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
«13» червня 2025 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: «Система автоматизованого вибору алгоритмів шифрування та стиснення для різних типів даних з використанням нечіткої логіки»

Виконавець: студентка IV курсу, групи КБ-41

_____ **Валерія ГАЙДУК**
(підпис) (ім'я, прізвище)

	Підпис	**Ім'я ПРІЗВИЩЕ**
Керівник		Юрій БАБЕНКО
Нормоконтроль		Інна МИХАЛЬЧУК

Київ 2025

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
_____Іван ПАРХОМЕНКО
«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студентці **КБ-41** **Гайдук Валерії Вікторівні**
(група) (прізвище ім'я по батькові)

Тема кваліфікаційної роботи «Система автоматизованого вибору алгоритмів шифрування та стиснення для різних типів даних з використанням нечіткої логіки»

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Характеристики різних типів даних, алгоритми шифрування та стиснення, методи нечіткої логіки, програмний засіб на основі нечіткої логіки.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Основи захисту та стиснення даних, аналіз алгоритмів шифрування та системи стиснення, застосування нечіткої логіки, розробка та тестування програмної автоматизованого вибору алгоритмів.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розроблена система автоматизованого вибору алгоритмів шифрування й стиснення для різних типів даних за допомогою нечіткої логіки.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 28 листопада 2024 року

Завдання видав _____ (підпис) Юрій БАБЕНКО (ім'я, прізвище)

Завдання прийняла до виконання _____ (підпис) Валерія ГАЙДУК (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 19.01.2025	*виконано*
2	Аналіз літератури	20.01.2025 – 09.02.2025	*виконано*
3	Обґрунтування вибору рішення	10.02.2025 – 16.02.2025	*виконано*
4	Вивчення властивостей шифрування та стиснення для різних типів даних	17.02.2025 – 09.03.2025	*виконано*
5	Формування стратегій та методологій	10.03.2025 – 23.03.2025	*виконано*
6	Розробка системи автоматизованого вибору алгоритмів	24.03.2025 – 13.04.2025	*виконано*
7	Тестування розробленого програмного рішення	14.04.2025 – 20.04.2025	*виконано*
8	Оформлення пояснювальної записки	21.04.2025 – 25.05.2025	*виконано*
9	Підготовка до захисту кваліфікаційної роботи	26.05.2025 – 13.06.2025	*виконано*

Завдання видав _____ (підпис) Юрій БАБЕНКО (ім'я, прізвище)

Завдання прийняла до виконання _____ (підпис) Валерія ГАЙДУК (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 58 сторінок основного тексту, 1 таблицю та 6 рисунків. Список використаних джерел містить 31 найменування і займає 3 сторінки.

Метою роботи є оптимізація процесу вибору найбільш відповідних алгоритмів шифрування та стиснення для різних типів файлів, а також забезпечення ефективного захисту та обробки даних відповідно до їхніх специфічних характеристик.

Для досягнення зазначеної мети були поставлені наступні завдання:

- провести аналіз типових форматів даних та їх характеристик з погляду особливостей захисту та стиснення;
- дослідити наявні алгоритми шифрування та стиснення, їх переваги та недоліки для різних типів даних;
- створити механізм автоматизованого вибору оптимальних алгоритмів на основі нечіткої логіки;
- реалізувати програмне забезпечення для практичного використання розробленого механізму та провести експериментальне дослідження ефективності запропонованого рішення.

Об'єктом дослідження є процес вибору методів стиснення та шифрування даних в інформаційних системах з урахуванням їх типу та інших характеристик.

Предметом дослідження є методи прийняття рішень щодо вибору оптимальних алгоритмів стиснення та шифрування даних з використанням нечіткої логіки.

Практичною цінністю отриманих результатів є програмне рішення, яке спрямоване на автоматизований вибір алгоритмів шифрування та стиснення для різних типів даних з використанням нечіткої логіки.

Ключові слова: шифрування, стиснення, нечітка логіка.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AES	–	Advanced Encryption Standard
ASCII	–	American Standard Code for Information Interchange
CTR	–	Click-Through Rate
DES	–	Data Encryption Standard
GCM	–	Galois/Counter Mode
JPEG	–	Joint Photographic Experts Group
LZMA	–	Lempel-Ziv-Markov chain Algorithm
PDF	–	Portable Document Format
PNG	–	Portable Network Graphics
PPM	–	Prediction by Partial Matching
RC4	–	Rivest Cipher 4
UTF	–	Unicode Transformation Format
XML	–	EXtensible Markup Language
ZIP	–	Lempel Ziv Welch
СНЛВ	–	Системи нечіткого логічного виведення

ВСТУП

Актуальність роботи. Дедалі більше зростає потреба в оптимізації процесів захисту та обробки даних в умовах постійного збільшення обсягів інформації різних типів. Сучасні цифрові системи працюють з різноманітними форматами даних, які мають свої особливості та вимоги до обробки. Вибір оптимального алгоритму шифрування та стиснення для конкретного типу даних є важливим завданням, оскільки неправильний вибір може призвести до надмірних витрат обчислювальних ресурсів, зниження продуктивності системи або недостатнього рівня захисту. Традиційні підходи до вибору алгоритмів часто не враховують специфічні особливості різних форматів даних та можуть бути неоптимальними. Використання методів нечіткої логіки дозволяє створювати інтелектуальні системи прийняття рішень, які здатні враховувати множину критеріїв та адаптуватися до різних умов, що особливо актуально для задач обробки неоднорідних даних.

Метою кваліфікаційної роботи є оптимізація процесу вибору найбільш відповідних алгоритмів шифрування та стиснення для різних типів файлів, а також забезпечення ефективного захисту та обробки даних відповідно до їхніх специфічних характеристик.

Для досягнення зазначеної мети були поставлені наступні завдання:

- провести аналіз типових форматів даних та їх характеристик з погляду особливостей захисту та стиснення;
- дослідити наявні алгоритми шифрування та стиснення, їх переваги та недоліки для різних типів даних;
- створити механізм автоматизованого вибору оптимальних алгоритмів на основі нечіткої логіки;
- реалізувати програмне забезпечення для практичного використання розробленого механізму та провести експериментальне дослідження ефективності запропонованого рішення.

Об'єкт дослідження: процес вибору методів стиснення та шифрування даних в інформаційних системах з урахуванням їх типу та інших характеристик.

Предмет дослідження: методи прийняття рішень щодо вибору оптимальних алгоритмів стиснення та шифрування даних з використанням нечіткої логіки.

Методи дослідження кваліфікаційної роботи бакалавра: аналіз наукової літератури та документації; порівняльний аналіз алгоритмів шифрування та стиснення; експериментальне дослідження та статистичний аналіз результатів.

Практичною цінністю роботи є програмне рішення, яке спрямоване на автоматизований вибір алгоритмів шифрування та стиснення для різних типів даних з використанням нечіткої логіки.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ СТРУКТУРИ ДАНИХ ТА КРИТЕРІЇВ ВИБОРУ МЕТОДІВ ЇХ ЗАХИСТУ

1.1 Систематизація та структурний аналіз різнотипних форматів даних

Сучасні інформаційні системи обробляють дані різноманітних форматів, кожен з яких має унікальну структуру, сферу застосування та специфічні вимоги до захисту. Розуміння внутрішньої архітектури кожного формату є фундаментальною основою для розробки ефективних механізмів їх захисту та обробки. Цифрові дані можна класифікувати за різними критеріями, але в контексті захисту інформації найбільш доречною є класифікація за форматом представлення, що визначає структурні особливості та вразливості файлів.

Текстові файли є одним із найпоширеніших типів даних, що характеризуються послідовністю символів, закодованих за певним стандартом. Такі файли (.txt, .csv, .xml, .json) зазвичай мають просту лінійну структуру і відносно невеликий розмір. Основними форматами кодування текстових файлів є ASCII та Unicode (UTF-8, UTF-16). Особливість текстових форматів полягає у відносній простоті структури, що одночасно є і перевагою, і недоліком з погляду безпеки.

ASCII-формат використовує 7-бітове кодування і містить 128 символів, що обмежує можливості представлення нелатинських алфавітів. Unicode своєю чергою розширює можливості кодування, підтримуючи більше мільйона унікальних символів [1]. UTF-8 використовує змінну довжину кодування (від 1 до 4 байтів на символ), що робить його ефективним для зберігання текстів латиницею, але менш компактним для символів інших алфавітів [2]. Структурні особливості текстових файлів визначають специфіку їх обробки та захисту – такі

файли легко піддаються аналізу та модифікації, що створює додаткові вимоги до їх захисту.

Документи формату Word (DOCX) мають значно складнішу структуру порівняно з простими текстовими файлами. DOCX є контейнерним форматом, що базується на Open XML і фактично являє собою ZIP-архів, який містить набір XML-файлів та медіаконтент [3]. Основними компонентами DOCX-файлу є:

- document.xml – основний документ, що містить текст та розмітку;
- styles.xml – визначення стилів форматування;
- numbering.xml – параметри нумерації та списків;
- media/ – каталог із вбудованими зображеннями та іншими медіаресурсами;
- settings.xml – налаштування документа;
- додаткові XML-файли для метаданих, тем, шрифтів тощо [4].

Така складна компонентна структура DOCX-формату створює додаткові вектори атак та вимагає комплексного підходу до захисту. Особливу увагу слід приділяти захисту метаданих, які можуть містити конфіденційну інформацію про автора, організацію та історію редагування документа. Формат DOCX також підтримує вбудовування макросів, які можуть бути потенційним джерелом вразливостей і використовуватися для доставлення зловмисного коду.

Растрові зображення (JPEG, PNG) характеризуються двовимірними масивами пікселів з різними колірними моделями та методами стиснення. JPEG (Joint Photographic Experts Group) використовує стиснення з втратами, що дозволяє досягти високого рівня компресії шляхом вилучення надлишкової візуальної інформації [5]. Процес стиснення JPEG включає такі етапи:

- перетворення колірної моделі з RGB в YCbCr;
- субдискретизація хроматичних компонентів [6];
- DCT (Discrete Cosine Transform) – дискретне косинусне перетворення блоків 8x8 пікселів;
- ентропійне кодування (зазвичай кодування Гаффмана) [7].

PNG (Portable Network Graphics) використовує стиснення без втрат на основі алгоритму Deflate, що робить його ідеальним для зображень з однорідними областями кольору, графіки та текстових зображень. Структурні особливості PNG включають підтримку альфа-каналу, керування гамма-корекцією та вбудовані метадані. Формат складається з сигнатури файлу та послідовності фрагментів, кожен з яких має визначений тип та призначення [8].

Відмінності між форматами стиснення з втратами (JPEG) та без втрат (PNG) є критичними при виборі методів захисту. Алгоритми шифрування, застосовані до вже стиснених з втратами зображень, можуть призвести до додаткової деградації якості при повторному збереженні. Крім того, стеганографічні методи по-різному застосовуються до цих форматів – JPEG чутливий до модифікацій через особливості алгоритму стиснення, тоді як PNG надає більше можливостей для приховування інформації [9].

PDF-документи (Portable Document Format) являють собою складні контейнери, що можуть містити текст, векторну та растрову графіку, мультимедійний вміст та інтерактивні елементи. Структура PDF базується на наборі об'єктів, організованих у дерево, з таблицею перехресних посилань, що дозволяє довільний доступ до об'єктів. Ключовими компонентами PDF-файлу є:

- заголовок – визначає версію PDF;
- тіло – містить об'єкти документа (текст, графіка, шрифти);
- таблиця перехресних посилань – забезпечує швидкий доступ до об'єктів;
- трейлер – містить посилання на каталог документа та таблицю перехресних посилань [10].

PDF-формат підтримує вбудовані механізми безпеки, включаючи шифрування вмісту, обмеження доступу та цифрові підписи [11]. Однак складність формату створює численні потенційні вразливості, особливо пов'язані з інтерактивними елементами, такими як JavaScript. PDF може містити приховані дані в метаданих, коментарях або навіть у невидимих шарах документа, що створює додаткові ризики для конфіденційності.

Порівняльний аналіз вразливостей різних форматів даних виявляє специфічні ризики для кожного типу. Текстові файли найбільш вразливі до безпосереднього читання та модифікації через їхню просту структуру. Документи формату DOCX мають ризики, пов'язані з макросами, метаданими та складною структурою XML. Зображення (JPEG, PNG) чутливі до стеганографічних атак та можуть містити приховані дані в метаданих EXIF (Exchangeable Image File Format) або в найменш значущих бітах [12]. PDF-файли уразливі через вбудований JavaScript, форми та можливості включення виконуваного коду.

У табл. 1.1 представлено порівняння основних структурних характеристик розглянутих форматів, що впливають на вибір методів їх захисту.

Таблиця 1.1

Структурні характеристики різних форматів даних

Характеристика	**Текстові файли**	**Файли DOCX**	**JPEG**	**PNG**	**PDF**
Структура	Лінійна послідовність символів	Контейнер ZIP з XML-файлами	Стиснені блоки DCT	Послідовність фрагментів	Дерево об'єктів
Стиснення	Зазвичай без стиснення	ZIP	З втратами	Без втрат	Різні методи
Метадані	Мінімальні	Розширені	EXIF	Вбудовані	Розширені
Можливість вбудовування коду	Ні	Макроси	Ні	Ні	JavaScript

Розуміння внутрішньої структури різних форматів даних дозволяє визначити оптимальні підходи до їх захисту та розробити ефективні алгоритми шифрування та стиснення, що враховують специфіку кожного типу. Особливо

важливим є баланс між рівнем захисту, продуктивністю обробки та збереженням структурної цілісності даних.

Саме різноманітність структурних характеристик цифрових форматів та відсутність чітких меж між категоріями даних обумовлює необхідність застосування методів нечіткої логіки для створення адаптивних систем вибору криптографічних алгоритмів. Такий підхід дозволяє формалізувати знання про особливості захисту різних типів даних та створити інтелектуальну систему прийняття рішень, що враховує складність та багатоваріантність параметрів цифрових форматів.

Для ефективного вибору алгоритмів шифрування та стиснення необхідно враховувати не лише загальні характеристики форматів, але й конкретні сценарії використання даних, вимоги до швидкодії та рівня захисту. Системи автоматизованого вибору алгоритмів, що базуються на нечіткій логіці, дозволяють враховувати множину таких параметрів та обирати оптимальне рішення для кожного типу даних.

1.2 Параметри даних, що впливають на вибір методів захисту

Вибір методів захисту даних є складним процесом, який вимагає врахування численних параметрів. Кожен тип даних має свої унікальні характеристики, що безпосередньо впливають на ефективність обраних методів захисту. Розуміння цих параметрів дає можливість вибрати найбільш підхожий до обробки кожного типу інформації, забезпечуючи належний рівень безпеки, а також оптимізацію ресурсів. У цьому підрозділі буде розглянуто критичні параметри даних, які найбільше впливають на вибір ефективних методів захисту.

