Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи
ОР Магістр

на тему: Дослідження часової ефективності базових конструкцій мов Objective C та Swift під IOS
за освітньою програмою Інженерія програмного забезпечення
зі спеціальності: 121 – Інженерія програмного забезпечення
Виконав: студент групи: ПЗ2421 (951м)

_____ / Сергій НАВКА /

Керівник: _____ / доцент Олександр ІВАНОВ /

Нормоконтролер: _____ / доцент Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Дніпро – 2026 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technologies»

Explanatory Note
to Master's Thesis
Magister

on the topic: Research on the time efficiency of basic Objective C and Swift language constructs under iOS

according to educational curriculum Software engineering
in the Speciality: 121 Software engineering

Done by the student / Serhii NAVKA /

of the group: PZ 2421 (951m)

Scientific Supervisor: / Associate Professor Olexandr IVANOV /

Normative controller : / Associate Professor Svitlana VOLKOVA /

Dnipro – 2026

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні тенології
Рівень вищої освіти: Магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН

Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу Магістр

студенту Нору Олексію Сергійовичу

1. Тема роботи: «Дослідження ефективності алгоритмів шифрування та дешифрування даних»

Керівник роботи: Куроп’ятник Олена Сергіївна

затверджені наказом 1401 ст від 02.10.2025 року

2. Строк подання студентом роботи: 09.01.2026 року
3. Вихідні дані до роботи: перелік криптографічних алгоритмів
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):

4.1 Оглядпредметної області та постанока задачі: розглянути призначення криптографічних алгоритмів і актуальність оцінювання їх ефективності в сучасних інформаційних системах; сформулювати мету та задачі дослідження.

4.2 Дослідження алгоритмів та методики оцінювання: обґрунтувати вибір криптографічних алгоритмів для аналізу та описати методику проведення експериментів і отримання результатів.

4.3 Розробка програмного забеспечення: C#-додаток для шифрування/розшифрування файлів і порівняльного тестування симетричних, асиметричних та гібридних алгоритмів із журналюванням і експортом результатів; подати архітектуру, модулі, UML та опис інтерфейсу користувача.

4.4 Експерементальні дослідження симетричних асиметричних та гібридних алгоритмів: описати умови та хід експериментів, підготовку даних, отримані результати та їх порівняльний аналіз; сформулювати висновки за підсумками експериментів.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень):

презентація, демонстраційне відео

КАЛЕНДАРНИЙ ПЛАН

1	Вступ, формулювання мети та завдань дослідження	01.09.2025 – 06.09.2025	
2	Огляд літератури та аналіз симетричних, асиметричних та гібридних алгоритмів шифрування	07.09.2025 – 20.09.2025	
3	Аналіз методик експериментального оцінювання криптографічних алгоритмів та підходів до стандартизації сценаріїв запуску	21.09.2025 – 08.10.2025	
4	Формування вимог до програмного продукту та постановка задачі	09.10.2025 – 12.10.2025	
5	Розробка та узгодження технічного завдання	13.10.2025 – 21.10.2025	30 %
6	Проєктування архітектури системи оцінювання криптографічних алгоритмів	22.10.2025 – 29.10.2025	
7	Розробка математичної моделі оцінювання криптографічних алгоритмів	30.10.2025 – 09.11.2025	
8	Реалізація симетричних, асиметричних та гібридних алгоритмів шифрування	10.11.2025 – 28.11.2025	
9	Програмна реалізація модуля вибору криптографічних алгоритмів	29.11.2025 – 08.12.2025	
10	Проведення експериментальних серій шифрування/дешифрування та фіксація результатів вимірювань	09.12.2025 – 20.12.2025	
11	Тестування та відлагодження програмного продукту	21.12.2025 – 30.12.2025	60 %
12	Аналіз результатів досліджень та формування висновків	31.12.2025 – 04.01.2026	
13	Оформлення пояснювальної записки та програмної документації	05.01.2026 – 13.01.2026	100 %
14	Розробка демонстраційних матеріалів та підготовка до захисту	14.01.2026 – 20.01.2026	
15	Подання та захист кваліфікаційної роботи	21.01.2026	

Студент _____ Олексій НОР

Керівник роботи _____ Олена КУРО’ПЯТНИК

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавр: 143 с., 57 рис., 13 табл., 1 додаток, 35 джерел.

Об’єкт дослідження – процеси шифрування та дешифрування даних у прикладних сценаріях оброблення з урахуванням впливу апаратної платформи та режиму виконання.

Мета роботи – розробка інструментальної програмної системи для стандартизованого проведення експериментів і порівняльного аналізу криптографічних алгоритмів, а також формування критеріїв та рекомендацій щодо оцінювання ефективності за сукупністю метрик.

Методи дослідження – аналіз сучасних криптографічних підходів і сценаріїв застосування; проєктування модульної архітектури; бенчмаркінг з вимірюванням часу, середньої швидкості обробки, пікового завантаження CPU, використання RAM та коефіцієнта зміни обсягу даних; валідація коректності результатів; нормалізація метрик і формування інтегральних показників для порівняння та ранжування.

Одержані результати – виконано серії експериментальних вимірювань ефективності симетричних, асиметричних і гібридних криптографічних алгоритмів на носіях M.2/SSD/HDD. Результати систематизовано у таблицях і графіках та проведено порівняльний аналіз за сукупністю метрик. Встановлено, що AES є найпродуктивнішим для масового шифрування; ChaCha20 і Blowfish – доцільні альтернативи в програмних реалізаціях; DES/3DES/Serpent суттєво поступаються; для коротких асиметричних операцій ECC демонструє найкращий баланс швидкодії та компактності.

Ключові слова: КРИПТОГРАФІЧНІ АЛГОРИТМИ, СИМЕТРИЧНЕ ШИФРУВАННЯ, АСИМЕТРИЧНЕ ШИФРУВАННЯ, ГІБРИДНІ СХЕМИ, БЕНЧМАРКІНГ, ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ, МЕТРИКИ ПРОДУКТИВНОСТІ, ВАЛІДАЦІЯ, ЖУРНАЛЮВАННЯ, ПОРІВНЯЛЬНИЙ АНАЛІЗ.

ЗМІСТ

ВСТУП 10
1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 16
1.1 Актуальність задачі оцінювання ефективності криптоалгоритмів 16
1.2 Вимоги до продуктивності та ресурсів у сучасних інформаційних системах 20
1.3 Критерії ефективності та показники оцінювання 22
1.4 Аналіз підходів до порівняння криптографічних алгоритмів 31
1.5 Постановка задачі, мета та задачі дослідження 37
Висновки до розділу 40
2 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ТА МЕТОДИКА ОЦІНЮВАННЯ 42
2.1 Обґрунтування вибору алгоритмів для дослідження 42
2.2 Вибір методів дослідження та експериментів 48
2.3 Показники, що вимірюються, та спосіб їх отримання 52
2.4 Аналітична підсистема оцінювання 59
2.5 Вхідні дані та вихідні результати дослідження 64
Висновки до розділу 69
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ 71
3.1 Призначення системи, вимоги та сценарії використання 71
3.2 Архітектура системи та опис модулів 76
3.3 Реалізація збору показників і журналювання 79
3.4 Формат збереження та експорт результатів 84
3.5 UML-моделювання системи 88
3.6 Опис інтерфейсу користувача та приклади роботи системи 95
Висновки до розділу 99
4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕНННЯ СИМЕТРИЧНИХ АСИМЕТРИЧНИХ ТА ГІБРИДНИХ АЛГОРИТМІВ 101
4.1 Мета та логіка експериментальної частини 101
4.2 Умови проведення експериментів і підготовка вхідних даних 102
4.3 Експериментальні дослідження 105
4.4 Порівняльний аналіз отриманих результатів 130
4.5 Висновки щодо ефективності моделей 134

Висновки до розділу 139
ВИСНОВКИ 141
ПЕРЕЛІК ПОСИЛАНЬ 144
ДОДАТОК А 148
ДОДАТОК Б 141
ДОДАТОК В 11176
ДОДАТОК Г 7147
Тези конференції УДУНТ 2025 7147
Тези конференції «МОЛОДА АКАДЕМІЯ – 25» 740

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AES (Advanced Encryption Standard) – симетричний блочний шифр.

DES (Data Encryption Standard) – симетричний блочний шифр.

3DES (Triple DES) – потрійне застосування DES.

Serpent – симетричний блочний шифр.

Blowfish – симетричний блочний шифр.

ChaCha20 – симетричний потоковий шифр.

RSA – асиметричний алгоритм на основі факторизації.

ElGamal – асиметричний алгоритм на основі дискретного логарифмування.

ECC (Elliptic Curve Cryptography) – криптографія на еліптичних кривих.

AES_6Core – багатопотокова реалізація AES (із використанням кількох ядер CPU).

CPU (Central Processing Unit) – центральний процесор.

RAM (Random Access Memory) – оперативна пам’ять.

SSD (Solid-State Drive) – твердотільний накопичувач.

HDD (Hard Disk Drive) – жорсткий диск.

M.2 – форм-фактор накопичувача (часто NVMe/SSD).

ОС – операційна система.

ПК – персональний комп’ютер.

I/O (Input/Output) – введення/виведення даних.

с – секунда.

мс – мілісекунда.

байт (B) – одиниця обсягу даних.

КБ / МБ / ГБ / ТБ – кілобайт / мегабайт / гігабайт / терабайт.

біт – одиниця довжини ключа/криптографічних параметрів.

МБ/с – мегабайт за секунду (швидкість обробки/передавання).

% – відсоток (завантаження CPU тощо).

ГГц / МГц – гігагерц / мегагерц (частота процесора/пам’яті).

t – час виконання операції (шифрування/розшифрування).

v – середня швидкість обробки даних, (МБ/с).

VCPU_peak – пікове завантаження CPU, (%).

VRAM_peak – пікове використання RAM, (МБ).

S – обсяг даних (вхідний/вихідний), (МБ, ГБ).

n – кількість файлів у наборі.

k – коефіцієнт розширення даних (відношення обсягу після перетворення до початкового).

Шифрування – перетворення відкритих даних у шифротекст із використанням ключа.

Розшифрування (дешифрування) – відновлення вихідних даних із шифротексту за ключем.

Відкритий текст – дані до шифрування.

Шифротекст – результат шифрування (зашифровані дані).

Симетричний алгоритм – шифрування й розшифрування виконуються одним ключем.

Асиметричний алгоритм – використовується пара ключів (відкритий/закритий).

Гібридне шифрування – поєднання асиметричного обміну ключем і симетричного шифрування даних.

Ключ – параметр, що визначає результат криптоперетворення.

Ключова пара – сукупність відкритого та закритого ключів (для асиметрії).

Продуктивність – здатність алгоритму обробляти дані за одиницю часу.

Ресурсоспоживання – використання обчислювальних ресурсів (CPU, RAM, підсистема зберігання).

Сценарій експерименту – формалізований порядок дій і умов запуску (набір даних, носій, алгоритм, режим).

Валідація результату – перевірка коректності перетворення.

Відтворюваність – можливість повторити експеримент і отримати порівнювані результати за тих самих умов.

ВСТУП

У сучасному світі цифровізація та стрімке зростання обсягів даних, що обробляються й передаються інформаційними системами, вимагають надійних рішень для забезпечення конфіденційності та цілісності інформації. Сфери на кшталт хмарних сервісів, Інтернету речей, корпоративних мереж, фінансових транзакцій і мультимедійних платформ постійно стикаються з ризиками несанкціонованого доступу та витоку даних, тому ключову роль у захисті відіграють криптографічні алгоритми шифрування та дешифрування. Вони забезпечують безпечне зберігання й обмін інформацією, однак їхня ефективність у практичних умовах суттєво залежить від низки факторів: типу й обсягу даних, режимів роботи, розміру ключів, реалізації алгоритму, а також обчислювальних ресурсів системи (CPU, RAM, швидкодія підсистеми введення-виведення). Розробка комплексних засобів для дослідження продуктивності та порівняння криптографічних алгоритмів є актуальною, оскільки дозволяє обґрунтовано обирати оптимальні методи захисту даних, досягаючи балансу між рівнем безпеки та швидкодією у високонавантажених і вбудованих системах.

Створення дослідницької платформи для аналізу ефективності алгоритмів шифрування та дешифрування даних відповідає сучасним вимогам інформаційної безпеки. Такий інструмент поєднує теорію з практикою, дозволяючи оцінити швидкодію алгоритмів у реальних сценаріях і обґрунтовано обирати оптимальні рішення. Розроблений програмний продукт може бути інтегрований у корпоративні системи, хмарні сервіси та вбудовані пристрої, забезпечуючи баланс між захистом і ресурсоспоживанням.

Актуальність роботи: сутність задачі, що розв’язується у цій роботі, полягає у розробці комплексної системи для дослідження ефективності алгоритмів шифрування та дешифрування даних (зокрема AES, DES/3DES, ChaCha20, RSA, ECC, ElGamal, Serpent) у середовищі C# з урахуванням різних режимів роботи, розмірів ключів і типів вхідних даних (файли різних форматів, текстові повідомлення, масиви даних, потоки). Значення цієї проблематики

зумовлене зростанням обсягів інформації та підвищенням вимог до захисту у сучасних системах (корпоративні мережі, хмарні сервіси, IoT), де неправильний вибір криптоалгоритму або параметрів може призводити до надмірних витрат процесорного часу, пам’яті та збільшення часу обробки даних. На відміну від підходів, що обмежуються теоретичним порівнянням або використанням одного «універсального» алгоритму, пропонована система забезпечує експериментальне вимірювання швидкодії та ресурсоспоживання, що дає змогу обґрунтовано підбирати оптимальний алгоритм і параметри для конкретного сценарію використання.

Порівняно з відомими рішеннями, такими як стандартні засоби шифрування файлів або окремі демонстраційні приклади криптографічних бібліотек, пропонований проєкт вирізняється комплексним підходом до оцінювання ефективності алгоритмів шифрування та дешифрування. Більшість аналогів зосереджується на базовій перевірці працездатності алгоритмів або на вимірюванні швидкодії без єдиної методики, без урахування різних режимів роботи, розмірів ключів, типів вхідних даних і реальних сценаріїв (файли, папки, потоки даних). Натомість розроблена платформа забезпечує стандартизований бенчмаркінг із контролем коректності розшифрування, збором метрик продуктивності (час, швидкість, CPU/RAM), а також можливістю порівняння симетричних і асиметричних алгоритмів у однакових умовах. Актуальність роботи підтверджується її науково-прикладною спрямованістю та здатністю усунути протиріччя між високим рівнем криптографічного захисту і вимогами до швидкодії, надаючи обґрунтовані рекомендації для систем, у яких неефективний вибір алгоритму призводить до перевитрат ресурсів і зниження продуктивності.

Доцільність роботи полягає у створенні дослідницької платформи, яка не лише забезпечує об’єктивне порівняння алгоритмів шифрування та дешифрування, а й сприяє виробленню практичних підходів до вибору оптимальних криптографічних рішень з урахуванням типу даних, режимів роботи, розмірів ключів і обмежень обчислювальних ресурсів. Такий підхід

дозволяє заповнити прогалини у розумінні того, як параметри реалізації та характеристики навантаження впливають на швидкодію й ресурсоспоживання, що є критично важливим для реальних систем із високими вимогами до безпеки та продуктивності.

Об’єкт дослідження – процеси шифрування та дешифрування даних із використанням криптографічних алгоритмів (симетричних і асиметричних) на різних типах даних (текстові повідомлення, файли різних форматів, набори числових даних, потоки/пакети даних, каталоги файлів) у середовищі програмної реалізації.

Предметом дослідження є методи та показники оцінювання ефективності симетричних, асиметричних і гібридних криптографічних алгоритмів шифрування та дешифрування (час виконання, швидкість обробки, завантаження CPU, використання RAM), а також залежність цих показників від режимів роботи, розмірів ключів і характеристик вхідних даних у прикладних сценаріях.

Мета роботи – розробка комплексної дослідницької платформи для порівняльного аналізу ефективності алгоритмів шифрування та дешифрування даних з урахуванням режимів роботи, розмірів ключів, типів вхідних даних і ресурсних обмежень, а також розробка підходу до обґрунтованого вибору оптимального криптографічного алгоритму та параметрів під конкретний сценарій використання.

Користь від реалізації проєкту полягає у наданні розробникам і дослідникам інструменту для кількісної оцінки ефективності алгоритмів шифрування та дешифрування, виявлення емпіричних залежностей між параметрами (режими роботи, розміри ключів, типи даних) і продуктивністю, а також у створенні універсального формату (контейнера) з метаданими для коректного розшифрування та перевірки цілісності даних. Користувачі отримують практичні рекомендації щодо оптимального вибору криптографічних методів і налаштувань під конкретні сценарії, а наукова спільнота – основу для подальших досліджень у сфері інформаційної безпеки, де ефективність

криптографічної обробки безпосередньо впливає на швидкодію та ресурсоспоживання систем.

Методи дослідження – теоретичний аналіз класифікації криптографічних алгоритмів, розробка й реалізація тестових сценаріїв у середовищі C# для шифрування/дешифрування тексту, файлів і папок із різними обсягами даних; бенчмаркінг продуктивності з вимірюванням часу виконання, швидкості обробки (МБ/с), завантаження CPU та споживання RAM; перевірка коректності дешифрування та цілісності даних, статистична обробка результатів і побудова узагальнених висновків щодо залежності ефективності від параметрів алгоритмів (розмір ключа, режим, тип даних).

Наукова новизна полягає у розробці підходу до порівняльного оцінювання ефективності алгоритмів шифрування та дешифрування на основі інтегрального аналізу показників продуктивності (час, швидкість обробки, завантаження CPU, споживання RAM) і параметрів алгоритмів (тип, режим роботи, розмір ключа, тип даних). У межах роботи створено програмну платформу з уніфікованим форматом збереження результатів і криптографічним контейнером з метаданими, що забезпечує коректне дешифрування та контроль цілісності, а також дозволяє формувати практичні рекомендації щодо вибору оптимального алгоритму для конкретних сценаріїв використання в середовищі C#.

Практичне значення роботи полягає у створенні програмного продукту, який може бути інтегрований у корпоративні інформаційні системи, хмарні сервіси, IoT та вбудовані пристрої для забезпечення конфіденційності й цілісності даних під час зберігання та передавання. Отримані результати дають змогу розробникам обґрунтовано обирати алгоритми шифрування та параметри їх застосування (режим роботи, розмір ключа) з урахуванням вимог до швидкодії й ресурсоспоживання, зменшуючи накладні витрати на криптографічну обробку та підвищуючи загальну продуктивність систем.

Апробація результатів дослідження проводилася на семінарах кафедри комп’ютерних інформаційних технологій Українського державного університету науки і технологій (м. Дніпро, 2025 р.), де було презентовано

ключові модулі програмної платформи та отримані емпіричні залежності щодо швидкодії й ресурсоспоживання під час шифрування та дешифрування. Результати також апробовано в межах внутрішніх тестувань на реальних файлах і наборах даних різних форматів (текстові документи, мультимедіа, архіви), що дозволило перевірити коректність роботи системи та практичну придатність рекомендацій щодо вибору криптографічних алгоритмів.

Нор О.С. Дослідження ефективності алгоритмів шифрування та дешифрування даних // Тези доповідей XIX Міжнародної науково-практичної конференції «СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ». – Дніпро: УДУНТ, 2025. – С. 161.

Нор О.С. Аналіз методів генерації тестів // Збірник тез Міжнародної науково-технічної конференції студентів і молодих учених «Молода академія - 25». Том 2. – Дніпро: Український державний університет науки і технологій, 2025. – С. 44.

Експлуатаційне призначення: розроблений програмний продукт призначений для використання дослідниками, розробниками програмного забезпечення та фахівцями з інформаційної безпеки як інструмент для аналізу й оптимізації процесів шифрування та дешифрування даних у реальних задачах, таких як захист файлів і резервних копій, передавання даних у корпоративних мережах, хмарних сервісах та IoT-пристроях. Система орієнтована на фахівців, які працюють з різними типами даних і потребують обґрунтованого вибору криптографічних алгоритмів та параметрів для забезпечення балансу між рівнем захисту і продуктивністю. Практичне застосування полягає в інтеграції платформи у процеси розробки та тестування, де вона дозволяє проводити бенчмаркінг, порівнювати алгоритми за швидкодією й ресурсоспоживанням, візуалізувати результати та формувати рекомендації щодо оптимального використання шифрування в конкретних сценаріях.

Кінцеві користувачі – розробники програмного забезпечення та фахівці з інформаційної безпеки – отримують нематеріальну вигоду у вигляді підвищення

конфіденційності та цілісності даних, зменшення ризиків витоку інформації й підвищення надійності систем завдяки коректному застосуванню шифрування та дешифрування. Додатково платформа дає змогу обґрунтовано підбирати алгоритми та параметри з урахуванням продуктивності, що сприяє скороченню накладних витрат на криптографічну обробку та стабільній роботі сервісів під навантаженням. Дослідники, у свою чергу, отримують зручний інструмент для експериментів, порівняння реалізацій, оцінки впливу режимів роботи та розміру ключів, а також формування узагальнених висновків щодо ефективності криптоалгоритмів. Таким чином, платформа слугує мостом між теоретичними основами криптографії та практичними застосуваннями, забезпечуючи корисні рекомендації для всіх учасників процесу захисту даних.

Розробка цього проєкту базується на аналізі сучасних криптографічних алгоритмів шифрування та дешифрування, режимів їх роботи, методології бенчмаркінгу й підходів до оцінювання продуктивності та ресурсоспоживання. У подальших розділах роботи розглянуто теоретичні основи криптографічного захисту, методологію дослідження, проєктування програмної платформи, експериментальні результати та висновки, що підтверджують її практичну цінність і відповідність поставленим цілям.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність задачі оцінювання ефективності криптоалгоритмів

Актуальність задачі оцінювання ефективності криптографічних алгоритмів зумовлена тим, що криптографічний захист у сучасних інформаційних системах перестав бути факультативним елементом і перетворився на базову передумову довіри до процесів зберігання та оброблення даних. Для систем, які працюють із персональною, фінансовою, медичною або корпоративною інформацією, забезпечення конфіденційності й цілісності є не лише вимогою безпеки, а й необхідною умовою відповідності нормативним та організаційним правилам, підтримання репутаційної стійкості й мінімізації ризиків витоку даних. У цьому контексті криптографічні механізми інтегруються в ключові контури функціонування програмних продуктів та інфраструктур і безпосередньо впливають на їхню надійність як технічних систем.

Додатковим чинником актуальності є загальна тенденція до ускладнення архітектур інформаційних систем. Поширення розподілених сервісів, мікросервісних підходів, віртуалізації, контейнеризації та гібридних моделей розгортання призводить до зростання кількості точок взаємодії між компонентами, а відповідно і до збільшення площини потенційних атак. За таких умов криптографія виконує роль універсального механізму довіри, який застосовується як на рівні даних, так і на рівні сервісної взаємодії. Водночас це означає, що криптографічні операції виконуються частіше, у різноманітніших режимах і в ширшому спектрі середовищ, ніж у традиційних монолітних системах, тому питання їх ефективності набуває системного характеру і впливає на сумарну керованість інфраструктури [1].

Криптографія застосовується як для захисту даних під час зберігання, так і під час передавання. У першому випадку криптографічні перетворення використовуються для захисту файлів, резервних копій, архівів, баз даних і репозиторіїв, де загрози можуть бути пов’язані як із фізичним доступом до носія, так і з компрометацією облікових записів або внутрішніми інцидентами в межах

організації. У другому випадку криптографія забезпечує безпеку взаємодії між компонентами розподілених систем, сервісами та користувачами, протидіючи перехопленню, підміні, повтору повідомлень і несанкціонованому втручанню в канали зв’язку [2, 3]. Отже, криптографічні алгоритми виступають не ізольованими математичними інструментами, а практичними засобами, від яких залежить цілісність архітектури безпеки інформаційної системи загалом.

