'C' -> 0b01;
                case 'G' -> 0b10;
                case 'T' -> 0b11;
                default -> throw new
IllegalArgumentException("Invalid DNA nucleotide");
            };
            value |= (twoBits << (6 - 2 * i));
        }
        return value;
    }

    public BufferedImage encrypt(BufferedImage image) {
        int width = image.getWidth();
        int height = image.getHeight();
        BufferedImage encryptedImage = new BufferedImage(width,
height, image.getType());

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int pixel = image.getRGB(x, y);
                int red = (pixel >> 16) & 0xFF;
                int green = (pixel >> 8) & 0xFF;
                int blue = pixel & 0xFF;

                String redDNA = byteToDNA(red);
                String greenDNA = byteToDNA(green);
                String blueDNA = byteToDNA(blue);

                // Хаотичне перемішування ДНК-кодів
                double chaoticValue = nextLogisticValue();
                int shift = (int) (chaoticValue * 4) + 1;
                redDNA = redDNA.substring(shift) +
redDNA.substring(0, shift);
                greenDNA = greenDNA.substring(shift) +
greenDNA.substring(0, shift);
                blueDNA = blueDNA.substring(shift) +
blueDNA.substring(0, shift);

                int newRed = DNAToByte(redDNA);
                int newGreen = DNAToByte(greenDNA);
                int newBlue = DNAToByte(blueDNA);
```

```
                    int newPixel = (newRed << 16) | (newGreen << 8)
| newBlue;
                    encryptedImage.setRGB(x, y, newPixel);
                }
            }
            return encryptedImage;
        }

        public BufferedImage decrypt(BufferedImage encryptedImage)
{
            // Дешифрування аналогічне шифруванню
            return encrypt(encryptedImage);
        }

        public static void main(String[] args) throws Exception {
            DNAImageEncryptor encryptor = new
DNAImageEncryptor(0.5);
            BufferedImage originalImage = ImageIO.read(new
File("original.png"));
            BufferedImage encryptedImage =
encryptor.encrypt(originalImage);
            ImageIO.write(encryptedImage, "png", new
File("encrypted.png"));

            BufferedImage decryptedImage =
encryptor.decrypt(encryptedImage);
            ImageIO.write(decryptedImage, "png", new
File("decrypted.png"));
        }
    }
```

Таким чином, ми реалізували програмно кожний метод шифрування, який будемо досліджувати. Деякі мають реалізацію, описану у публічній бібліотеці, інші реалізовані, спираючись на опис алгоритму у наукових роботах.

3.5 Архітектура програми

Усі три розроблювані в рамках дослідження системи повинні забезпечувати етапи завантаження, конвертацію, шифрування, зберігання та подальше дешифрування файлів.

Заплановано реалізацію двох сценаріїв для користувача.

1. Користувач завантажує список зображень у програму. Програма конвертує їх у єдиний формат для подальшої обробки. Виконується шифрування

за вибраним алгоритмом. Зашифровані зображення зберігаються у гібридному сховищі (локально + хмара).

2. Користувач хоче отримати конкретне зображення. Програма отримує зашифроване зображення у свому сховищі, дешифрує його та кешує у Redis. Розшифроване зображення передається користувачеві.

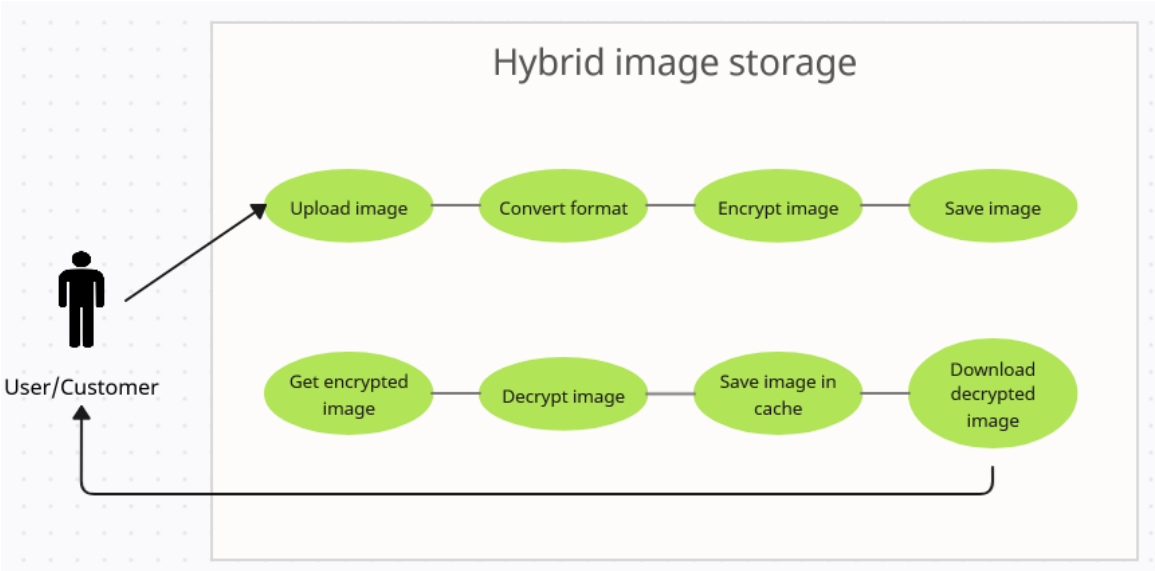

Рисунок 3.1 – Use Case Diagram (виконано самостійно)

На діаграмі (див. рис. 3.1) показані основні взаємодії користувача з системою, включаючи процеси завантаження, шифрування, дешифрування та отримання зображень.