Одним із основних параметрів, що визначають вибір методів захисту, є розмір файлу [13]. Важливість цього параметра зумовлена тим, що великі обсяги даних потребують більше обчислювальних ресурсів для їх обробки, а тому вимагають алгоритмів, які оптимізовані для швидкої обробки. Водночас для невеликих файлів потрібно мінімізувати час обробки при збереженні високої

якості шифрування і стиснення. Для великих файлів, де ефективність стиснення має більшу вагу, обираються алгоритми, що забезпечують високе стиснення без значних втрат якості даних.

Структура даних також є важливим критерієм. Деякі формати, такі як текстові файли, мають лінійну структуру і менший розмір, що дозволяє застосовувати алгоритми шифрування, які ефективно обробляють ці файли з мінімальними витратами часу. Інші формати, наприклад, документи формату DOCX або PDF, мають складнішу внутрішню структуру, з великою кількістю компонентів (текст, графіка, метадані), що вимагає застосування більш спеціалізованих алгоритмів для збереження цілісності даних при їх захисті.

Формати даних значно відрізняються між собою, і вибір методів захисту залежить від того, з яким саме форматом ми працюємо. Для текстових файлів завдяки їх простій структурі й відносно невеликому розміру, можна застосовувати стандартні алгоритми шифрування та стиснення. Наприклад, популярні алгоритми, такі як AES (Advanced Encryption Standard) або DES (Data Encryption Standard) для шифрування, а для стиснення – алгоритми на основі архівації, такі як ZIP або LZ4.

У той час як для зображень вибір методів захисту потребує врахування особливостей стиснення. Формати, які мають стиснення з втратами, такі як JPEG, використовують спеціалізовані алгоритми, які зменшують розмір файлу шляхом видалення частини даних, що незначно впливають на якість зображення. Такі файли вимагають особливого підходу до шифрування, оскільки повторне стиснення може призвести до подальших втрат якості. Формати без втрат (наприклад, PNG) надають більше можливостей для збереження якості даних і можуть використовувати стандартні методи шифрування без значних втрат [14].

PDF-документи, які є складними контейнерами для тексту, графіки та інших елементів, вимагають іншого підходу. Завдяки тому, що PDF підтримує вбудовані механізми безпеки та має вразливості через JavaScript, необхідним є застосування спеціалізованих алгоритмів шифрування, що забезпечують захист не лише вмісту, але й додаткових метаданих та інтерфейсів документа [15].

У виборі методів захисту важливим параметром є швидкодія обробки. З урахуванням великого обсягу даних, швидкість обробки може суттєво вплинути на ефективність системи. Для реальних систем, де потрібна обробка даних у реальному часі або для великих обсягів інформації, важливо вибирати алгоритми, які оптимізовані для швидкості виконання, забезпечуючи мінімальні затримки при шифруванні та стисненні. Це особливо важливо для систем, що використовують дані для обробки відео чи великих баз даних.

Цілісність даних є ще одним важливим параметром при виборі методів захисту. Для деяких форматів, таких як PDF або DOCX, де є метадані або приховані елементи, необхідно застосовувати методи шифрування, які захищають як основний вміст файлу, так і ці додаткові елементи. Це допомагає уникнути потенційних вразливостей, коли зловмисники можуть змінити або знищити приховану частину даних.

У системах автоматизованого вибору методи нечіткої логіки представляють ефективний інструмент для прийняття оптимальних рішень в умовах невизначеності та множинності критеріїв. На відміну від класичних методів, які оперують чіткими значеннями, нечітка логіка дозволяє працювати з лінгвістичними змінними та систематизувати експертні знання у вигляді нечітких правил, що відображають реальний процес прийняття рішень людиною.

Запропонована в даній роботі концепція використання нечіткої логіки базується на створенні системи нечіткого виведення, де вхідними параметрами є характеристики даних (розмір, формат, структура), а вихідними – рекомендації щодо оптимальних алгоритмів шифрування та стиснення. Такий підхід дозволяє врахувати нечіткі границі між категоріями даних та створити більш гнучку систему, здатну приймати рішення в умовах невизначеності, які характерні для задач обробки різнотипних даних.

Основними перевагами застосування нечіткої логіки для розв’язання задачі автоматизованого вибору криптографічних алгоритмів є:

- можливість формалізації неоднозначних критеріїв вибору у вигляді нечітких множин і правил. Наприклад, поняття "високий рівень захисту" або "прийнятна швидкодія" можуть бути представлені як нечіткі лінгвістичні змінні;
- здатність одночасно враховувати множину різнорідних параметрів та їх взаємодію в процесі прийняття рішень;
- адаптивність до змінних умов експлуатації через можливість динамічного налаштування функцій приналежності та правил виведення;
- забезпечення плавного переходу між категоріями, що дозволяє уникнути різких змін у рекомендаціях системи при незначних змінах вхідних параметрів.

Використання нечіткої логіки дозволяє створити інтелектуальну систему, яка не лише автоматизує процес вибору оптимальних алгоритмів шифрування та стиснення, але й забезпечує гнучкість та адаптивність рішень до специфічних вимог конкретної задачі. Така система здатна аналізувати множину параметрів для різнотипних даних, визначати їх взаємозв'язки та пропонувати найбільш ефективні комбінації алгоритмів з урахуванням наявних обмежень та пріоритетів.

Особливо ефективним є використання нечіткої логіки для оптимізації балансу між протилежними вимогами, такими як високий рівень захисту та швидкодія, або ефективність стиснення та збереження якості даних.

1.3 Порівняльний аналіз методів захисту

Розглянуті параметри даних визначають вимоги до методів їх захисту та обробки. Однак для прийняття оптимального рішення щодо вибору алгоритмів необхідно також провести порівняльний аналіз самих методів шифрування та стиснення, їх ефективності для різних типів даних, а також обґрунтувати доцільність використання нечіткої логіки для автоматизації цього процесу. Такий аналіз дозволить сформувати основу для створення інтелектуальної системи вибору криптографічних алгоритмів, що є ключовим завданням даної роботи.

Для забезпечення ефективного захисту та обробки різних типів даних необхідно враховувати не лише їх структурні особливості, а й специфіку алгоритмів, що можуть бути застосовані до них. У цьому розділі будуть розглянуті особливості різних методів обробки даних та визначені критерії, за якими доцільно обирати оптимальні алгоритми для кожного типу інформаційних об'єктів.

1.3.1 Особливості методів обробки різних типів даних

Кожен тип даних має свої характерні особливості, які впливають на ефективність застосування до них різних алгоритмів шифрування та стиснення. Зокрема, текстові дані через їхню структуру добре піддаються стисненню методами, що використовують статистичні закономірності у тексті. Такі алгоритми як Deflate, LZMA (Lempel-Ziv-Markov chain Algorithm) або Bzip2 показують високу ефективність для текстових даних, оскільки вони побудовані на принципах пошуку повторюваних послідовностей, які часто зустрічаються у текстах.

Зображення, залежно від формату, потребують різних підходів. Для форматів без втрат, таких як PNG, ефективними є алгоритми стиснення без втрат (наприклад, LZ4), що зберігають усі дані без змін. Для форматів з втратами, таких як JPEG, додаткове стиснення може призвести до погіршення якості, тому часто доцільніше фокусуватися на швидших алгоритмах шифрування при мінімальній додатковій обробці.

Документи складних форматів, таких як DOCX або PDF, потребують особливого підходу через їхню компонентну структуру. Оскільки вони вже містять стиснені компоненти (наприклад, DOCX базується на ZIP-архіві), додаткове стиснення може бути неефективним. Для таких форматів важливішим є вибір шифрування, що зберігає структурну цілісність даних та захищає метадані.

1.3.2 Критерії вибору оптимальних алгоритмів

При виборі оптимальних алгоритмів шифрування та стиснення для різних типів даних необхідно враховувати низку критеріїв, що визначають ефективність обробки та рівень захисту інформації. У цьому випадку виділимо шість основних критеріїв:

1. Тип даних: кожен формат має свої структурні особливості, що визначають ефективність застосування до нього різних алгоритмів.
2. Розмір даних: для великих файлів важливішим може бути ступінь стиснення, тоді як для невеликих – швидкість обробки.
3. Вимоги до швидкодії: у системах реального часу або з обмеженими обчислювальними ресурсами перевага надається швидким алгоритмам, навіть якщо вони забезпечують нижчий ступінь стиснення чи рівень захисту.
4. Вимоги до рівня захисту: для критично важливих даних перевага надається алгоритмам з вищим рівнем безпеки, навіть якщо вони працюють повільніше.
5. Обчислювальні ресурси: доступність обчислювальних ресурсів визначає можливість застосування ресурсомістких алгоритмів.
6. Сценарій використання: різні сценарії, такі як передача даних, зберігання, архівація, висувають різні вимоги до алгоритмів.

Для різних типів файлів оптимальними будуть різні комбінації алгоритмів шифрування та стиснення. Наприклад, для текстових даних ефективною може бути комбінація Bzip2 для стиснення та AES для шифрування, тоді як для зображень у форматі JPEG краще використовувати швидкий алгоритм шифрування RC4 без додаткового стиснення.

Також важливо враховувати порядок застосування алгоритмів. Зазвичай спочатку виконується стиснення, а вже потім шифрування, оскільки шифрування збільшує ентропію даних і робить їх менш придатними для подальшого стиснення.

Взаємозалежність критеріїв вибору та їх нечітка природа створюють складну задачу оптимізації, яку важко змоделювати традиційними методами. Саме тому в даній роботі, використовуючи методи нечіткої логіки, такі критерії як високий рівень захисту, швидкодія або великий розмір файлу представляються у вигляді нечітких множин з відповідними функціями приналежності.

1.3.3 Обґрунтування застосування нечіткої логіки

Вибір оптимальних алгоритмів є класичною задачею багатофакторного вибору, де необхідно враховувати множину параметрів з різними ваговими коефіцієнтами. У таких умовах стандартні методи прийняття рішень можуть бути недостатньо гнучкими.

Нечітка логіка пропонує ефективний підхід для розв’язання таких задач. Для вибору оптимальних алгоритмів можна використовувати систему, де вхідними параметрами є характеристики даних та вимоги до обробки, а вихідними – рекомендації щодо вибору алгоритмів шифрування та стиснення. Така система базується на множині нечітких правил, що відображають інформацію про ефективність різних алгоритмів для різних типів даних та умов їх обробки.

Наприклад, правило може мати вигляд: "ЯКЩО файл є текстовим, І розмір файлу малий, І вимоги до швидкодії високі, ТО рекомендується використовувати алгоритм LZ4 для стиснення та AES для шифрування".

Перевагою такого підходу є адаптивність системи до різних умов та можливість врахування множини факторів, що впливають на вибір алгоритмів. Крім того, нечітка система може бути легко розширена додатковими правилами або адаптована до нових типів даних чи алгоритмів.

Отже, застосування методів нечіткої логіки дозволяє створити інтелектуальну систему прийняття рішень, яка здатна не лише обрати найбільш

доцільні алгоритми для кожного типу даних, але й оптимізувати баланс між суперечливими вимогами.

Висновки за розділом 1

У першому розділі проведено комплексне дослідження структурних властивостей різних типів цифрових даних та визначено ключові критерії вибору методів їх захисту. Систематизація та структурний аналіз цифрових форматів виявив, що кожен тип даних має унікальні характеристики, які безпосередньо впливають на ефективність застосування методів шифрування та стиснення. Текстові файли характеризуються лінійною структурою та відносно невеликим розміром, що робить їх придатними для застосування стандартних алгоритмів. Документи формату DOCX мають складну контейнерну структуру на основі ZIP-архіву з набором XML-файлів та мультимедійним вмістом, що вимагає особливого підходу до їх захисту. Зображення JPEG і PNG відрізняються за методами стиснення (з втратами та без втрат відповідно), що значно впливає на вибір оптимальних алгоритмів шифрування. PDF-документи представляють складні контейнери з ієрархічною структурою об'єктів, що можуть включати різноманітний вміст та інтерактивні елементи, створюючи додаткові вимоги до їх захисту.

Визначено основні параметри даних, що впливають на вибір методів захисту: розмір файлу, структура даних, специфіка формату, вимоги до швидкодії та цілісності даних, а також особливості застосування відповідних алгоритмів. Виявлено, що ці параметри мають взаємопов'язаний характер та часто являють собою нечіткі множини з розмитими границями, що обґрунтовує доцільність застосування методів нечіткої логіки для автоматизованого вибору криптографічних алгоритмів.

Порівняльний аналіз методів захисту різнотипних даних показав, що ефективність алгоритмів шифрування та стиснення суттєво залежить від типу даних та сценарію використання. Для текстових даних ефективними є алгоритми

стиснення, що використовують статистичні закономірності тексту, тоді як для зображень з втратами додаткове стиснення може призвести до погіршення якості.

Обґрунтовано застосування нечіткої логіки для розв’язання задачі автоматизованого вибору криптографічних алгоритмів. На відміну від класичних методів, нечітка логіка дозволяє представити інформацію у вигляді нечітких правил та створити адаптивну систему прийняття рішень, здатну враховувати множину гетерогенних параметрів. Такий підхід забезпечує гнучкість та адаптивність.

Проведений аналіз створює теоретичне підґрунтя для розробки системи автоматизованого вибору на основі нечіткої логіки, що дозволить оптимізувати процеси захисту та обробки різних типів цифрових даних з урахуванням їх структурних особливостей та специфічних вимог.

РОЗДІЛ 2
ВИКОРИСТАННЯ НЕЧІТКОЇ ЛОГІКИ ДЛЯ АВТОМАТИЗОВАНОГО ВИБОРУ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ

2.1 Аналіз сучасних алгоритмів шифрування

Методи шифрування відіграють критичну роль у забезпеченні захисту даних від несанкціонованого доступу. Для правильного застосування цих методів необхідно розуміти їх внутрішні механізми роботи, переваги, обмеження та ефективність для різних типів даних.

Криптографічні алгоритми традиційно поділяються на дві основні категорії: симетричні та асиметричні. Симетричні алгоритми використовують один ключ для шифрування і дешифрування, тоді як асиметричні базуються на парі математично пов'язаних ключів – публічному та приватному [16]. Крім того, симетричні шифри додатково класифікуються на блокові та потокові, залежно від принципу їх роботи з даними.

Блокові шифри обробляють дані фіксованими блоками, застосовуючи до кожного блоку однакове перетворення. Найбільш поширеними представниками цього класу є AES, DES, Triple DES та Blowfish. Потокові шифри, такі як RC4, обробляють дані по одному біту або байту, що надає їм певні переваги в специфічних сценаріях використання [17].