Важливо підкреслити, що вимоги до захисту дедалі частіше поширюються не лише на первинні дані, а й на похідні артефакти їх оброблення. До таких артефактів належать журнали подій, технічні дампи, кеші, тимчасові файли, резервні копії та проміжні результати аналітики. Оскільки ці об’єкти можуть містити чутливу інформацію або дозволяти відновлення первинних даних, їх захист перетворюється на обов’язковий елемент життєвого циклу даних [4]. Це підвищує частоту застосування криптографічних операцій і робить ефективність алгоритмів важливим чинником, який впливає на швидкість процесів супроводу, відновлення після збоїв і регулярного адміністрування систем.

Упровадження криптографічного захисту неминуче супроводжується накладними витратами, оскільки будь-яке перетворення даних потребує додаткових обчислень, ускладнює оброблення потоків і додає нові етапи до життєвого циклу роботи з інформацією. Ці витрати проявляються у конкретному середовищі експлуатації: на клієнтських пристроях із обмеженими ресурсами, на серверних вузлах з високою конкуренцією запитів, у хмарних інфраструктурах із динамічним масштабуванням, а також у системах, де суттєву роль відіграє підсистема введення та виведення та різні типи накопичувачів. У результаті навіть за умови коректно обраного рівня криптографічної стійкості рішення може виявитися експлуатаційно дорогим: погіршувати час відгуку, збільшувати навантаження на обчислювальні ресурси, знижувати пропускну здатність системи та ускладнювати виконання вимог до рівня сервісу. У практичних проєктах це створює ситуацію, коли безпека починає сприйматися як фактор, що обмежує продуктивність, а не як складова, узгоджена з інженерними вимогами [5, 6].

Окремого розгляду потребує вплив криптографії на масштабованість та економічну доцільність експлуатації. У розподілених середовищах збільшення кількості користувачів або зростання обсягу даних може супроводжуватися непропорційним збільшенням накладних витрат, якщо криптографічні механізми стають вузьким місцем оброблення. У такому випадку організація захисту перестає бути суто питанням політик безпеки і перетворюється на інженерну задачу, пов’язану з плануванням ресурсів, оптимізацією потоків оброблення та забезпеченням стабільної роботи під навантаженням. Саме тому оцінювання ефективності криптоалгоритмів є складовою прийняття архітектурних рішень, а не лише етапом підтвердження коректності впровадження.

Актуальність пов’язана із тим, що вибір криптографічного алгоритму в реальних застосуваннях не зводиться до формального переліку стандартизованих примітивів. Під алгоритмом у прикладному сенсі фактично розуміється сукупність рішень: спосіб керування ключами, режим застосування, характер оброблення даних, можливість паралельного виконання, особливості взаємодії з пам’яттю та введенням і виведенням, а також ефективність конкретної програмної реалізації. Тому два рішення, які формально забезпечують порівнюваний рівень захисту, можуть суттєво відрізнятися за поведінкою в однакових умовах використання. Звідси випливає, що вибір типового або рекомендованого за замовчуванням механізму не гарантує оптимальності в конкретній системі, а відсутність системного порівняння перетворює процес вибору на евристичне рішення, недостатньо обґрунтоване з позицій інженерної доцільності [7].

Додатково слід враховувати, що криптографічні алгоритми застосовуються в системах із різними пріоритетами та різною критичністю даних. Для одних систем ключовим є збереження керованості та стабільності роботи у довготривалих процесах зберігання й оброблення великих масивів інформації. Для інших визначальним є забезпечення оперативності та низької затримки під час взаємодії користувачів із сервісом. У такій ситуації

ефективність криптографічного механізму має оцінюватися не у відриві від сценарію, а в контексті конкретного режиму застосування. Це підсилює вимогу до порівняльного аналізу як до процедури, яка має враховувати умови виконання, характер даних і практичні обмеження інфраструктури.

Ефективність криптографічних рішень має багатокомпонентний характер і не може бути адекватно описана одним параметром. У прикладній системі важливо враховувати сукупність взаємопов’язаних факторів: характер використання обчислювальних ресурсів, вплив на паралельну обробку, стабільність роботи за зміни умов і масштабів даних, а також супутні ефекти, що виникають у процесі формування захищених подань інформації та їх подальшого зберігання й передавання. На практиці ці фактори нерідко вступають у суперечність: рішення, яке є більш прийнятним в одному аспекті, може вимагати суттєвих компромісів в іншому. Відтак оцінювання ефективності криптоалгоритмів доцільно розглядати як комплексну дослідницьку задачу, зорієнтовану на обґрунтований вибір механізмів для визначених сценаріїв експлуатації [8].

У контексті інженерної практики важливо також, що системне оцінювання ефективності виконує роль засобу підвищення прозорості технічних рішень. Воно дозволяє аргументовано пояснювати вибір криптографічних механізмів, формувати узгоджені рекомендації для різних режимів роботи та забезпечувати повторюваність висновків під час модернізації системи або міграції на іншу інфраструктуру. Таким чином, результати порівняльного аналізу є корисними не лише на етапі розроблення, а й у процесі супроводу, коли необхідно перевіряти відповідність застосованих механізмів зміненим умовам експлуатації та оновленим вимогам до захисту.

Отже, актуальність проблематики визначається фундаментальною суперечністю між необхідністю забезпечення високого рівня захисту інформації та вимогою зберігати належні експлуатаційні характеристики інформаційної системи. Розв’язання цієї суперечності неможливе декларативно й потребує методично коректного порівняння альтернатив з урахуванням контексту

застосування, архітектури системи та умов її функціонування. Саме тому в межах магістерського дослідження доцільно обґрунтовувати вибір криптографічних алгоритмів через системне оцінювання їх ефективності, що створює логічну основу для формування вимог і переходу до наступного підрозділу, присвяченого аналізу експлуатаційних потреб сучасних інформаційних систем [9, 10].

1.2 Вимоги до продуктивності та ресурсів у сучасних інформаційних системах

Вимоги до продуктивності та ресурсів у сучасних інформаційних системах формуються не абстрактно, а під впливом конкретних сценаріїв експлуатації, у межах яких криптографічні перетворення мають виконуватися без порушення функціональної придатності та без деградації якості сервісу. На практиці криптографічні алгоритми застосовуються в різнорідних режимах оброблення даних. Найпростішим є шифрування окремого файла, однак навіть у цьому випадку важливими стають вимоги до швидкості виконання операції та до прогнозованості поведінки системи за різних типів даних. Більш складним є шифрування папки або пакета файлів, де зростає роль пакетної обробки, керування чергою операцій і взаємодії з підсистемою введення та виведення. Окрему групу становлять задачі резервного копіювання й архівації, для яких характерні великі обсяги інформації, тривалі цикли оброблення та потреба підтримувати стабільний режим роботи протягом усього процесу. Нарешті, у сценаріях обміну повідомленнями та керування ключами криптографічні механізми часто працюють у режимі коротких операцій, що виконуються часто й у взаємодії з мережевими протоколами та прикладною логікою, а отже критичною стає здатність системи забезпечувати оперативність і мінімізувати затримки в обробленні запитів.

У зазначених сценаріях продуктивність слід розглядати як здатність системи підтримувати необхідний темп оброблення інформації та відповідати вимогам користувацьких і сервісних процесів. Для потокових і пакетних режимів

принциповим є забезпечення достатньої пропускної спроможності при тривалих навантаженнях, щоб криптографічний захист не перетворювався на вузьке місце в загальному ланцюгу оброблення даних. Для інтерактивних і транзакційних режимів більш значущою стає швидкість реакції, оскільки навіть незначні затримки, що повторюються багаторазово, здатні істотно впливати на сприйняття системи користувачем і на здатність сервісів виконувати договірні зобов’язання щодо рівня обслуговування. Окремої уваги потребує масштабованість: зі зростанням обсягів даних, кількості одночасних запитів або паралельних процесів криптографічні операції мають зберігати прийнятну ефективність і не спричиняти лавиноподібного зростання витрат, яке підриває економічну доцільність рішення [11].

Поряд із вимогами до продуктивності, у реальних інформаційних системах визначальними є обмеження ресурсів, оскільки криптографічні перетворення конкурують за ресурси з іншими компонентами програмного комплексу. Насамперед йдеться про процесорні ресурси, адже інтенсивні обчислення можуть витісняти прикладні потоки, підвищувати загальне навантаження та знижувати стабільність системи під піковими запитами. Не менш важливою є потреба в оперативній пам’яті, оскільки оброблення даних, буферизація та службові структури можуть призводити до зростання споживання пам’яті, що особливо критично для середовищ із жорсткими обмеженнями або високою щільністю розміщення сервісів. Значний внесок у кінцевий результат робить підсистема зберігання, оскільки шифрування файлів і масивів даних нерозривно пов’язане з операціями читання та запису, а отже характеристики носія та загальна організація введення і виведення визначають фактичні умови, в яких працює криптографічний механізм. У різних інфраструктурах ці умови можуть істотно відрізнятися, тому порівняння алгоритмів без урахування середовища зберігання здатне давати спотворені висновки щодо їх практичної придатності [12, 13].

Окрім власне продуктивності й ресурсів, важливими є вимоги до практичності застосування криптографічних рішень у розробленні та

експлуатації. Система має демонструвати стабільність і передбачуваність роботи в повторюваних умовах, щоб результати оцінювання не залежали від випадкових коливань середовища, а зроблені висновки можна було відтворити та перевірити. Необхідною є також можливість автоматизації випробувань і організації журналювання, оскільки без систематичного збору інформації про перебіг операцій складно забезпечити прозорість експериментів, контроль коректності виконання та накопичення даних для подальшого аналізу. У підсумку вимоги до криптографічних алгоритмів у сучасних інформаційних системах доцільно розглядати як сукупність очікувань щодо здатності підтримувати ефективну обробку даних у різних сценаріях, працювати в межах доступних обчислювальних і пам'яттєвих ресурсів, коректно взаємодіяти з підсистемою зберігання та бути придатними до відтворюваного експериментального оцінювання. Саме ці аспекти становитимуть основу для подальшої перевірки в межах даної роботи під час порівняння обраних криптографічних рішень у типових умовах використання.

1.3 Критерії ефективності та показники оцінювання

У цьому підрозділі задаються критерії ефективності криптографічних алгоритмів і формалізуються показники, які дають змогу коректно порівнювати реалізації в однакових умовах. Нехай A – обраний алгоритм або реалізація алгоритму, d_i – тестовий об'єкт даних, S_i – обсяг даних для d_i у байтах. Кожен експеримент задається конфігурацією c, яка фіксує середовище виконання, параметри алгоритму, тип носія, режим оброблення та інші контрольовані умови. Для кожної конфігурації c збирається набір спостережень, що описують часові характеристики, інтенсивність використання обчислювальних ресурсів і зміну розміру даних після перетворення [14, 15].

Базовим часовим показником для симетричних перетворень є тривалість операції шифрування, яку визначають як різницю моментів завершення та початку процедури перетворення даних за формулою (1.1):

$$T_{enc}(A, d_i, c) = t_{end} - t_{start}. \quad (1.1)$$

де t_{start} – момент початку операції шифрування;

t_{end} – момент завершення операції шифрування.

Аналогічно визначається тривалість розшифрування вдповдно до (1.2):

$$T_{dec}(A, d_i, c) = t'_{end} - t'_{start}. \quad (1.2)$$

де t'_{start} – момент початку операції розшифрування;

t'_{end} – момент завершення операції шифрування.

Для асиметричних схем окремо фіксується тривалість побудови ключової пари, оскільки цей етап має іншу природу навантаження та суттєво впливає на придатність алгоритму в інтерактивних сценаріях; відповідний показник задається формулою (1.3):

$$T_{key}(A, c) = t_{end}^{key} - t_{start}^{key}. \quad (1.3)$$

де t_{start}^{key} – момент початку генерації ключів;

t_{end}^{key} – момент завершення генерації ключів.

На практиці часові величини доцільно розкладати на складові, щоб відокремити власне криптографічне перетворення від супровідних операцій; такий розклад задається у (1.4):

$$T_{enc} = T_{io} + T_{core} + T_{aux}, \quad (1.4)$$

де T_{io} – описує витрати на введення та виведення;

T_{core} – час виконання криптографічного ядра;

T_{aux} – допоміжні дії (підготовки буферів, службових структур).

Такий поділ важливий для коректної інтерпретації, оскільки однакова реалізація може демонструвати різну поведінку в залежності від того, чи домінує підсистема зберігання, чи обчислювальний етап.

Похідним показником, який інтерпретується як здатність алгоритму обробляти дані з певним темпом, є швидкість перетворення [16, 17]. Для одного об’єкта даних швидкість шифрування та розшифрування визначається як відношення обсягу до тривалості (1.5):

$$V_{enc}(A, d_i, c) = \frac{S_i}{T_{enc}(A, d_i, c)}, \qquad V_{dec}(A, d_i, c) = \frac{S_i}{T_{dec}(A, d_i, c)}. \tag{1.5}$$

де S_i – обсяг данних для тестового об’єкта d_i;

$T_{enc}(A, d_i, c)$ – тривалість шифрування за (1.1);

$T_{dec}(A, d_i, c)$ – тривалість розшифрування за (1.2).

Для серії з n об’єктів доцільно застосовувати узгоджене агрегування, яке відповідає інженерному змісту задачі пакетної обробки (1.6):

$$\overline{V}_{enc}(A, c) = \frac{\sum_{i=1}^{n} S_i}{\sum_{i=1}^{n} T_{enc}(A, d_i, c)}, \qquad \overline{V}_{dec}(A, c) = \frac{\sum_{i=1}^{n} S_i}{\sum_{i=1}^{n} T_{dec}(A, d_i, c)}. \tag{1.6}$$

де n – кількість об’єктів у серіхї;

S_i – обсяг i-го об’єкта;

$T_{enc}(A, d_i, c)$, $T_{dec}(A, d_i, c)$ – тривалості операцій для i-го об’єкта.

Таке визначення коректніше за просте усереднення швидкостей по файлах, оскільки воно автоматично враховує різний обсяг об’єктів і відображає фактичну пропускну здатність конвеєра обробки.

Окремий клас показників пов’язаний із використанням обчислювальних ресурсів. Нехай $u(t)$ – миттєве завантаження процесора у відсотках у момент часу t протягом виконання операції, а T – її тривалість [18]. Тоді середній рівень

завантаження можна формально задати як середнє за часом (1.7), а пікове – як максимум на інтервалі виконання (1.8):

$$U_{avg} = \frac{1}{T}\int_0^T u(t)dt, \tag{1.7}$$

$$U_{max} = \max_{t\in[0,T]} u(t). \tag{1.8}$$

де $u(t)$ – миттєве завантаження CPU;

T – тривалість операції.

Корисним є також показник сумарної процесорної роботи, що відповідає інтегралу завантаження (1.9):

$$W_{cpu} = \int_0^T u(t)dt, \tag{1.9}$$

де W_{cpu} – сумарна «процесорна робота» на інтервалі виконання;

$u(t), dt$ – визначені вище.

Аналогічно, якщо $m(t)$ – обсяг зайнятої оперативної пам’яті у момент часу t, то середнє аксимальне використання пам’яті визначаються співвідношеннями (1.10):

$$M_{avg} = \frac{1}{T}\int_0^T m(t)dt, \qquad M_{max} = \max_{t\in[0,T]} m(t). \tag{1.10}$$

де $m(t)$ – використання RAM у момент часу t;

M_{avg} – середнє використання RAM;

M_{max} – пікове використання RAM;

T – тривалість операції.

Ці формалізації важливі тим, що дають змогу відрізняти алгоритми, які працюють швидко за рахунок агресивного використання ресурсів, від

алгоритмів, які є більш ощадними, але можуть вимагати більшого часу на виконання.

Показники, що описують зміну розміру даних після перетворення, фіксують накладні витрати, пов’язані зі структурою шифротексту, службовими заголовками, вирівнюванням блоків та іншими конструктивними елементами формату [19, 20]. Нехай S_{in} – розмір вхідних даних, S_{out} – розмір вихідних даних після перетворення. Тоді коефіцієнт розширення визначається за формулою (1.11), а відносні накладні витрати – за формулою (1.12):

$$K = \frac{S_{out}}{S_{in}}. \tag{1.11}$$

$$O = \frac{S_{out} - S_{in}}{S_{in}} = K - 1. \tag{1.12}$$

де S_{in} – розмір вхідних даних;

S_{out} – розмір результату після перетворення;

K – коефіцієнт розширення;

O – відносні накладні витрати за розміром.

Для практичної інтерпретації ці величини відповідають тому, наскільки зростає обсяг зберігання і трафік у системі при переході до захищеного представлення даних, що є важливим для систем резервного копіювання, архівації та масового зберігання.

Оскільки експериментальні дані завжди містять варіативність, у підрозділі також потрібно формально задати спосіб узагальнення результатів серії повторів. Нехай для фіксованих A, d_i, c виконано r повторів, а отримані значення певного показника позначено як $x_1, x_2, \dots, x_r$. Тоді вибіркове середнє визначається за (1.13), дисперсія – за (1.14), стандартне відхилення – за (1.15):

$$\overline{x} = \frac{1}{r}\sum_{j=1}^{r} x_j, \tag{1.13}$$

$$s^2 = \frac{1}{r-1}\sum_{j=1}^{r}(x_j - \overline{x})^2, \tag{1.14}$$

$$s = \sqrt{s^2}. \tag{1.15}$$

де r – кількість повторів;

x_j – результат j-го повтору;

$\overline{x}$ – вибіркове середнє;

s^2 – вибіркова дисперсія;

s – стандартне відхилення.

Для оцінювання надійності середнього використовується довірчий інтервал (1.16):

$$\overline{x} \pm t_{1-\frac{a}{2},r-1} \cdot \frac{s}{\sqrt{r}}, \tag{1.16}$$

де α – рівень значущості,

$t_{1-\frac{a}{2},r-1}$ – квантиль розподілу Стьюдента з $r-1$ ступенями вільності.

s – стандартне відхилення за (1.15);

r – кількість повторів.

Для зіставлення стабільності різних конфігурацій корисним є безрозмірний коефіцієнт варіації (1.17), а для виявлення викидів – міжквартильний розмах (1.18):

$$CV = \frac{s}{\overline{x}}, \tag{1.17}$$

$$IQR = Q_3 - Q_1. \tag{1.18}$$

де CV – кофіцієнт варіації;

Q_1, Q_3 – перший і третій квартилі;

IQR – міжквартильний розмах.

Значення x можна вважати аномальним, якщо виконується нерівність $x < Q_1 - 1.5 \cdot IQR$ або $x > Q_3 + 1.5 \cdot IQR$. Таке формулювання підсилює відтворюваність висновків і робить процедуру обробки даних прозорою.

Для підсумкового порівняння алгоритмів у багатокритеріальній постановці доцільно задати нормалізацію показників, оскільки вони мають різні одиниці виміру та різний напрям оптимальності [21]. Нехай $x_k(A, c)$ – значення k-го показника для алгоритму A у конфігурації c, а $x_k^{min}(c)$ і $x_k^{max}(c)$ – мінімальне та максимальне значення цього показника серед усіх алгоритмів у тій самій конфігурації. Тоді для показників, де більші значення є кращими, нормалізація визначається за (1.19), а для показників, де менші значення є кращими – за (1.20):

$$z_k(A, c) = \frac{x_k(A, c) - x_k^{min}(c)}{x_k^{max}(c) - x_k^{min}(c)}, \tag{1.19}$$

$$z_k(A, c) = \frac{x_k^{max}(c) - x_k(A, c)}{x_k^{max}(c) - x_k^{min}(c)}. \tag{1.20}$$

де $z_k(A, c)$ – нормалізоване значення k-го показника;

$x_k(A, c)$ – початкове значення показника;

$x_k^{min}(c), x_k^{max}(c)$ – межі значень у конфігурації c.

Після нормалізації вводиться інтегральний індекс з вагами w_k, який задається формулою (1.21):

$$I(A, c) = \sum_{k=1}^{m} w_k z_k(A, c), \qquad \sum_{k=1}^{m} w_k = 1. \tag{1.21}$$

де $I(A, c)$ – інтегральний індекс ефективності;

m – кількість показників;

w_k – вага k-го показника;

$z_k(A, c)$ – нормалізоване значення за (1.19 – 1.20).

Як альтернативу, що жорсткіше штрафує слабкі місця, можна застосувати зважене геометричне усереднення (1.22):

$$l_g(A,c) = \prod_{k=1}^{m} [z_k(A,c)]^{w_k}. \tag{1.22}$$

де $l_g(A,c)$ – геометричний інтегральний індекс;

інші позначення відповідають (1.21).

У термінах часткового порядку корисним є також поняття домінування за Парето. Нехай $x(A,c)$ – вектор значень показників для алгоритму A у конфігурації c. Алгоритм A домінує алгоритм B, якщо для всіх показників він не гірший і хоча б за одним – кращий, що формально задається умовами $\forall k: x_k(A,c) \le x_k(B,c)$ та $\exists k: x_k(A,c) < x_k(B,c)$ для показників, де менше означає краще, або відповідно з оберненими нерівностями для показників іншого типу. Такий підхід дозволяє аргументовано пояснювати, чому в деяких сценаріях неможливо виділити єдиного універсального лідера без фіксації пріоритетів [22].

Для забезпечення однозначності подальших розрахунків і порівнянь у межах підрозділу зафіксовано: часові характеристики виконання криптографічних операцій (1.1 – 1.4), показники темпу оброблення даних (1.5 – 1.6), величини використання обчислювальних ресурсів (1.7 – 1.10), параметри зміни обсягу даних після перетворення (1.11 – 1.12), статистичні характеристики узагальнення результатів повторів (1.13 – 1.18) та підходи до нормалізації й інтегрального зіставлення (1.19 – 1.22). Узагальнений перелік прийнятих позначень, одиниць подання та змістової інтерпретації показників наведено в таблиці 1.1.

Таблиця 1.1 – Система показників та позначень для оцінювання ефективності криптографічних перетворень

Позначення	Одиниці	Інтерпритація
T_{enc}	с	Тривалість шифрування для заданих даних і конфігурації
T_{dec}	с	Тривалість розшифрування для заданих даних і конфігурації
T_{key}	с	Тривалість генерації ключового матеріалу для асиметричних схем
V_{enc}, V_{dec}	байт/с	Темп оброблення даних, що відображає пропускну здатність перетворень
U_{avg}, U_{max}	відсоток	Середнє та пікове завантаження процесора під час операції
W_{cpu}	відсоток с	Інтегральна процесорна робота за час виконання
M_{avg}, M_{max}	байт	Середнє та пікове використання оперативної пам’яті
K	безрозмірно	Відношення розміру вихідних даних до вхідних, характеризує розширення
O	безрозмірно	Відносні накладні витрати за розміром у порівнянні з початковими даними
$\overline{x}, s, SE$	як у x	Усереднення та розкид результатів повторів, надійність оцінки
I, I_g	безрозмірно	Інтегральний індекс для ранжування з урахуванням пріоритетів сценарію

Сформульовані критерії ефективності та формалізовані показники оцінювання створюють методичну основу для коректного зіставлення криптографічних алгоритмів у заданих умовах. Водночас наявність різнорідних показників і відмінностей у сценаріях застосування зумовлює потребу перейти

від простого фіксування результатів до обґрунтованого вибору підходів порівняння та узагальнення, які забезпечують інтерпретованість і відтворюваність висновків. Саме тому в наступному підрозділі розглядаються підходи до порівняльного аналізу криптографічних алгоритмів та обґрунтовується вибір методики, що найбільш адекватно відповідає меті даної роботи.

1.4 Аналіз підходів до порівняння криптографічних алгоритмів

У сучасних інформаційних системах криптографічні алгоритми застосовуються не як ізольовані математичні конструкції, а як інженерні механізми, що функціонують у складі складних програмно-апаратних комплексів. Тому питання їх порівняння виходить за межі формального переліку властивостей і перетворюється на задачу обґрунтованого вибору, де важливо врахувати як внутрішню структуру алгоритму, так і умови, в яких він фактично експлуатується. На практиці саме різниця у контексті застосування призводить до ситуації, коли одна й та сама криптографічна технологія може демонструвати прийнятні характеристики в одному середовищі та створювати неприйнятні обмеження в іншому. Це зумовлено поєднанням кількох чинників: специфікою програмної реалізації, характером оброблення даних, архітектурою взаємодії компонентів, особливостями підсистеми зберігання та впливом системних накладних витрат, які супроводжують виконання криптографічних операцій [23].