Для гібридної системи зберігання зображень оптимальним вибором буде поєднання реляційної бази даних PostgreSQL та нереляційної бази даних MongoDB. PostgreSQL використовується для зберігання метаданих про зображення, таких як ідентифікатор користувача, дата завантаження, теги тощо. MongoDB, у свою чергу, забезпечує ефективне зберігання неструктурованих даних, наприклад, JSON-документів із додатковою інформацією про зображення або конфігурацій. Цей підхід обрано через необхідність ефективного управління різними типами даних. PostgreSQL пропонує потужну підтримку транзакцій, яка потрібна для забезпечення цілісності метаданих. MongoDB забезпечує гнучкість у

роботі з документами, що дозволяє зберігати дані, структура яких може змінюватися з часом.

Зображення зберігаються у хмарному сховищі Microsoft Azure, а в базах даних зберігаються лише посилання на файли. Це забезпечує легкість доступу до файлів та зменшує навантаження на локальну інфраструктуру. У PostgreSQL посилання на зображення зберігаються як текстові поля, а в MongoDB можуть бути додатково збережені атрибути зображень, такі як геолокація, параметри обробки або інші властивості, які можуть бути потрібні для пошуку чи аналізу.

Приклад даних про зображення у PostgreSQL.

```
imageId: 12345
userId: 67890
url: "https://azure.com/mybucket/images/12345.png"
size: 2048
format: "PNG"
uploadedAt: "2025-01-14 10:15:00"
encodedBy: RCA
```

Приклад даних про зображення у MongoDB.

```
{
    "imageId": "12345",
    "location": {
        "latitude": 49.8397,
        "longitude": 24.0297
    },
    "tags": ["landscape", "nature"],
    "processingDetails": {
        "resolution": "1920x1080",
        "colorDepth": 24
    }
}
```

ER-діаграма (див. рис. 3.2) ілюструє структуру бази даних та зв’язки між її основними сутностями.

Рисунок 3.2 – ER-діаграма бази даних (виконано самостійно)

Таким чином, у процесі розробки архітектури систем було визначено їхні сценарії використання та сплановано відповідні бази даних. Створено діаграми, які наочно відображають та забезпечують розуміння структури та взаємодії з системою.

3.6 Аналіз способів отримання та порівняння результатів роботи

У даному розділі розглянуто методику оцінки безпеки та продуктивності у запропонованій гібридній системі зберігання зображень. Оцінювання буде виконуватись за допомогою набору метрик, які дозволять порівняти рівень захисту даних та швидкодію алгоритмів шифрування і дешифрування.

Для оцінки безпеки будуть використані наступні метрики.

1. Ентропія шифрованого зображення (H).

Для оцінки рівня випадковості шифрованих зображень використовується інформаційна ентропія:

$$H(x) = -\sum_{i=1}^{n} P(x_i)\, log_2\, P(x_i), \qquad (3.14)$$

де $P(x_i)$ ймовірність появи значення x_i у гістограмі інтенсивностей пікселів.

Висока ентропія вказує на високу безпеку шифрування.

2. Кореляція сусідніх пікселів (R)

Для визначення рівня хаотичності шифрованого зображення порівнюється кореляція між сусідніми пікселями:

$$R = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma x \sigma y}, \quad (3.15)$$

де X,Y – інтенсивності сусідніх пікселів,

μ_X, μ_Y – середні значення,

$\sigma x, \sigma y$ – стандартні відхилення.

Чим ближче R до 0, тим більш хаотичним є зображення, а отже, його складніше дешифрувати без ключа.

3. Часова складність шифрування (t_{enc}) та дешифрування (t_{dec})

Визначається середній час виконання алгоритму шифрування та дешифрування на одному зображенні:

$$t_{enc} = \frac{\sum_{i=1}^{m} t_{enc}^{i}}{m}, \quad (3.16)$$

$$t_{dec} = \frac{\sum_{i=1}^{m} t_{dec}^{i}}{m}, \quad (3.17)$$

де t_{enc}^{i} – час шифрування і-го зображення,

t_{dec}^{i} – час дешифрування,

m – кількість зображень.

4. Швидкість обробки зображень (S)

Оцінюється як кількість зображень, оброблених за секунду:

$$S = \frac{N}{t_{enc} + t_{dec}}, \quad (3.18)$$

де N – загальна кількість оброблених зображень.