AES є на сьогодні найбільш поширеним та рекомендованим стандартом симетричного шифрування, прийнятим Національним інститутом стандартів і технологій США у 2001 році. AES працює з блоками даних розміром 128 біт та підтримує ключі довжиною 128, 192 або 256 біт. Алгоритм базується на мережі замін та перестановок і складається з повторюваних раундів, які включають такі основні операції: SubBytes – заміна байтів, ShiftRows – зсув рядків, MixColumns – змішування стовпців та AddRoundKey – додавання ключа раунду [18].

Одна з основних переваг AES – оптимальне співвідношення між безпекою та продуктивністю. Завдяки ефективній архітектурі, алгоритм демонструє високу швидкодію як при програмній, так і при апаратній реалізації. Сучасні процесори часто включають спеціальні інструкції для прискорення AES-операцій, наприклад AES-NI – Advanced Encryption Standard New Instructions, що дозволяє досягати надзвичайно високої пропускної здатності – до декількох гігабайтів на секунду на звичайному персональному комп'ютері [19].

З погляду криптографічної стійкості, AES вважається надзвичайно надійним. За більш ніж два десятиліття інтенсивного криптоаналізу не було виявлено практичних атак, які б компрометували повну версію алгоритму. Найкращі відомі атаки на AES-256 мають складність порядку 2^{254} [20], що значно перевищує обчислювальні можливості сучасних і прогнозованих комп'ютерних систем.

При застосуванні до різних типів даних AES демонструє свою універсальність. Для текстових файлів він забезпечує швидке та надійне шифрування без значного збільшення розміру даних. У випадку роботи з зображеннями або мультимедійними файлами, які вже містять стиснені дані, наприклад, JPEG та MP3, можуть виникати незначні затримки через необхідність обробки великих обсягів інформації.

DES – це алгоритм, який був стандартом шифрування в США з 1977 до 2000 року, коли його замінив AES. Алгоритм працює з блоками даних розміром 64 біти та використовує ключ довжиною 56 біт [21]. DES базується на мережі Фейстеля, яка складається з 16 раундів, де в кожному раунді половина блоку модифікується за допомогою функції, що залежить від іншої половини та підключа [22].

Недоліком DES є недостатня довжина ключа, що робить його вразливим до атак повного перебору. В 1998 році було продемонстровано можливість зламу DES за 56 годин з використанням спеціалізованого обладнання [23]. На сучасному рівні розвитку обчислювальної техніки час, необхідний для зламу

DES, скоротився до лічених годин, що робить його застосування неприйнятним для захисту конфіденційних даних.

3DES (Triple DES) був розроблений як тимчасове розв'язання проблеми вразливості DES. Він застосовує алгоритм DES тричі до кожного блоку даних, використовуючи два або три різні ключі, що ефективно збільшує довжину ключа до 112 або 168 біт відповідно [24]. Це значно підвищує криптографічну стійкість, але ціною суттєвого зниження продуктивності – 3DES приблизно в три рази повільніший за базовий DES.

Через свою низьку швидкодію 3DES не є оптимальним вибором для шифрування великих обсягів даних або для систем з обмеженими обчислювальними ресурсами. Однак він може бути виправданим рішенням для забезпечення зворотної сумісності з застарілими системами або для захисту невеликих за обсягом критично важливих даних, де швидкість шифрування не є визначальним фактором.

Blowfish – це симетричний блоковий шифр, розроблений Брюсом Шнаєром у 1993 році знову ж таки як альтернатива застарілому DES. Алгоритм працює з блоками розміром 64 біти та підтримує ключі змінної довжини від 32 до 448 біт. Blowfish також базується на мережі Фейстеля і включає 16 раундів обробки даних [25].

Особливістю Blowfish є використання ключ-залежних S-блоків замін та P-блоків перестановок, які генеруються під час ініціалізації ключа. Цей процес вимагає значних обчислювальних ресурсів і включає шифрування приблизно 4 кілобайт даних, що робить ініціалізацію ключа відносно повільною операцією. Однак після завершення ініціалізації алгоритм працює з високою швидкістю, особливо при шифруванні великих обсягів даних з одним ключем.

Blowfish чудово підходить для захисту текстових даних та невеликих файлів, де ініціалізація ключа не створює значних затримок. Для обробки великих обсягів даних довготривала ініціалізація може бути компенсована високою швидкістю подальшого шифрування. Однак для сценаріїв, де ключі

часто змінюються, Blowfish може бути менш ефективним через витрати часу на ініціалізацію.

RC4 – це потоковий шифр, розроблений Рональдом Рівестом у 1987 році. RC4 обробляє дані побайтово, що робить його особливо ефективним для шифрування даних змінної довжини. Алгоритм надзвичайно простий у реалізації та використовує ключі довжиною від 8 до 2048 біт [26].

Принцип роботи RC4 базується на генерації псевдовипадкової послідовності байтів, яка потім об'єднується з відкритим текстом за допомогою операції XOR. Принцип роботи складається з алгоритмів генерації ключового потоку та псевдовипадкових чисел.

RC4 демонструє високу швидкодію та низькі вимоги до обчислювальних ресурсів, що робить його привабливим для використання в системах з обмеженою продуктивністю. Однак протягом років експлуатації були виявлені серйозні вразливості в алгоритмі, особливо пов'язані з початковими байтами згенерованого ключового потоку. Ці вразливості призвели до успішних атак на протоколи, що використовують RC4, такі як WEP та ранні версії TLS.

Через виявлені вразливості RC4 більше не рекомендується для використання в нових системах. Сучасні стандарти безпеки, такі як TLS 1.3, повністю відмовилися від RC4 на користь більш надійних алгоритмів. Однак розуміння принципів роботи RC4 залишається важливим для аналізу застарілих систем та історичного контексту розвитку криптографії.

Асиметричні алгоритми шифрування, такі як RSA, базуються на математичних властивостях односторонніх функцій та використовують пару ключів – публічний для шифрування та приватний для дешифрування. RSA, розроблений у 1977 році, базується на складності факторизації великих чисел і залишається одним з найбільш поширених асиметричних алгоритмів [27].

Принцип роботи RSA полягає у використанні добутку двох великих простих чисел для створення модуля, який є частиною обох ключів [28]. Безпека алгоритму базується на обчислювальній складності розкладання цього модуля на оригінальні прості множники.

RSA значно повільніший за симетричні алгоритми – на кілька порядків, що обмежує його застосування для шифрування великих обсягів даних. Зазвичай він використовується для безпечного обміну ключами або цифрового підписування, а не для безпосереднього шифрування вмісту файлів. У гібридних системах шифрування часто застосовується для захисту симетричного ключа, який потім використовується для шифрування основних даних за допомогою швидшого симетричного алгоритму, такого як AES.

Важливо також враховувати режими роботи блокових шифрів, які визначають, як алгоритм застосовується до послідовності блоків даних. Найпоширенішими режимами є:

- ECB (Electronic Codebook) – найпростіший режим, де кожен блок шифрується незалежно. Цей режим не рекомендується для більшості застосувань, оскільки однакові блоки відкритого тексту шифруються в однакові блоки шифротексту, що може розкривати патерни у вихідних даних [29].
- CBC (Cipher Block Chaining) – режим, де кожен блок відкритого тексту комбінується з результатом шифрування попереднього блоку перед застосуванням алгоритму шифрування, що забезпечує залежність між блоками та підвищує безпеку, але вимагає послідовної обробки блоків.
- CTR (Click-Through Rate, Counter) – режим, який перетворює блоковий шифр на потоковий шляхом шифрування послідовних значень лічильника та комбінування результатів з блоками відкритого тексту. Цей режим дозволяє паралельну обробку блоків, що підвищує продуктивність [30].
- GCM (Galois/Counter Mode) – режим, що забезпечує не лише конфіденційність, але й автентифікацію даних. Він комбінує режим CTR з поліноміальним хешуванням для генерації ідентифікаційного тегу. Цей режим є стандартним для багатьох сучасних протоколів безпеки через його ефективність та додаткові гарантії безпеки.

Аналіз сучасних алгоритмів шифрування показує, що не існує універсального рішення, яке було б оптимальним для всіх типів даних та сценаріїв використання.

2.2 Аналіз сучасних алгоритмів стиснення

Стиснення даних є важливим елементом обробки інформації, що дозволяє зменшити обсяг даних для зберігання та передачі, одночасно зберігаючи їх цілісність. Різні типи даних потребують специфічних підходів до стиснення, оскільки їхні внутрішні структури та характеристики суттєво відрізняються.

Алгоритми стиснення можна розділити на дві основні категорії. Алгоритми без втрат забезпечують повне відновлення початкових даних, що особливо важливо для текстових документів, програмного коду та структурованих даних. Алгоритми з втратами жертвують частиною інформації для досягнення вищого коефіцієнта стиснення, що є допустимим для медіаконтенту, де незначні втрати якості можуть бути непомітними.

Deflate є одним з найпоширеніших алгоритмів стиснення без втрат, що лежить в основі популярних форматів ZIP, gzip та PNG. Цей алгоритм працює у два етапи: спочатку знаходить повторювані послідовності байтів і замінює їх посиланнями на попередні екземпляри, а потім застосовує кодування Гаффмана для оптимізації представлення символів та посилань. Він демонструє хороший баланс між ступенем стиснення та швидкістю роботи, що робить його універсальним рішенням для багатьох типів даних, особливо для текстових файлів, програмного коду та структурованих документів. Для зображень з різкими контрастами, текстом або лініями, алгоритм Deflate, який використовується в PNG, забезпечує хороші результати, особливо за наявності великих однотонних областей або обмеженої палітри кольорів.

LZ4 – це універсальний алгоритм стиснення без втрат, який використовує модифікований словниковий підхід, але з оптимізованими структурами даних та пошуковими алгоритмами, що забезпечують декомпресію зі швидкістю понад 5 ГБ/с на сучасних процесорах. Така особливість LZ4 ідеальним для сценаріїв, де критичною є швидкість обробки даних, наприклад, у мережевій передачі даних або системах передачі даних у реальному часі [31].

Хоча коефіцієнт стиснення LZ4 нижчий порівняно з іншими алгоритмами, його надзвичайна швидкість компенсує цей недолік у багатьох практичних застосуваннях. Дослідження показують, що для деяких типів даних комбінація швидкості та ступеня стиснення LZ4 забезпечує вищу ефективну пропускну здатність системи порівняно з повільнішими алгоритмами, що мають вищий коефіцієнт стиснення.

Bzip2 є також алгоритмом стиснення без втрат, який використовує перетворення Барроуза-Вілера у поєднанні з кодуванням Гаффмана та моделлю переходу рангів. Цей алгоритм забезпечує значно вищий ступінь стиснення порівняно з Deflate, але працює повільніше на етапах компресії та декомпресії. Bzip2 особливо ефективний для звичайних текстів із високою повторюваністю слів та фраз.

LZMA є одним із найефективніших алгоритмів стиснення без втрат з погляду коефіцієнта стиснення. Цей алгоритм використовується у форматі 7z та xz і поєднує модифікований словниковий метод з контекстним моделюванням та арифметичним кодуванням. LZMA використовує складну систему передбачення символів на основі ланцюгів Маркова, що дозволяє досягти надзвичайно високого ступеня стиснення, але потребує значних обчислювальних ресурсів.

PPM (Prediction by Partial Matching) представляє сімейство адаптивних статистичних алгоритмів стиснення, що використовують контекстне моделювання для передбачення наступних символів. PPM будує модель даних, яка оцінює ймовірність появи кожного символу на основі контексту символів, що йому передують. Ці ймовірності потім використовуються для арифметичного кодування, що дозволяє досягти надзвичайно високого ступеня стиснення для текстових даних. Як і LZMA, алгоритми PPM потребують значних обчислювальних ресурсів та працюють відносно повільно.

Для медіаформатів, таких як зображення, аудіо та відео, широко використовуються алгоритми стиснення з втратами, специфічні для кожного типу даних. Такі алгоритми враховують особливості сприйняття та жертвують

найменш важливими аспектами інформації для досягнення значно вищого коефіцієнта стиснення.

Важливо розуміти, що застосування додаткового стиснення до вже стиснених даних зазвичай неефективне або навіть шкідливе. Наприклад, спроба застосувати Deflate або LZMA до JPEG-зображень дає мінімальний виграш у розмірі через високу ентропію вже стиснених даних. На додаток, для форматів з втратами, додаткове стиснення та декомпресія можуть призвести до подальшої деградації якості при повторному збереженні.

Порівняльний аналіз основних алгоритмів стиснення для різних типів даних демонструє, що не існує універсального алгоритму, який би був оптимальним для всіх типів даних та сценаріїв використання. Вибір алгоритму залежить від багатьох факторів.

Текстові дані та документи найкраще стискаються алгоритмами PPM та LZMA, які забезпечують максимальний коефіцієнт стиснення, але потребують значних обчислювальних ресурсів. Для сценаріїв, де важлива швидкість, більш доцільним є використання LZ4 або Deflate, попри менш ефективне стиснення.

Структуровані дані, такі як XML, JSON або CSV, демонструють високу надлишковість через повторювані теги, розділові знаки та структурні елементи. Для таких даних ефективними є алгоритми на основі словникових методів, зокрема LZMA та Bzip2, які здатні виявляти та кодувати такі повторення.

Бінарні дані, такі як виконувані файли, бази даних або архіви, зазвичай вже мають оптимізовану структуру з мінімальною надлишковістю, що ускладнює їх подальше стиснення. Для таких даних найефективнішими є алгоритми з високим коефіцієнтом стиснення, такі як LZMA або Bzip2, хоча досягнутий коефіцієнт стиснення зазвичай нижчий порівняно з текстовими чи структурованими даними.

Для складних форматів, таких як DOCX або PDF, які вже містять стиснені компоненти (DOCX базується на ZIP-архіві з алгоритмом Deflate), додаткове стиснення всього файлу часто малоефективне. У таких випадках більш

доцільним є вибірковий підхід, при якому стисненню піддаються лише окремі компоненти, які ще не були стиснені.

Важливим аспектом вибору алгоритму стиснення є його взаємодія з алгоритмами шифрування. Шифрування зазвичай збільшує ентропію даних, ускладнюючи або унеможливлюючи подальше стиснення. Тому оптимальним підходом є спочатку стиснення даних для зменшення їх обсягу, а потім шифрування для забезпечення безпеки. Зворотний порядок зазвичай призводить до низької ефективності стиснення через високу ентропію шифрованих даних.

Наявність нечіткої логіки дозволяє створити адаптивну систему прийняття рішень, яка враховує не лише чіткі параметри (розмір файлу, формат), але й нечіткі характеристики (прийнятна швидкодія, допустимі втрати якості). Такий підхід забезпечує оптимальний вибір алгоритмів стиснення з урахуванням специфіки кожного типу даних та конкретних вимог до їх обробки.