Крім того, порівняння криптографічних алгоритмів не можна зводити до одноразового вимірювання або до оцінки за одним параметром, оскільки в реальних задачах важливими є не лише швидкісні характеристики, а й здатність алгоритму працювати стабільно, передбачувано та без критичного впливу на інші підсистеми. Відсутність чіткої методики порівняння призводить до ризику некоректних висновків, коли отримані результати відображають не властивості алгоритмів, а випадкові особливості експерименту або неконтрольовані відмінності середовища. Звідси випливає необхідність організовувати аналіз як

послідовний процес, у якому кожний етап має власну роль: від первинного обґрунтування очікуваної поведінки до отримання емпіричних даних і їх узагальнення у формі, придатній для прийняття рішення.

У межах порівняльного аналізу криптографічних алгоритмів доцільно розрізняти три рівні міркувань: теоретичний, експериментальний та рівень узагальнення результатів для прийняття рішення. Теоретичний рівень використовується для попереднього відсіву та пояснення очікуваної поведінки алгоритмів, експериментальний рівень забезпечує емпіричну релевантність у конкретному середовищі, а багатокритеріальне узагальнення перетворює набір різнорідних спостережень на ранжування та рекомендації для сценаріїв застосування [24].

Теоретичний підхід ґрунтується на побудові моделі витрат, у якій час виконання подається як сума вартостей базових операцій, зважених на їх кількість. Нехай O є множиною типів елементарних операцій, наприклад побітові операції, табличні підстановки, циклічні зсуви, множення у скінченних полях, модульні множення та піднесення до степеня [25]. Для алгоритму A і задачі i кількість операцій типу j позначимо $n_j^{A,i}$ а вартість однієї такої операції у вибраному обчислювальному середовищі позначимо a_j. Тоді теоретично оцінена трудомісткість може бути записана за формулою (1.23):

$$C^{A,i} \sum_{j \in O} a_j n_j^{A,i} . \tag{1.23}$$

де $C^{A,i}$ – оцінена трудомісткість алгоритму A для задачі i;

O – множина типів елементарних операцій;

a_j – «вага» (вартість) однієї операції типу j у даному середовищі;

$n_j^{A,i}$ – кількість операцій типу j, які виконує алгоритм A у задачі i.

Для асиметричних схем ключовою є вартість великих арифметичних операцій. Порядкову оцінку вартості модульного піднесення до степеня можна подати формулою (1.24):

$$C_{exp}^{K} \sim M_k \log k\,, \tag{1.24}$$

де $C_{exp}^{K}(k)$ порядкова оцінка вартості операції піднесення до степеня для розрядності

$M(k)$ – вартість множення великих чисел розрядності

$\sim$ – позначає оцінку за порядком зростання (асимптотичний характер).

Експериментальний підхід базується на відтворюваних випробуваннях у фіксованих умовах. Для алгоритму A, конфігурації c, тестового об'єкта i та повтору r фіксується значення показника $x_k^{A,c,i,r}$, де k нумерує показники. Узагальнення повторів задається вибірковим середнім за флрмулою (1.25):

$$\overline{x}_k^{A,c,i} = \frac{1}{R}\sum_{r=1}^{R} x_k^{A,c,i,r}\,, \tag{1.25}$$

де $\overline{x}_k^{A,c,i}$ – середнє значення показника k для A у конфігурації c на об'єкті i;

R – кількість повторів;

$x_k^{A,c,i,r}$ – значення показника k у повторі r.

Розкид результатів доцільно характеризувати вибірковою дисперсією за формулою (1.26):

$$s_k^{2\,A,c,i} = \frac{1}{R-1}\sum_{r=1}^{R}\left(x_k^{A,c,i,r} - \overline{x}_k^{A,c,i}\right)^2. \tag{1.26}$$

де $s_k^{2\,A,c,i}$ – вибіркова дисперсія показника k;

інші позначення – як у (1.25).

Для пакетної обробки серії об’єктів зручним є агрегування по i. Якщо S_i є обсягом даних тестового об’єкта, тоді узагальнена характеристика темпу оброблення визначається як відношення сумарного обсягу до сумарного часу. Це задається формулою (1.27):

$$\overline{V}^{A,c} = \frac{\sum_{i=1}^{N} S_i}{\sum_{i=1}^{N} \overline{T}^{A,c,i}}. \tag{1.27}$$

де $\overline{V}^{A,c}$ – узагальнена швидкість оброблення (МБ/с або байт/с) для A у конфігурації c;

S_i – обсяг i-го тестового об’єкта;

$\overline{T}^{A,c,i}$ – середній час оброблення об’єкта i (після узагальнення повторів);

N – кількість об’єктів у серії.

Якщо показник інтерпретується так, що менше значення є кращим, тоді додатне Δ означає покращення. Для показників з протилежним напрямом оптимальності знак у чисельнику змінюють місцями, щоб додатне значення завжди означало перевагу.

Коректність порівняння залежить від забезпечення однакових умов. Це реалізується через фіксацію конфігурації c, яка містить опис середовища виконання, параметрів алгоритму, режимів оброблення та правил підготовки даних. Формально вимога однаковості означає, що для всіх алгоритмів A множина конфігурацій c і множина тестових об’єктів i є спільними, тобто кожне порівняння виконується для одних і тих самих c та i. Додатково вводиться регламент повторів R і незмінна процедура збору результатів, щоб мінімізувати систематичні відхилення між серіями.

Оскільки показники мають різні одиниці виміру та різний напрям оптимальності, вводиться нормалізація [26]. Для показника типу «більше – краще» нормалізоване значення визначається формулою (1.28) а межі нормалізації визначаються формулою (1.28):

$$z_k^{A,c} = \frac{\overline{x}_k^{A,c} - x_{k,min}^c}{x_{k,max}^c - x_{k,min}^c}, \tag{1.28}$$

$$x_{k,min}^c = \min_A \overline{x}_k^{A,c}, \qquad x_{k,max}^c = \max_A \overline{x}_k^{A,c}. \tag{1.29}$$

де $z_k^{A,c}$ – нормалізоване значення показника k для алгоритму A у конфігурації c;

$\overline{x}_k^{A,c}$ – узагальнене (наприклад, середнє) значення показника k;

$x_{k,min}^c, x_{k,max}^c$ – мінімум/максимум серед усіх алгоритмів у тій самій конфігурації c.

Для показника типу «менше – краще» використовується інверсія за формулою (1.30):

$$z_k^{A,c} = \frac{x_{k,min}^c - \overline{x}_k^{A,c}}{x_{k,max}^c - x_{k,min}^c}, \tag{1.30}$$

де $z_k^{A,c}$ – нормалізоване значення, уніфіковане так, що більше завжди означає краще;

інші позначення – як у (1.28 – 1.29).

Далі вводяться ваги пріоритетів показників для конкретного сценарію. Умова на ваги задається формулою (1.31):

$$\sum_{k=1}^{m} w_k^c = 1, \qquad w_k^c \geq 0. \tag{1.31}$$

де w_k^c – ага показника k у конфігурації/сценарії c;

m – кількість показників.

Найпоширеніший варіант узагальнення – зважена сума (інтегральний індекс), що задається формулою (1.32):

$$I^{A,c} = \sum_{k=1}^{m} w_k^c z_k^{A,c}. \quad (1.32)$$

де $I^{A,c}$ – нтегральний (сценарний) індекс ефективності алгоритму A у сценарії c;

$z_k^{A,c}$ – нормалізовані показники (див. (1.28 – 1.30));

w_k^c – ваги (див. (1.31)).

Щоб сильніше «штрафувати» слабкі місця, можна застосувати зважене геометричне узагальнення за формулою (1.33):

$$I_g^{A,c} = \prod_{k=1}^{m} \left(z_k^{A,c}\right)^{w_k^c}. \quad (1.33)$$

де $I_g^{A,c}$ – геометричний інтегральний індекс;

інші позначення – як у (1.32).

Альтернативою індексам є метод ідеальної точки, де визначаються найкращі нормалізовані значення визначаються за формулою (1.34). Відстані до ідеальної та антиідеальної точок задаються формулою (1.35). Коефіцієнт близькості до ідеалу визначається формулою (1.36):

$$z_k^{+,c} = \max_A z_k^{A,c}, \qquad z_k^{-,c} \min_A z_k^{A,c}. \quad (1.34)$$

$$d_+^{A,c} = \sqrt{\sum_{k=1}^{m} w_k^c \left(z_k^{A,c} - z_k^{+,c}\right)^2}, \qquad d_-^{A,c} = \sqrt{\sum_{k=1}^{m} w_k^c \left(z_k^{A,c} - z_k^{-,c}\right)^2}. \quad (1.35)$$

$$C^{A,c} = \frac{d_-^{A,c}}{d_-^{A,c} + d_-^{A,c}}, \quad (1.36)$$

де $z_k^{+,c}$ – найкраще значення показника k у конфігурації c;

$z_k^{-,c}$ – найгірше значення показника k у конфігурації c;

$d_+^{A,c}$ – відстань до ідеальної точки;

$d_-^{A,c}$ – відстань до антиідеальної точки;

$C^{A,c}$– показник близкості до ідеалу.

Якщо потрібно отримати глобальну оцінку з урахуванням важливості різних сценаріїв, вводяться ваги сценаріїв π_c, що задовольняють формулою (1.37) а інтегральна оцінка алгоритму по всіх сценаріях задається формулою (1.38):

$$\sum_{c=1}^{C} \pi_c = 1, \qquad \pi_c \geq 0. \tag{1.37}$$

$$I^A = \sum_{c=1}^{C} \pi_c I^{A,c}, \tag{1.38}$$

де π_c – важливість сценарью c у загальному профілі застосування;

$I^{A,c}$ – сценарій індекс ефективності (див. (1.32));

I^A – інтегральна оцінка алгоритму по всіх сценаріях;

C – кількість сценаріїв.

Така конструкція дозволяє формувати рекомендації не для абстрактного середнього випадку, а для визначеного профілю застосування.

1.5 Постановка задачі, мета та задачі дослідження

Об’єктом дослідження є застосування криптографічних перетворень у сучасних інформаційних системах під час захисту даних у типових режимах експлуатації. У межах цього об’єкта криптографічні алгоритми розглядаються не як абстрактні математичні конструкції, а як практичні механізми, які функціонують у складі програмно-апаратного середовища та взаємодіють з підсистемами введення-виведення, оброблення даних, організацією виконання задач і ресурсними обмеженнями платформи. Предметом дослідження є методи

та засоби порівняльного оцінювання ефективності криптографічних алгоритмів і їх конфігурацій у контрольованих умовах виконання, а також процедурні правила отримання, валідації, упорядкування й узагальнення експериментальних результатів, що забезпечують формування обґрунтованих висновків щодо доцільності використання конкретних криптографічних рішень у визначених сценаріях.

Метою роботи є розроблення та обґрунтування підходу до порівняльного оцінювання ефективності криптографічних алгоритмів на основі експериментальних досліджень у контрольованих умовах виконання, а також створення інструментальної системи, яка забезпечує автоматизоване виконання серій випробувань, фіксацію результатів в уніфікованому форматі, їх аналітичне опрацювання та підготовку практичних рекомендацій щодо вибору алгоритмів відповідно до умов застосування. Запропонований підхід передбачає опору на формалізовану методику, яка однозначно визначає порядок виконання операцій, склад контрольованих умов і правила інтерпретації отриманих даних.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

1. Проаналізувати предметну область застосування криптографічних перетворень у сучасних інформаційних системах та визначити вимоги, що впливають на вибір алгоритмів у практичних сценаріях.

2. Обґрунтувати вибір криптографічних алгоритмів і конфігурацій, що підлягають дослідженню, з урахуванням їх поширеності та типових режимів використання.

3. Сформувати методику експериментального оцінювання, визначивши сценарії виконання, правила однаковості умов і процедури перевірки коректності результатів.

4. Розробити інструментальну систему для запуску експериментів, збору показників, журналювання контексту виконання та формування артефактів, придатних для подальшого аналізу.

5. Реалізувати механізми підготовки та експорту результатів у форматах, що забезпечують подальше узагальнення і відтворюваність дослідження.

6. Провести серії експериментів у визначених умовах та отримати узгоджені дані для порівняльного аналізу алгоритмів у межах заданих сценаріїв.

7. Виконати аналітичне узагальнення результатів і сформувати висновки та рекомендації щодо доцільності застосування конкретних алгоритмів для відповідних умов експлуатації.

Наступним завданням є реалізація програмних компонентів системи, які виконують криптографічні операції та забезпечують коректний перебіг експериментальних серій за уніфікованими правилами запуску. Окремо формується набір тестових даних і описуються сценарії використання, які відображають характерні режими оброблення в інформаційних системах, включно з обробленням одиничних об’єктів, пакетною обробкою наборів даних, а також сценаріями, де важлива оперативність виконання коротких операцій. Після підготовки середовища виконуються експериментальні серії в різних умовах, що дозволяє дослідити поведінку алгоритмів при зміні параметрів експлуатації та виявити залежності, які мають практичне значення. Завершальним етапом є аналітичне опрацювання результатів, їх узагальнення у формі порівняльних представлень і ранжувань, а також формулювання висновків і рекомендацій, орієнтованих на обґрунтований вибір криптографічних рішень у конкретних умовах застосування. Додатково передбачається підготовка опису використання інструментальної системи та регламенту відтворення експериментів, що підсилює практичну цінність роботи.

У роботі фіксується конфігурація апаратної та програмної платформи, а також правила виконання експериментів, які забезпечують порівнюваність між алгоритмами в межах заданого середовища. Передбачається коректність використаних реалізацій криптографічних перетворень і дотримання єдиних налаштувань, що впливають на хід виконання операцій. Також приймається, що обрані тестові дані та сценарії є репрезентативними для типових режимів застосування у сучасних інформаційних системах і відображають характерні особливості оброблення та зберігання інформації. Водночас узагальнення висновків на інші платформи або інші режими експлуатації потребує

додаткового підтвердження шляхом повторення експериментів у відповідних умовах, оскільки результати можуть залежати від особливостей середовища виконання та організації підсистеми зберігання.

Очікуваними результатами роботи є методично узгоджена процедура порівняльного аналізу криптографічних алгоритмів, інструментальна система для проведення експериментів і фіксації результатів, а також систематизований набір підсумкових матеріалів, який забезпечує практичну інтерпретацію. До таких матеріалів належать порівняльні таблиці та графічні представлення, які відображають поведінку алгоритмів у різних умовах, узагальнені ранжування альтернатив за визначеними правилами узагальнення, а також висновки щодо сильних і слабких сторін кожного рішення в межах типових сценаріїв. Практичний результат доповнюється рекомендаціями щодо доцільності застосування алгоритмів залежно від характеру задачі, умов експлуатації та обмежень інформаційної системи, а також описом правил використання розробленої системи для повторення експериментів і перевірки висновків.

Узагальнену логіку виконання дослідження доцільно подати як перетворення від вхідних даних до вихідних результатів через формалізований процес. Вхідними даними є обрані криптографічні алгоритми та їх параметри, визначені сценарії застосування, підготовлені тестові дані й фіксовані умови середовища виконання. Процес реалізується через стандартизовані експериментальні серії, у межах яких виконуються криптографічні операції за єдиними правилами, здійснюється реєстрація результатів і подальше аналітичне узагальнення на основі узгоджених процедур. Вихідними результатами є структуровані порівняльні матеріали, узагальнені висновки та практичні рекомендації, які можуть бути використані для обґрунтованого вибору криптографічних рішень у межах визначених умов застосування.

Висновки до розділу

У розділі здійснено огляд предметної області та обґрунтовано актуальність порівняльного оцінювання ефективності криптографічних алгоритмів у

контексті сучасних інформаційних систем. Показано, що криптографічний захист є інтегрованою складовою життєвого циклу даних і використовується як під час зберігання, так і під час передавання інформації, а також поширюється на похідні артефакти оброблення, зокрема журнали, резервні копії та проміжні результати. Обґрунтовано, що в реальних умовах вибір криптографічного рішення має інженерний характер і визначається не лише криптографічною стійкістю, а й експлуатаційними наслідками, які проявляються через накладні витрати на обчислення та взаємодію з підсистемами введення-виведення, зберігання і керування ресурсами.

Сформульовано ключові вимоги до продуктивності та ресурсів, які визначають практичну придатність криптографічних перетворень у різних режимах експлуатації, включно з обробленням одиничних об’єктів даних, пакетними процесами, резервним копіюванням та сценаріями коротких операцій, характерних для асиметричних механізмів. Визначено, що ефективність криптографічних алгоритмів є багатокритеріальною характеристикою, тому її коректне оцінювання потребує формалізації системи показників, процедур їх узгодженого вимірювання та правил інтерпретації результатів. Запропоновано формальну основу для зіставлення алгоритмів, яка поєднує часові характеристики, ресурсні прояви та параметри зміни представлення даних, доповнені статистичним узагальненням повторних запусків і математичними схемами нормалізації та інтегрального порівняння.

2 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ТА МЕТОДИКА ОЦІНЮВАННЯ

2.1 Обґрунтування вибору алгоритмів для дослідження

Формування набору криптографічних алгоритмів для дослідження в межах магістерської роботи розглядається як методично важливий етап, оскільки саме від коректності цього вибору залежить репрезентативність експериментів і практична цінність подальших висновків. У прикладних інформаційних системах криптографія використовується як інфраструктурний механізм, який має забезпечувати захист даних у різних режимах експлуатації, від локального шифрування файлів і каталогів до організації захищеної взаємодії між сервісами та користувачами. Тому набір алгоритмів у дослідженні має одночасно відображати поширені практики використання, бути технічно зіставним у межах єдиного інструментального середовища та охоплювати різні принципи побудови криптографічних перетворень, що дозволяє виявляти не випадкові відмінності реалізацій, а закономірності поведінки алгоритмів різних класів. У даній роботі до набору включено симетричні та асиметричні алгоритми, а також враховано гібридну логіку їх спільного застосування, що відповідає реальним сценаріям, де симетричне шифрування виконує основне перетворення даних, а асиметричні механізми забезпечують розповсюдження або захист сеансових ключів. Саме такий підхід прямо узгоджується з концепцією дослідження, де аналіз будується на порівнянні поведінки алгоритмів у різних конфігураціях використання.

Критерії відбору алгоритмів у дослідженні ґрунтуються на чотирьох взаємопов’язаних принципах. Алгоритми мають бути релевантними для типових задач захисту даних у сучасних інформаційних системах, тобто реально застосовуватися для шифрування файлів, потоків даних або коротких повідомлень, а також використовуватися в задачах керування ключами та організації довіри. Вони мають мати стабільні програмні реалізації, які дозволяють відтворювати експерименти, забезпечувати однакові умови запуску та коректно інтерпретувати результати без змішування властивостей алгоритму з артефактами нестабільної реалізації. Алгоритми мають бути поширеними або

концептуально значущими, щоб результати порівняння можна було пов’язати з практикою, зокрема з типовими бібліотеками та платформами розроблення. Набір має включати різні класи і різні принципи побудови, оскільки порівняння лише близьких за дизайном рішень не дає повної картини і не дозволяє сформувати обґрунтовані рекомендації для різних сценаріїв експлуатації. У підсумку сформований набір включає AES, Serpent, Blowfish, ChaCha20, DES, 3DES, RSA, ElGamal та ECC, що забезпечує охоплення ключових груп сучасної прикладної криптографії.

В системі захисту доцільно розрізняти симетричні та асиметричні алгоритми, а також гібридні схеми, у яких вони комбінуються. Симетричні алгоритми є базовим інструментом для шифрування масивів даних, оскільки забезпечують ефективне перетворення великих обсягів інформації за рахунок відносно простих внутрішніх операцій і можливості оптимізації на рівні програмної реалізації або апаратної підтримки. Асиметричні алгоритми використовуються там, де необхідно організувати обмін ключами, виконати криптографічну ідентифікацію або забезпечити можливість шифрування для одержувача без попереднього спільного секрету. Практичний підхід до застосування криптографії у більшості систем полягає у гібридній організації: симетричний алгоритм перетворює самі дані, а асиметричний застосовується до невеликого обсягу службової інформації, передусім до сеансового ключа, що дозволяє поєднувати ефективність симетричних шифрів з перевагами асиметричного керування ключами.

У межах симетричної групи як базовий і водночас еталонний алгоритм обрано AES. Його включення зумовлене тим, що AES є загальновизнаним стандартом і широко використовується в прикладних системах, що робить його природною точкою відліку для порівняння. Структурно AES належить до підстановково-перестановочних мереж, де кожен раунд поєднує нелінійні підстановки, перестановки та лінійні перетворення, а також додавання раундових ключів. Такий дизайн забезпечує добре збалансовані криптографічні властивості і, що важливо для прикладного аналізу, дозволяє реалізаціям

ефективно використовувати оптимізації, у тому числі апаратну підтримку, яка часто присутня на сучасних процесорах. Саме тому AES доцільно включати не лише як популярний алгоритм, а і як практичний маркер того, як стандартизована і оптимізована криптографія поводиться в реальних умовах програмного виконання.

Алгоритм Serpent включено як концептуально важливий представник альтернативної лінії дизайну, який також розглядався в межах відбору стандарту AES і часто характеризується як більш консервативний за структурою. Для Serpent типовою є орієнтація на підвищений запас міцності за рахунок більшої кількості раундів і специфіки раундових перетворень. З методичної точки зору Serpent є цінним, оскільки дозволяє дослідити компроміс між запасом криптографічної стійкості та практичною придатністю у сценаріях, де важливі експлуатаційні характеристики. Крім того, Serpent має іншу організацію внутрішніх операцій, що робить його чутливим до відмінностей програмних оптимізацій і дозволяє виявляти, наскільки результати порівняння залежать від особливостей реалізації, а не лише від формального класу алгоритму.

Blowfish включено до дослідження як представника класичних блочних шифрів з фейстелевою структурою та історично широким застосуванням. Його методична цінність полягає в тому, що Blowfish демонструє підхід, де значну роль відіграє процедура підготовки ключового матеріалу і формування ключезалежних таблиць, що впливає на поведінку алгоритму в різних режимах використання. Водночас Blowfish має особливість, яка є принципово важливою для прикладних систем, а саме обмежений розмір блока, що у сучасних умовах розглядається як фактор ризику при шифруванні дуже великих масивів даних у деяких режимах роботи. Тому Blowfish у даному дослідженні виступає не як універсальна рекомендація, а як показовий приклад алгоритму, який може бути зручним у певних умовах, але потребує обережного застосування при роботі з великими обсягами або довготривалими потоками даних.

ChaCha20 обрано як представника потокових шифрів нового покоління, побудованих на операціях додавання, циклічних зсувів і побітового виключного

або. Включення ChaCha20 є принципово важливим для коректного порівняння, оскільки цей алгоритм відрізняється від класичних блочних шифрів не лише внутрішньою структурою, а і сценарієм використання. На практиці ChaCha20 часто розглядають як сильний кандидат для середовищ, де апаратна підтримка для блочних стандартів відсутня або де необхідна передбачувана поведінка програмної реалізації без залежності від спеціалізованих інструкцій. Потокова природа ChaCha20 робить його природним для шифрування даних у режимі послідовного оброблення та для задач, де важлива стабільність роботи при різних профілях введення та виведення. Таким чином, ChaCha20 у наборі алгоритмів дозволяє зіставити два підходи до симетричного шифрування та оцінити, як відмінності дизайну відображаються на поведінці в прикладному середовищі.