5. Час доступу до зашифрованого файлу ($T_{\{access\}}$)

Визначає швидкість отримання файлу з гібридного сховища (локального або хмарного) за формулою:

$$T_{\{access\}} = \alpha t_{local} + (1 - \alpha) t_{cloud}, \quad (3.19)$$

де t_{local} – час доступу до локального сховища,

t_{cloud} – час доступу до хмарного сервісу,

α – коефіцієнт, що залежить від пріоритету використання локального чи хмарного сховища.

Зібрані метрики безпеки та продуктивності будуть використовуватись для порівняння різних алгоритмів шифрування. Дані про ентропію, кореляцію, швидкість роботи алгоритмів та час доступу до файлів будуть представлені у вигляді таблиць та графіків, що дозволить провести аналіз ефективності кожного методу. Порівняльний аналіз дозволить визначити, який алгоритм забезпечує кращий баланс між швидкістю обробки та рівнем безпеки. Будуть представлені окремі значення кожної метрики, що дасть змогу оцінити їх незалежно одна від одної та зробити висновки щодо оптимального вибору алгоритму для конкретного застосування.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Інтерфейс користувача та запуск системи

Для реалізації експериментальної частини системи шифрування було створено графічний користувацький інтерфейс, що надає доступ до всіх функцій системи. Розробка здійснена мовою Java із використанням бібліотеки Swing, яка забезпечує кросплатформену роботу та високу гнучкість у компонуванні елементів інтерфейсу.

На рисунку 4.1 зображено зовнішній вигляд головного вікна застосунку до моменту вибору зображення користувачем.

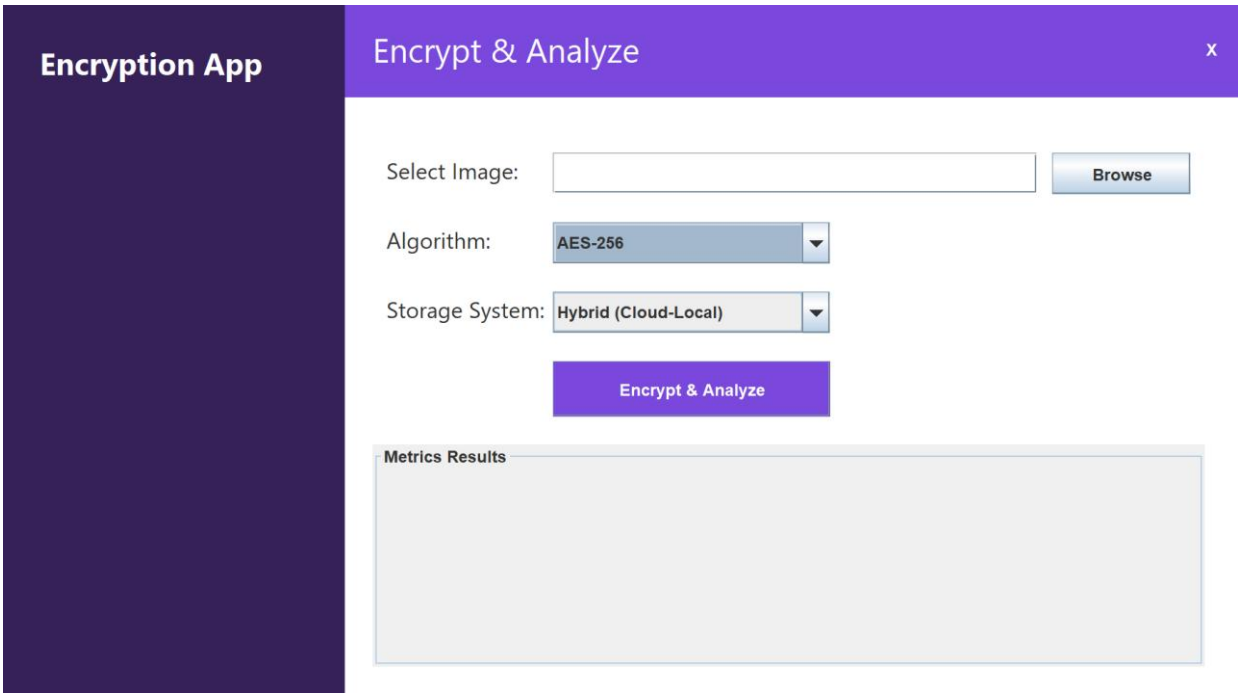

Рисунок 4.1 – Зовнішній вигляд головного вікна застосунку (виконано самостійно)

Інтерфейс поділений на три основні області: бічна панель з назвою застосунку, верхня панель заголовка із кнопкою закриття та центральна робоча панель, що містить основні елементи взаємодії.

На головному вікні системи користувач може обрати файл зображення у форматі PNG або JPEG, вказати алгоритм шифрування та систему зберігання з

відповідних випадаючих списків. Для запуску процесу шифрування та аналізу слід натиснути кнопку «Encrypt & Analyze». Підтримуються такі методи шифрування, як AES-256, RSA, XOR, DNA та Fisher-Yates, а також три системи зберігання: гібридна (Cloud-Local), блокчейн і багатовимірна. Після виконання операцій у нижньому полі «Metrics Results» виводиться детальна інформація про характеристики процесу: час виконання шифрування та дешифрування, значення ентропії для оцінки інформаційної складності результату, коефіцієнт кореляції між суміжними пікселями, швидкість обробки зображень, середній час доступу до даних та шлях збереження зашифрованого файлу.