2.3 Принципи побудови систем нечіткого логічного виведення

Системи нечіткого логічного виведення (СНЛВ) є ефективним інструментом для розв'язання багатьох задач, що базуються на теорії нечітких множин, яка дозволяє формалізувати якісні поняття та експертні знання за допомогою функцій приналежності та нечітких правил.

Принципи побудови СНЛВ включають кілька ключових етапів. Першим етапом є процес перетворення чітких вхідних величин у нечіткі множини. На цьому етапі для кожної вхідної змінної визначаються лінгвістичні позначки та відповідні їм функції приналежності. Наприклад, для змінної "розмір файлу" можуть бути визначені позначення "малий", "середній" і "великий", а для змінної "вимоги до безпеки" – "низькі", "середні" і "високі".

Функції приналежності відображають ступінь відповідності значення вхідної змінної кожному з визначених понять. Найчастіше використовуються трикутні, трапецієподібні або гауссові функції приналежності, вибір яких залежить від характеру конкретної змінної та експертних оцінок. Наприклад, для

змінної "розмір файлу" може бути використана трапецієподібна функція приналежності, що відображає поступовий перехід між категоріями.

Другим ключовим етапом є формування бази правил типу "ЯКЩО-ТО". Правила пов'язують вхідні змінні з вихідними рекомендаціями щодо вибору алгоритмів. База правил є ключовим компонентом СНЛВ, оскільки саме вона визначає логіку прийняття рішень. Для ефективного функціонування системи автоматизованого вибору криптографічних алгоритмів необхідно сформувати повний набір правил, що охоплює різні комбінації вхідних параметрів. Експертна оцінка фахівців у галузі інформаційної безпеки та обробки даних є критично важливою для формування такої бази правил.

Третім етапом є поєднання підумов у нечітких правилах продукції. На цьому етапі визначається ступінь істинності умов для кожного правила на основі значень функцій приналежності вхідних змінних. Для цього зазвичай використовуються операції нечіткого логічного "І" (мінімум) та "АБО" (максимум).

Четвертим етапом є активізація підвисновків у нечітких правилах продукції. Тут визначається ступінь істинності кожного з підвисновків на основі ступеня істинності відповідної умови та заданого методу нечіткої імплікації. Найчастіше використовуються методи мінімуму або множення.

П'ятим етапом є нагромадження висновків нечітких правил продукції. На цьому етапі об'єднуються всі ступені істинності висновків для отримання нечіткої множини для кожної вихідної змінної. Для акумуляції зазвичай використовується операція максимуму.

Завершальним етапом роботи СНЛВ є процес перетворення нечіткої множини у чітке число. Найбільш поширеними методами для перетворення є центр ваги, центр максимумів та середнє значення максимумів.

Для задачі автоматизованого вибору криптографічних алгоритмів доцільно використовувати метод центра ваги, який забезпечує більш плавний перехід між рекомендаціями при незначних змінах вхідних параметрів. Цей метод обчислює

зважене середнє значення вихідної змінної, де вагами є ступені приналежності до відповідної нечіткої множини.

Побудові СНЛВ сприяє модульна архітектура системи, де окремі модулі відповідають за аналіз специфічних характеристик даних та формування відповідних рекомендацій. Важливим аспектом є також можливість адаптації системи до нових умов шляхом корегування параметрів функцій приналежності та бази правил на основі результатів експериментальних досліджень. Це забезпечує гнучкість системи та її здатність враховувати змінні вимоги до обробки даних у різних умовах експлуатації.

2.4 Взаємодія алгоритмів шифрування та стиснення

Порядок застосування операцій шифрування та стиснення суттєво впливає на результат обробки даних. З теоретичного погляду, спочатку слід виконувати стиснення, а потім шифрування. Це зумовлено фундаментальними властивостями даних процесів: стиснення зменшує надлишковість даних, тоді як шифрування збільшує ентропію інформації, роблячи її статистично близькою до випадкової послідовності.

При застосуванні стиснення до вже зашифрованих даних ефективність значно знижується, оскільки шифрування маскує статистичні закономірності та повторювані послідовності, які є основою для роботи алгоритмів стиснення. У результаті, стиснення може не лише виявитися неефективним, але й у деяких випадках навіть збільшити розмір зашифрованих даних через додавання службової інформації.

Однак, у деяких специфічних випадках можуть виникати ситуації, коли доцільно змінити цей порядок або застосувати комбіновані підходи. Наприклад, при роботі з форматами, що вже містять стиснені компоненти (JPEG, DOCX, PDF), додаткове стиснення перед шифруванням може бути малоефективним і призводити до зайвих витрат обчислювальних ресурсів. У таких випадках раціональним є безпосереднє шифрування даних без попереднього стиснення.

Слід також враховувати проблеми, що виникають при взаємодії різних комбінацій алгоритмів. Одна з ключових проблем – це несумісність деяких алгоритмів шифрування та стиснення при спільному застосуванні. Наприклад, блокові шифри можуть вносити помилки при шифруванні стиснених даних, якщо розмір блоку не узгоджений з форматом стиснених даних. Потокові шифри, такі як RC4, зазвичай краще взаємодіють з алгоритмами стиснення, оскільки обробляють дані побайтово, не змінюючи їх структуру.

Інша проблема – це вплив помилок при передачі або зберіганні даних на результат дешифрування. Якщо помилка виникає в зашифрованих стиснених даних, вона може призвести до неможливості відновлення всього файлу після дешифрування через особливості алгоритмів стиснення. Ця проблема особливо актуальна для алгоритмів стиснення з використанням арифметичного кодування або адаптивних словників.

Для розв'язання цих проблем можуть використовуватися різні підходи. Один із них – це застосування методів завадостійкого кодування перед шифруванням стиснених даних, що дозволяє виявляти та виправляти помилки. Інший підхід – це використання спеціальних режимів шифрування, таких як CFB (Cipher Feedback) або OFB (Output Feedback), які мінімізують поширення помилок при дешифруванні.

Висновки за розділом 2

У другому розділі роботи було досліджено теоретичні основи впровадження нечіткої логіки для автоматизованого вибору криптографічних алгоритмів. Проведений аналіз показав, що сучасні методи шифрування значно відрізняються за своїми характеристиками, включаючи криптографічну стійкість, швидкодію та ефективність для різних типів даних. Симетричні алгоритми, такі як AES, DES, Triple DES, Blowfish та RC4, мають свої унікальні властивості та сфери оптимального застосування. Асиметричні алгоритми, хоча

й повільніші, забезпечують додаткові механізми захисту, такі як цифрове підписування та безпечний обмін ключами.

Паралельно з цим було розглянуто алгоритми стиснення даних: Deflate, LZ4, Bzip2, LZMA, PPM. Кожен з цих алгоритмів демонструє різну ефективність залежно від типу даних, їх структури та вимог до збереження якості інформації.

Важливим аспектом дослідження стало вивчення взаємодії алгоритмів шифрування та стиснення. Встановлено, що порядок застосування цих процесів критично впливає на кінцевий результат. Оптимальною стратегією є застосування спочатку стиснення для зменшення надлишковості даних, а потім шифрування для забезпечення конфіденційності.

Для застосування великої кількості критеріїв доцільним буде використання систем нечіткого логічного виведення. Проведене дослідження створює методологічну основу для розробки інтелектуальної системи автоматизованого вибору алгоритмів шифрування та стиснення, що здатна адаптуватися до різноманітних типів даних, враховувати специфічні вимоги до їх обробки та забезпечувати оптимальний баланс між безпекою, швидкодією та ефективністю використання обчислювальних ресурсів.

РОЗДІЛ 3
РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ВИБОРУ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ

У цьому розділі детально розглядається архітектура та програмна реалізація системи автоматизованого вибору алгоритмів шифрування і стиснення на основі методів нечіткої логіки. Для кращого розуміння практичної реалізації, код головної форми, яка містить основну логіку інтерфейсу користувача та керування процесами навчання, підбору і застосування алгоритмів, наведено в Додатку А.

3.1 Архітектура системи автоматизованого вибору

Архітектура розробленої системи побудована за принципом модульної структури з чітким розподілом функціональних обов'язків між компонентами. Система реалізована як Windows Forms додаток з використанням платформи .NET, що забезпечує зручний графічний інтерфейс користувача та високу продуктивність обробки даних.

Основою архітектури системи є структура інтерфейсу користувача з вкладками, де кожна вкладка відповідає за окремий етап роботи. Перша вкладка забезпечує процес навчання системи, друга реалізує інтелектуальний підбір оптимальних алгоритмів, а третя надає можливості для ручного застосування обраних методів обробки даних. Така структура відображає логічний процес роботи з системою: від збору даних та навчання до практичного застосування отриманих знань.

Центральним компонентом архітектури є головна форма MainForm, яка виконує роль координатора всіх процесів системи і містить основну логіку додатка та керує взаємодією між різними модулями. У рамках головної форми реалізовано систему управління колекціями алгоритмів, де кожен тип алгоритму

представлений відповідним інтерфейсом. Алгоритми шифрування реалізовано через інтерфейс IEncryptionAlgorithm, який включає методи AES, DES, Triple DES, Blowfish та RC4. Алгоритми стиснення представлені інтерфейсом ICompressionAlgorithm з реалізаціями Deflate, LZ4, Bzip2, LZMA та PPM.

Модуль навчання системи являє собою комплексний механізм аналізу файлів різних типів та оцінки ефективності різних комбінацій алгоритмів. Інтерфейс модуля навчання (рис. 3.1) містить мінімальний набір елементів управління: поле для введення шляху до теки з файлами для аналізу, поле для введення ключового слова та кнопку запуску процесу навчання. Архітектура цього модуля передбачає ітеративний процес обробки файлів у вказаній користувачем директорії. Для кожного файлу система обчислює унікальний хеш-ідентифікатор за допомогою алгоритму SHA-256, що дозволяє уникнути повторної обробки вже проаналізованих файлів. Це забезпечує ефективність роботи системи та запобігає дублюванню результатів навчання.

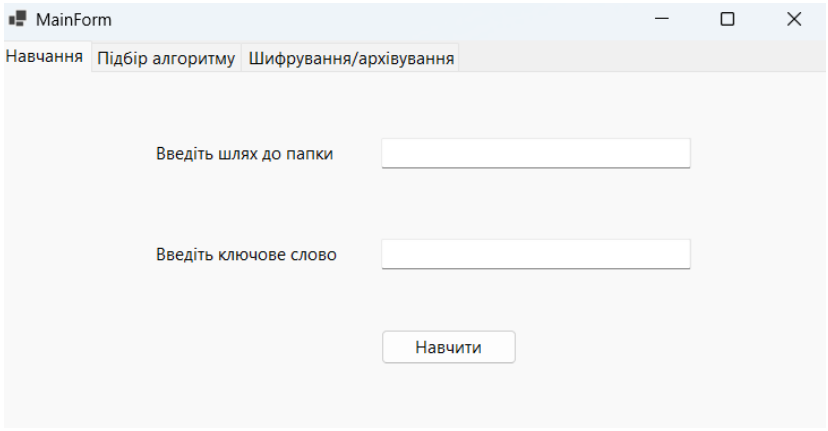

Рисунок 3.1 – Інтерфейс модуля навчання системи

Процес навчання організовано як повний перебір комбінацій алгоритмів шифрування та стиснення для кожного файлу. Система тестує спочатку шифрування, потім стиснення, та навпаки. Для кожної комбінації система вимірює час виконання операцій за допомогою високоточного таймера Stopwatch та обчислює коефіцієнт стиснення як відношення розміру результативного файлу до початкового. Всі отримані дані зберігаються у текстовому файлі бази

даних results_database.txt у структурованому форматі, що включає розширення файлу, його початковий розмір, використану комбінацію алгоритмів, кінцевий розмір, коефіцієнт стиснення та час обробки.

Модуль інтелектуального підбору алгоритмів реалізує ядро системи нечіткої логіки для прийняття рішень. Архітектура цього модуля базується на аналізі накопичених під час навчання даних та застосуванні нечітких множин для оцінки ефективності алгоритмів. Система використовує функції належності для перетворення вхідних параметрів, де час виконання класифікується як швидкий, середній або повільний, а коефіцієнт стиснення оцінюється як високий, середній або низький. Графічний інтерфейс модуля інтелектуального підбору (рис. 3.2) надає користувачеві можливість вибору файлу для аналізу та критерію оптимізації через перемикач, що дозволяє налаштувати систему на пріоритет швидкості виконання або якості стиснення.

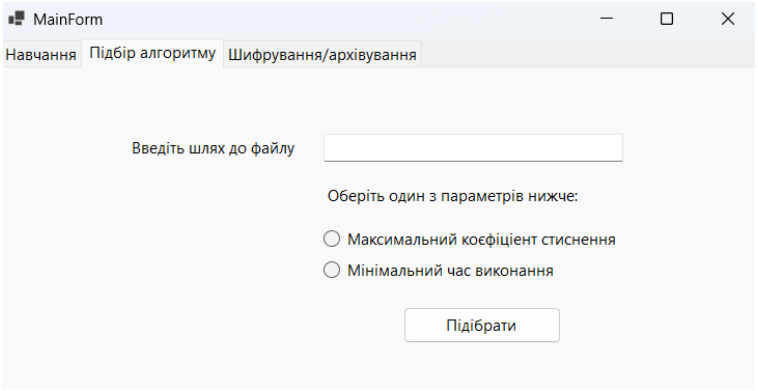

Рисунок 3.2 – Інтерфейс модуля підбору алгоритмів з критеріями оптимізації

Важливою особливістю архітектури є система динамічного визначення ваг критеріїв залежно від типу файлу. Система містить словник fileTypeWeights, який зберігає попередньо налаштовані коефіцієнти важливості часу виконання та якості стиснення для різних розширень файлів. Для текстових файлів пріоритет надається стисненню, для зображень важливішою є швидкість обробки, а для документів встановлено збалансований підхід. Ця адаптивність дозволяє системі враховувати специфіку різних типів даних та надавати більш точні рекомендації.

Архітектура модуля підбору також включає механізм сегментації даних за розміром файлів. Система автоматично групує результати навчання на категорії: файли менш ніж 1 МБ, середні від 1 до 10 МБ та великі понад 10 МБ. Це дозволяє системі надавати рекомендації, що враховують не лише тип файлу, але і його розмір, оскільки ефективність алгоритмів може суттєво відрізнятися залежно від обсягу даних.

Модуль практичного застосування алгоритмів забезпечує гнучкий інтерфейс для виконання операцій шифрування та стиснення згідно з вибором користувача або рекомендаціями системи. Архітектура цього модуля передбачає підтримку як прямих операцій (шифрування та архівування), так і зворотних (розшифрування та розархівування). Система може виконувати операції як окремо, так і в комбінації, при цьому користувач має можливість змінювати послідовність застосування алгоритмів за допомогою спеціального механізму перемикання. Інтерфейс модуля практичного застосування алгоритмів (рис. 3.3) забезпечує повний контроль над процесом обробки файлів через систему вибору типу операцій, випадаючі списки для конкретизації алгоритмів та індикатори послідовності виконання операцій.