DES включено до дослідження як історично значущий стандарт, що тривалий час використовувався у промислових системах і донині може зустрічатися як елемент спадкової сумісності. З науково-методичної точки зору DES важливий як контрольний приклад, який демонструє обмеження класичних підходів у сучасних умовах і дозволяє показати, як застарілі криптографічні рішення впливають на інженерні характеристики системи. DES має фейстелеву структуру з фіксованою кількістю раундів і специфічним набором перестановок і підстановок, що робить його добре визначеним для порівняння, але водночас відображає обмеження, через які алгоритм вважається неприйнятним для сучасних вимог безпеки. У роботі DES доцільно інтерпретувати як реперний орієнтир, який дозволяє контрастно відобразити різницю між сучасними та спадковими підходами в симетричній криптографії.

Алгоритм 3DES, або потрійний DES, включено як проміжне рішення, яке історично використовувалося для продовження життєвого циклу DES шляхом багаторазового застосування базового перетворення. Методичний сенс 3DES полягає у тому, що він дозволяє дослідити наслідки прямого нарощування складності шляхом повторення раундів з різними ключами, що є типовим інженерним компромісом у періоди переходу між стандартами. 3DES широко

відомий як сумісний з інфраструктурами, що опиралися на DES, і тривалий час застосовувався у фінансових та корпоративних системах, де оновлення криптографії відбувається поступово. У рамках дослідження 3DES важливий як представник застарілого, але ще зустрічного у практиці класу рішень, що дає змогу порівнювати сучасні алгоритми не лише між собою, а й із тим рівнем криптографії, який залишився в багатьох системах через вимоги сумісності.

Для повноти картини та відповідності реальним сценаріям застосування до набору включено також асиметричні алгоритми RSA, ElGamal та ECC. Їх включення обґрунтоване тим, що в прикладних системах асиметрична криптографія рідко використовується для прямого шифрування великих даних, однак вона критично важлива для побудови довірчих відносин, організації обміну ключами і захисту службових елементів гібридних схем. Таким чином, аналіз асиметричних алгоритмів у даній роботі доповнює симетричну частину й дозволяє формувати рекомендації, які відповідають реальній інженерній практиці гібридного шифрування.

RSA є класичним і найбільш поширеним прикладом асиметричної криптографії, який підтримується практично всіма платформами та криптографічними бібліотеками. Його принципова основа полягає у використанні арифметики великих цілих чисел і складності факторизації, а центральною операцією є модульне піднесення до степеня. RSA є показовим для дослідження з двох причин. Він репрезентує традиційну система асиметричного шифрування та підпису, яка історично закріпилася як базова для інфраструктур відкритих ключів. Його обчислювальна природа, пов’язана з великою кількістю операцій над великими числами, робить RSA чутливим до реалізаційних оптимізацій і дозволяє наочно продемонструвати відмінність між симетричними і асиметричними криптографічними механізмами з позицій експлуатаційної поведінки. У роботі RSA розглядається як еталонний представник класичних схем, від якого зручно відштовхуватися при порівнянні з більш сучасними підходами.

ElGamal включено як представника асиметричних схем, що базуються на задачі дискретного логарифмування у мультиплікативних групах. На відміну від RSA, ElGamal є ймовірнісним шифром, де шифрування використовує випадковий параметр, що впливає на структуру шифротексту і на поведінку алгоритму при багаторазовому повторенні операцій. Для методики дослідження це важливо, оскільки дозволяє врахувати клас алгоритмів, де випадковість є невід'ємною частиною процесу, а отже, експериментальні серії мають бути організовані так, щоб коректно відображати типову поведінку алгоритму, а не одиничні реалізації. Крім того, ElGamal є показовим з точки зору практичного використання, оскільки історично застосовувався як криптографічний примітив для побудови протоколів, і його включення дозволяє зіставити різні підходи в асиметричній криптографії у межах одного експериментального середовища.

ECC, або криптографія на еліптичних кривих, включена як сучасний напрям асиметричних перетворень, який забезпечує високий рівень криптографічної стійкості за рахунок складності задачі дискретного логарифмування в групах точок еліптичних кривих. Практичний інтерес ECC пояснюється тим, що ці алгоритми широко використовуються в сучасних протоколах, особливо там, де важлива компактність ключових даних і ефективність операцій у середовищах з обмеженими ресурсами. З позиції структури обчислень ECC спирається на операції додавання точок і множення точки на скаляр, що формує іншу обчислювальну система порівняно з RSA та ElGamal. Для даного дослідження ECC є критично важливим компонентом, оскільки дозволяє порівняти класичні підходи асиметричної криптографії з сучасними, а також сформувати рекомендації щодо вибору механізмів для різних сценаріїв застосування гібридних схем.

Щоб забезпечити коректність порівняння, у межах дослідження параметри застосування алгоритмів узгоджуються на рівні сценаріїв і процедур використання. Для симетричних алгоритмів це означає єдині правила роботи з файлами та каталогами, однаковий підхід до оброблення даних і збереження результатів, а також узгоджені режими виконання, включно з можливістю

дослідження одночасної обробки для оцінювання впливу паралельності. Для асиметричних алгоритмів узгодження передбачає однакові принципи роботи з ключовими парами та однакові процедури багаторазового виконання коротких операцій, що дозволяє отримувати зіставні дані в межах одного інструментального середовища. Важливо, що в роботі ключі розглядаються насамперед як параметри алгоритмів у рамках експериментальної постановки, тоді як повноцінна інфраструктура управління ключами свідомо не є предметом даного підрозділу, оскільки така інфраструктура є окремим великим напрямом дослідження і виходить за межі сформульованої задачі.

Сформований набір алгоритмів створює достатню основу для подальшого експериментального дослідження та аналітичного узагальнення, а вихідними артефактами підрозділу можуть слугувати узагальнювальна таблиця відповідності алгоритмів їх класам, сценаріям застосування та мотивам включення до дослідження, а також концептуальна схема розподілу ролей симетричних і асиметричних перетворень у типових інформаційних системах.

2.2 Вибір методів дослідження та експериментів

Побудова методики дослідження та організація експериментів у даній роботі ґрунтується на порівняльному експериментальному підході як основному способі отримання об’єктивних даних про поведінку криптографічних алгоритмів у прикладному середовищі. Такий підхід дає змогу зіставляти альтернативи в однакових умовах виконання та робити висновки, які мають практичну значущість для інформаційних систем, де криптографічні перетворення є частиною реальних процесів оброблення, зберігання та передавання даних. Водночас експериментальні процедури доповнюються теоретичним аналізом, який виконує пояснювальну функцію: дозволяє інтерпретувати спостережувану поведінку алгоритмів через їх внутрішні принципи побудови, типові операції та характер взаємодії з програмно-апаратним середовищем. Сукупність цих підходів забезпечує не лише фіксацію

результатів, а й формування аргументованих пояснень причин відмінностей між алгоритмами у різних сценаріях застосування.

Ключовою методичною вимогою до експериментів є відтворюваність і прозорість процедур. Це означає, що для кожного запуску має бути однозначно визначено, які вхідні дані використовуються, які параметри алгоритмів задані, у якому середовищі виконується оброблення та за якими правилами реєструється результат. Для досягнення цього у роботі вводиться регламент проведення експерименту як формалізована послідовність кроків, де етапи підготовки даних, запуску криптографічних операцій, перевірки коректності та збереження результатів виконуються однаково для всіх алгоритмів. Прозорість забезпечується журналюванням, у межах якого фіксуються характеристики середовища виконання, параметри запуску та ознаки коректності завершення операцій. Така організація дає можливість повторити експериментальну серію в тому ж середовищі або перенести її на іншу платформу, зберігаючи порівнюваність процедур.

Експерименти орієнтовані на відтворення типових сценаріїв експлуатації криптографічних механізмів в інформаційних системах. Розглядається сценарій оброблення одиничного об’єкта даних, який відповідає поширеним випадкам шифрування окремого файла або окремого повідомлення та дозволяє аналізувати поведінку алгоритму для коротких і відносно незалежних операцій. Відтворюється пакетна обробка наборів файлів, що характерно для задач захисту каталогів, колекцій документів, а також для процесів, де дані формуються багатьма різнорідними об’єктами. Відтворюється режим резервного копіювання та архівації, де криптографічні перетворення працюють як довготривалий процес і важливими стають стабільність виконання та узгодженість роботи з підсистемою зберігання. Розглядаються короткі операції, характерні для асиметричних перетворень і задач керування ключами, де важливо оцінити придатність алгоритмів у повторюваних, частих і транзакційних викликах, які типово супроводжують роботу захищених протоколів і сервісів.

У роботі виділяється фактор середовища зберігання, оскільки в задачах шифрування файлів результативність криптографічних операцій нерозривно пов’язана з організацією читання та запису, а різні типи накопичувачів по-різному впливають на перебіг оброблення. Окремим фактором є обсяг і структура вхідних даних, оскільки алгоритми можуть демонструвати різну поведінку залежно від того, чи обробляються великі суцільні масиви або множина невеликих об’єктів з частими зверненнями до файлової системи. Ще одним фактором є режим виконання обробки, оскільки послідовне виконання та паралельна організація задачі по-різному розкривають властивості реалізацій і їх взаємодію з обчислювальними ресурсами. Додатково враховується фактор платформи виконання у випадку, якщо дослідження передбачає порівняння результатів на різних обчислювальних машинах або в різних конфігураціях середовища, що дозволяє оцінити переносимість висновків і стійкість спостережуваних закономірностей.

Для забезпечення коректності експерименту вводяться контрольовані умови та правила стандартизації. Фіксується програмне середовище, що включає операційну систему, версії бібліотек криптографічних реалізацій та залежних компонентів, а також налаштування, які можуть впливати на виконання обчислювальних задач. Встановлюється єдина процедура підготовки даних і запуску операцій, щоб усунути відмінності, пов’язані не з алгоритмами, а з різним способом подання вхідних даних чи різними діями перед запуском. Для мінімізації впливу зовнішніх чинників регламентується контроль фонових процесів і забезпечується стабільність середовища виконання, зокрема через проведення експериментів у порівнюваних умовах завантаженості системи. Окремо визначається порядок виконання серій, який зменшує ризик систематичних зсувів, коли один і той самий алгоритм отримує перевагу або недолік через послідовність запусків, нагрівання системи, накопичення тимчасових даних або інші ефекти тривалої роботи. Завдяки цьому порівняння будується на максимально «чистій» основі, де відмінності між алгоритмами пов’язані насамперед з їх властивостями та характером реалізації.

Процедура повторів застосовуються для підвищення надійності результатів і зменшення впливу випадкових коливань середовища. Кожен сценарій дослідження передбачає багаторазове відтворення запусків за однакових умов, що дозволяє отримати узгоджену картину поведінки алгоритмів і зменшити залежність висновків від одиничних випадкових відхилень. Перед основним збором результатів передбачається етап прогріву та стабілізації виконання, який зменшує вплив первинних ефектів на кшталт ініціалізації середовища, підготовки внутрішніх структур та першого звернення до даних. Усі результати серій зберігаються у вигляді сирих записів журналу з фіксацією умов запуску, що дозволяє проводити подальше аналітичне опрацювання, виконувати перевірку коректності та повторювати обчислення узагальнень без втрати первинної інформації.

Критерії завершення експерименту та перевірка коректності виконання є необхідною умовою достовірності. Для симетричних і асиметричних перетворень перевіряється коректність зворотного перетворення, тобто відповідність отриманих після розшифрування даних вихідному вмісту. Для файлових сценаріїв додатково контролюється цілісність результатів, що включає перевірку відсутності помилок запису, коректність формування вихідних об’єктів і узгодженість структури оброблених наборів. У методиці передбачено правила обробки збоїв, ситуацій перевищення допустимого часу виконання та виняткових випадків, які можуть виникати через помилки введення, обмеження доступу до ресурсів або нестабільність середовища. Такі випадки фіксуються в журналі як окремі результати з описом причини, що запобігає прихованому спотворенню картини та дозволяє коректно відокремлювати успішні серії від некоректних запусків.

Для документування методики та забезпечення її подальшого відтворення в роботі доцільно навести два ключові артефакти. Діаграма процесу експерименту наочно фіксує послідовність етапів підготовки, запуску, реєстрації результатів, валідації та збереження даних. Таблиця факторів експерименту систематизує змінні параметри дослідження, встановлює відповідність між

факторами й варійованими умовами, окреслює очікуваний характер їх впливу та визначає спосіб контролю, що підвищує прозорість дизайну і спрощує перевірку методичної коректності. У сукупності описані принципи визначають науково коректну процедуру проведення експериментів, орієнтовану на зіставність умов, відтворюваність результатів і зменшення похибок, що створює основу для подальшого збору даних і їх аналітичного узагальнення в наступних підрозділах.

2.3 Показники, що вимірюються, та спосіб їх отримання

Показники доцільно структурувати у чотири взаємодоповнювальні групи, кожна з яких відображає окремий аспект поведінки криптографічних алгоритмів у прикладному середовищі. Часові характеристики фіксують динаміку виконання криптографічних операцій і демонструють, як алгоритм поводиться в різних сценаріях перетворення даних. Показники ресурсного профілю описують інтенсивність використання обчислювальних ресурсів та дають змогу оцінити вплив криптографічних операцій на керованість і стабільність роботи платформи загалом. Показники зміни представлення даних характеризують особливості формування вихідного захищеного подання й є важливими для систем зберігання та передавання інформації, оскільки криптографічний формат може спричиняти додаткові накладні витрати під час зберігання та обміну. Показники якості експерименту та стабільності серій забезпечують контроль повторюваності та узгодженості отриманих даних, без чого неможливо сформувати науково коректні висновки навіть за умови технічно правильного виконання окремих вимірювань [27].

Отримання часових характеристик організовується як вимірювання інтервалів між фіксованими точками початку та завершення операції. Нехай момент початку операції позначено як t_{staer}, а момент завершення як t_{end}. Тоді тривалість шифрування визначається як $T_{enc} = t_{end} - t_{start}$, а тривалість розшифрування як $T_{dec} = t_{end} - t_{start}$, де у кожному випадку часові точки відповідають одному й тому самому рівню абстракції. У даній методиці часові точки прив’язуються до виконання криптографічної процедури в прикладному

середовищі так, щоб інтервал відображав завершену корисну дію: для файлів це означає, що старт фіксується безпосередньо перед запуском перетворення даних, а завершення фіксується після отримання повністю сформованого результату, готового до збереження або подальшої обробки. Для пакетної обробки наборів файлів застосовується той самий принцип, але часовий інтервал охоплює послідовність операцій для всього набору як єдину задачу. Це дозволяє оцінювати алгоритм у режимі, який відповідає реальному використанню, де важливими є не лише внутрішні обчислення, а й взаємодія з обробленням множини об’єктів та з підсистемою введення і виведення.

Для коректності часових вимірювань використовується монотонне джерело часу з високою роздільною здатністю, що виключає спотворення, пов’язані зі зміною системного часу. Додатково уніфікується склад дій, які входять у вимірюваний інтервал: у нього не включаються дії користувацького інтерфейсу та дії, що не є частиною алгоритмічного перетворення, наприклад ручний вибір файлу. Натомість включаються етапи, які є невід’ємною частиною прикладної операції шифрування або розшифрування в системі, зокрема формування вихідного об’єкта даних і його коректне завершення. Такий підхід робить часові характеристики інтерпретованими з позицій практичного використання.

Окремі правила застосовуються для асиметричних перетворень і операцій з ключовим матеріалом, оскільки ці операції мають іншу природу. Виділяється тривалість формування ключової пари, яка визначається як $T_{key} = t_{end} - t_{start}$ для процедури генерації ключів. Для коротких операцій шифрування і розшифрування, що повторюються багаторазово, застосовується серійний запуск, у якому обчислюється час виконання сукупності однакових операцій у межах одного сценарію, після чого результат інтерпретується як характеристика типового короткого виклику. Така організація зменшує вплив накладних витрат середовища виконання на одиничну операцію та дозволяє коректніше порівнювати алгоритми, у яких короткі виклики можуть мати різну внутрішню структуру [28].

Для інтерпретації часових результатів у задачах оброблення даних, є темп перетворення, який описує обсяг оброблених даних за одиницю часу. Нехай обсяг вхідних даних позначено як S, тоді темп шифрування можна задати як $V_{enc} = S \div T_{enc}$, а темп розшифрування як $V_{dec} = S \div T_{dec}$. Для пакетної обробки набору файлів темп доцільно визначати як відношення сумарного обсягу оброблених даних до сумарного часу виконання серії, оскільки саме так описується пропускна здатність у реальному процесі. Цей показник є інтерпретованим, оскільки дозволяє порівнювати алгоритми при різних структурах наборів даних і різній кількості об'єктів, фокусуючись на загальному ефекті для задачі.

Фіксація ресурсного профілю виконується для того, щоб оцінити, як криптографічні операції впливають на обчислювальне середовище і наскільки вони можуть обмежувати паралельне виконання інших задач. Для центрального процесора доцільно реєструвати як середній рівень використання ресурсу за час операції, так і пікові значення, що відображають можливі короткочасні перевантаження. Узгоджений спосіб оцінювання навантаження процесора базується на співвідношенні процесорного часу, витраченого процесом, до тривалості операції у реальному часі. Нехай витрачений процесорний час позначено як C, а тривалість операції як T, тоді узагальнений показник інтенсивності використання процесора можна подати як $U = C \div T$. Для багатопотокового виконання цей показник інтерпретується з урахуванням того, що процесорний час акумулюється по потоках, а тому значення відображає сумарну обчислювальну роботу, виконану процесом під час операції. Пікові значення навантаження фіксуються за допомогою періодичного опитування стану процесу з однаковим інтервалом дискретизації для всіх запусків, причому часові мітки опитування синхронізуються з початком і завершенням операції, щоб оцінка відповідала саме експериментальній дії.

Для оперативної пам'яті фіксується профіль використання, який характеризує, чи потребує алгоритм значних буферів, таблиць або проміжних структур. Практично значущими є два узагальнення: типовий рівень

використання пам’яті за час операції та максимальний обсяг пам’яті, до якого процес піднімався під час виконання. Уніфікованою основою служить показник резидентного обсягу пам’яті процесу, який відображає фактичне розміщення в оперативній пам’яті і є порівнюваним між запусками. Для кожного запуску реєструються значення, отримані шляхом періодичного опитування процесу під час виконання, а також виділяється найбільше зафіксоване значення як характеристика пікового споживання. Синхронізація моментів збору для процесора і пам’яті забезпечує, що обидва профілі відображають один і той самий часовий інтервал виконання криптографічної операції, а не довільний фрагмент роботи системи [29, 30].

Оцінювання зміни обсягу даних після перетворення є необхідним, оскільки криптографічні алгоритми та способи пакування результатів можуть збільшувати обсяг збережених або переданих даних, що впливає на вимоги до дискового простору, каналу зв’язку та часу введення і виведення. Нехай обсяг вхідних даних позначено як S_{in}, а обсяг вихідних даних як S_{out}. Тоді коефіцієнт зміни обсягу задається як $K = S_{out} \div S_{in}$, а відносна накладна зміна обсягу як $O = (S_{out} - S_{in}) \div S_{in}$. Для коректності цього показника правила вимірювання фіксують, що S_{in} визначається як фактичний розмір вхідного об’єкта даних, а S_{out} як розмір сформованого криптографічного результату в тому форматі, в якому він зберігається або передається системою. Окремо контролюється, щоб у S_{out} не потрапляли зовнішні представлення, які залежать від засобу збереження, наприклад не обов’язкове кодування для текстового подання, якщо воно не використовується у реальному сценарії. Також уніфікується підхід до службових компонентів криптографічного формату, щоб порівняння між алгоритмами було коректним: якщо результат завжди містить службову частину, вона повинна входити у вимірювання, оскільки є частиною фактичного виходу системи.

Окрема увага приділяється показникам якості експерименту та стабільності серій, оскільки будь-яке вимірювання в реальному середовищі підпадає під вплив фонового навантаження, планувальника операційної системи та інших випадкових факторів. Для кожного сценарію передбачається

повторення запусків за однакових умов, а результати узагальнюються статистично. Нехай x_r є значенням певного показника у повторі з номером r, тоді середнє значення серії визначається як $\overline{x} = (\sum x_r) \div R$, де R є кількістю повторів. Розкид результатів описується дисперсією $s^2 = (\sum(x_r - \overline{x})^2) \div (R - 1)$ та стандартним відхиленням. Для інтерпретації стабільності зручно використовувати відносний розкид, який задається як $cv = s \div \overline{x}$ і дозволяє зіставляти стабільність між серіями для різних алгоритмів. Такий опис дає змогу відрізняти ситуації, коли алгоритм демонструє передбачувану поведінку, від ситуацій, коли результати є чутливими до випадкових коливань середовища. Усі ці узагальнення мають сенс лише за умови коректного очищення даних, тому методика передбачає процедури валідації та відсіву некоректних запусків [31].

Валідація результатів починається з перевірки коректності перетворення. Для симетричних алгоритмів перевіряється повна відповідність даних після розшифрування вихідним даним. Для файлових сценаріїв це реалізується через порівняння вмісту або через порівняння контрольних представлень, що дає однозначний критерій правильності. Для асиметричних сценаріїв перевіряється коректність виконання операцій на рівні протоколу експерименту, зокрема можливість відновити початкове повідомлення, якщо це передбачено сценарієм, або коректність роботи з ключовим матеріалом у межах заданих процедур. Запуски, що завершилися помилкою, перериванням або некоректним результатом, не включаються до узагальнень як повноцінні значення показників, але фіксуються в журналі як окремий тип події, щоб не втратити інформацію про відмови та їх можливі причини. Для роботи з аномальними значеннями застосовуються прозорі правила очищення, які не маскують дані, а пояснюють їх. Наприклад, можуть використовуватися робастні оцінки, що спираються на медіану та медіанне абсолютне відхилення, або правила на основі квартилів, які дозволяють ідентифікувати значення, що суттєво вибиваються з основної маси спостережень і, як правило, відповідають зовнішнім впливам, а не властивостям алгоритму [32].

Результати збору даних доцільно подавати у двох формах: у вигляді узагальнювальної таблиці показників та у вигляді прикладу запису журналу експерименту. Узагальнювальна таблиця фіксує для кожного показника спосіб отримання, одиниці вимірювання, інтерпретацію та обмеження застосування, що забезпечує однозначність подальшого аналізу. Узагальнений перелік показників і відповідні правила їх трактування наведено в таблиці 2.1.

Таблиця 2.1. Показники експерименту та спосіб їх отримання

Показник	Спосіб отримання	Одиниці	Інтерпретація	Обмеження
Тривалість шифрування T_{enc}	Різниця монотонних часових міток для операції шифрування	секунда	Час виконання прикладної операції шифрування у заданому сценарії	Залежить від стабільності середовища та режиму введення і виведення
Тривалість розшифрування T_{dec}	Різниця монотонних часових міток для операції розшифрування	секунда	Час відновлення даних у заданому сценарії	Аналогічні обмеження для середовища та носія

Продовження таблиці 2.1

Показник	Спосіб отримання	Одиниці	Інтерпретація	Обмеження
Тривалість формування ключів T_{key}	Різниця часових міток для процедури генерації ключів	секунда	Придатність алгоритму для сценаріїв, де потрібне швидке створення ключового матеріалу	Значною мірою залежить від реалізації та джерела випадковості
Темп перетворення V_{enc}, V_{dec}	Відношення обсягу даних до тривалості операції	байт за секунду	Пропускна здатність перетворення у режимі серії або пакета	Коректний за узгоджених правил вимірювання обсягу
Використання процесора U	Відношення процесорного часу процесу до тривалості операції	частка або відсоток	Інтенсивність обчислювальної роботи під час операції	Потребує однакових умов фонового навантаження
Пікове використання пам’яті	Максимум резидентного обсягу пам’яті під час операції	байт	Найвища потреба в оперативній пам’яті під час виконання	Залежить від буферизації та реалізації

Продовження таблиці 2.1

Показник	Спосіб отримання	Одиниці	Інтерпретація	Обмеження
Коефіцієнт зміни обсягу U	Відношення розміру вихідних даних до вхідних	безрозмірний	Накладні витрати формату захищених даних	Має враховувати лише фактичний формат збереження або передавання
Стабільність серії s, cv	Статистичні узагальнення повторів	як у показника	Відтворюваність і чутливість до випадкових впливів	Потребує достатньої кількості повторів і коректного очищення

Сформована система показників і уніфікована процедура їх отримання забезпечують зіставність результатів між алгоритмами та сценаріями виконання. Однак самі по собі зібрані дані ще не дають підстав для узагальнених висновків, доки не буде визначено правила їх попередньої обробки, нормалізації та агрегування.