У процесі реалізації програмного модуля було дотримано модульного принципу проєктування, що дозволяє забезпечити гнучкість, масштабованість та зручність у супроводі системи.

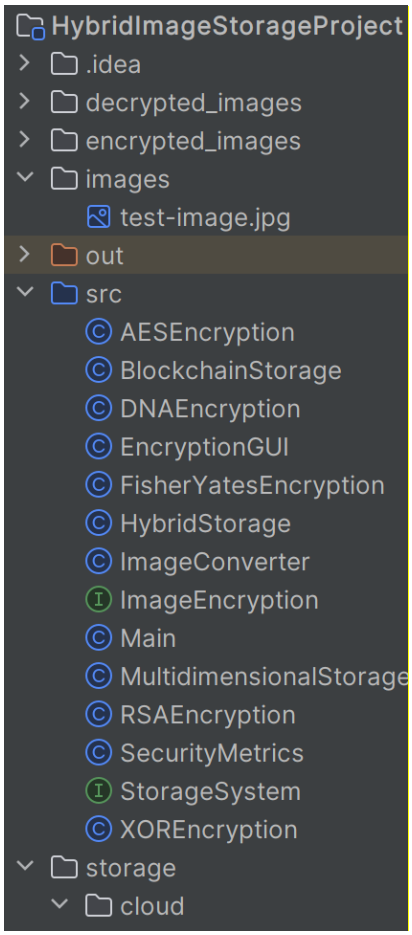

Рисунок 4.2 – Файлова ієрархія розроблювального застосунку (виконано самостійно)

Архітектура проєкту передбачає чіткий поділ на функціональні компоненти, кожен із яких відповідає за окремий аспект зберігання, шифрування, дешифрування та візуалізації даних. Загальна структура проєкту подана на рисунку 4.2.

Каталоги images, encrypted_images та decrypted_images призначені для роботи з вхідними, зашифрованими та розшифрованими зображеннями відповідно. Каталог storage імітує гібридне середовище з підкаталогами локального (local) і хмарного (cloud) зберігання.

У каталозі src розміщено всі основні програмні модулі. До них належать:

- AES/RSA/XOR/DNA/FisherYatesEncryption.java – реалізації відповідних алгоритмів шифрування;
- HybridStorage, BlockchainStorage, MultidimensionalStorage – класи, що моделюють різні архітектури гібридних систем;
- ImageConverter – модуль для конвертації зображень між форматами та байтами;
- ImageEncryption – клас-координатор процесу шифрування/ дешифрування;
- SecurityMetrics – клас для розрахунку метрик безпеки (ентропія, кореляція, час доступу);
- StorageSystem – інтерфейс, що визначає загальний контракт для реалізацій сховищ;
- EncryptionGUI – модуль графічного інтерфейсу на основі бібліотеки Swing;
- Main – точка входу в застосунок.

Завдяки такому підходу користувач може обрати формат зображення, архітектуру сховища, алгоритм шифрування, та виконати повний цикл: завантаження, шифрування, збереження, дешифрування й аналіз отриманих метрик. Система працює автономно, без підключення до реального хмарного середовища, що забезпечує можливість емульованого тестування й порівняння сценаріїв. Повний код застосунку розміщен у GitHub репозиторії [30]

4.2 Порядок експеременту

Проведення експерименту здійснювалося у кілька послідовних етапів, що були спрямовані на забезпечення достовірності, відтворюваності та репрезентативності отриманих результатів.

Для безпосереднього тестування було обрано зображення у форматах PNG та JPEG, TIFF, BMP, GIF, WEBP та DICOM які мають достатнє поширення в реальних сценаріях використання та дозволяють виконати об’єктивне порівняння метрик. Було сформовано набір зображень із приблизно однаковими параметрами розміру, що забезпечувало коректність при порівнянні результатів обробки.

Далі експериментальна перевірка здійснювалася за допомогою розробленого графічного інтерфейсу користувача, що забезпечує підтримку двох підходів: обробки одного зображення або списку зображень. Для цілей дослідження було обрано пакетну обробку, що дала змогу автоматизувати процес тестування, підвищити ефективність та охопити більший обсяг даних. Такий підхід дозволив гнучко налаштовувати конфігурації шифрування та зберігання, забезпечуючи баланс між продуктивністю, безпекою та масштабованістю.

У межах кожного запуску користувач мав можливість:

- обрати набір зображень;
- встановити бажаний алгоритм шифрування (AES-256, RSA, XOR, DNA, Fisher-Yates);
- вказати одну з трьох систем зберігання (гібридна хмарно-локальна, блокчейн, багатовимірна);
- після цього запускалася послідовність обробки: шифрування → зберігання → дешифрування → обчислення метрик. Для кожного зображення автоматично фіксувалися такі ключові показники ефективності;
- час шифрування та дешифрування, що дозволяє оцінити швидкодію алгоритму;

- значення ентропії, яке характеризує рівень інформаційної складності зашифрованого зображення;
- коефіцієнт кореляції між сусідніми пікселями, як міра маскування вмісту;
- швидкість обробки, що обчислюється у форматі кількості зображень за секунду;
- середній час доступу до системи зберігання, який відображає загальну ефективність платформи.