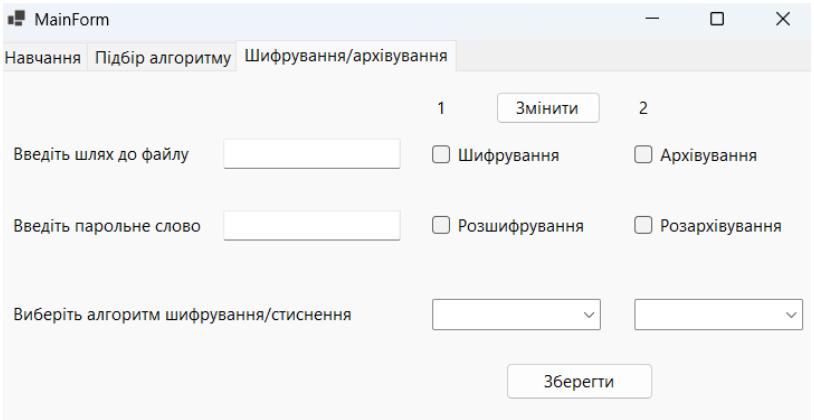

Рисунок 3.3 – Інтерфейс модуля застосування шифрування та стиснення

Важливим аспектом архітектури є система обробки помилок та валідації даних. Кожен модуль містить механізми перевірки коректності вхідних параметрів, існування файлів та директорій, а також прав доступу до файлової

системи. Система використовує метод HasWriteAccess для перевірки можливості запису у вказану директорію, що запобігає виникненню помилок під час роботи з файлами. Всі винятки обробляються централізовано з відображенням інформативних повідомлень користувачеві.

Архітектура інтерфейсу користувача побудована за принципом інтуїтивної навігації та зручності використання. Кожна вкладка містить мінімальний набір елементів управління, необхідних для виконання відповідних функцій. Перша вкладка включає поля для введення шляху до теки та пароля, а також кнопку запуску процесу навчання. Друга вкладка надає можливість вибору файлу для аналізу та критерію оптимізації через радіокнопки, результати відображаються у табличному вигляді з виділенням найкращої комбінації. Третя вкладка містить поля для введення шляху до файлу та пароля, чекбокси для вибору типу операцій, випадаючі списки для вибору конкретних алгоритмів та систему індикаторів послідовності виконання операцій.

Система зберігання даних організована у вигляді текстового файлу з чітко структурованим форматом записів. Кожен запис містить всю необхідну інформацію для подальшого аналізу: тип файлу, розмір, комбінацію алгоритмів, результати обробки та метрики продуктивності. Такий підхід забезпечує простоту реалізації, портативність системи та можливість ручного аналізу даних при необхідності.

Загальна архітектура системи забезпечує масштабованість та можливість подальшого розширення функціональності. Модульна структура дозволяє легко додавати нові алгоритми шифрування та стиснення без зміни основної логіки системи. Архітектура інтерфейсу забезпечує можливість інтеграції додаткових критеріїв оцінки та методів нечіткої логіки. Така гнучкість робить систему адаптованою до майбутніх потреб та технологічних змін у сфері криптографії та стиснення даних.

3.2 Програмна реалізація компонентів системи на основі методів нечіткої логіки

Центральним компонентом системи є механізм нечіткого логічного виведення, який аналізує накопичені під час навчання дані та надає рекомендації щодо оптимальних комбінацій алгоритмів для конкретних типів файлів.

Основою програмної реалізації є словник типів файлів з відповідними вагами критеріїв fileTypeWeights, який визначає важливість швидкості виконання та якості стиснення для різних форматів. Структура даних містить кортежі значень, де перший елемент відповідає вазі часу виконання, а другий – вазі коефіцієнта стиснення. Для текстових файлів з розширенням .txt встановлено ваги (0.3, 0.7), що означає більшу важливість стиснення порівняно зі швидкістю обробки. Для графічних форматів .jpg та .jpeg визначено зворотні пропорції (0.7, 0.3), оскільки зображення вже є стисненими.

Програмна реалізація включає систему збору навчальних даних через метод training_Click, який забезпечує всебічне тестування комбінацій алгоритмів. Система створює унікальні хеші файлів за допомогою методу ComputeFileHash для уникнення повторного аналізу одних і тих же файлів. Процес навчання включає перебір усіх можливих комбінацій алгоритмів шифрування та стиснення з урахуванням порядку їх застосування.

Для кожної комбінації система використовує об'єкт Stopwatch для точного вимірювання часу виконання операцій. Результати зберігаються у текстовому файлі results_database.txt у структурованому форматі, де кожен запис містить розширення файлу, початковий розмір, комбінацію алгоритмів, кінцевий розмір, коефіцієнт стиснення та час виконання, розділені символом вертикальної риски. Такий формат забезпечує легкість обробки даних та подальшого аналізу даних.

Центральним компонентом нечіткого логічного виведення є метод selection_Click, який реалізує процес аналізу та вибору оптимальних алгоритмів. Система спочатку зчитує параметри вхідного файлу та визначає його розмір для подальшої сегментації результатів навчання. Метод GroupResultsByFileSize

розділяє накопичені дані на три категорії: малі файли розміром менше ніж 1 МБ, середні файли від 1 до 10 МБ та великі файли понад 10 МБ.

Така класифікація дозволяє системі враховувати різну ефективність алгоритмів залежно від обсягу даних. Для забезпечення стійкості системи реалізовано метод GetClosestGroup, який знаходить найбільш релевантну групу навіть у випадку відсутності точної відповідності. Алгоритм надає перевагу сусіднім за розміром групам, що забезпечує більшу точність рекомендацій.

Система підтримує два режими оптимізації: мінімізацію часу виконання та максимізацію ефективності стиснення. Для обох оптимальними вважаються найменші значення, що відповідають кращому результату. Система автоматично сортує результати відповідно до обраного критерію та виділяє найкращий варіант зеленим кольором у табличному представленні.

Програмна реалізація включає розвинену систему візуалізації результатів через метод ShowResults, який створює динамічну форму з таблицею результатів. Система формує таблицю з налаштованими колонками для відображення типу файлу, комбінації алгоритмів, коефіцієнта стиснення та оцінки за обраним критерієм.

Особливою функцією є автоматичне виділення найкращого результату кольором та відображення підсумкової інформації у нижній панелі форми. Система адаптує відображення відповідно до обраного критерію оптимізації, показуючи або час виконання, або коефіцієнт стиснення як ключову метрику.

Програмна реалізація включає функціональну систему обробки файлів через метод save_Click, який забезпечує практичне застосування результатів аналізу. Система підтримує різні комбінації операцій: тільки шифрування, тільки стиснення, або комбінацію обох операцій у різній послідовності та з можливістю зворотного процесу.

Метод change_Click дозволяє користувачеві динамічно змінювати порядок операцій між шифрування-стиснення та стиснення-шифрування. Система зберігає оброблені файли з унікальними назвами, додаючи суфікс "_processed" для уникнення конфліктів імен.

Також програмний засіб включає всебічну систему підтвердження дійсності введених даних та обробки винятків. Метод HasWriteAccess перевіряє права доступу до директорій перед початком операцій, що запобігає виникненню помилок під час збереження файлів. Система перевіряє існування файлів, коректність паролів та доступність вибраних алгоритмів. Кожен критичний метод огорнуто в блоки try-catch з інформативними повідомленнями про помилки. Система автоматично створює необхідні файли бази даних та відновлює роботу після збоїв, забезпечуючи стабільність функціонування.

Архітектура системи забезпечує легке додавання нових алгоритмів шифрування та стиснення через списки encryptionAlgorithms та compressionAlgorithms. Модульна структура дозволяє інтегрувати додаткові критерії оцінки без зміни основної логіки системи. Система підтримує розширення словника fileTypeWeights для нових типів файлів, а також модифікацію алгоритмів групування за розміром файлів. Гнучка архітектура забезпечує адаптацію до майбутніх потреб користувачів та інтеграцію з іншими системами.

Програмна реалізація демонструє високу ефективність у автоматизації процесу вибору алгоритмів, значно скорочуючи час прийняття рішень користувачами. Система накопичує знання про поведінку різних алгоритмів на різних типах даних, створюючи базу для майбутніх рекомендацій. Інтегрована система візуалізації дозволяє користувачам розуміти логіку прийняття рішень та при необхідності коригувати параметри оцінки. Система забезпечує баланс між автоматизацією та контролем користувача, що робить її придатною для широкого спектра застосувань у сфері захисту та стиснення цифрової інформації.

3.3 Методика тестування та оцінки ефективності розробленої системи

Методика тестування розробленої системи автоматизованого вибору криптографічних алгоритмів базується на комплексному підході до оцінки функціональності, точності та надійності всіх компонентів програмного

забезпечення. Процес тестування організовано відповідно до архітектури системи та включає перевірку кожного модуля окремо, а також інтегрованого функціонування системи в цілому. Експериментальна база включала 100 файлів різних форматів загальним обсягом 0.9 ГБ, що забезпечило репрезентативність результатів для широкого спектра практичних застосувань.

Першим етапом тестування є перевірка модуля навчання системи, який відповідає за збір та структурування даних для подальшого аналізу методами нечіткої логіки. Тестування цього модуля розпочинається з підготовки репрезентативного набору тестових файлів різних форматів та розмірів. Тестовий набір включає текстові файли з розширенням .txt розміром від кількох кілобайтів до декількох мегабайтів, графічні файли форматів .jpg, .jpeg та .png різної роздільної здатності, документи .pdf та .docx з різним вмістом. Такий підхід забезпечує охоплення основних типів даних, які система повинна обробляти згідно з технічними вимогами.

Процедура тестування модуля навчання починається з введення шляху до підготовленої директорії з тестовими файлами та парольної фрази у відповідні поля інтерфейсу. Система повинна коректно обробити всі файли у директорії, застосувати кожну комбінацію алгоритмів шифрування та стиснення, а також зберегти результати у структурованому форматі бази даних.

Критично важливим аспектом тестування модуля навчання є перевірка цілісності оброблюваних файлів. Система повинна забезпечувати збереження оригінальних файлів без будь-яких змін, оскільки всі операції виконуються з копіями даних у пам'яті. Для верифікації цілісності використовується порівняння хеш-сум файлів до та після процесу навчання за допомогою алгоритму SHA-256. Ідентичність хеш-сум підтверджує, що оригінальні файли залишилися незмінними під час аналізу їх характеристик системою.

Результатом успішного функціонування модуля навчання є створення файлу results_database.txt, який містить структуровані записи про ефективність кожної комбінації алгоритмів для кожного типу файлів (рис. 3.4). Аналіз вмісту цього файлу дозволяє оцінити повноту збору даних та коректність вимірювання

параметрів продуктивності. Кожен запис повинен містити інформацію про розширення файлу, початковий розмір, застосовану комбінацію алгоритмів, висхідний розмір, коефіцієнт стиснення та час виконання операцій.

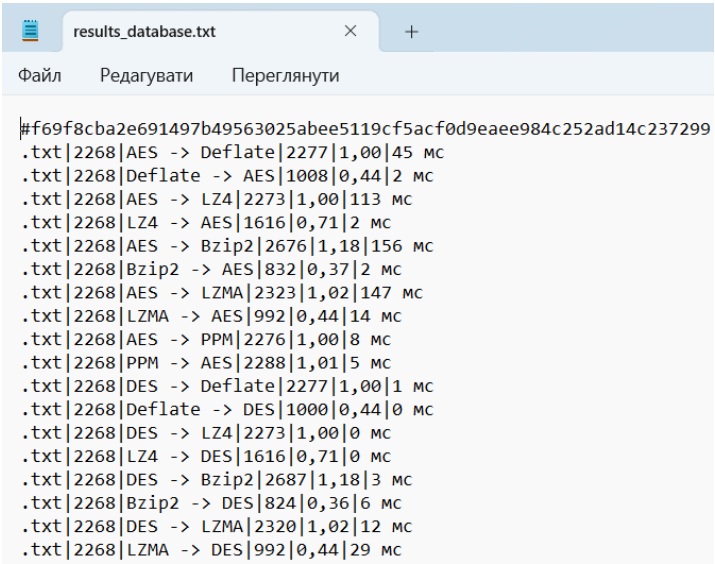

Рисунок 3.4 – Збереження результатів навчання у файлі

Другий етап тестування зосереджується на модулі інтелектуального підбору алгоритмів, який реалізує систему нечіткої логіки для прийняття рішень. Тестування цього модуля починається з підготовки контрольних файлів з відомими характеристиками та очікуваними результатами аналізу. Процедура включає вибір тестового файлу через інтерфейс системи та активацію процесу аналізу. Система повинна коректно визначити тип файлу за його розширенням, застосувати відповідні ваги критеріїв згідно з налаштуваннями для конкретного формату даних та надати рекомендації щодо оптимальної комбінації алгоритмів.

Ключовим індикатором якості роботи модуля підбору алгоритмів є відповідність отриманих рекомендацій теоретичним очікуванням базуючись на характеристиках різних типів файлів. Для текстових файлів система повинна надавати перевагу алгоритмам з високим коефіцієнтом стиснення, навіть якщо це призводить до дещо більшого часу обробки. Для графічних файлів, особливо у форматах JPEG, пріоритет має надаватися швидкості обробки, оскільки такі файли вже є стисненими й додаткове стиснення може бути неефективним.

Результати роботи модуля підбору відображаються у табличному форматі, де представлені всі проаналізовані комбінації алгоритмів з відповідними оцінками ефективності (рис. 3.5, 3.6). Найкраща комбінація виділяється окремо у нижній частині вікна результатів. Тестування включає перевірку коректності розрахунків оцінок ефективності, правильності сортування результатів та відповідності рекомендованої комбінації критеріям оптимізації, обраним користувачем. Як можна побачити частіше фігурують алгоритми Deflate та AES. А виграш шляхом стиснення складає в середньому 5% від вхідного розміру.