2.4 Аналітична підсистема оцінювання

Аналітична підсистема оцінювання в межах даної роботи виконує роль зв’язувальної ланки між експериментальною частиною та підсумковими висновками. Її призначення полягає в тому, щоб перетворити сирі журнальні записи запусків на інтерпретовані узагальнення, які дозволяють коректно порівнювати криптографічні алгоритми у різних умовах використання та

формувати практичні рекомендації. На відміну від етапу збору даних, який фіксує фактичну поведінку реалізацій, аналітична підсистема формалізує правила підготовки, зіставлення та узагальнення результатів. Таке розмежування є принциповим: воно підвищує відтворюваність дослідження, оскільки одна й та сама сукупність експериментальних записів може бути повторно проаналізована за незмінними правилами, а також забезпечує прозорість, оскільки кожний перехід від первинних даних до підсумкових висновків має описану процедуру.

Вхідними даними аналітичної підсистеми є набір записів запусків, сформований інструментальною системою, де для кожного запуску зафіксовано ідентифікатор алгоритму, сценарій, умови виконання, атрибути об’єкта оброблення, часові характеристики, профіль використання ресурсів, зміна обсягу результату, а також ознаки коректності завершення. Для формального опису зручно розглядати множину алгоритмів A, множину сценаріїв C та множину показників K. Нехай $x_{a,c,s,r,k}$ позначає значення показника k для алгоритму a у сценарії c, для конкретного об’єкта оброблення s та повтору r. Завдання аналітичної підсистеми полягає у відображенні набору x у узагальнені оцінки, рейтинги й рекомендації, зберігаючи при цьому коректність порівняння і не змішуючи властивості алгоритмів із випадковими впливами середовища.

Обробка даних починається зі структуризації та перевірки повноти записів. На цьому етапі здійснюється контроль наявності обов’язкових полів, перевіряється цілісність зв’язків між записами, а також відсіюються або маркуються запускі, що завершилися помилкою чи мають ознаки некоректного виконання. Одночасно виконується узгодження форматів журналів: значення приводяться до єдиних типів, стандартизуються ідентифікатори сценаріїв і алгоритмів, синхронізуються назви полів, що походять з різних модулів системи. Після цього дані групуються за алгоритмами, сценаріями та умовами виконання, що формує базову багатовимірну структуру, придатну для подальшого аналізу. Практично це означає, що аналітична підсистема переходить від плоского журналу подій до наборів спостережень, згрупованих за ключовими вимірами, і створює основу для порівняння за однакових умов.

Наступним етапом є нормалізація та уніфікація напрямку оптимальності. Оскільки різні показники мають різні шкали і різний сенс, пряме порівняння значень без перетворення є некоректним. Тому для кожного показника вводиться правило перетворення до безрозмірної форми та однонапрямної інтерпретації, коли більше значення означає кращий результат. Якщо для певного показника природною є інтерпретація менше означає краще, використовується інверсійне перетворення. Формально це можна задати так: для кожного сценарію $c \in C$ та показника $k \in K$ визначається опорний інтервал на основі спостережуваних значень серед усіх алгоритмів, після чого обчислюється нормалізоване значення $z_{A,c,k}$ у межах від нуля до одиниці [33]. Для показників із напрямком більше означає краще застосовується перетворення що виуонуються за формулою (2.1):

$$z_{A,c,k} = \frac{x_{A,c,k} - x_{c,k}^{min}}{x_{c,k}^{max} - x_{c,k}^{min}}, \tag{2.1}$$

де $z_{A,c,k}$ – нормалізоване значення k-го показника для алгоритму A у сценарії c (у межах [0;1]);

$x_{A,c,k}$ – агреговане (табличне) значення k-го показника для алгоритму A у сценарії c;

$x_{c,k}^{min}$ – мінімальне значення показника k серед усіх алгоритмів у сценарії c;

$x_{c,k}^{max}$ – максимальне значення показника k серед усіх алгоритмів у сценарії c;

Для показників із напрямком бменьше означає краще застосовується перетворення що виуонуються за формулою (2.2):

$$z_{A,c,k} = \frac{x_{c,k}^{max} - x_{A,c,k}}{x_{c,k}^{max} - x_{c,k}^{min}}, \tag{2.2}$$

де $z_{A,c,k}$ – нормалізоване значення k-го показника (у межах [0;1]);

$x_{A,c,k}$ – агреговане значення k-го показника для алгоритму A у сценарії c;

$x_{c,k}^{min}$ – мінімальне значення показника k серед усіх алгоритмів у сценарії c;

$x_{c,k}^{max}$ – максимальне значення показника k серед усіх алгоритмів у сценарії c;

Ці правила гарантують, що подальше узагальнення будується на порівнюваних величинах і що жоден показник не домінує лише через масштаб значень.

Після нормалізації здійснюється агрегація результатів і формування інтегральних оцінок. На першому рівні агрегуються повтори, що зменшує вплив випадкових коливань середовища. Для кожного алгоритму a, сценарію c, об'єкта s і показника k обчислюється узагальнене значення, наприклад у вигляді середнього або медіани, залежно від прийнятого правила робастності. На другому рівні агрегуються результати по наборах даних, коли сценарій включає множину об'єктів. Тут важливо, щоб узагальнення відображало реальну задачу: для показників, що залежать від обсягу даних, допустимим є зважування вкладом об'єкта в загальний обсяг, тоді як для показників, що відображають короткі операції, зручніше зберігати однакову вагу повторів як однорідних спостережень. У підсумку для кожного алгоритму та сценарію формується вектор нормалізованих і агрегованих значень $z_{A,c,k}$.

Інтегральна оцінка в межах сценарію визначається як узагальнення по множині показників з урахуванням пріоритетів сценарію. Для цього вводяться ваги $w_{c,k}$, які відображають відносну важливість кожного показника в конкретному сценарії і задовольняють умову $\sum_{k\in K} w_{c,k} = 1$. Тоді інтегральний бал алгоритму у сценарії задається за формулою (2.3):

$$I(A,c) = \sum_{k\in K} w_{c,k} z_{A,c,k}. \qquad (2.3)$$

де $I(A, c)$ – сценарний індекс ефективності алгоритму A у сценарії c;

$w_{c,k}$ – вага (пріоритет) показника k для сценарію c, причому $\sum_{k \in K} w_{c,k} = 1$ та $w_{c,k} \geq 0$;

$z_{A,c,k}$ – нормалізоване значення показника k (див. (2.1 – 2.2));

K – множина показників оцінювання.

Таке визначення дозволяє отримувати рейтинг алгоритмів у межах одного сценарію та одночасно зберігати інтерпретованість, оскільки вклад кожного показника контролюється вагами. За потреби, щоб сильніше штрафувати слабкі сторони, може застосовуватися геометричне узагальнення, яке задаєється за фомулою (2.4):

$$I_g(A, c) = \prod_{k \in K} (z_{A,c,k})^{w_{c,k}}, \tag{2.4}$$

де $I_g(A, c)$ – геометричний сценарний індекс ефективності алгоритму A у сценарії c;

$z_{A,c,k}$ – нормалізація значення показника k (див. (2.1 – 2.2));

$w_{c,k}$ – ваги показників у сценарії c ($\sum w_{c,k} = 1, w_{c,k} \geq 0$);

K – множина показників.

Побудова рейтингів і сценарний вибір реалізуються як перетворення інтегральних оцінок у впорядкований список альтернатив. У межах кожного сценарію формується порядок алгоритмів за спаданням, а також, за необхідності, формується супровідне пояснення, яке вказує на ключові причини позиції алгоритму, наприклад через найвагоміші внески та найслабші складові вектора $z_{A,c,k}$. Далі здійснюється узагальнення по сценаріях, якщо робота передбачає загальний висновок для сукупності умов [34]. Для цього вводяться ваги сценаріїв π_c, що відображають важливість сценаріїв для прикладної області, і визначається глобальна оцінка за формулою (2.5):

$$I_A = \sum_{c \in C} \pi_c I(A, c). \quad (2.5)$$

де π_c – важливість (вага) сценарію c у загальному профілі застосування, причому $\sum_{c \in C} \pi_c = 1$ та $\pi_c \geq 0$;

$I(A, c)$ – сценарний індекс ефективності (див. (2.3));

I_A – інтегральна оцінка алгоритму A по всіх сценаріях;

C – множина сценаріїв.

У такий спосіб рейтинги набувають практичного сенсу: рекомендація формується не для абстрактної ситуації, а для зафіксованого профілю використання, який визначається сценаріями та їх важливістю.

Окремий компонент аналітичної підсистеми призначений для аналізу компромісів і недомінованих альтернатив. У багатокритеріальних задачах часто відсутній єдиний алгоритм, який був би кращим за всіма показниками одночасно. Тому поряд із рейтингами на основі інтегрального бала доцільно застосовувати аналіз Парето, який виділяє множину рішень без однозначного домінування. Для цього визначається домінування: алгоритм A домінує алгоритм B у сценарії c, якщо для всіх показників $z_{A,c,k} \geq z_{B,c,k}$ і для принаймні одного показника нерівність є строгою. Алгоритми, які не домінуються жодним іншим, утворюють множину Парето. Її наявність означає, що остаточний вибір має спиратися на пріоритети сценарію або на додаткові практичні обмеження, і саме тут інтегральні правила з вагами виконують роль інструмента уточнення рішення. Таким чином, аналітична підсистема не лише ранжує, а й пояснює природу компромісів, що підвищує наукову обґрунтованість висновків [35].

2.5 Вхідні дані та вихідні результати дослідження

Вхідні дані дослідження складаються з тестових наборів, що репрезентують типові сценарії застосування криптографії в інформаційних системах. До таких наборів належать як одиничні об’єкти даних, так і колекції файлів різних типів, що дає змогу відтворити сценарій пакетної обробки, а також

дані, наближені до практик резервного копіювання й архівації. Опис тестових наборів має містити їх склад, принципи формування, логіку структурування каталогів, а також правила відбору вмісту, щоб уникнути випадкового перекосу в бік певного типу даних. Варіативність наборів задається через керовані зміни структури та складу даних, що дозволяє дослідити поведінку алгоритмів в умовах, близьких до реальної експлуатації. Для забезпечення порівнюваності важливо, щоб усі алгоритми обробляли тотожні набори даних, а зміни між серіями були документовані як частина протоколу експерименту. Окремо фіксуються параметри запусків та конфігурація середовища, включно з описом платформи виконання, програмного оточення, версій залежностей та режимів доступу до підсистеми зберігання. Така фіксація є необхідною умовою інтерпретованості результатів, оскільки відмінності середовища можуть суттєво впливати на перебіг криптографічних операцій.

Вхідні налаштування інструментальної системи оформлюються як конфігураційні профілі запуску, що відокремлюють опис експерименту від реалізаційного коду. У типовому випадку конфігурація визначає перелік алгоритмів, що запускаються в межах серії, порядок обробки даних, сценарії виконання, режими оброблення для одиничних та пакетних задач, а також правила журналювання й перевірки коректності. Параметри сценаріїв та режимів задаються у формі, яка дозволяє автоматизовано відтворювати експеримент крок за кроком без ручного втручання, що мінімізує ризик випадкових помилок оператора та підвищує повторюваність. Регламент відтворення експерименту має описувати послідовність дій від підготовки даних і завантаження профілю до виконання серій і формування вихідних матеріалів, включно з правилами перевірки, що підтверджують коректність завершення кожного запуску. У такий спосіб конфігураційні профілі виступають ключовим артефактом, який дозволяє відновити дослідження у тому самому вигляді або перенести його на інше середовище зі збереженням структури процедур.

Вихідні артефакти експериментів формуються на кількох рівнях узагальнення. Базовим рівнем є сирі журнали запусків, у яких для кожної

операції фіксуються параметри середовища, сценарій, обраний алгоритм, характеристики вхідних даних, результати виконання та ознаки коректності. Саме сирі журнали є первинним джерелом істини для подальшої перевірки та повторного аналізу, тому вони мають зберігатися без втрат і без ручного редагування. Наступним рівнем є зведені таблиці, які агрегують записи за алгоритмами, сценаріями та умовами виконання і подають результати у вигляді, придатному для порівняльного аналізу. Додатково формуються графічні представлення, що ілюструють закономірності та відмінності між алгоритмами для різних сценаріїв, оскільки візуалізація підсилює інтерпретацію та дозволяє виявляти тренди, які складно помітити у сирих таблицях. Завершальним рівнем є підсумкові рейтинги й рекомендації, які формуються аналітичною підсистемою на основі узгоджених правил нормалізації, агрегування та сценарного узагальнення. Важливо, що рекомендації подаються не як універсальна відповідь для всіх умов, а як результат інтерпретації в межах визначених сценаріїв використання, що відповідає практичній природі задачі вибору криптографічних механізмів.

Формати збереження та подання результатів визначаються вимогами до автоматизованої обробки, зручності перевірки та придатності до включення в текст роботи й додатки. Журнали доцільно зберігати у структурованому машинозчитуваному форматі, який підтримує однозначні поля, часові мітки та ідентифікатори серій, що дозволяє відтворювати аналітичні обчислення без ручної підготовки. Зведені таблиці варто подавати у форматі, сумісному з табличними процесорами та інструментами аналізу даних, щоб забезпечити можливість незалежної перевірки й повторного використання. Підсумкові звіти й графічні матеріали доцільно формувати у форматах, придатних для включення в документ магістерської роботи та для подальшого поширення, з обов’язковим збереженням версійності, щоб було зрозуміло, до якої серії експериментів належить конкретний графік або таблиця. Узгоджені формати є важливою частиною методики, оскільки забезпечують прозорість переходу від виконання експериментів до представлення результатів у тексті роботи.

Вимоги до відтворюваності підсумовують мінімальний набір умов і матеріалів, необхідний для повторення дослідження та перевірки висновків. Для повного відтворення потрібні тестові набори або інструкція їх формування, конфігураційні профілі запусків, опис середовища виконання та версій програмних компонентів, а також інструментальна система, що реалізує процедури запуску та журналювання. Для перевірки висновків достатнім є мінімальний набір матеріалів, що включає сирі журнали запусків, зведені таблиці, правила аналітичного узагальнення та підсумкові рейтинги з поясненням сценарної інтерпретації. Узагальнений перелік матеріалів, потрібних для відтворення та перевірки результатів, наведено в таблиці 2.2. У роботі доцільно додавати ці матеріали до додатків або надавати їх як супровідний пакет, щоб незалежний дослідник міг відтворити принаймні ключові етапи аналізу й переконатися, що висновки відповідають зафіксованим даним.

Таблиця 2.2. Артефакти дослідження, формати та призначення

Артефакт	Формат	Зміст	Де використовується в роботі
Опис тестових наборів	Текстовий документ	Склад наборів, принципи формування, структура даних	Розділ про методику, додатки
Тестові дані	Структура файлів	Набори для одиничних і пакетних сценаріїв	Проведення експериментів, перевірка

Продовження таблиці 2.2.

Артефакт	Формат	Зміст	Де використовується в роботі
Профілі запуску	Конфігураційний файл	Алгоритми, сценарії, режими, правила запуску	Відтворення експериментів
Сирі журнали запусків	Структурований журнал	Записи кожного запуску з параметрами та результатами	Аналіз, перевірка коректності, додатки
Зведені таблиці	Табличний файл	Узагальнення результатів за сценаріями та алгоритмами	Розділ експериментів, додатки
Графічні матеріали	Файли зображень	Візуалізація порівнянь і тенденцій	Розділ експериментів

Наведена таблиця узагальнює склад артефактів дослідження та фактично виконує роль матриці простежуваності методики. Вона демонструє, що кожний етап роботи має матеріальне підтвердження у вигляді конкретного файлу або набору даних, а також визначає, як саме ці матеріали інтегруються у структуру магістерської роботи. Така систематизація дозволяє однозначно відтворити шлях від вхідних даних до підсумкових висновків, оскільки для кожного результату можна встановити джерело походження, правила отримання та місце використання у тексті.

Окремо важливо підкреслити, що артефакти поділяються на первинні та похідні. Первинні артефакти формуються безпосередньо під час виконання

експериментів і не повинні піддаватися ручному редагуванню, оскільки вони відображають фактичний перебіг процедур. До них належать тестові набори, профілі запуску та сирі журнали, які забезпечують доказову базу дослідження. Похідні артефакти утворюються на етапі аналітичної обробки, коли первинні дані перетворюються на зведені таблиці, графічні матеріали, рейтинги та рекомендації. Такий поділ є методично доцільним, оскільки забезпечує прозорість і можливість перевірки, коли будь-який підсумковий висновок можна співвіднести з відповідними первинними записами запусків.

Для підвищення відтворюваності доцільно встановити єдині правила ідентифікації та збереження артефактів. Це включає узгоджені найменування серій і сценаріїв, фіксацію конфігурації середовища для кожного запуску, збереження версій профілів запуску та окреме зберігання матеріалів, що відносяться до різних серій експериментів. У сукупності такі правила дозволяють уникнути змішування результатів, забезпечують коректне зіставлення даних і спрощують підготовку додатків до роботи, де читач може перевірити ключові етапи методики без звернення до вихідного коду.

Висновки до розділу

У розділі сформовано цілісну науково обґрунтовану основу для подальшого порівняльного аналізу криптографічних алгоритмів у прикладному середовищі. Обґрунтовано склад досліджуваних рішень так, щоб він відображав реальні практики використання симетричних і асиметричних перетворень, а також їх гібридне поєднання в інформаційних системах. Показано, що доцільність включення кожного алгоритму визначається не лише його формальною належністю до класу, а й інженерними особливостями застосування: характером оброблення даних, взаємодією з підсистемою введення-виведення, вимогами до керування ключовим матеріалом і стабільністю програмної реалізації.

Запропоновано підхід до організації експериментів як стандартизованої процедури, у якій правила підготовки даних, запуску операцій, фіксації

результатів і перевірки коректності є єдиними для всіх альтернатив. Така організація забезпечує порівнюваність спостережень і зменшує ризик змішування властивостей алгоритму з випадковими ефектами середовища виконання. Сформовано систему показників, яка охоплює часові характеристики, темп оброблення, ресурсний профіль та особливості вихідного подання даних після перетворення, а також визначено принципи отримання цих показників у межах одного експериментального протоколу. Узагальнений перелік показників і правила їх трактування подано в таблиці 2.1, що забезпечує однозначність інтерпретації під час подальшого аналізу.

Окремо визначено логіку аналітичної обробки результатів, яка перетворює сирі журнальні записи на інтерпретовані узагальнення, придатні для ранжування та формування рекомендацій. Уніфікація напрямку оптимальності, нормалізація різнорідних величин і сценарне агрегування створюють прозорий місток між первинними даними та підсумковими висновками, а також дозволяють коректно працювати з компромісами між показниками. Зафіксовано склад артефактів, потрібних для відтворення й перевірки результатів, і систематизовано їх формати та призначення в таблиці 2.2. У підсумку розділ задає чіткий протокол дослідження: від вибору алгоритмів і визначення вимірюваних характеристик до правил збереження, перевірки та узагальнення даних, що забезпечує наукову коректність і відтворюваність наступної експериментальної частини.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ

3.1 Призначення системи, вимоги та сценарії використання

Застосування криптографічних алгоритмів у сучасних інформаційних системах відбувається в умовах, де безпекові вимоги тісно поєднані з експлуатаційними характеристиками програмного середовища. У практичних сценаріях шифрування і розшифрування виконуються як частина робочих процесів зберігання, оброблення та передавання даних, а отже впливають на узгодженість взаємодії компонентів, стабільність сервісів і керованість ресурсів платформи. За таких умов коректне порівняння криптографічних рішень потребує не окремих разових тестів, а стандартизованої процедури експериментів, у межах якої однаково визначаються межі операцій, порядок підготовки даних, правила фіксації показників і критерії прийнятності результатів. Без єдиних правил вимірювання зіставлення алгоритмів стає методично нестійким, оскільки на результати суттєво впливають відмінності в сценаріях виконання, організації введення і виведення, ініціалізації бібліотек, а також у способі документування параметрів запуску.

У межах даного дослідження під системою розуміється інструментальна система оцінювання ефективності криптографічних алгоритмів, яка забезпечує повний цикл експериментального дослідження від формалізованого опису умов запуску до отримання структурованих результатів, придатних для подальшого аналізу. Система призначена для керування експериментами за визначеними сценаріями, виконання криптографічних операцій через уніфіковані механізми виклику, синхронізованого збору характеристик виконання, перевірки коректності отриманих результатів, збереження експериментальних артефактів і підготовки вихідних матеріалів для узагальнення у вигляді таблиць, графіків та підсумкових рекомендацій. Таким чином, система виступає не лише засобом виконання перетворень, а й засобом стандартизації процедури дослідження, що забезпечує порівнюваність і відтворюваність висновків.

Межі системи визначаються предметом роботи та спрямовані на оцінювання ефективності алгоритмів у типових прикладних умовах. Система охоплює організацію запусків, збір і фіксацію даних виконання та формування артефактів, однак не замінює криптоаналітичні дослідження, не виконує повного аудиту безпеки реалізацій, не моделює загрози у вигляді атак і не проектує промислові криптографічні протоколи. Також система не підміняє інфраструктурні аспекти експлуатаційної безпеки, включно з політиками доступу, процедурою розповсюдження ключів у реальних мережах чи регламентами організаційного контролю. У межах дослідження приймається, що використовуються коректні реалізації криптографічних перетворень, а порівняння здійснюється на основі їх поведінки в контрольованих умовах виконання, які можуть бути формально зафіксовані та повторені.

Функціональні вимоги визначають перелік дій, які система має забезпечувати для реалізації методики дослідження. Насамперед система повинна підтримувати основні сценарії експериментів, які відповідають типовим прикладним задачам використання криптографії. До таких сценаріїв належать шифрування і розшифрування одиничного файла, пакетна обробка каталогу або набору файлів, довготривалі процеси на кшталт архівації чи резервного копіювання, а також короткі асиметричні операції й дії, пов’язані з ключовим матеріалом. Кожний сценарій має реалізовуватися як відтворювана процедура з однозначно визначеними межами операцій і правилами формування вихідних даних.

Система повинна забезпечувати налаштування запусків через профіль, у якому фіксуються вибір алгоритмів і їх конфігурацій, вибір наборів даних та параметри виконання сценарію, а також порядок запусків у межах серії. Така форма налаштування необхідна для уніфікації експериментів і мінімізації ручного втручання, яке потенційно призводить до методичних розбіжностей. На основі профілю система має автоматизувати виконання серії, реалізуючи послідовний запуск комбінацій умов, керування переходами між запусками і фіксацію контексту кожного запуску в журналі.

Окремою функціональною вимогою є валідація результатів, оскільки експериментальні дані мають бути придатними для аналізу лише за умови коректного виконання криптографічних перетворень. Система повинна підтверджувати коректність розшифрування та цілісність відновлених даних відповідно до прийнятого регламенту, а також маркувати або відсіювати запуски з ознаками некоректності. Паралельно система має формувати набір артефактів, що включає сирі журнали запусків, зведені результати, матеріали для графіків і таблиць та експортні набори, підготовлені для подальшої аналітичної обробки.

Нефункціональні вимоги описують властивості системи як інженерного інструмента, від яких залежить наукова коректність та практична придатність отриманих результатів. Ключовою вимогою є відтворюваність, що передбачає фіксацію параметрів запуску, версій програмних компонентів, конфігурацій виконання та умов середовища у вигляді, який дозволяє повторити серію експериментів незалежно від суб’єктивних дій користувача. Не менш важливою є надійність, тобто здатність системи коректно реагувати на помилки введення, збої виконання, проблеми доступу до файлів, нестачу прав або переривання процесу, зберігаючи при цьому повний контекст подій для подальшого розбору.