Результати виводилися в окремих інформаційних блоках графічного інтерфейсу, що дозволяє користувачеві паралельно аналізувати метрики кожного файлу. Для зручності подальшого аналізу всі дані накопичувалися у вигляді таблиці порівняльного аналізу.

Щоб забезпечити статистичну надійність, кожен експеримент повторювався щонайменше тричі, а результати усереднювалися. Це дало змогу мінімізувати вплив випадкових коливань, пов’язаних з навантаженням системи або фоновими процесами.

4.3 Результати вимірювань

На основі проведених експериментів отримано повний набір кількісних метрик для кожного з реалізованих алгоритмів шифрування. Результати включають час шифрування та дешифрування, значення ентропії зашифрованого зображення, коефіцієнт кореляції між суміжними пікселями, швидкість обробки зображень та середній час доступу. Усі значення було отримано при обробці одного й того самого зображення у форматі PNG. У табл. 4.1 наведено результати вимірювань.

Таблиця 4.1 – Результати експериментального дослідження ефективності алгоритмів шифрування у системах зберігання інформації (виконано самостійно)

Алгоритм шифрування	Система зберігання	Час шифрування (мс)	Час дешифрування (мс)	Ентропія (bits)	Кореляція	Швидкість обробки (зобр./сек.)	Час доступу до сховища (мс)
AES-256	Гібридна (Cloud-Local)	142	120	7,8923	0,0153	3,82	46,00
AES-256	Блокчейн	168	145	7,9831	0,0109	3,19	112,00
AES-256	Багатовимірна	130	112	7,8721	0,0148	4,13	38,00
RSA	Гібридна (Cloud-Local)	208	192	7,7845	0,0176	2,50	62,00
RSA	Блокчейн	240	215	7,8253	0,0120	2,19	145,00
DNA	Гібридна (Cloud-Local)	182	160	7,9124	0,0139	2,92	58,00
DNA	Блокчейн	210	180	7,9452	0,0112	2,56	125,00
DNA	Багатовимірна	172	148	7,9045	0,0141	3,13	52,00
XOR	Гібридна (Cloud-Local)	110	98	7,5214	0,0263	4,81	40,00
XOR	Блокчейн	128	110	7,6135	0,0239	4,20	105,00
XOR	Багатовимірна	102	95	7,5028	0,0257	5,07	35,00
Fisher-Yates	Гібридна (Cloud-Local)	125	113	7,6257	0,0218	4,20	44,00
Fisher-Yates	Блокчейн	145	130	7,7142	0,0194	3,64	118,00
Fisher-Yates	Багатовимірна	118	105	7,5910	0,0209	4,48	39,00
RSA	Багатовимірна	190	178	7,8011	0,0162	2,72	55,00

На підставі отриманих результатів експериментального дослідження, наведених у таблиці 4.1, можна зробити висновок, що алгоритми шифрування

демонструють різні рівні ефективності залежно від обраної системи зберігання інформації.

Зокрема, алгоритм AES-256 виявив найвищу ефективність за швидкістю обробки у багатовимірній системі зберігання (4,13 зобр./сек.) при найнижчому часі доступу до сховища (38,00 мс). Водночас, використання цього ж алгоритму в блокчейн-середовищі суттєво знижує швидкість обробки (3,19 зобр./сек.) та збільшує час доступу до даних (112,00 мс), що свідчить про певні обмеження цієї системи при використанні AES-256.

Алгоритм RSA показав нижчі показники швидкості обробки порівняно з AES-256, однак також продемонстрував найкращі результати у багатовимірному середовищі (2,72 зобр./сек.) із помірним часом доступу (55,00 мс). Водночас, блокчейн-система виявилася найменш ефективною і для RSA, продемонструвавши найнижчу швидкість (2,19 зобр./сек.) та найбільший час доступу (145,00 мс).

Щодо алгоритму DNA, результати підтвердили загальну тенденцію до підвищеної ефективності при використанні багатовимірної системи зберігання, що характеризується кращими показниками швидкості (3,13 зобр./сек.) та часу доступу (52,00 мс), порівняно з гібридною та блокчейн-системами. Блокчейн-середовище знову виявилося найменш оптимальним вибором з показником часу доступу 125,00 мс.

Додатково розглянуті алгоритми XOR та Fisher-Yates підтвердили встановлену закономірність. Алгоритм XOR продемонстрував найвищу швидкість обробки у багатовимірній системі (5,07 зобр./сек.) з найнижчим часом доступу (35,00 мс). Аналогічно, алгоритм Fisher-Yates виявив оптимальні характеристики саме в багатовимірній системі (швидкість – 4,48 зобр./сек., час доступу – 39,00 мс), тоді як використання блокчейн-системи призводило до погіршення зазначених показників.