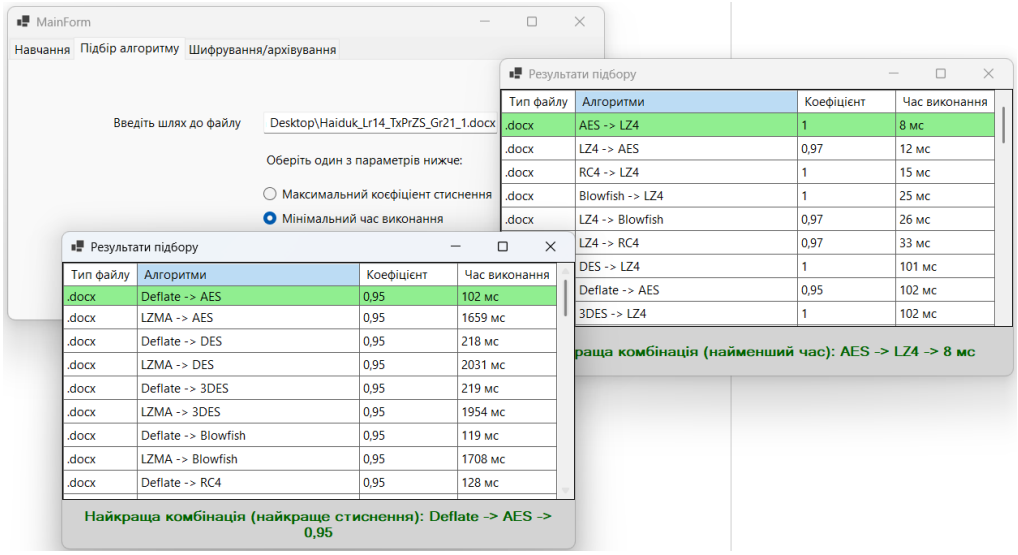

Рисунок 3.5 – Відображення результату підбору для текстового файлу .docx

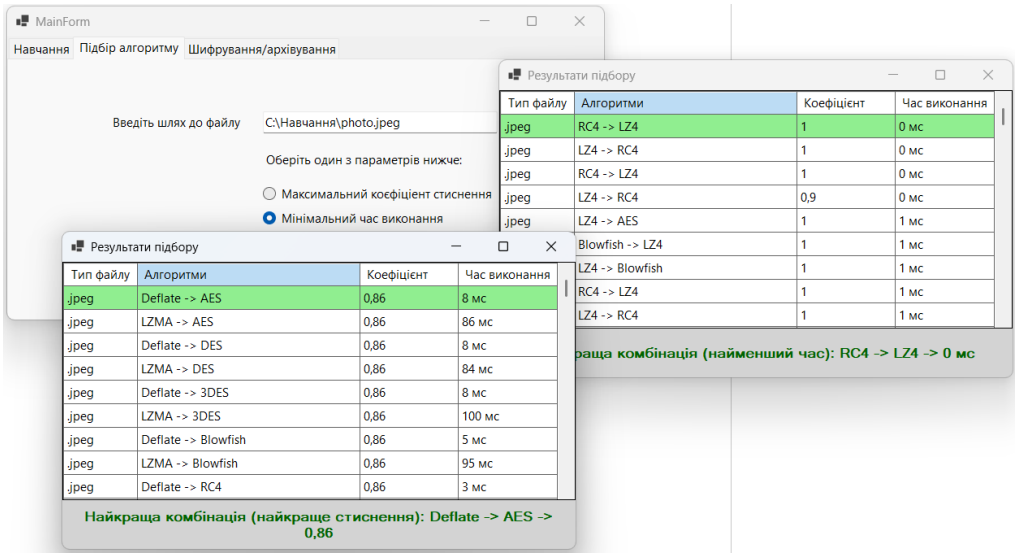

Рисунок 3.6 – Відображення результату підбору для зображення .jpeg

Особливу увагу під час тестування модуля підбору приділено перевірці роботи системи нечіткої логіки. Функції належності для часу виконання та коефіцієнта стиснення тестуються з використанням граничних значень та типових діапазонів параметрів. Система повинна коректно класифікувати час виконання менше ніж 10 мілісекунд як швидкий з максимальною оцінкою, час від 10 до 50 мілісекунд як середній з градуальним зниженням оцінки, та час понад 50 мілісекунд як повільний з мінімальною оцінкою.

Третій етап тестування охоплює модуль практичного застосування алгоритмів, який дозволяє користувачеві виконувати операції шифрування та стиснення згідно з отриманими рекомендаціями або власними налаштуваннями. Тестування цього модуля включає перевірку всіх можливих комбінацій операцій: окреме шифрування, окреме стиснення, послідовне застосування обох операцій у різному порядку, а також відповідні зворотні операції розшифрування та розархівування.

Процедура тестування модуля застосування починається з підготовки тестових файлів різних типів та розмірів, для яких відомі очікувані результати обробки. Інтерфейс модуля дозволяє гнучко налаштовувати параметри обробки через систему прапорців для вибору типу операцій та випадаючі списки для конкретизації алгоритмів. Критично важливим є тестування функціональності перемикання послідовності застосування алгоритмів, яка реалізована через кнопку зміни порядку операцій.

Найважливішим аспектом тестування модуля застосування є перевірка оборотності операцій та збереження цілісності даних. Для кожного тестового файлу виконується повний цикл обробки: застосування обраної комбінації алгоритмів шифрування та стиснення, а потім відповідних зворотних операцій. Результівний файл повинен бути ідентичним оригіналу, що підтверджується порівнянням хеш-сум або побайтовим порівнянням вмісту файлів.

Тестування оборотності операцій включає перевірку обох можливих послідовностей застосування алгоритмів. Якщо спочатку застосовано шифрування, а потім стиснення, то зворотні операції повинні виконуватися у

відповідному порядку: спочатку розархівування, потім розшифрування. Система повинна автоматично визначати правильну послідовність зворотних операцій базуючись на поточних налаштуваннях інтерфейсу.

Оцінка точності рекомендацій системи нечіткої логіки здійснюється через порівняння автоматично обраних комбінацій алгоритмів з результатами експертного аналізу або еталонними рішеннями для конкретних типів файлів. Високий ступінь відповідності отриманий на практиці свідчить про ефективність застосовуваних методів нечіткої логіки та коректність налаштування функцій належності й системи ваг критеріїв.

Завершальним етапом тестування є перевірка стабільності роботи системи під час тривалих сесій використання та обробки великої кількості файлів. Система повинна демонструвати стійкість до накопичення помилок, ефективне управління пам'яттю та збереження продуктивності навіть після обробки сотень файлів різних типів і розмірів.

3.4 Аналіз результатів експериментальних досліджень та шляхи подальшого вдосконалення системи

Експериментальні дослідження розробленої системи автоматизованого вибору алгоритмів шифрування та стиснення виявили ряд значущих результатів, які дозволяють оцінити ефективність застосовуваних методів нечіткої логіки та визначити перспективи подальшого розвитку. Аналіз отриманих даних засвідчив, що система демонструє задовільну функціональність у рамках поставлених завдань, проте має певні обмеження, які потребують ретельного розгляду з погляду практичного застосування в реальних умовах.

Результати тестування модуля навчання системи показали, що процес збору та структурування даних відбувається правильно. Система успішно генерує базу даних результатів з повною інформацією про ефективність різних комбінацій алгоритмів. Однак детальний аналіз накопичених даних виявив значну варіативність показників ефективності залежно від конкретного вмісту

файлів, а не лише їх формату. Це свідчить про те, що класифікація за розширенням файлу є недостатньо точною для забезпечення оптимального вибору алгоритмів. Наприклад, текстові файли з різним ступенем ентропії демонструють кардинально відмінні коефіцієнти стиснення, що не враховується поточною системою класифікації.

Аналіз роботи модуля інтелектуального підбору алгоритмів засвідчив ефективність застосовуваних методів нечіткої логіки для багатокритеріального прийняття рішень. Система здатна надавати обґрунтовані рекомендації, які у більшості випадків відповідають теоретичним очікуванням щодо оптимальності вибору. Проте експериментальні дослідження виявили обмеженість використовуваних функцій належності, які базуються на фіксованих граничних значеннях без урахування динамічних характеристик системи. Трапецієподібні функції належності для часу виконання з граничними значеннями 10 та 50 мілісекунд виявилися занадто статичними для різних апаратних конфігурацій та розмірів файлів.

Критичним аспектом експериментальних досліджень стало виявлення проблем з безпекою та практичністю застосування системи в реальних умовах. Використання кодового слова для шифрування створює додаткові ризики, пов'язані з необхідністю передачі та зберігання паролів захищеними каналами зв'язку. В умовах корпоративного або державного використання такий підхід створює значні уразливості в системі захисту інформації. Експериментальне тестування показало, що система не включає механізмів управління ключами, що є критичним недоліком для практичного застосування.

Економічний аналіз розробленої системи засвідчив її високу вартість впровадження у великих організаціях. Необхідність проведення етапу навчання для кожного типу даних та специфічних умов використання вимагає значних обчислювальних ресурсів та часу фахівців. Експериментальні розрахунки показали, що для повноцінного навчання системи на репрезентативному наборі корпоративних даних може знадобитися кілька днів безперервної роботи

потужних обчислювальних систем. Такі витрати можуть не виправдовуватися для організацій з обмеженими IT-бюджетами.

Дослідження застарілості алгоритмів, включених до системи, виявило серйозні обмеження з погляду сучасних стандартів кібербезпеки. Алгоритми DES та RC4, які використовуються в системі для демонстрації різноманітності підходів, вважаються небезпечними за сучасними стандартами та не рекомендуються для використання в продуктивних системах. Навіть Triple DES поступово виводиться з експлуатації через обмежену довжину ключа та вразливості до певних типів атак. Експериментальне тестування підтвердило, що ці алгоритми демонструють високу швидкість роботи, але їх використання створює неприйнятні ризики безпеки.

Аналіз поведінки системи на великих файлах виявив проблеми масштабованості та ефективності використання пам'яті. Експериментальні дослідження показали, що обробка файлів розміром понад 100 МБ призводить до значного уповільнення роботи системи та збільшення споживання оперативної пам'яті. Це пов'язано з тим, що поточна реалізація завантажує весь файл у пам'ять перед обробкою, що є неефективним для великих обсягів даних.

Перспективи подальшого вдосконалення системи включають кілька ключових напрямків розвитку. Найбільш критичним є модернізація криптографічної основи системи з переходом на сучасні стандарти шифрування. Доцільно зосередитися на алгоритмах AES з різними режимами роботи, а також включити підтримку асиметричного шифрування для розв'язання проблем управління ключами. Перспективним напрямком є інтеграція квантово-стійких алгоритмів шифрування з огляду на майбутні загрози від квантових обчислень.

Значного покращення потребує система класифікації файлів з переходом від простої категоризації за розширенням до аналізу змісту файлів. Доцільно реалізувати механізми аналізу ентропії даних, структурних характеристик файлів та статистичних властивостей вмісту. Такий підхід дозволить системі більш точно передбачати ефективність різних алгоритмів стиснення та приймати більш обґрунтовані рішення.

Модернізація системи нечіткої логіки має включати адаптивні функції належності, які автоматично налаштовуються залежно від характеристик конкретної обчислювальної системи та типів оброблюваних даних. Перспективним є застосування методів машинного навчання для автоматичного налаштування параметрів нечітких множин на основі накопичуваних даних про ефективність роботи системи.

Критично важливим напрямком розвитку є розробка системи управління ключами з підтримкою сучасних стандартів криптографії. Система повинна включати механізми генерації, розподілу, зберігання та відкликання криптографічних ключів з дотриманням принципів мінімальних привілеїв та розподіленого довір'я. Доцільно розглянути інтеграцію з наявними системами управління ключами та інфраструктурою відкритих ключів.

Архітектурні покращення системи мають включати перехід до потокової обробки даних замість завантаження повних файлів у пам'ять. Це дозволить системі ефективно працювати з файлами будь-якого розміру без обмежень на обсяг доступної оперативної пам'яті. Перспективним є також розпаралелювання обчислень для використання переваг багатоядерних процесорів та розподілених обчислювальних систем.

Розвиток системи має включати створення модульної архітектури з можливістю легкого додавання нових алгоритмів шифрування та стиснення без модифікації основного коду. Доцільно реалізувати систему плагінів, яка дозволить розширювати функціональність системи третіми сторонами та адаптувати її до специфічних потреб різних галузей застосування.

Подальші дослідження мають зосередитися на розробці методів оцінки якості рекомендацій системи та автоматичного покращення точності прогнозування через механізми зворотного зв'язку. Перспективним є створення системи моніторингу ефективності прийнятих рішень з автоматичним коригуванням параметрів нечіткої логіки на основі реальних результатів використання.

Розроблена система автоматизованого вибору алгоритмів шифрування та стиснення демонструє потенціал застосування методів нечіткої логіки для розв'язання задач багатокритеріальної оптимізації у сфері захисту інформації. Проте її практичне застосування потребує суттєвої модернізації з урахуванням сучасних вимог безпеки, масштабованості та економічної ефективності. Визначені напрямки подальшого розвитку створюють основу для створення промислової системи, здатної ефективно функціонувати в реальних умовах корпоративного та державного використання.

Висновки за розділом 3

У третьому розділі представлено комплексне дослідження розробки та впровадження розробленої системи автоматизованого вибору на основі методів нечіткої логіки. Створена архітектура демонструє ефективний підхід до інтеграції теоретичних принципів нечіткої логіки з практичними потребами обробки різнотипних даних.

Архітектурне рішення, побудоване за принципом модульної структури з використанням трьох вкладок в інтерфейсі користувача, що забезпечує логічну послідовність процесів від навчання системи до практичного застосування отриманих знань. Центральна роль головної форми як координатора всіх процесів системи дозволяє ефективно управляти взаємодією між різними компонентами та забезпечує масштабованість рішення. Реалізація алгоритмів через інтерфейси IEncryptionAlgorithm та ICompressionAlgorithm створює гнучку основу для подальшого розширення функціональності без зміни базової архітектури.

Словник типів файлів з відповідними вагами критеріїв забезпечує адаптивний підхід до оцінки ефективності алгоритмів залежно від специфіки конкретних форматів даних. Механізм поділу даних за розміром файлів дозволяє системі враховувати різну ефективність алгоритмів для різних обсягів інформації, що підвищує точність рекомендацій.

Результати експериментальних досліджень підтвердили функціональну спроможність розробленої системи та виявили ключові аспекти для подальшого вдосконалення. Методика тестування забезпечила комплексну оцінку всіх компонентів системи, включаючи перевірку цілісності даних, оборотності операцій та стабільності роботи під навантаженням. Особливо важливим результатом стало підтвердження збереження оригінальних файлів під час процесу навчання та коректності виконання зворотних операцій розшифрування та розархівування.

Аналіз отриманих результатів виявив значущі обмеження поточної реалізації, які потребують врахування при практичному застосуванні системи. Класифікація файлів виключно за розширенням показала недостатню точність для забезпечення оптимального вибору алгоритмів, оскільки файли одного формату можуть мати кардинально різні характеристики залежно від вмісту. Використання застарілих алгоритмів шифрування створює серйозні ризики безпеки, а відсутність системи управління ключами обмежує можливості практичного застосування в корпоративному середовищі.