Вимога модульності визначає, що система має дозволяти додавання нових алгоритмів або сценаріїв без переробки інших частин, збереження єдиних правил вимірювання та стабільності формату журналів. Вимога прозорості означає повне журналювання подій і рішень системи, включно з параметрами запуску, статусами виконання і результатами перевірок коректності. Портативність визначає, що для перенесення системи в інше середовище має бути очевидним перелік необхідних залежностей, процедур підготовки та умов запуску. Окремо підкреслюється вимога до продуктивності як властивості інструмента: система повинна організовувати виконання експериментів таким чином, щоб накладні витрати самої інструменталізації не спотворювали досліджувані закономірності та не ускладнювали інтерпретацію результатів.

Застосовність системи визначається обмеженнями виконувальної платформи, зокрема характеристиками операційної системи, доступними

бібліотеками та правилами доступу до носіїв і файлової структури. Система передбачає коректне налаштування середовища виконання, наявність необхідних криптографічних компонентів та достатні права для читання і запису даних у визначених каталогах. До припущень дослідження належить використання коректних криптографічних реалізацій, а також відносна стабільність середовища у межах серій запусків, оскільки суттєві зовнішні зміни можуть впливати на результати і знижувати їх порівнюваність.

Окремо фіксуються обмеження щодо типів даних і структури тестових наборів, які використовуються в експериментах. Система орієнтована на оброблення файлових об’єктів та їх колекцій, отже специфічні формати, нестандартні схеми зберігання або нестійкі до перезапису носії можуть вимагати додаткових умов, що виходять за межі базової постановки. Також обмеження можуть стосуватися структури каталогів, правил іменування вихідних артефактів та вимог до вільного простору для формування результатів експериментів.

Сценарії використання визначають практичний регламент роботи з системою та забезпечують зв’язок між вимогами і реалізованими функціями. У сценарії шифрування одиничного файла користувач задає вхідний об’єкт, обирає алгоритм і конфігурацію, визначає шлях збереження результату та запускає операцію. Система виконує перетворення за стандартизованою процедурою, фіксує контекст запуску, виконує перевірку коректності шляхом контрольного відновлення даних і формує запис у журналі разом із відповідними артефактами, необхідними для подальшого порівняльного аналізу.

У сценарії пакетної обробки каталогу або набору файлів користувач визначає джерело даних і правила формування вибірки, після чого система послідовно застосовує обраний алгоритм до елементів набору відповідно до регламенту сценарію. На рівні системи забезпечується узгоджене журналювання запусків, фіксація службових подій і результатів перевірок, а також формування зведених матеріалів, що відображають узагальнену поведінку алгоритму для заданого набору.

Сценарій довготривалих процесів, пов’язаних з архівацією або резервним копіюванням, передбачає оброблення даних у режимі, де суттєву роль відіграє послідовність операцій читання, перетворення і запису. У цьому сценарії система повинна забезпечувати стабільність виконання, коректність проміжних станів, повноту журналювання та можливість відновити контекст кожного етапу, якщо виникне збій. Результатом є набір артефактів, що дозволяє порівнювати алгоритми з позицій їх практичної придатності у довготривалих процесах.

Сценарій коротких асиметричних операцій і роботи з ключовим матеріалом орієнтований на операції, де ключовими є коректність процедур генерації ключів, виконання коротких перетворень і формування вихідних представлень, придатних для використання в прикладних протоколах або гібридних схемах. Система забезпечує стандартизований порядок виконання дій, фіксує результати операцій і підтверджує їх коректність, формуючи записи у журналі та експортні дані для подальшого узагальнення.

Для кожного сценарію система формує набір артефактів, які включають журнал запуску, службові записи про параметри виконання і перевірки, а також результати у форматі, придатному для порівняння в аналітичній частині роботи. Наявність таких артефактів є необхідною умовою відтворюваності: вони дозволяють співвіднести кожний підсумковий висновок із конкретним сценарієм, конкретними умовами запуску та зафіксованими результатами перевірки коректності.

Наприкінці підрозділу доцільно передбачити два допоміжні матеріали, що підсилюють структурованість опису системи. Перший матеріал є таблицею відповідності ролей і дій користувачів із вхідними даними та отриманими результатами, що забезпечує прозоре відображення призначення системи з позиції різних учасників процесу. Другий матеріал є таблицею відповідності вимог і засобів їх реалізації, яка пов’язує сформульовані функціональні та нефункціональні вимоги з архітектурними рішеннями та механізмами системи. Для логічного переходу до UML-моделювання також доцільно подати короткий

перелік сценаріїв як основу для побудови Use Case-діаграми, де кожний сценарій відповідає окремій групі варіантів використання.

3.2 Архітектура системи та опис модулів

Інструментальна система оцінювання ефективності криптографічних алгоритмів спроектована як модульне програмне рішення, орієнтоване на відтворюваність експериментів, уніфікацію процедур запуску та прозорість формування результатів. Архітектура ґрунтується на принципі розділення відповідальностей, за якого кожний функціональний компонент виконує чітко визначену роль у загальному конвеєрі обробки. Такий підхід мінімізує взаємозалежності між частинами системи, спрощує супровід і забезпечує можливість розширення без зміни базової логіки виконання експериментів та формування артефактів.

Ключовим архітектурним рішенням є відокремлення керування експериментами від реалізації криптографічних операцій та від підсистем збору й збереження результатів. Запуск серій визначається конфігураційними профілями, що формалізують сценарій, перелік алгоритмів, параметри виконання та правила повторів. На основі профілю формується план запусків, після чого центральний керувальний компонент ініціалізує виконання серії та координує переходи між підготовкою даних, виконанням операцій, валідацією коректності та реєстрацією результатів. Завдяки цьому система забезпечує однакові правила оброблення даних для різних алгоритмів і підтримує коректність порівняння в межах заданих умов.

Функціонування системи організовано як послідовність етапів, де вихід одного етапу виступає входом наступного. На початку користувач задає профіль запуску, який інтерпретується модулем конфігурації та перетворюється на формалізований план серії. Далі оркестратор експериментів запускає відповідні сценарії обробки, що визначають порядок роботи з даними та межі операцій, які підлягають вимірюванню. Криптографічні перетворення виконуються через адаптери алгоритмів, які інкапсулюють специфіку реалізацій і надають

уніфікований інтерфейс виконання операцій. Паралельно модуль збору показників синхронізовано фіксує характеристики виконання згідно з методикою, після чого модуль валідації підтверджує коректність результатів і формує ознаки придатності запуску для подальшого аналізу. Сформований структурований запис запуску передається до підсистеми журналювання та сховища результатів, де забезпечується цілісність і простежуваність даних. На завершальному етапі модуль експорту формує похідні представлення, придатні для подальшого аналітичного опрацювання та включення у звітні матеріали.

Модуль конфігурації та профілів запуску забезпечує опис експериментів у формалізованому вигляді. Він відповідає за зчитування профілів, перевірку узгодженості параметрів та формування плану серії як переліку запусків з однозначно визначеними умовами виконання. Оркестратор експериментів є центральним керувальним компонентом, який забезпечує послідовне виконання плану серії, контроль станів, маршрутизацію результатів і обробку виняткових ситуацій. Він ініціалізує модулі збору показників, запускає валідацію та гарантує, що кожний запуск завершується формуванням структурованого запису або коректно зареєстрованої помилки з описом контексту.

Модуль сценаріїв реалізує типові режими застосування криптографічних перетворень, зокрема оброблення одиничного файла, пакетну обробку набору файлів, довготривалі процеси на кшталт архівації та резервного копіювання, а також короткі асиметричні операції і процедури, пов’язані з ключовим матеріалом. У межах сценарію задається порядок читання і запису даних, логіка формування вихідних об’єктів і межі операції, що фіксуються як часові інтервали вимірювання. Криптографічний модуль реалізовано у вигляді набору адаптерів, які надають уніфікований інтерфейс виконання операцій шифрування, розшифрування та, за потреби, процедур генерації ключів. Це дозволяє відокремити логіку сценарію від деталей реалізації алгоритму та забезпечує можливість додавання нових алгоритмів шляхом реалізації стандартного контракту без модифікації інших компонентів.

Модуль збору показників відповідає за уніфіковану реєстрацію характеристик виконання операцій. Він синхронізується зі сценарієм і фіксує часові характеристики, профіль використання ресурсів і ознаки зміни представлення даних на виході відповідно до прийнятої методики. Важливо, що цей модуль застосовує однакові правила збору даних для всіх алгоритмів, забезпечуючи порівнюваність результатів. Модуль валідації забезпечує контроль коректності виконання криптографічних перетворень, реалізує перевірку відповідності результатів розшифрування вихідним даним і формує ознаки успішності, що дозволяють відокремлювати коректні запуски від тих, які є непридатними для подальшого аналізу.

Підсистема журналювання та реєстрації подій формує сирі результати експериментів у структурованому вигляді. Вона фіксує контекст запуску, параметри середовища та сценарію, результати вимірювань і статус валідації, забезпечуючи простежуваність, оскільки кожний підсумок може бути співвіднесений з конкретним запуском і умовами його виконання. Сховище результатів і менеджер артефактів відповідають за організацію збереження даних експериментів, структурування матеріалів за серіями та сценаріями, контроль цілісності та зберігання конфігурацій, що використовувалися під час запуску. Модуль експорту формує похідні представлення результатів, придатні для подальшої обробки й включення у звіт, зокрема шляхом підготовки зведених таблиць і матеріалів для порівняльного аналізу.

Взаємодія між модулями реалізується на основі узгоджених структур даних і контрактів. Основною одиницею обміну є опис запуску, що містить параметри сценарію, алгоритму та середовища, а також структурований запис результату, який включає зафіксовані показники, статус виконання і результат валідації. На рівні інтерфейсів фіксуються правила, за якими сценарій викликає криптографічний адаптер, модуль вимірювань синхронізується з межами операції, а модулі журналювання і сховища забезпечують узгоджене збереження результатів. Для підтримки розширюваності система передбачає стандартизований механізм підключення нових алгоритмів і сценаріїв:

додавання нового алгоритму реалізується шляхом створення адаптера з уніфікованим набором операцій, а додавання нового сценарію здійснюється шляхом підключення нового сценарного компонента, що використовує ті самі модулі збору показників, валідації та журналювання. Завдяки цьому зберігається порівнюваність результатів і цілісність методики дослідження.

3.3 Реалізація збору показників і журналювання

Підсистема збору показників і журналювання в межах інструментальної системи розглядається як ключовий елемент, що забезпечує методичну коректність порівняльного дослідження криптографічних алгоритмів. Її призначення полягає у тому, щоб перетворити виконання криптографічних операцій у стандартизований експериментальний процес, результати якого є порівнюваними, перевірюваними та придатними до відтворення. Відмінність між алгоритмами у практичному середовищі може проявлятися не лише через їх внутрішню структуру, а й через різний характер взаємодії з пам’яттю, введенням і виведенням, а також через особливості реалізації бібліотек. Саме тому вимірювання мають виконуватися за уніфікованою процедурою, де однаково визначено межі операцій, порядок дій, правила фіксації характеристик виконання та критерії коректності результату. Базовою одиницею обліку в цій процедурі є експериментальний запуск, під яким розуміється завершений цикл виконання конкретної операції за фіксованих умов, включно з заданим алгоритмом і його конфігурацією, визначеним сценарієм оброблення, характеристиками вхідних даних і середовища виконання. Для кожного запуску система формує цілісний запис, який поєднує контекст запуску, зафіксовані характеристики виконання, результат перевірки коректності та службову інформацію, необхідну для подальшої інтерпретації.

Логіка роботи підсистеми реалізується як послідовність узгоджених етапів, що відтворюються для всіх алгоритмів і сценаріїв без винятку. На етапі підготовки визначаються параметри запуску і формується контекст, що включає обраний алгоритм, конфігурацію сценарію, джерело вхідних даних, правила

формування вихідних матеріалів, а також умови виконання. Далі здійснюється виконання криптографічної операції у межах формально заданих меж вимірювання, після чого відбувається фіксація характеристик виконання і перевірка коректності перетворень. Завершальним етапом є формування запису журналу та збереження пов’язаних артефактів у структурованому вигляді. Така схема дозволяє розділити власне виконання операції та процедури контролю, а також забезпечити прозоре походження кожного результату. Окремо враховується вимога мінімального впливу інструменталізації на результати, що означає обмеження допоміжних операцій під час вимірювання, уникнення зайвих перетворень даних поза межами необхідного сценарію та забезпечення однакових правил оброблення для всіх порівнюваних альтернатив. Підсистема не повинна змінювати характер виконання криптографічної операції, а має лише спостерігати, фіксувати та документувати її перебіг у стандартизованому форматі.

Методична коректність значною мірою визначається тим, як встановлено межі вимірювання, оскільки саме вони задають, які дії вважаються частиною операції, а які належать до підготовчих або завершальних процедур. Для сценарію шифрування межі вимірювання визначаються так, щоб охопити виконання перетворення даних у рамках обраного алгоритму та формування вихідного представлення, тоді як для сценарію розшифрування межі повинні охоплювати відновлення даних із вихідного представлення та завершення операції з формуванням результату, придатного до подальшої перевірки. У межах запуску розрізняються етапи ініціалізації криптографічного компонента, організації доступу до вхідних даних, виконання перетворення, запису вихідного результату та завершальних дій, пов’язаних із коректним закриттям потоків і формуванням підсумку запуску. Оскільки деякі підготовчі процедури можуть мати суттєві накладні витрати, у методиці фіксується правило включення або невключення таких процедур до інтервалу вимірювання, аби порівняння не залежало від випадкових відмінностей у підготовці, а відображало поведінку алгоритму в заданому сценарії. Узгодженість меж вимірювання підтримується

для всіх типових сценаріїв, зокрема для оброблення одиничного файла, пакетної обробки набору файлів у каталозі, тривалих процесів на кшталт архівації або резервного копіювання, а також для коротких асиметричних операцій і процедур, пов’язаних із ключовим матеріалом. Для кожного сценарію фіксується, у який момент відносно виклику криптографічної операції здійснюється старт і завершення вимірювання, щоб забезпечити однакове трактування результатів у різних серіях та виключити систематичні зсуви.

Стандартизація процедур виконання реалізується через єдиний порядок запуску для всіх алгоритмів, що передбачає використання одного джерела параметрів, однакові правила підготовки вхідних даних, узгоджену логіку створення вихідних матеріалів та незмінну процедуру перевірки коректності. Якщо експерименти виконуються серіями, визначаються правила повторного запуску, порядок проходження комбінацій умов та вимога фіксації контексту кожного запуску, щоб уникнути втрати інформації про середовище, сценарій або конфігурацію алгоритму. Додатково може застосовуватися процедура прогріву виконання, яка спрямована на стабілізацію умов запуску, зокрема станів кешування або ініціалізації бібліотек, за умови, що правила прогріву є однаковими для всіх порівнюваних альтернатив і прозоро задокументовані в журналі. Стандартизація у цьому сенсі розглядається як необхідна умова для порівнянності, оскільки дозволяє відокремити власне вплив алгоритмічних рішень від випадкових чинників середовища.

Підсистема збору ресурсного профілю орієнтована на фіксацію того, як виконання криптографічної операції впливає на обчислювальну платформу і наскільки інтенсивно залучаються ключові ресурси. У контексті даного дослідження ресурсні характеристики інтерпретуються як ознака експлуатаційної придатності: алгоритм може демонструвати прийнятні результати за одним аспектом, але створювати небажані навантаження в іншому, що є критичним для систем з конкуренцією процесів або обмеженим середовищем. Важливою методичною умовою є прив’язка спостережень за ресурсами до інтервалу виконання операції, що забезпечує змістовність таких

спостережень і дозволяє співставляти різні запуски. Підсистема може використовувати підхід фіксації значень у контрольних точках або підхід періодичного спостереження протягом інтервалу виконання, після чого застосовуються правила вибору репрезентативного значення для журналу. У методиці також описуються можливі джерела похибок, зокрема фонові процеси та коливання системних станів, і спосіб їх урахування під час інтерпретації висновків через узгоджені умови запусків та повне документування контексту.

Коректність інтерпретації результатів неможлива без строгих правил визначення вхідних і вихідних даних кожного запуску. Підсистема має однозначно визначати, що є вхідним об’єктом експерименту, чи це один файл, чи множина файлів, а також фіксувати правила включення або виключення файлів у каталозі, якщо застосовується пакетний сценарій. Аналогічно визначається вихідний об’єкт, включно зі структурою розміщення результатів, правилами іменування та заходами, що унеможливлюють випадковий перезапис або змішування результатів різних запусків. Окремим принципом є узгодженість наборів даних між алгоритмами в межах серії, що означає незмінність вхідних матеріалів протягом серії та їх повторне використання за однакових умов для всіх порівнюваних альтернатив. Це забезпечує, що відмінності у результатах відображають різницю в алгоритмічних рішеннях, а не випадкову зміну даних.

На основі базових спостережень підсистема формує похідні показники, які використовуються для узагальнення результатів і подальшого порівняльного аналізу. Такі похідні характеристики мають чітке призначення, оскільки дозволяють перейти від сирих спостережень до інтерпретованих узагальнень, придатних для зіставлення між запусками, сценаріями та алгоритмами. Для одиничного файла похідні значення формуються безпосередньо з результатів запуску, тоді як для набору файлів застосовуються правила агрегування, які визначають, які величини накопичуються як сумарні, які підлягають усередненню, а які повинні відображати найгірший або найкращий випадок у межах запуску, залежно від їх змісту. У тексті підрозділу доцільно пояснити сенс

кожної похідної характеристики з позиції сценаріїв експлуатації, підкресливши, що інтерпретація має опиратися на єдині правила формування та порівняння.

Центральним елементом документування є журнал експериментів, який має забезпечувати повноту і однозначність опису запусків, придатність до відтворення та можливість машинної обробки. Запис журналу містить ідентифікатори серії та запуску, опис сценарію і алгоритму, контекст алгоритму з параметрами конфігурації, контекст середовища виконання з релевантними умовами, контекст даних із характеристиками вхідного об’єкта та правилами формування вихідних матеріалів, результати вимірювання у базовому і похідному поданні, статус виконання та результат перевірки коректності, а також повідомлення про службові події, попередження і діагностичні записи. Важливим аспектом є керування сумісністю формату журналу при змінах, тому у методиці визначаються правила версіонування, які дозволяють зберігати можливість інтерпретації раніше сформованих записів і коректного порівняння між серіями, виконаними в різні моменти розвитку системи.

Валідація коректності перетворень є обов’язковою складовою підсистеми, оскільки відокремлює випадки, коли операція формально виконана, але результат не відповідає вимогам правильності, від запусків, придатних до аналізу. Для симетричних сценаріїв перевірка орієнтована на підтвердження відповідності відновлених даних вихідним, що дозволяє гарантувати змістовність зафіксованих характеристик виконання. Для асиметричних сценаріїв валідація пов’язана з перевіркою коректності виконання операцій та узгодженості результатів у межах обраної процедури, включно з ключовим матеріалом і вихідним представленням. За результатами валідації запуск маркується як придатний до аналізу або як непридатний, при цьому розрізняються випадки помилки виконання, невдалої перевірки коректності та неповноти даних. Політика поводження з некоректними запусками визначається правилами серії та передбачає однозначне документування контексту, щоб забезпечити можливість повторення та локалізації причини, а також прозоре виключення таких запусків із подальших узагальнень.

Політика обробки помилок і логування подій доповнює журналювання, забезпечуючи керованість системи в умовах відхилень і збоїв. У підрозділі доцільно визначити класи помилок, пов’язані з параметрами запуску, доступом до файлів, винятками криптографічних реалізацій, помилками читання або запису, а також порушеннями очікуваного формату даних. Для кожної категорії формулюються правила фіксації, які мають забезпечити достатність інформації для повторення ситуації та діагностики, не розкриваючи при цьому чутливі відомості. З міркувань безпечності журналювання важливо окремо підкреслити, що ключовий матеріал та інші конфіденційні дані не повинні потрапляти до журналу у вигляді, який допускає їх відновлення, а система має обмежувати журналювання до опису контексту і параметрів у допустимому обсязі.

3.4 Формат збереження та експорт результатів

Підсистема збереження та експорту результатів є завершальним елементом інструментальної системи, який перетворює виконані експериментальні запуски на впорядкований набір матеріалів, придатних для подальшого аналізу та підготовки звітних фрагментів магістерської роботи. Її призначення полягає у забезпеченні простежуваності походження кожного результату, однозначного віднесення записів до конкретних умов виконання, а також у формуванні єдиного інформаційного контуру, в якому сирі журнали, зведені дані та підсумкові матеріали узгоджуються між собою. У межах науково коректного дослідження збереження результатів не розглядається як технічна дрібниця, оскільки саме структура сховища й дисципліна форматів визначають можливість незалежної перевірки, повторного аналізу та відтворення серій без ручних уточнень і неоднозначних інтерпретацій.

Організація сховища результатів базується на принципі ієрархічного групування за логікою проведення експериментів. Верхній рівень структури відповідає серіям запусків, які відображають завершені набори експериментів, виконані за певним профілем умов. Усередині серії дані доцільно групувати за сценаріями виконання, оскільки сценарій визначає регламент оброблення даних

і, відповідно, інтерпретацію результатів. Наступним рівнем є групування за алгоритмами та їх конфігураціями, що забезпечує природне розділення результатів для різних криптографічних альтернатив і дозволяє уникнути змішування артефактів при масових серіях. Найнижчим рівнем структури виступає конкретний запуск як атомарна одиниця обліку, для якої зберігається повний набір пов’язаних матеріалів.

Правила іменування каталогів і файлів мають забезпечувати однозначність, читабельність і стабільність. Доцільно застосовувати уніфіковані ідентифікатори серії та запуску, а також короткі маркери сценарію й алгоритму, які не залежать від локалізованих підписів у тексті роботи. Важливо, щоб імена не містили неоднозначних символів і не залежали від налаштувань операційної системи. Практичним рішенням є використання системи іменування, у якій спочатку задається ідентифікатор серії, далі позначення сценарію, потім позначення алгоритму і лише після цього номер або ідентифікатор запуску. Така схема дозволяє швидко знаходити потрібні фрагменти, спрощує автоматизовану обробку та мінімізує ризик помилкового аналізу не тих даних.

Окремо доцільно розділяти область сирих даних та область похідних матеріалів. Сирі журнали і первинні артефакти запусків мають зберігатися у стабільній структурі без ручного редагування, оскільки вони є доказовою базою дослідження. Натомість зведені таблиці, підготовлені вибірки для аналітики та звітні матеріали доцільно зберігати у підкаталогах експорту, що містять уже агреговані й нормалізовані дані, сформовані на основі сирих записів. Такий поділ забезпечує чітке розмежування між первинними даними і результатами їх оброблення, що є важливим для прозорості методики.

Формати збереження визначають, наскільки дані є придатними для автоматизованого аналізу та наскільки легко їх перевіряти незалежним дослідником. Сирі журнали запусків доцільно зберігати у структурованому форматі, який підтримує явне представлення контексту запуску, полів результатів і службових статусів. Такий формат має забезпечувати однозначне трактування типів даних, стабільність назв полів і можливість валідації схеми

запису. Принциповою вимогою є те, що журнал повинен містити повний контекст, включно з описом сценарію, конфігурацією алгоритму, характеристиками вхідних даних у термінах, необхідних для відтворення, параметрами середовища виконання, результатами перевірки коректності та статусом завершення. Усі ці елементи мають зберігатися так, щоб їх можна було прочитати як людиною, так і програмними засобами без додаткових перетворень.

Зведені таблиці та агреговані представлення доцільно зберігати у форматах, орієнтованих на аналітичну обробку, де кожний рядок відповідає запуску або узагальненню запусків за певними умовами, а стовпці відповідають уніфікованим полям контексту й результатів. Важливо забезпечити сумісність між сирими журналами і зведеними таблицями через сталі ключі ідентифікації, що дозволяють у будь-який момент перейти від агрегованого рядка до первинних записів запуску. Для читабельності й коректного імпорту в аналітичні інструменти доцільно уніфікувати кодування, правила подання числових значень, форматування дат і часу та порядок полів, щоб уникнути залежності від регіональних налаштувань середовища.