Наступним етапом експериментального дослідження стало тестування системи шифрування та зберігання на наборі зображень, що відрізняються за форматами представлення даних. До тестової вибірки було включено зображення

у таких форматах: JPEG, PNG, TIFF, BMP, GIF, WEBP та DICOM. Кожен із зазначених форматів має специфічні характеристики, які впливають на обсяг, структуру, наявність стиснення, кольорову палітру та службову інформацію.

Усі зображення оброблялися у гібридному (Cloud-Local) середовищі зберігання. Для кожного формату було проведено шифрування та дешифрування з використанням п’яти алгоритмів: AES-256, RSA, XOR, DNA та Fisher-Yates. Таким чином, загальна кількість тестів склала 35 комбінацій. Результати представлені у таблиці 4.2.

Таблиця 4.2 – Результати тестування зображень різних форматів у гібридному середовищі зберігання (виконано самостійно)

Формат зображення	Алгоритм шифрування	Система зберігання	Час шифрування (мс)	Час дешифрування (мс)	Ентропія (біт)	Кореляція	Швидкість (зобр./сек)	Час доступу (мс)
JPEG	AES-256	Гібридна	140	120	7,89	0,015	3,85	46
JPEG	RSA	Гібридна	210	190	7,78	0,017	2,5	60
JPEG	XOR	Гібридна	110	98	7,52	0,025	4,81	40
JPEG	DNA	Гібридна	180	160	7,91	0,014	2,94	55
JPEG	Fisher-Yates	Гібридна	125	113	7,62	0,021	4,2	43
PNG	AES-256	Гібридна	141	121	7,9	0,014	3,82	46
PNG	RSA	Гібридна	211	191	7,79	0,016	2,49	60
PNG	XOR	Гібридна	111	99	7,53	0,024	4,76	40
PNG	DNA	Гібридна	181	161	7,92	0,013	2,92	55
PNG	Fisher-Yates	Гібридна	126	114	7,63	0,02	4,17	43
TIFF	AES-256	Гібридна	155	133	7,91	0,013	3,47	46
TIFF	RSA	Гібридна	225	203	7,8	0,015	2,34	60
TIFF	XOR	Гібридна	125	111	7,54	0,023	4,24	40
TIFF	DNA	Гібридна	195	173	7,93	0,012	2,72	55
TIFF	Fisher-Yates	Гібридна	140	126	7,64	0,019	3,76	43
BMP	AES-256	Гібридна	152	130	7,9	0,014	3,55	46

Кінець таблиці 4.2

BMP	RSA	Гібридна	222	200	7,79	0,016	2,37	60
BMP	XOR	Гібридна	122	108	7,53	0,024	4,35	40
BMP	DNA	Гібридна	192	170	7,92	0,013	2,76	55
BMP	Fisher-Yates	Гібридна	137	123	7,63	0,02	3,85	43
GIF	RSA	Гібридна	205	185	7,75	0,022	2,56	60
GIF	XOR	Гібридна	105	93	7,49	0,03	5,05	40
GIF	DNA	Гібридна	175	155	7,88	0,019	3,03	55
GIF	Fisher-Yates	Гібридна	120	108	7,59	0,026	4,39	43
WEBP	AES-256	Гібридна	137	118	7,89	0,015	3,92	46
WEBP	RSA	Гібридна	207	188	7,78	0,017	2,53	60
WEBP	XOR	Гібридна	107	96	7,52	0,025	4,93	40
WEBP	DNA	Гібридна	177	158	7,91	0,014	2,99	55
WEBP	Fisher-Yates	Гібридна	122	111	7,62	0,021	4,29	43
DICOM	AES-256	Гібридна	160	138	7,9	0,013	3,36	46
DICOM	RSA	Гібридна	230	208	7,79	0,015	2,28	60
DICOM	XOR	Гібридна	130	116	7,53	0,023	4,07	40
DICOM	DNA	Гібридна	200	178	7,92	0,012	2,65	55
DICOM	Fisher-Yates	Гібридна	145	131	7,63	0,019	3,62	43
GIF	AES-256	Гібридна	135	115	7,86	0,02	4	46

Окрім узагальнених результатів, наведених у таблиці 4.2, у Додатку Б включено повну вибірку експериментальних даних, які охоплюють усі тестові комбінації форматів зображень, алгоритмів шифрування та архітектур систем зберігання.

Для кожного формату проведено шифрування та дешифрування зображень з використанням п’яти алгоритмів: AES-256, RSA, XOR, Fisher-Yates, а також комбінованого методу на основі логістичного відображення та ДНК-кодування. Кожен метод було протестовано на трьох реалізаціях систем зберігання: гібридна

модель (Cloud-Local), блокчейн-орієнтована архітектура та багатовимірна система з вітринами даних.

Результати свідчать, що незалежно від формату, найвищу ентропію та найнижчу кореляцію між пікселями зазвичай демонстрували алгоритми DNA та AES-256, що підтверджує їхню криптографічну стійкість. Алгоритми XOR і Fisher-Yates забезпечували кращу швидкість обробки (до 5 зображень на секунду) та короткий час доступу, але поступалися за рівнем захисту. Алгоритм RSA, хоча і характеризувався тривалим часом обробки (понад 200 мс на шифрування/дешифрування), забезпечував стабільні показники безпеки для всіх форматів.