Визначені перспективи подальшого розвитку системи включають модернізацію криптографічної основи з переходом на сучасні стандарти шифрування, розробку системи аналізу змісту файлів замість простої категоризації за розширенням, створення адаптивних функцій належності та інтеграцію механізмів управління ключами. Архітектурні покращення мають зосередитися на потоковій обробці даних для підвищення масштабованості та створенні модульної системи плагінів для легкого розширення функціональності.

Розроблена система створює надійну основу для подальшого розвитку інтелектуальних систем захисту інформації, здатних адаптуватися до специфічних вимог різних галузей застосування та забезпечувати ефективну обробку великих обсягів різнотипних даних.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи була розроблена система автоматизованого вибору алгоритмів шифрування та стиснення для різних типів цифрових даних з використанням методів нечіткої логіки. Аналіз структурних властивостей текстових файлів, документів формату DOCX і PDF, а також растрових зображень у форматах JPEG і PNG, дозволив визначити ключові параметри, які впливають на вибір оптимальних методів захисту та обробки інформації. До них входять: розмір файлу, структура даних, вимоги до швидкодії та цілісності, а також індивідуальні особливості алгоритмів стиснення та шифрування. Порівняння методів показав, що ефективність алгоритмів залежить від типу даних і контексту їх використання, що обґрунтовує необхідність застосування адаптивного підходу.

Застосування методів нечіткої логіки стало теоретичною та практичною основою для побудови інтелектуальної системи. Реалізована система включає компоненти навчання, інтелектуального підбору та практичного застосування алгоритмів, демонструючи свою гнучкість і адаптивність. Методика тестування підтвердила функціональність системи, ефективність нечіткого логічного виведення, а також можливість збереження цілісності оброблюваних даних.

Водночас експериментальні дослідження виявили низку обмежень, серед яких недосконаліть класифікації файлів, використання алгоритмів шифрування без системи управління ключами, а також проблеми масштабованості при обробці великих файлів. Порушення цілісності електронних цифрових підписів під час стиснення підкреслює необхідність ретельного підходу до вибору місця застосування методів на практиці. З економічної точки зору необхідні значні ресурси для навчання системи, що потребує оптимізації.

Подальші напрями вдосконалення системи включають модернізацію криптографічної бази, розвиток методів класифікації за файлів, адаптивне налаштування функцій належності нечіткої логіки, розробку комплексної

системи управління ключами, а також архітектурні покращення для потокової обробки та масштабованості. Впровадження методів машинного навчання для автоматичного налаштування параметрів також розглядаються як перспективні напрямки.

Отже, виконана робота забезпечила теоретичну базу і практичний інструментарій для розробки адаптивних систем захисту інформації, що враховують особливості різних типів цифрових даних. Попри виявлені обмеження, розроблена система має значний потенціал для подальшого розвитку і впровадження в реальних умовах, сприяючи підвищенню ефективності та безпеки обробки інформації в сучасних цифрових системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is ASCII (American Standard Code for Information Interchange) [Електронний ресурс]. URL: https://www.techtarget.com/whatis/definition/ASCII-American-Standard-Code-for-Information-Interchange
2. Character encoding: Types, UTF-8, Unicode, and more explained [Електронний ресурс]. URL: https://lokalise.com/blog/what-is-character-encoding-exploring-unicode-utf-8-ascii-and-more/
3. Open XML Formats and file name extensions [Електронний ресурс]. URL: https://support.microsoft.com/en-us/office/open-xml-formats-and-file-name-extensions-5200d93c-3449-4380-8e11-31ef14555b18
4. Інформація про файл DOCX - Що таке формат файлу DOCX [Електронний ресурс]. URL: https://kb.fileformat.app/uk/extension/docx-file-info/
5. JPEG (Joint Photographic Experts Group): Print-Ready Files Explained [Електронний ресурс]. URL: https://www.mondaymerch.com/resources/print-ready-files/jpeg-(joint-photographic-experts-group)-print-ready-files-explained
6. What is Chroma Subsampling in JPEG [Електронний ресурс]. URL: https://www.educative.io/blog/chroma-subsampling
7. Implementation of JPEG XS entropy encoding and decoding on FPGA [Електронний ресурс]. URL: https://link.springer.com/article/10.1007/s11554-023-01410-8
8. Deflate/Inflate Compression [Електронний ресурс]. URL: http://www.libpng.org/pub/png/spec/1.2/PNG-Compression.html
9. JPEG і PNG [Електронний ресурс]. URL: https://www.adobe.com/ua/creativecloud/file-types/image/comparison/jpeg-vs-png.html
10. The Structure of a PDF File [Електронний ресурс]. URL: https://medium.com/@jberkenbilt/the-structure-of-a-pdf-file-6f08114a58f6

11. Digital Signatures in a PDF [Електронний ресурс]. URL: https://www.adobe.com/devnet-docs/acrobatetk/tools/DigSig/Acrobat_DigitalSignatures_in_PDF.pdf
12. Bharat Sinha. Comparison of PNG & JPEG Format for LSB Steganography [Електронний ресурс] / Bharat Sinha // International Journal of Science and Research (IJSR). – 2015. – Т. 4, № 4. – С. 198–201. URL: https://www.ijsr.net/archive/v4i4/29031501.pdf
13. What is File Size & How Does It Affect Data Transfer Speed [Електронний ресурс]. URL: https://www.lenovo.com/us/en/glossary/what-is-file-size/?orgRef=https%253A%252F%252Fwww.google.com%252F
14. What is image compression [Електронний ресурс]. URL: https://www.cloudflare.com/learning/performance/glossary/what-is-image-compression/
15. Why using JavaScript in PDF files is a security risk [Електронний ресурс]. URL: https://www.locklizard.com/document-security-blog/javascript-security-pdf/
16. Cryptography and its Types [Електронний ресурс]. URL: https://www.geeksforgeeks.org/cryptography-and-its-types/
17. Difference between Block Cipher and Stream Cipher [Електронний ресурс]. URL: https://www.geeksforgeeks.org/difference-between-block-cipher-and-stream-cipher/
18. Advanced Encryption Standard (AES) [Електронний ресурс]. URL: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/
19. Яка різниця між AES і AES-NI [Електронний ресурс]. URL: https://kort.aqua-svit.com.ua/yaka-riznicya-mizh-aes-i-aes-ni/
20. Information Security and Privacy [Електронний ресурс] / ред.: E. Foo, D. Stebila. – Cham : Springer International Publishing, 2015. URL: https://doi.org/10.1007/978-3-319-19962-7
21. AES vs DES Encryption: Why Advanced Encryption Standard (AES) has replaced DES, 3DES and TDEA [Електронний ресурс]. URL: https://www.precisely.com/blog/data-security/aes-vs-des-encryption-standard-3des-tdea

22. Data Encryption Standard (DES) | Set 1 [Електронний ресурс]. URL: https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/
23. Smid M. E. Development of the Advanced Encryption Standard [Електронний ресурс] / Miles E. Smid // Journal of Research of the National Institute of Standards and Technology. – 2021. – Т. 126. URL: https://doi.org/10.6028/jres.126.024
24. Про стандарт Triple DES (3DES) [Електронний ресурс]. URL: https://rubydevelopers.org/t/triple-des-3des/424
25. Blowfish Algorithm with Examples [Електронний ресурс]. URL: https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/
26. What is RC4 Encryption [Електронний ресурс]. URL: https://www.geeksforgeeks.org/what-is-rc4-encryption/
27. RSA Algorithm in Cryptography: Rivest Shamir Adleman Explained [Електронний ресурс]. URL: https://www.splunk.com/en_us/blog/learn/rsa-algorithm-cryptography.html#:~:text=How%20does%20RSA%20work%3F,a%20substantial%20difference%20between%20them
28. Blockchain - Elliptic Curve Cryptography [Електронний ресурс]. URL: https://www.geeksforgeeks.org/blockchain-elliptic-curve-cryptography/
29. Block Cipher modes of Operation [Електронний ресурс]. URL: https://www.geeksforgeeks.org/block-cipher-modes-of-operation/
30. Understanding Compression Methods: From Arithmetic Coding to Zstandard [Електронний ресурс]. URL: https://itnext.io/understanding-compression-methods-df970a9133c9
31. Data Compression: LZ77 vs. LZ4 vs. LZ4HC [Електронний ресурс]. URL: https://www.baeldung.com/cs/lz77-lz4-lz4hc-algorithms-difference