Окремою вимогою є версіонування формату. Оскільки інструментальна система може еволюціонувати, формат журналу має містити ознаку версії схеми, яка дозволяє інтерпретувати записи, створені на різних етапах розроблення, без втрати змісту. Це також спрощує підтримку зворотної сумісності в аналітичній підсистемі та виключає ситуації, коли результати стають непридатними до повторного аналізу через несумісність полів.

Експорт результатів розглядається як формалізований процес підготовки даних до подальшої аналітики та представлення у вигляді матеріалів для тексту роботи. На відміну від простого копіювання файлів, експорт має забезпечувати відбір релевантних даних, перевірку їх повноти та узгодженості, а також формування уніфікованого набору, який може бути використаний без ручних виправлень. У межах експорту система формує агреговані представлення, в яких

записи з різних сценаріїв та алгоритмів приводяться до узгодженого набору полів, зберігаючи при цьому зв’язок із первинними журналами.

Перевірка повноти експорту полягає у контролі наявності обов’язкових складових для кожної серії: журналів запусків, інформації про конфігурацію, результатів валідації, а також супровідних даних, необхідних для інтерпретації. Такий контроль дозволяє виявити незавершені або некоректні серії ще до початку аналітичної обробки, що підвищує якість дослідження і зменшує ризик помилкових висновків. Додатково експорт повинен зберігати узгодженість ідентифікаторів між сирими записами та зведеними даними, щоб кожне узагальнення можна було перевірити шляхом повернення до первинних артефактів.

Повторне використання даних забезпечується тим, що експортуються не лише підсумкові таблиці, але й супровідні описи контексту, які пояснюють, як саме сформовано дані, які правила відбору застосовано та які записи було виключено внаслідок валідації або помилок виконання. Для звітності доцільно формувати окремий набір вихідних матеріалів, орієнтований на вставку у текст, зокрема зведення у вигляді таблиць і графічні матеріали, при цьому їх походження має бути прив’язане до експортованих аналітичних даних. Така прив’язка забезпечує узгодженість між числовими підсумками, графіками та поясненнями в тексті, що особливо важливо під час рецензування.

Пакет відтворюваності є сукупністю матеріалів, достатніх для того, щоб сторонній перевіряючий міг повторити експериментальні серії, перевірити коректність методики та відтворити ключові результати без неформальних уточнень. Основою такого пакета є профілі запуску, які фіксують сценарії, конфігурації алгоритмів, правила формування наборів даних і параметри виконання. Далі необхідним є опис середовища виконання, який включає відомості про програмне оточення, залежності, версії бібліотек і суттєві налаштування, що можуть впливати на відтворення. Також у пакет мають входити сирі журнали запусків і структура даних у сховищі, щоб перевіряючий міг зіставити підсумкові матеріали з первинними записами.

Для повноцінного відтворення доцільно додати інструкцію, яка описує порядок дій, починаючи від підготовки середовища та розміщення тестових наборів і завершуючи запуском серій та формуванням експорту. Така інструкція повинна фіксувати логіку відтворення, правила вибору профілю, порядок формування вихідних каталогів і спосіб перевірки того, що серія виконалась коректно. Важливо, щоб пакет відтворюваності не вимагав від перевіряючого здогадуватись про приховані припущення: усі ключові правила повинні бути явно задокументовані, а структура артефактів – узгодженою і передбачуваною. Додатково доцільно передбачити засоби підтвердження цілісності матеріалів, які дозволяють перевірити, що вхідні дані та отримані артефакти не зазнали випадкових змін у процесі перенесення або збереження.

3.5 UML-моделювання системи

Use Case-діаграма формалізує зовнішні вимоги до системи у термінах акторів та їхніх цілей. На ній визначається, хто взаємодіє із системою, які функції є доступними для кожної ролі, а також які сценарії є ключовими для виконання дослідження. У межах цієї роботи до типових акторів належать дослідник, який налаштовує профілі запусків, ініціює серії експериментів, контролює журнали та виконує експорт; розробник, який розширює систему шляхом додавання нових алгоритмів або сценаріїв; рецензент або перевіряючий, який відтворює виконання за заданим профілем і перевіряє повноту артефактів. За необхідності також може бути виділено адміністратора середовища, який відповідає за права доступу, підготовку носіїв і коректність структури каталогів.

Рисунок 3.1 – Use Case-діаграма

Activity-діаграма фіксує регламент виконання експериментальної серії як послідовність дій та умовних переходів. Її головна методична цінність полягає у демонстрації того, що запуск серій не є «ручним набором операцій», а підпорядкований відтворюваному конвеєру: система приймає профіль, готує контекст і дані, виконує операцію, синхронізовано збирає показники, перевіряє коректність, формує журнал і зберігає артефакти, після чого або переходить до наступного запуску, або завершує серію з формуванням експорту. Така діаграма також є доречною для документування правил обробки помилок, оскільки дозволяє явно показати гілки, які активуються у випадку некоректного виконання, невдалої валідації або неможливості доступу до даних.

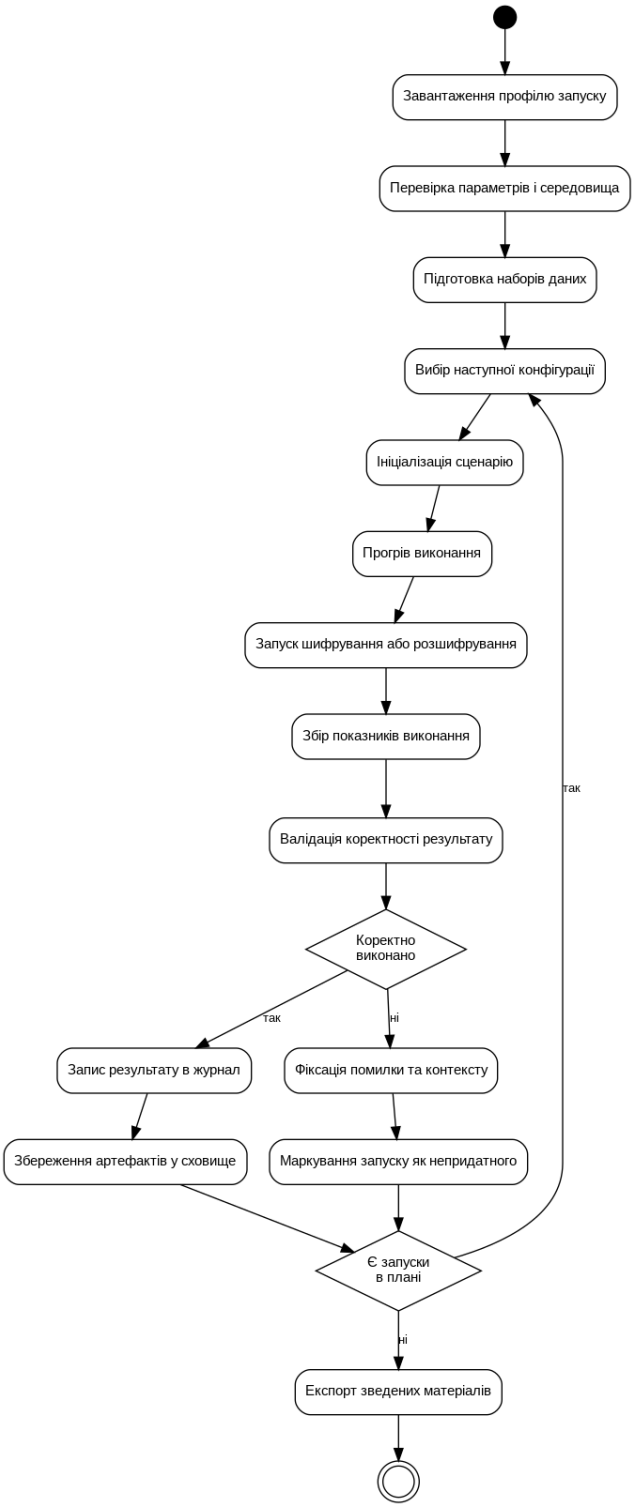

Рисунок 3.2 – Activity-діаграма

Component-діаграма формалізує модульний склад системи та залежності між її частинами, що є необхідним для обґрунтування архітектурних рішень та демонстрації модульності й розширюваності. У цій діаграмі система розглядається як набір компонентів із визначеними інтерфейсами, де кожний

компонент відповідає за окрему ділянку функціональності: керування профілями та конфігураціями, оркестрацію серій, виконання сценаріїв, криптографічні адаптери, збір показників, модуль валідації, журналювання, сховище результатів і компонент експорту. Окремо доцільно відобразити зовнішні залежності: криптографічну бібліотеку або провайдер криптоопераций, файлову систему, а також засоби отримання системної інформації, якщо вони використовуються для фіксації контексту запуску.

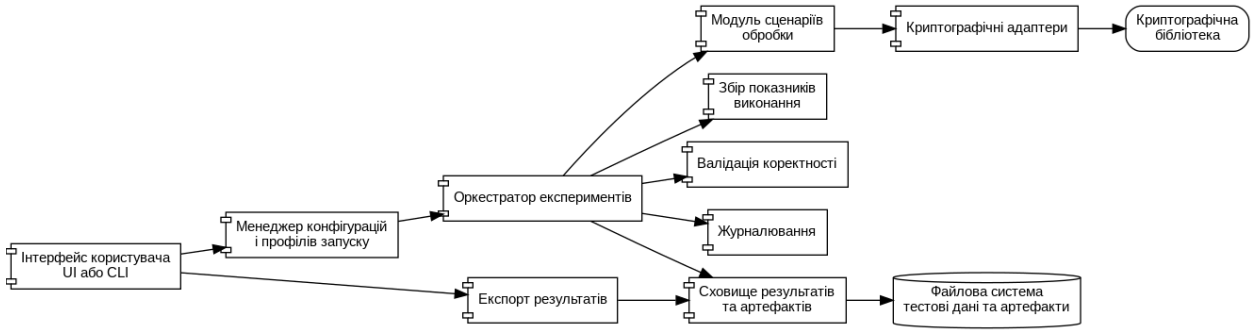

Рисунок 3.3 – Component-діаграма

Class-діаграма використовується для фіксації статичної структури даних і сервісів, що забезпечують виконання експериментів, збереження результатів та експорт. Вона дає змогу формально описати ключові сутності, з якими працює система, а також зв’язки між ними, що особливо важливо для пояснення структури журналу, формату збереження та повторного використання даних в аналітичній підсистемі. У контексті інструментальної системи до основних класів доцільно віднести профіль запуску, опис сценарію, конфігурацію алгоритму, опис набору даних, план запусків, контекст запуску, запис результату, об’єкт результату валідації, подію журналу та компоненти, що реалізують збереження й експорт.

Окрему увагу варто приділити інтерфейсам розширення. Для забезпечення модульності доречно мати уніфікований інтерфейс адаптера алгоритму, який інкапсулює виконання криптографічної операції у стандартизованих межах, а також інтерфейс сценарію, який визначає правила оброблення одиничного файла, набору файлів або короткої операції.

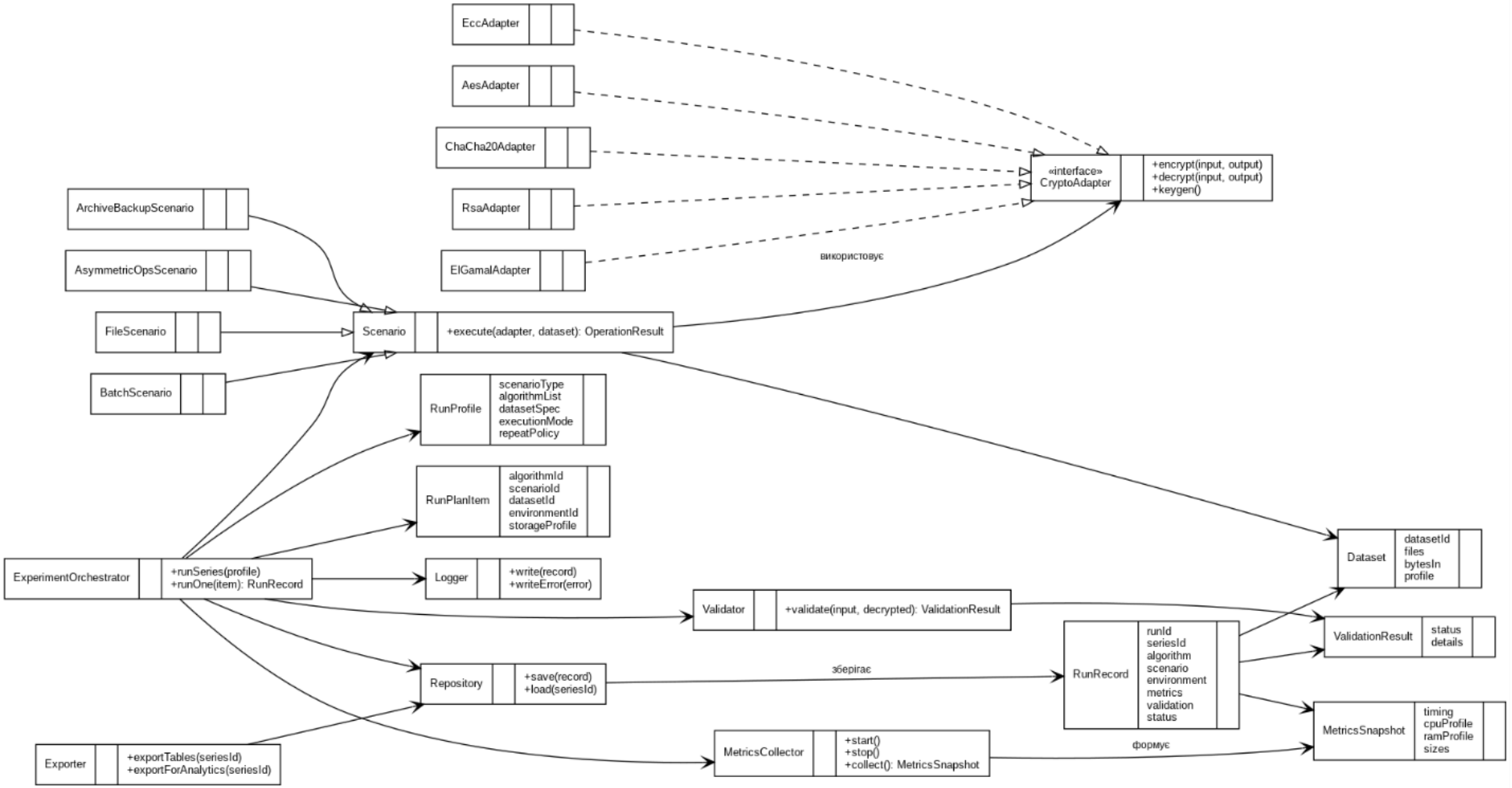

Рисунок 3.4 – Class-діаграма

Sequence-діаграми фіксують динаміку взаємодії компонентів у часі, тобто те, як саме реалізується виконання ключових операцій з точки зору обміну повідомленнями між модулями. Для інструментальної системи мінімально необхідною є діаграма «Запуск одного експерименту», оскільки вона демонструє, як ініціація запуску перетворюється на послідовність викликів: отримання параметрів профілю, підготовка контексту, виконання сценарію, запуск криптоадаптера, синхронний збір показників, валідація результату, формування запису журналу та збереження артефактів. Така діаграма є особливо цінною для підтвердження, що вимірювання, валідація та журналювання не є «розрізненими фрагментам», а інтегровані в єдиний контрольований процес, який повторюється у кожному запуску.

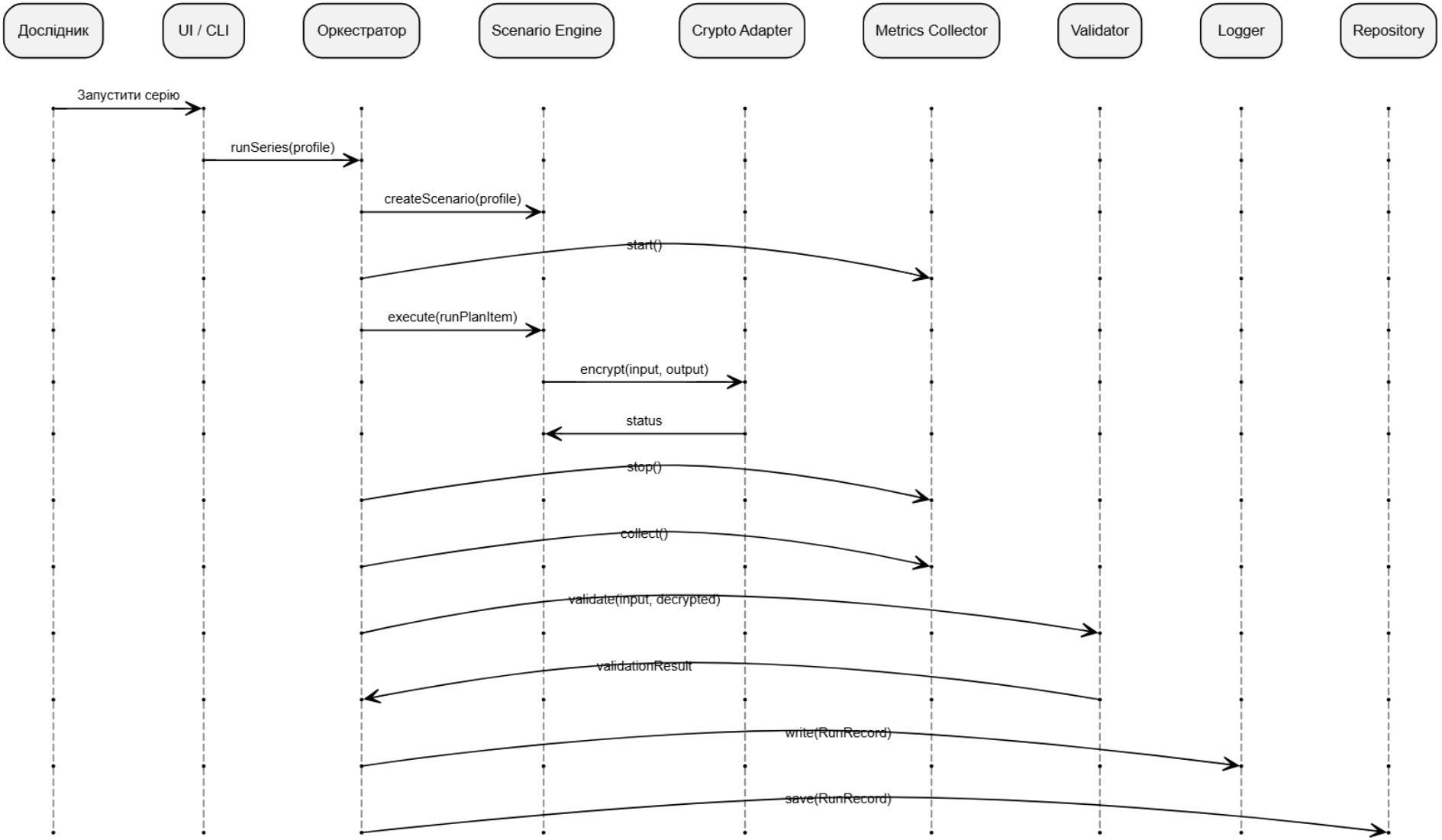

Рисунок 3.5 – Sequence-діаграми

3.6 Опис інтерфейсу користувача та приклади роботи системи

Інтерфейс організовано за принципом уніфікованої структури керування, що повторюється для різних алгоритмів і сценаріїв. У випадку симетричних алгоритмів ключовими елементами є засоби вибору вхідного об’єкта, засоби керування ключовим матеріалом, керування операцією перетворення та область відображення перебігу виконання і журналу. Вхідний об’єкт може задаватися як окремий файл або як каталог, що відповідає двом базовим сценаріям експлуатації, характерним для практичних задач захисту даних. Для зручності та зменшення ризику помилок інтерфейс передбачає пряме введення шляху, а також підтримку швидкого вибору об’єкта через взаємодію з файловою системою. Така організація знижує ймовірність того, що користувач випадково передасть некоректний об’єкт або запустить операцію в неузгоджених умовах.

Керування ключовим матеріалом у симетричних сценаріях подається як окрема підзадача інтерфейсу, оскільки саме коректність ключа та службових параметрів безпосередньо визначає можливість відновлення даних і придатність результатів для експериментального аналізу. Інтерфейс надає засоби генерації ключових параметрів, їх відображення у формі, придатній для повторного використання, а також механізми введення або відновлення вже сформованих значень. Це робить процедуру запуску контрольованою: користувач або приймає автоматично згенерований ключовий матеріал, або свідомо задає його, зберігаючи узгодженість між шифруванням і розшифруванням у межах серії.

Операційний блок керування, як правило, представлений діями запуску шифрування та розшифрування, що дозволяє явно відмежувати два різні типи перетворення і уникнути неявних режимів. Для довших операцій інтерфейс доповнюється індикаторами перебігу виконання, які відображають стан оброблення та забезпечують спостережуваність процесу. Паралельно передбачено текстову область журналу, у якій накопичується інформація про об’єкти оброблення, про статус виконання та про результати перевірок. Важливо, що журнал в інтерфейсі не замінює системне збереження результатів,

а є «вікном спостереження», яке дозволяє користувачеві контролювати хід експерименту та оперативно виявляти некоректні запуски.

Для асиметричних алгоритмів структура інтерфейсу адаптується до специфіки коротких операцій і роботи з ключами. Тут центральним елементом стає блок генерації пари ключів із можливістю представлення публічного та приватного компонентів у формі, придатній для копіювання, відтворення й документування. Також інтерфейс містить області для введення повідомлення та відображення результату перетворення. Такий підхід відповідає практиці використання асиметричних механізмів, де ключовою є коректність ключового матеріалу й контроль того, що зворотна операція відновлює первинний зміст у межах прийнятого регламенту.

Профілі та сценарії у межах інтерфейсу задаються через сукупність параметрів, які визначають, що саме обробляється, яким алгоритмом і в якому режимі виконання. Навіть якщо користувач працює з різними формами або режимами, логіка залишається сталою: спочатку визначаються вхідні дані та контекст, далі запускається операція, після чого результати документуються і готуються до подальшого аналізу. Саме сталість цієї логіки є важливою для відтворюваності, оскільки зменшує вплив «людського фактора» на структуру і повноту отриманих артефактів.

Типові робочі процеси доцільно описувати як регламенти дій користувача, які відображають практичну експлуатацію системи та одночасно підтверджують, що інтерфейс підтримує методику дослідження. Перший процес відповідає запуску операції для одного файла. У цьому сценарії користувач обирає файл як вхідний об’єкт, задає або генерує ключовий матеріал, після чого ініціює шифрування. Система формує вихідний об’єкт у заздалегідь визначеній структурі, відображає перебіг виконання, а після завершення фіксує результати й статус коректності у журналі. Далі користувач запускає зворотну операцію розшифрування для отриманого захищеного подання, що дозволяє підтвердити правильність циклу перетворення. Цей сценарій є базовим, оскільки він

демонструє повний замкнений цикл та дозволяє відразу відокремити коректні налаштування від помилкових.

Другий процес стосується пакетної обробки каталогу або набору файлів. Його особливість полягає в тому, що інтерфейс повинен забезпечити контрольований обхід множини об’єктів, стабільність правил включення файлів до оброблення та однозначну фіксацію результатів. Користувач задає каталог як вхідний об’єкт і ініціює операцію. Система послідовно або за визначеним регламентом обробляє файли, формуючи вихідну структуру, що зберігає зв’язок між вихідними і отриманими об’єктами. У журналі відображаються події по мірі виконання, а у випадку помилок система не приховує їх, а документує із зазначенням, на якому етапі виник збій та які наслідки це має для придатності результатів. Такий сценарій є принципово важливим для експериментальної частини, оскільки саме він найбільш наближений до практик резервного копіювання, архівації або масового захисту користувацьких колекцій даних.