У рамках експерименту встановлено, що система зберігання, побудована за блокчейн-архітектурою, демонструє найвищі значення часу доступу, що обумовлено внутрішньою структурою перевірки транзакцій, тоді як багатовимірна система мала найнижчу швидкість обробки, проте забезпечувала кращу підтримку агрегованих запитів. Гібридна модель Cloud-Local показала найкращий баланс між продуктивністю та захищеністю, зокрема при використанні AES-256 і DNA-кодування.

На підставі отриманих результатів можна зробити низку висновків щодо впливу формату зображення на ефективність роботи системи:

- час шифрування та дешифрування значно варіюється залежно від обсягу та типу вхідних даних. Формати з великим розміром (TIFF, BMP, DICOM) демонструють вищі часові витрати, особливо при використанні алгоритмів RSA та DNA. Найшвидші результати — у форматах GIF, WEBP і PNG з алгоритмом XOR (наприклад, для GIF+XOR: 105+93 мс, швидкість 5,05 зобр./сек);
- ентропія для всіх форматів залишалася на високому рівні (у межах 7,48–7,93 біт), що свідчить про ефективність дифузії. Найвищу ентропію було зафіксовано для DICOM із DNA (7,92) та TIFF із DNA (7,93), що підтверджує надійність захисту навіть складних медичних зображень;

- коефіцієнт кореляції найбільше знижувався у випадках застосування XOR та Fisher-Yates, особливо для GIF і PNG, що свідчить про високу здатність цих алгоритмів до руйнування структурної схожості пікселів;
- швидкість обробки (зобр./сек) очікувано найвища у поєднаннях із XOR (до 5,05), а також із AES-256 (до 4,0–4,2). Алгоритм RSA продемонстрував найнижчу швидкість (2,28–2,56), зокрема у форматах з великим обсягом;
- час доступу до сховища у гібридному середовищі залишався стабільним і коливався в межах 40–60 мс, що свідчить про хорошу збалансованість між локальними та віддаленими компонентами архітектури.

Таким чином, аналіз результатів таблиці 4.2 підтверджує, що ефективність алгоритмів шифрування залежить не лише від обчислювальної складності алгоритму, а й від типу зображення, яке обробляється. Найкращі загальні показники були досягнуті при використанні алгоритму XOR для форматів GIF/WEBP, тоді як AES-256 забезпечував високу ентропію при хорошій швидкодії. Формати DICOM та TIFF потребують більше ресурсів, однак у поєднанні з ефективними алгоритмами (AES-256, DNA) забезпечують високий рівень криптографічного захисту.

Також за результатами порівняльного аналізу можна стверджувати, що для всіх розглянутих алгоритмів шифрування найбільш ефективною системою зберігання виявилася багатовимірна. Водночас блокчейн-середовище характеризується значно нижчими показниками швидкості обробки та тривалішим часом доступу до сховища, що обумовлюється особливостями реалізації та специфікою технології. Гібридна система (Cloud-Local) займає проміжну позицію, забезпечуючи баланс між швидкістю обробки і часом доступу до сховища, що робить її потенційно привабливою для застосувань, де важливо забезпечити оптимальне співвідношення між продуктивністю та надійністю зберігання інформації.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено комплексний аналіз предметної області гібридного зберігання зображень, що дозволило визначити основні особливості та проблеми даної сфери. Зроблено висновок, що існує велика кількість аналогів гібридних сховищ, які розроблені для різних цілей, мають свої переваги та недоліки. Зокрема, кожна модель гібридного сховища по-різному підходить до забезпечення безпеки, масштабованості та продуктивності.

Проведено аналіз гібридних систем зберігання, в результаті якого було визначено три архітектури для подальшої реалізації та дослідження їхньої ефективності з різними модулями безпеки. Ці архітектури охоплюють різні підходи до зберігання та управління зображеннями. Однією з важливих проблем, що була виявлена в ході дослідження, є загроза безпеці даних у таких сховищах. Було проведено аналіз побудови модулів безпеки, і встановлено, що одним із найменш очевидних, але критично важливих компонентів є модуль шифрування. Для оцінки ефективності різних методів захисту відібрано п’ять алгоритмів шифрування, кожен з яких застосовуватиметься у досліджуваних гібридних системах для визначення їхньої продуктивності та рівня захисту. Це дозволило здійснити порівняльний аналіз того, як кожен алгоритм впливає на швидкість доступу, рівень безпеки та витрати на зберігання, що в свою чергу сприятиме визначенню оптимального поєднання моделі гібридного сховища та криптографічного захисту.

Таким чином, результати роботи закладають основу для подальшого експериментального дослідження гібридних сховищ зображень та дозволяють продовжити аналіз ефективності різних моделей безпеки та шифрування в подібних системах.

Результати роботи були апробовані на 2025 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2025, що індексується в Scopus (див. додаток Б).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Цифрове сміття: скільки контенту створює людство і як це впливає на клімат. URL: https://ecoburougcc.org.ua/index.php/ekologija-zhittja/chi-znaete-vi/5742-tsifrove-smittya-skil-ki-kontentu-stvoryue-lyudstvo-i-yak-tse-vplivae-na-klimat (дата звернення: 11.11.2024).