ДОДАТОК А

Код головної форми

```
// Ваги критеріїв для різних типів файлів: (вагаЧасу, вагаСтиснення)
private Dictionary<string, (double timeWeight, double compressionWeight)> fileTypeWeights = new Dictionary<string, (double, double)>
{
    { ".txt", (0.3, 0.7) },      // Текст — головне стиснення
    { ".jpg", (0.7, 0.3) },      // Зображення — головне швидкість
    { ".jpeg", (0.7, 0.3) },
    { ".pdf", (0.5, 0.5) },      // Баланс
    { ".png", (0.6, 0.4) },      // Трохи більше часу
    { ".docx", (0.4, 0.6) }      // Змішаний, але важливе стиснення
};
public Form1()
{
    InitializeComponent();
}
// Ініціалізація списків доступних алгоритмів
private readonly List<IEncryptionAlgorithm> encryptionAlgorithms = new List<IEncryptionAlgorithm>
{
    new AESEncryption(),
    new DESEncryption(),
    new TripleDESEncryption(),
    new BlowfishEncryption(),
    new RC4Encryption()
};
private readonly List<ICompressionAlgorithm> compressionAlgorithms = new List<ICompressionAlgorithm>
{
    new DeflateCompression(),
    new LZ4Compression(),
    new Bzip2Compression(),
    new LZMACompression(),
    new PPMCompression()
};
// Метод для перевірки доступу до директорії для запису
private bool HasWriteAccess(string directoryPath)
{
    try
    {
        string testFilePath = Path.Combine(directoryPath, "test.txt");
        // Спробуємо створити і записати тестовий файл
        File.WriteAllText(testFilePath, "test");
        File.Delete(testFilePath);
        return true;
    }
```

```
    catch
    {
        return false;
    }
}
// Метод для обчислення унікального хешу файлу
private string ComputeFileHash(string filePath)
{
    using (var sha256 = System.Security.Cryptography.SHA256.Create())
    {
        using (var stream = File.OpenRead(filePath))
        {
            var hash = sha256.ComputeHash(stream);
            return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();
        }
    }
}
// Обробка кнопки "Навчити"
private void training_Click(object sender, EventArgs e)
{
    // Отримуємо шлях до папки та пароль від користувача
    string folderPath = PathToTheFolder.Text; // Текстбокс для шляху до папки
    string password = PasswordWord1.Text;  // Текстбокс для введення пароля
    // Перевіряємо, чи існує вказана папка
    if (!Directory.Exists(folderPath))
    {
        MessageBox.Show("Папку не знайдено. Введіть коректний шлях.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    // Вказуємо шлях до файлу бази даних для запису результатів навчання
    string trainingFilePath = Path.Combine(Directory.GetCurrentDirectory(), "results_database.txt");
    try
    {
        // Перевірка доступу до директорії
        string directoryPath = Path.GetDirectoryName(trainingFilePath);
        if (!Directory.Exists(directoryPath) || !HasWriteAccess(directoryPath))
        {
            MessageBox.Show("Немає прав на запис у директорію. Перевірте права доступу.",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        // Перевірка чи існує файл і створення, якщо не існує
        if (!File.Exists(trainingFilePath))
        {
            File.Create(trainingFilePath).Close(); // Створення і закриття файлу
        }
        // Аналізуємо всі файли в папці
        foreach (var filePath in Directory.GetFiles(folderPath))
        {
            // Обчислюємо унікальний хеш для файлу
            string fileHash = ComputeFileHash(filePath);
```

```
        // Перевіряємо, чи файл вже є в базі даних
        if (File.Exists(trainingFilePath) && File.ReadAllText(trainingFilePath).Contains(fileHash))
        {
            continue; // Пропускаємо обробку, якщо файл уже проаналізовано
        }
        // Збираємо базову інформацію про файл
        var fileInfo = new FileInfo(filePath);
        string fileExtension = fileInfo.Extension; // Розширення файлу
        long fileSizeBefore = fileInfo.Length;      // Розмір до обробки
        // Додаємо хеш файлу у файл бази даних
        File.AppendAllText(trainingFilePath, $"#{fileHash}{Environment.NewLine}");
        // Перебираємо всі комбінації алгоритмів шифрування і стиснення
        foreach (var encryption in encryptionAlgorithms)
        {
            foreach (var compression in compressionAlgorithms)
            {
                // Зчитуємо вміст файлу
                byte[] fileData = File.ReadAllBytes(filePath);
                // Ініціалізація stopwatch
                var stopwatch = System.Diagnostics.Stopwatch.StartNew();
                // Комбінація: спочатку шифрування, потім стиснення
                byte[] encryptedData = encryption.Encrypt(fileData, password);
                byte[] compressedData = compression.Compress(encryptedData);
                stopwatch.Stop(); // Зупинка таймера
                // Обчислюємо результати
                long fileSizeAfter = compressedData.Length;
                double compressionRatio = (double)fileSizeAfter / fileSizeBefore;
                long processingTime = stopwatch.ElapsedMilliseconds; // Отримуємо час у
мілісекундах
                // Записуємо результати у файл бази даних
                string                              resultEncryptThenCompress                              =
$"{fileExtension}|{fileSizeBefore}|{encryption.Name}                                                    ->
{compression.Name}|{fileSizeAfter}|{compressionRatio:F2}|{processingTime} мс";
                File.AppendAllText(trainingFilePath,              resultEncryptThenCompress              +
Environment.NewLine);
                // Комбінація: спочатку стиснення, потім шифрування
                stopwatch.Restart(); // Перезапускаємо таймер
                byte[] compressedDataFirst = compression.Compress(fileData);
                byte[] encryptedDataAfterCompression = encryption.Encrypt(compressedDataFirst,
password);
                stopwatch.Stop(); // Зупинка таймера
                // Обчислюємо результати
                fileSizeAfter = encryptedDataAfterCompression.Length;
                compressionRatio = (double)fileSizeAfter / fileSizeBefore;
                processingTime = stopwatch.ElapsedMilliseconds; // Отримуємо час у мілісекундах
                // Записуємо результати у файл бази даних
                string                              resultCompressThenEncrypt                              =
$"{fileExtension}|{fileSizeBefore}|{compression.Name}                                                   ->
{encryption.Name}|{fileSizeAfter}|{compressionRatio:F2}|{processingTime} мс";
                File.AppendAllText(trainingFilePath,              resultCompressThenEncrypt              +
Environment.NewLine);
            }
```

```
            }
        }
        // Повідомляємо користувача про завершення процесу навчання
        MessageBox.Show("Навчання завершено!", "Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (UnauthorizedAccessException)
    {
        MessageBox.Show("Немає прав для створення або запису в файл. Перевірте права доступу.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка при навчанні: {ex.Message}", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
// Обробка кнопки "Підбір алгоритму"
private void selection_Click(object sender, EventArgs e)
{
    string filePath = FilePath1.Text;  // Зчитуємо шлях до файлу
    // Перевіряємо, чи існує файл
    if (!File.Exists(filePath))
    {
        MessageBox.Show("Файл не знайдено. Введіть коректний шлях.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    // Отримуємо інформацію про файл
    FileInfo fileInfo = new FileInfo(filePath);
    long selectedFileSize = fileInfo.Length;
    string selectedFileExtension = fileInfo.Extension.ToLower();
    // Читаємо базу даних результатів навчання
    string trainingFilePath = Path.Combine(Directory.GetCurrentDirectory(), "results_database.txt");
    if (!File.Exists(trainingFilePath))
    {
        MessageBox.Show("База даних результатів навчання не знайдена. Спочатку проведіть навчання.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    string[] databaseLines = File.ReadAllLines(trainingFilePath);
    // Перевірка вибору критерію
    bool isTimeBased = MinimumExecutionTime.Checked;
    bool isCompressionRatioBased = MaximumCompressionRatio.Checked;
    if (!isTimeBased && !isCompressionRatioBased)
    {
        MessageBox.Show("Оберіть критерій оптимізації (час або коефіцієнт стиснення).", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    // Фільтруємо результати за типом файлу
    var relevantResults = GetResultsForFileType(databaseLines, selectedFileExtension);
```

```
    if (relevantResults.Count == 0)
    {
        MessageBox.Show($"Немає даних навчання для файлів типу {selectedFileExtension}. Спочатку проведіть навчання на файлах цього типу.", "Інформація", MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }
    // Групуємо результати за розміром файлів
    var groupedResults = GroupResultsByFileSize(relevantResults);
    // Знаходимо найбільш підходящу групу
    var relevantGroup = GetRelevantGroup(groupedResults, selectedFileSize);
    if (relevantGroup.Count == 0)
    {
        // Якщо немає точної відповідності, беремо найближчу групу
        relevantGroup = GetClosestGroup(groupedResults, selectedFileSize);
    }
    if (relevantGroup.Count == 0)
    {
        MessageBox.Show("Немає підходящих даних для аналізу.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    // Збираємо дані для таблиці
    List<AlgorithmResult> tableData = new List<AlgorithmResult>();
    // Перебираємо всі записи у вибраній групі результатів
    foreach (var line in relevantGroup)
    {
        var parts = line.Split('|');
        if (parts.Length < 6) continue;
        string fileExtension = parts[0];
        string algorithmCombination = parts[2];
        long processingTime = long.Parse(parts[5].Replace(" мс", ""));
        double compressionRatio = double.Parse(parts[4]);
        // Додаємо до таблиці
        tableData.Add(new AlgorithmResult
        {
            FileExtension = fileExtension,
            AlgorithmCombination = algorithmCombination,
            ProcessingTime = processingTime,
            CompressionRatio = compressionRatio,
            Score = isTimeBased ? processingTime : compressionRatio
        });
    }
    if (tableData.Count == 0)
    {
        MessageBox.Show("Немає даних для аналізу.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    // Знаходимо найкращий результат
    AlgorithmResult bestResult;
    if (isTimeBased)
```

```
    {
        // Для часу: найменший час
        bestResult = tableData.OrderBy(x => x.ProcessingTime).First();
    }
    else
    {
        // Для коефіцієнта стиснення: найменший коефіцієнт (краще стиснення)
        bestResult = tableData.OrderBy(x => x.CompressionRatio).First();
    }
    // Сортуємо результати за обраним критерієм
    if (isTimeBased)
    {
        tableData = tableData.OrderBy(x => x.ProcessingTime).ToList();
    }
    else
    {
        tableData = tableData.OrderBy(x => x.CompressionRatio).ToList();
    }
    // Створюємо форму для відображення результатів
    ShowResults(tableData, bestResult, isTimeBased);
}
// Допоміжний клас для результатів
public class AlgorithmResult
{
    public string FileExtension { get; set; }
    public string AlgorithmCombination { get; set; }
    public long ProcessingTime { get; set; }
    public double CompressionRatio { get; set; }
    public double Score { get; set; }
}
// Метод для отримання результатів для конкретного типу файлу
private List<string> GetResultsForFileType(string[] databaseLines, string fileExtension)
{
    var results = new List<string>();
    foreach (var line in databaseLines)
    {
        if (line.StartsWith("#")) continue;
        var parts = line.Split('|');
        if (parts.Length >= 6 && parts[0].ToLower() == fileExtension)
        {
            results.Add(line);
        }
    }
    return results;
}
// Виправлена функція для групування результатів за розміром файлів
private Dictionary<string, List<string>> GroupResultsByFileSize(List<string> results)
{
    var groupedResults = new Dictionary<string, List<string>>();
    foreach (var line in results)
    {
        var parts = line.Split('|');
```

```
        if (parts.Length < 6) continue;
        long fileSize = long.Parse(parts[1]);
        // Розподіл за розмірами: < 1MB, 1MB-10MB, > 10MB
        string group;
        if (fileSize < 1048576) // < 1MB
            group = "Small";
        else if (fileSize < 10485760) // 1MB - 10MB
            group = "Medium";
        else // > 10MB
            group = "Large";
        if (!groupedResults.ContainsKey(group))
        {
            groupedResults[group] = new List<string>();
        }
        groupedResults[group].Add(line);
    }
    return groupedResults;
}
// Виправлена функція для вибору відповідної групи
private List<string> GetRelevantGroup(Dictionary<string, List<string>> groupedResults, long fileSize)
{
    string targetGroup;
    if (fileSize < 1048576)
        targetGroup = "Small";
    else if (fileSize < 10485760)
        targetGroup = "Medium";
    else
        targetGroup = "Large";
    return groupedResults.ContainsKey(targetGroup) ? groupedResults[targetGroup] : new List<string>();
}
// Функція для отримання найближчої групи, якщо точної немає
private List<string> GetClosestGroup(Dictionary<string, List<string>> groupedResults, long fileSize)
{
    // Пріоритет: спочатку сусідні групи, потім будь-які доступні
    if (fileSize < 1048576) // Шукаємо Small
    {
        if (groupedResults.ContainsKey("Medium"))
            return groupedResults["Medium"];
        if (groupedResults.ContainsKey("Large"))
            return groupedResults["Large"];
    }
    else if (fileSize < 10485760) // Шукаємо Medium
    {
        if (groupedResults.ContainsKey("Small"))
            return groupedResults["Small"];
        if (groupedResults.ContainsKey("Large"))
            return groupedResults["Large"];
    }
    else // Шукаємо Large
```

```
    {
        if (groupedResults.ContainsKey("Medium"))
            return groupedResults["Medium"];
        if (groupedResults.ContainsKey("Small"))
            return groupedResults["Small"];
    }
    // Якщо нічого не знайдено, повертаємо першу доступну групу
    return groupedResults.Values.FirstOrDefault() ?? new List<string>();
}
// Метод для відображення результатів
private void ShowResults(List<AlgorithmResult> tableData, AlgorithmResult bestResult, bool isTimeBased)
{
    // Створення форми результатів
    Form resultForm = new Form();
    resultForm.Text = "Результати підбору";
    resultForm.Size = new Size(650, 400);
    resultForm.StartPosition = FormStartPosition.CenterParent;
    resultForm.MinimumSize = new Size(600, 350);
    // Таблиця результатів
    DataGridView resultsTable = new DataGridView();
    resultsTable.Dock = DockStyle.Fill;
    resultsTable.ReadOnly = true;
    resultsTable.AllowUserToAddRows = false;
    resultsTable.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
    resultsTable.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
    resultsTable.ScrollBars = ScrollBars.Both;
    resultsTable.RowHeadersVisible = false;
    // Налаштування колонок залежно від обраного критерію
    if (isTimeBased)
    {
        // Для критерію часу: показуємо стиснення та час окремо
        resultsTable.DataSource = tableData.Select(row => new {
            FileExtension = row.FileExtension,
            Algorithms = row.AlgorithmCombination,
            Compression = Math.Round(row.CompressionRatio, 2),
            ProcessingTime = row.ProcessingTime + " мс"
        }).ToList();
    }
    else
    {
        // Для критерію стиснення: показуємо тільки стиснення та час
        resultsTable.DataSource = tableData.Select(row => new {
            FileExtension = row.FileExtension,
            Algorithms = row.AlgorithmCombination,
            Compression = Math.Round(row.CompressionRatio, 2),
            ProcessingTime = row.ProcessingTime + " мс"
        }).ToList();
    }
    // Встановлюємо заголовки колонок
    resultsTable.DataBindingComplete += (s, e) => {
        if (resultsTable.Columns.Count >= 4)
```

```
        {
            resultsTable.Columns[0].HeaderText = "Тип файлу";
            resultsTable.Columns[1].HeaderText = "Алгоритми";
            resultsTable.Columns[2].HeaderText = "Коефіцієнт стиснення";
            resultsTable.Columns[3].HeaderText = "Час виконання";
            // Встановлюємо ширину колонок
            resultsTable.Columns[0].FillWeight = 15;
            resultsTable.Columns[1].FillWeight = 45;
            resultsTable.Columns[2].FillWeight = 20;
            resultsTable.Columns[3].FillWeight = 20;
            // Виділяємо найкращий результат (перший рядок після сортування)
            if (resultsTable.Rows.Count > 0)
            {
                resultsTable.Rows[0].DefaultCellStyle.BackColor = Color.LightGreen;
            }
        }
    };
    // Панель для найкращої комбінації
    Panel bottomPanel = new Panel();
    bottomPanel.Dock = DockStyle.Bottom;
    bottomPanel.Height = 60;
    bottomPanel.BackColor = Color.LightGray;
    bottomPanel.Padding = new Padding(10);
    // Текст найкращої комбінації
    Label bestCombinationLabel = new Label();
    string criteriaText = isTimeBased ? "найменший час" : "найкраще стиснення";
    string bestValue = isTimeBased ? $"{bestResult.ProcessingTime} мс" :
$"{Math.Round(bestResult.CompressionRatio, 2)}";
    bestCombinationLabel.Text = $"Найкраща комбінація ({criteriaText}):
{bestResult.AlgorithmCombination} -> {bestValue}";
    bestCombinationLabel.Dock = DockStyle.Fill;
    bestCombinationLabel.ForeColor = Color.DarkGreen;
    bestCombinationLabel.Font = new Font("Microsoft Sans Serif", 10, FontStyle.Bold);
    bestCombinationLabel.TextAlign = ContentAlignment.MiddleCenter;
    bottomPanel.Controls.Add(bestCombinationLabel);
    // Додаємо компоненти до форми
    resultForm.Controls.Add(resultsTable);
    resultForm.Controls.Add(bottomPanel);
    resultForm.Show();
}
private void save_Click(object sender, EventArgs e)
{
    // Зчитуємо шлях до файлу з текстового поля
    string filePath = FilePath2.Text;
    // Зчитуємо парольну фразу
    string password = PasswordWord2.Text;
    // Перевірка: чи введено шлях до файлу
    if (string.IsNullOrWhiteSpace(filePath))
    {
        MessageBox.Show("Шлях до файлу не може бути порожнім.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
```

```
    }
    // Перевірка: чи введено парольну фразу
    if (string.IsNullOrWhiteSpace(password))
    {
        MessageBox.Show("Парольне слово не може бути порожнім.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    try
    {
        // Зчитуємо дані з файлу
        byte[] fileData = File.ReadAllBytes(filePath);
        byte[] resultData = null; // Змінна для зберігання результату
        // Отримуємо вибраний алгоритм шифрування (якщо обрано)
        var encryption = (this.encryption.Checked || decoding.Checked)
            ? encryptionAlgorithms.FirstOrDefault(a => a.Name ==
shyfruvannya.SelectedItem?.ToString())
            : null;
        // Отримуємо вибраний алгоритм стиснення (якщо обрано)
        var compression = (archiving.Checked || unzipping.Checked)
            ? compressionAlgorithms.FirstOrDefault(a => a.Name ==
arkhivuvannya.SelectedItem?.ToString())
            : null;
        // Перевірка: чи знайдено вибраний алгоритм шифрування
        if ((this.encryption.Checked || decoding.Checked) && encryption == null)
        {
            MessageBox.Show("Не знайдено вибраного алгоритму шифрування.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        // Перевірка: чи знайдено вибраний алгоритм стиснення
        if ((archiving.Checked || unzipping.Checked) && compression == null)
        {
            MessageBox.Show("Не знайдено вибраного алгоритму архівування.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        // Виконання обраних дій
        if (this.encryption.Checked && archiving.Checked) // Шифрування + архівування
        {
            if (label5.Text == "1") // Спочатку шифрування, потім стиснення
            {
                resultData = encryption.Encrypt(fileData, password); // Шифрування
                resultData = compression.Compress(resultData); // Стиснення
            }
            else // Спочатку стиснення, потім шифрування
            {
                resultData = compression.Compress(fileData); // Стиснення
                resultData = encryption.Encrypt(resultData, password); // Шифрування
            }
        }
        else if (decoding.Checked && unzipping.Checked) // Розшифрування + розархівування
```

```
        {
            if (label5.Text == "1") // Спочатку розшифрування, потім розархівування
            {
                resultData = encryption.Decrypt(fileData, password); // Розшифрування
                resultData = compression.Decompress(resultData); // Розархівування
            }
            else // Спочатку розархівування, потім розшифрування
            {
                resultData = compression.Decompress(fileData); // Розархівування
                resultData = encryption.Decrypt(resultData, password); // Розшифрування
            }
        }
        else if (this.encryption.Checked) // Тільки шифрування
        {
            resultData = encryption.Encrypt(fileData, password);
        }
        else if (decoding.Checked) // Тільки розшифрування
        {
            resultData = encryption.Decrypt(fileData, password);
        }
        else if (archiving.Checked) // Тільки стиснення
        {
            resultData = compression.Compress(fileData);
        }
        else if (unzipping.Checked) // Тільки розархівування
        {
            resultData = compression.Decompress(fileData);
        }
        // Збереження результату у файл
        string outputFilePath = Path.Combine(
            Path.GetDirectoryName(filePath),
            Path.GetFileNameWithoutExtension(filePath) + "_processed" + Path.GetExtension(filePath)
        );
        File.WriteAllBytes(outputFilePath, resultData);
        // Повідомлення про успішну обробку
        MessageBox.Show($"Файл успішно оброблено та збережено: {outputFilePath}", "Успіх",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Повідомлення про помилку
        MessageBox.Show($"Сталася помилка: {ex.Message}", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```