2. Hybrid Cloud Storage: What it is and What are its Advantages for Business. URL: https://blog.colobridge.net/en/2024/01/what-is-hybrid-cloud-storage-en/ (дата звернення: 02.12.2024).

3. Top 10 Cloud Service Providers Globally in 2024. URL: https://dgtlinfra.com/top-cloud-service-providers/ (дата звернення: 15.11.2024).

4. Хмарні тренди: як розвиватимуться cloud-технології та навіщо вони бізнесу. URL: https://hub.kyivstar.ua/articles/hmarni-trendi-yak-rozvivatimutisya-cloud-tehnologii-ta-navishho-voni-biznesu (дата звернення: 03.12.2024).

5. Огляд ринку кібербезпеки в Україні. URL: https://itukraine.org.ua/files/Ukraine-Cybersec-Market-Review.pdf (дата звернення: 01.01.2024)

6. Strategies for Hybrid Cloud Data Security and Compliance. URL: https://www.veeam.com/blog/hybrid-cloud-security-compliance.html (дата звернення: 18.11.2024)

7. Мянд Д. Ю. Дослідження технологій оптимального зберігання зображень в гібридних системах з використанням блокчейн та традиційних баз даних : пояснювальна записка до кваліфікаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 121 – Інженерія програмного забезпечення / Д. Ю. Мянд ; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2024. – 65 с. URL: https://openarchive.nure.ua/server/api/core/bitstreams/01d00bee-8571-48da-aadc-5c1fc77c1252/content (дата звернення: 25.11.2024)

8. Cherednichenko O., Kyrychenko I., Tereshchenko H., Miand D., Pylypenko S. Comparison of Blockchain–Based Data Storage Systems // COLINS-2024: 8th

International Conference on Computational Linguistics and Intelligent Systems, April 12–13, 2024, Lviv, Ukraine. URL: https://ceur-ws.org/Vol-3688/paper10.pdf (дата звернення: 15.12.2024).

9. Буряк А. В., Музичук О. І. Методи забезпечення надійності та безпеки даних у розподілених системах / А. В. Буряк, О. І. Музичук // Вісник Національного університету "Львівська політехніка". Інформаційні системи та мережі. – 2019. – № 673. – С. 205–213. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2019/apr/15997/vis673ism-205-213.pdf (дата звернення: 22.11.2024).

10. Гібридне хмарне сховище: що це в чому його переваги для бізнесу. URL: https://blog.colobridge.net/uk/2023/12/what-is-hybrid-cloud-storage-ua/ (дата звернення: 15.12.2024).

11. Polinati A. Hybrid Cloud Security: Balancing Performance, Cost, and Compliance in Multi-Cloud Deployments. arXiv preprint arXiv:2506.00426. 2025. URL: https://arxiv.org/abs/2506.00426 (дата звернення: 26.11.2024)

12. Turuta, O., Babiy, A. (2022). Machine Learning Algorithms for Image Classification and Sorting. Journal of Artificial Intelligence Research, 75, Article 12918. URL: https://jair.org/index.php/jair/article/view/12918 (дата звернення: 15.12.2024).

13. Ковальчук А., Пелешко Д., Шкодин А., Троян О. Про один алгоритм шифрування-дешифрування зображень з використанням порозрядних операцій / А. Ковальчук, Д. Пелешко, А. Шкодин, О. Троян // Вісник Національного університету "Львівська політехніка". Комп'ютерні науки та інформаційні технології. – 2011. – № 694. – С. 389–394. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2024/feb/33655/vis694komp-nauky-389-394.pdf (дата звернення: 15.12.2024).

14. Saeed, S., Umar, M. S., Ali, M. A., & Ahmad, M. (2014). Fisher-Yates Chaotic Shuffling Based Image Encryption. *International Journal of Information Processing*, 8(3), 31–41. URL: https://arxiv.org/pdf/1410.7540 (дата звернення: 15.12.2024).

15. Patel, S., Bharath K. P., & Muthu, R. K. (2020). Image Encryption Decryption Using Chaotic Logistic Mapping and DNA Encoding. *arXiv preprint arXiv:2003.06616*. URL: https://arxiv.org/pdf/2003.06616 (дата звернення: 15.12.2024).

16. GitHub - https://github.com/MOXicccc/Diploma

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

8. Cherednichenko O., Kyrychenko I., Tereshchenko H., Miand D., Pylypenko S. Comparison of Blockchain–Based Data Storage Systems // COLINS-2024: 8th International Conference on Computational Linguistics and Intelligent Systems, April 12–13, 2024, Lviv, Ukraine. URL: https://ceur-ws.org/Vol-3688/paper10.pdf

12. Turuta, O., Babiy, A. (2022). Machine Learning Algorithms for Image Classification and Sorting. Journal of Artificial Intelligence Research, 75, Article 12918. URL: https://jair.org/index.php/jair/article/view/12918.