Третій процес відображає виконання короткої асиметричної операції. Користувач ініціює генерацію ключової пари, після чого вводить повідомлення та запускає перетворення. Результат відображається у відповідному полі, а система надає можливість виконати зворотну операцію із використанням приватного компонента ключа. Для дослідницької постановки важливо, що інтерфейс у такому сценарії робить ключові етапи явними: створення ключів, виконання перетворення, отримання результату, перевірка відновлення. Це дозволяє документувати поведінку асиметричних механізмів у керованих умовах і формувати артефакти, які надалі можуть бути зіставлені між різними алгоритмами за однаковим регламентом.

Четвертий процес стосується експорту результатів і підготовки матеріалів для подальшого аналізу. У межах інтерфейсу користувач ініціює формування вихідних матеріалів, що включають збережені журнали, результати валідації та узгоджені зведення, придатні для імпорту в аналітичну підсистему або для використання при побудові таблиць і ілюстрацій у наступних розділах. Методична цінність такого процесу полягає в тому, що користувач не виконує

ручних перетворень даних, які могли б спотворити результати, а отримує стандартизований набір артефактів, пов’язаний із конкретними серіями запусків.

Вихідні матеріали, сформовані системою, слугують доказовою базою проведених запусків і одночасно є джерелом даних для експериментального аналізу. У межах цього підпункту доцільно навести характерні приклади таких матеріалів у вигляді фрагмента журналу та прикладу зведеного представлення. Фрагмент журналу має демонструвати, що запис містить контекст запуску, включно з ідентифікацією сценарію та алгоритму, описом вхідного об’єкта, результатом валідації, статусом завершення і службовими повідомленнями. Такий приклад важливий не стільки як «звіт користувачу», скільки як підтвердження того, що дані журналу є достатніми для перевірки методики і повторного відтворення умов запуску.

Приклад зведення, у свою чергу, має показувати, яким чином результати серії переходять від набору окремих запусків до узгодженого представлення, придатного для порівняльного аналізу. Тут важливо підкреслити, що зведення не існує «саме по собі»: воно має посилатися на первинні записи і зберігати можливість повернення до сирих даних, якщо необхідно уточнити причини відхилень. Саме так вибудовується простежуваність: кожний підсумок у подальших таблицях і графіках може бути співвіднесений із джерельними журналами і перевірений на коректність.

Далі доцільно явно пояснити, як ці матеріали інтегруються у структуру роботи. Журнали і зведення, сформовані інструментальною системою, виступають вхідними даними для експериментального розділу: вони використовуються для агрегування результатів, формування порівняльних представлень, побудови ілюстрацій та обґрунтування висновків. Таким чином, підрозділ про інтерфейс не є відокремленим описом «користувацьких кнопок», а демонструє, що система забезпечує повний цикл: від запуску до формування матеріалів, які безпосередньо застосовуються в аналізі.

Обробка помилок на рівні інтерфейсу є необхідною умовою надійності інструментальної системи, оскільки експерименти можуть виконуватися в

умовах, де виникають некоректні параметри, проблеми доступу до файлів, порушення структури вхідних даних або відмови зовнішніх залежностей. Інтерфейс має забезпечувати зрозуміле повідомлення про проблему, не приховуючи її і не створюючи ілюзії успішного виконання. Повідомлення повинні бути достатньо інформативними, щоб користувач міг визначити, чи є помилка наслідком неправильного вибору вхідного об’єкта, некоректного ключового матеріалу, неможливості запису результату або збою виконання алгоритму.

Важливо, що інтерфейсна обробка помилок повинна бути узгоджена з журналюванням. Тобто будь-яка значуща помилка, що відображається користувачу, має бути відображена і в артефактах запуску, щоб у подальшому було можливим відтворити ситуацію та пояснити, чому певний запуск є непридатним для аналізу. З методичної точки зору це запобігає потраплянню некоректних результатів у експериментальні зведення. Крім того, інтерфейс має підтримувати сценарії коректного повтору: якщо причина помилки усунена, користувач може повторити запуск у тих самих умовах, зберігаючи порівнянність серій.

Окрему увагу доцільно приділити випадкам, коли помилка не є фатальною для всієї серії. У такій ситуації інтерфейс має або запропонувати контрольоване продовження з фіксацією інциденту, або завершити серію у визначеному стані так, щоб результати не трактувалися двозначно. Таким чином, інтерфейс реалізує не лише функцію «зручного керування», але й функцію дисциплінування експерименту: він допомагає користувачу виконувати дослідження у передбачуваному режимі і забезпечує прозорість усіх відхилень.

Висновки до розділу

У розділі обґрунтовано та описано розробку інструментальної системи оцінювання ефективності криптографічних алгоритмів, орієнтованої на виконання порівняльних експериментів у прикладному середовищі. Визначено призначення системи, межі її застосування та сформульовано функціональні й

нефункціональні вимоги, ключовими серед яких є відтворюваність, прозорість процедур, надійність у випадках відхилень і модульність для подальшого розширення. Сценарії використання структуровано відповідно до типових практичних задач (одиничні та пакетні файлові операції, довготривалі процеси, короткі асиметричні перетворення й робота з ключовим матеріалом), що забезпечує методичну відповідність експериментальній частині дослідження.

Запропонована архітектура реалізує розділення відповідальностей між компонентами керування експериментами, сценарного виконання, криптографічних адаптерів, збору показників, валідації, журналювання, сховища та експорту. Така організація забезпечує уніфікований конвеєр запуску, зменшує вплив інструменталізації на результати та зберігає порівнюваність вимірювань між алгоритмами. Описано підсистему збору метрик і журналювання як основу доказовості результатів: кожний запуск документується з контекстом, показниками, статусом коректності та діагностичними подіями. Визначено принципи збереження, структурування та версіонування артефактів, а також механізм експорту, що формує узгоджені зведення та пакет відтворюваності. UML-моделювання (Use Case, Activity, Component, Class, Sequence) використано для формалізації вимог, процесів і взаємодії модулів, а опис інтерфейсу підтверджує практичну придатність системи для керованого виконання експериментів і підготовки матеріалів для подальшого аналізу.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕНННЯ СИМЕТРИЧНИХ АСИМЕТРИЧНИХ ТА ГІБРИДНИХ АЛГОРИТМІВ

4.1 Мета та логіка експериментальної частини

Експериментальна частина дослідження спрямована на отримання емпірично підтверджених висновків щодо поведінки криптографічних алгоритмів у прикладних умовах оброблення даних. Її мета полягає у тому, щоб на основі серій стандартизованих запусків визначити, як різні криптографічні підходи проявляють себе в типових сценаріях використання, а також сформувати обґрунтовані рекомендації для вибору алгоритму з урахуванням практичного контексту. Такий підхід є принциповим, оскільки формальні характеристики алгоритмів не завжди дозволяють передбачити їхню реальну поведінку у середовищі, де на результат впливають не лише властивості перетворення, а й організація даних, специфіка введення-виведення та особливості виконання програмної реалізації.

Логіка побудови експериментів базується на сценарній репрезентативності. Кожна серія запусків відповідає конкретній задачі, яка характерна для застосування криптографії в інформаційних системах: оброблення одиничного файлу, пакетна обробка наборів файлів, виконання тривалих процедур на кшталт архівації або резервного копіювання, а також короткі операції, властиві асиметричним перетворенням і роботі з ключовим матеріалом. Завдяки такій організації результати інтерпретуються не як абстрактні порівняння, а як інженерно значущі спостереження, що безпосередньо описують придатність алгоритму для певного режиму експлуатації.

Експерименти виконуються за єдиним регламентом, який визначає правила підготовки вхідних даних, межі операцій, порядок запусків і вимоги до перевірки коректності. У межах цього регламенту фіксуються характеристики виконання, що відображають часову поведінку операцій, профіль використання ресурсів обчислювальної платформи, а також зміни представлення даних після

перетворення. Окрему увагу приділено стабільності серій і придатності результатів до узагальнення, оскільки без контролю повторюваності та відсіювання некоректних запусків неможливо сформувати науково обґрунтовані висновки навіть за наявності великого обсягу спостережень.

Первинні дані експериментів формуються розробленою інструментальною системою, яка забезпечує стандартизоване виконання сценаріїв, фіксацію контексту запуску та формування артефактів, необхідних для подальшого аналізу. До таких артефактів належать профілі запусків, що описують параметри серій, журнали, які документують перебіг і результат виконання разом зі статусами та повідомленнями, а також експортовані зведення, підготовлені для аналітичної обробки та представлення у вигляді таблиць і графічних матеріалів. Важливою властивістю експериментальної частини є простежуваність: кожне узагальнення має бути відтворювано прив’язане до конкретних запусків і їхнього контексту, що створює підставу для перевірки та інтерпретації результатів.

У підсумку експериментальна частина виступає центральною ланкою, що переводить дослідження від постановки задачі та формалізованої методики до практичних висновків, придатних для застосування. Подальший виклад природно переходить до опису умов проведення експериментів і принципів підготовки вхідних даних, оскільки саме контроль середовища та дисципліна формування наборів визначають коректність і порівнюваність отриманих результатів.

4.2 Умови проведення експериментів і підготовка вхідних даних

Коректність порівняльного експерименту в криптографічних дослідженнях визначається не лише вибором алгоритмів і сценаріїв, а насамперед дисципліною умов виконання та відтворюваністю процедури отримання результатів. Тому експерименти в межах даної роботи проводяться в контрольованому контексті, де середовище запуску розглядається як частина методики. Контрольованість означає фіксованість суттєвих характеристик програмного оточення, стабільність конфігурації виконання та мінімізацію

сторонніх впливів, здатних спотворити результати. Практично це реалізується через уніфікований режим запуску інструментальної системи, стабільну конфігурацію залежностей, визначені правила доступу до даних і стандартизований порядок виконання серій. Важливим аспектом такого підходу є повна документованість контексту: кожний запуск супроводжується збереженням опису середовища та параметрів, достатніх для перевірки того, що результати отримані за однакових умов.

Підготовка вхідних даних виконується за принципом репрезентативності та незмінності наборів у межах експериментальних серій. Репрезентативність означає, що тестові набори відображають типові об’єкти, з якими працюють інформаційні системи: як одиничні файли, так і колекції файлів різної структури, що імітують пакетну обробку, а також набори, наближені до процесів архівації та резервного копіювання. Незмінність означає, що після формування набір використовується як еталонний і не піддається редагуванню, перейменуванню або переформуванню протягом усього циклу експериментів. Для цього вводяться правила, за якими набір описується через маніфест вхідних даних із фіксацією структури, складу та атрибутів, а контроль цілісності виконується на основі криптографічного хешування, що дозволяє виявляти навіть непомітні зміни вмісту. Додатково визначаються правила включення об’єктів до наборів, зокрема оброблення вкладених каталогів, відбір файлів за дозволеними типами та виключення службових або тимчасових елементів, які не мають дослідницької цінності й можуть створювати неузгодженість між серіями.

Однаковість умов між алгоритмами забезпечується через єдину процедуру запуску, яка уніфікує порядок дій для всіх сценаріїв і виключає ручні відхилення. У межах цієї процедури фіксується послідовність етапів: ініціалізація контексту, підготовка вхідного об’єкта, виконання криптографічної операції в заданих межах вимірювання, збір супровідних характеристик виконання, валідація коректності результату, формування та збереження журналу, накопичення артефактів у структурованому сховищі. Для забезпечення порівнюваності важливо, що кожний алгоритм застосовується до одного й того самого

еталонного набору даних у межах відповідної серії, а порядок виконання серій задається регламентом, який зменшує вплив випадкових факторів, пов’язаних зі станом системи. Особливу роль відіграє валідація коректності перетворень: результати криптографічних операцій приймаються до подальшого аналізу лише за умови підтвердження правильності відновлення даних у відповідних сценаріях або підтвердження коректності виконання коротких операцій для асиметричних перетворень. Некоректні запуски не змішуються з валідними результатами, а маркуються у журналі та зберігаються з повним контекстом, що дозволяє пояснити причину відхилення і, за необхідності, відтворити ситуацію.

Фіксація контексту в журналах розглядається як обов’язкова умова наукової прозорості. Журнал експерименту має відображати не лише факт виконання операції, а й умови, за яких вона виконувалась: ідентифікацію серії та запуску, опис сценарію, конфігурацію алгоритму та режиму застосування, характеристики вхідного об’єкта, відомості про підготовку вихідних матеріалів, статус завершення, результат валідації та службові повідомлення. Такий підхід забезпечує простежуваність: будь-яке узагальнення в подальшому аналізі може бути перевірене шляхом повернення до первинних записів, а неоднозначні випадки можуть бути інтерпретовані на основі збереженого контексту, а не на основі припущень.

Для подальшого аналізу визначається набір артефактів серії, які вважаються обов’язковими. До них належить профіль запуску, що відтворює параметри серії та порядок виконання; сирі журнали запусків як первинний доказовий матеріал; результати валідації, що підтверджують коректність перетворень і відмежовують некоректні запуски; маніфест вхідних даних із правилами формування набору та контрольними ознаками цілісності; структуроване сховище вихідних об’єктів, сформованих у процесі виконання; експортовані зведення, підготовлені для аналітичної обробки і подальшого представлення у вигляді таблиць та ілюстрацій. Наявність саме цього набору матеріалів дозволяє виконувати узагальнення без втрати простежуваності, а

також забезпечує можливість незалежної перевірки методики та відтворення експериментів у тому самому регламенті.

4.3 Експериментальні дослідження

Експериментальна частина дослідження була організована як послідовність серій вимірювань, спрямованих на кількісну оцінку продуктивності та ресурсоспоживання криптографічних алгоритмів у різних умовах. Основна увага приділялася впливу трьох груп чинників: обсягу та типу даних, характеристик апаратної платформи (тип носія інформації) та особливостей алгоритмічного підходу (симетричні, асиметричні й багатопотокові реалізації). Такий підхід дозволив максимально наблизити експерименти до реальних сценаріїв роботи прикладних інформаційних систем.

Усі вимірювання проводились на персональному комп’ютері в контрольованих умовах, із фіксованою конфігурацією програмного середовища та однаковими параметрами запуску для всіх алгоритмів. Для забезпечення порівнюваності результатів під час серій експериментів мінімізувалось фонове навантаження та використовувався сталий режим живлення. Тестування проводилося на ПК з такими характеристиками:

1. Процесор: Intel Core i5-9400F, 6 ядер, 6 потоків, частота 2.9 ГГц;

2. Оперативна пам’ять: 16 ГБ, DDR4 – 3200 МГц;

3. Накопичувачі: M.2 Kingston 1ТБ, SSD Samsung 500 ГБ, HDD Seagate BarraCuda 2ТБ;

4. Операційна система: Windows 10;

Дослідження симетричних алгоритмів на синтетичних наборах даних. Перший блок експериментів був присвячений аналізу поведінки симетричних алгоритмів при шифруванні файлів фіксованого розміру. Для цього було сформовано три синтетичні набори: файли обсягом 1 МБ, 10 МБ та 100 МБ. У кожному наборі дані не мали спеціальної внутрішньої структури, що дозволило відокремити власне обчислювальні властивості алгоритмів від впливу факторів файлу як об’єкта файлової системи.

Для кожного з цих наборів виконувалося шифрування та розшифрування з використанням алгоритмів AES, DES, 3DES, Serpent, ChaCha20 та Blowfish. Вимірювання проводилися на трьох типах носіїв: високошвидкісному накопичувачі M.2, твердотільному SSD та традиційному жорсткому диску HDD. Для кожної комбінації «носій – алгоритм – розмір файлу» фіксувалися час шифрування й розшифрування, середня швидкість обробки даних, пікове завантаження центрального процесора та пікове використання оперативної пам’яті. Узагальнені результати цих вимірювань наведено у таблицях 4.1 – 4.2.

Таблиця 4.1 – Симетричні алгоритми шифрування файлів обсягом 1, 10 та 100 МБ на різних носіях

Тип носія	Алгоритм	Об'єм даних	Шифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
M.2	AES	1 МБ	0,05	18,64	0	39,8
		10 МБ	0,03	305,58	15,8	78,9
		100 МБ	0,31	321,29	15	61,4
	DES	1 МБ	0,04	22,77	11,9	43,7
		10 МБ	0,37	26,94	14,7	79,6
		100 МБ	3,4	29,45	16,9	62,8
	ChaCha20	1 МБ	0,03	33,2	8,6	207,4
		10 МБ	0,14	70,74	20,3	207,5
		100 МБ	1,4	71,47	16,7	207,8
	Serpent	1 МБ	0,06	16,75	13,3	59,2
		10 МБ	0,46	21,66	16,9	59,2
		100 МБ	5	20,01	16,7	59,3
	3Des	1 МБ	0,04	23,68	18,2	59,4
		10 МБ	0,35	28,84	17,2	61,1
		100 МБ	3,46	28,87	16,6	61,1
	Blowfish	1 МБ	0,03	37,2	19,3	45
		10 МБ	0,16	61,63	16	45
		100 МБ	1,61	62,23	16,7	45

Продовження таблиці 4.1

Тип носія	Алгоритм	Об'єм даних	Шифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
SSD	AES	1 МБ	0,01	108,9	0	46,9
		10 МБ	0,08	118,88	6,1	83,8
		100 МБ	0,36	274,41	12,1	65,9
	DES	1 МБ	0,07	14,33	7,6	210,9
		10 МБ	0,37	26,76	16	66
		100 МБ	3,39	29,51	16,8	130,1
	ChaCha20	1 МБ	0,02	62,51	16,3	83
		10 МБ	0,16	62,7	13	83
		100 МБ	1,28	78,42	17,1	83
	Serpent	1 МБ	0,05	19,8	15,4	83,3
		10 МБ	0,46	21,64	16,9	83,4
		100 МБ	4,55	21,96	16,6	83,4
	3Des	1 МБ	0,07	13,76	10,8	83,4
		10 МБ	0,34	29,09	19,7	84
		100 МБ	3,43	29,16	16,9	84
	Blowfish	1 МБ	0,02	47,85	12,5	83,9
		10 МБ	0,16	62,44	19,5	83,9
		100 МБ	1,57	63,82	16,9	83,9
HDD	AES	1 МБ	0,03	31,58	16,3	48,1
		10 МБ	0,12	85,78	4,4	85
		100 МБ	0,38	264	11,7	67,1
	DES	1 МБ	0,53	1,88	1,5	108,8
		10 МБ	0,35	28,79	16,5	147,3
		100 МБ	3,4	29,4	16,9	124,1
	ChaCha20	1 МБ	0,03	34,96	8,8	76,8
		10 МБ	0,14	72,37	17	76,8
		100 МБ	1,3	76,7	17	76,8
	Serpent	1 МБ	0,06	17,34	13,6	76,9
		10 МБ	0,46	21,93	16,6	76,9
		100 МБ	4,65	21,49	16,4	77,3

Продовження таблиці 4.1

Тип носія	Алгоритм	Об'єм даних	Шифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
HDD	3Des	1 МБ	0,04	24,55	19,1	79,2
		10 МБ	0,34	29	16,6	80,6
		100 МБ	3,43	29,16	16,4	80,6
	Blowfish	1 МБ	0,04	25,06	19,3	48,6
		10 МБ	0,35	28,84	17,3	48,6
		100 МБ	1,58	63,14	16,8	48,6

Таблиця 4.2 – Симетричні алгоритми дешифрування файлів обсягом 1, 10 та 100 МБ на різних носіях

Тип носія	Алгоритм	Об'єм даних	Дешифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
M.2	AES	1 МБ	0,03	30,39	15,8	58,7
		10 МБ	0,07	142,05	11,1	77,1
		100 МБ	0,57	176,91	11,1	77,6
	DES	1 МБ	0,06	16,19	12,6	59,8
		10 МБ	0,4	25,14	15,7	78,7
		100 МБ	3,93	25,42	15,4	206,9
	ChaCha20	1 МБ	0,02	51,51	13,1	207,5
		10 МБ	0,14	70,76	20,2	207,6
		100 МБ	1,45	69,22	16,9	208
	Serpent	1 МБ	0,06	17,7	18,7	59,2
		10 МБ	0,48	20,77	16,8	59,2
		100 МБ	4,84	20,67	16,7	59,3
	3Des	1 МБ	0,05	22,2	17,4	60,7
		10 МБ	0,36	27,75	16,6	61,1
		100 МБ	3,6	27,76	16,8	44,9
	Blowfish	1 МБ	0,02	46,82	12,5	45
		10 МБ	0,17	58,14	16,6	45
		100 МБ	1,67	60,05	16,7	45

Продовження таблиці 4.2

Тип носія	Алгоритм	Об'єм даних	Дешифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
SSD	AES	1 МБ	0,03	36,96	9,5	63,9
		10 МБ	0,08	129,22	13,4	81,9
		100 МБ	0,61	164,95	9	210
	DES	1 МБ	0,06	16,7	13	228
		10 МБ	0,39	25,48	17,3	82
		100 МБ	3,92	25,52	15,6	82,1
	ChaCha20	1 МБ	0,02	57,7	15,6	83
		10 МБ	0,13	75,64	15,8	83,1
		100 МБ	1,27	78,93	16,8	83
	Serpent	1 МБ	0,05	19,34	15,2	83,4
		10 МБ	0,5	19,96	16,1	83,4
		100 МБ	4,77	20,98	16,8	83,4
	3Des	1 МБ	0,04	23,53	18,2	83,9
		10 МБ	0,36	27,98	16,8	84
		100 МБ	3,55	28,19	16,8	83,9
	Blowfish	1 МБ	0,02	49,61	13	83,9
		10 МБ	0,17	60,39	17,3	83,9
		100 МБ	1,71	58,38	15,7	83,9
HDD	AES	1 МБ	0,07	14,1	11	65,1
		10 МБ	0,18	55,31	5,8	83,1
		100 МБ	1,32	75,87	4,9	105,8
	DES	1 МБ	0,1	12,9	9,89	127
		10 МБ	0,49	20,37	12,7	183,5
		100 МБ	4,39	22,78	13,9	76,1
	ChaCha20	1 МБ	0,02	51,52	0	76,8
		10 МБ	0,14	74,04	17,4	76,8
		100 МБ	1,28	77,97	16,6	76,8
	Serpent	1 МБ	0,05	18,19	14,1	76,9
		10 МБ	0,49	20,61	16,6	77
		100 МБ	4,81	20,8	16,6	77,6

Продовження таблиці 4.2

Тип носія	Алгоритм	Об'єм даних	Дешифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
HDD	3Des	1 МБ	0,05	21,89	17	80,3
		10 МБ	0,36	27,41	16,4	80,6
		100 МБ	3,58	27,95	16,7	48,6
	Blowfish	1 МБ	0,04	24,08	12,7	48,6
		10 МБ	0,36	27,85	16,7	48,6
		100 МБ	1,66	60,18	16,9	48,6

Аналіз таблиць 4.1 – 4.2 показує, що для всіх розглянутих типів носіїв спостерігається очікувана лінійна залежність часу обробки від розміру файлів: при переході від 1 МБ до 100 МБ час шифрування зростає в десятки разів, а середня швидкість, навпаки, стабілізується й стає більш репрезентативною характеристикою алгоритму. Наприклад, для AES на носії M.2 середня швидкість шифрування файлів обсягом 100 МБ перевищує 320 МБ/с, тоді як для DES при тих самих умовах вона становить близько 30 МБ/с. Це означає, що AES у даному сценарії працює більш ніж у десять разів швидше за DES. Подібні співвідношення спостерігаються й у порівнянні з Serpent та 3DES, які демонструють швидкість на рівні 20–30 МБ/с для великих файлів, тоді як AES стабільно утримує порядок сотень мегабайт за секунду.

Окремий інтерес становлять ChaCha20 та Blowfish, які за середньою швидкістю часто займають проміжне положення між AES і повільнішими DES/3DES/Serpent. Для файлів обсягом 100 МБ на SSD та HDD ChaCha20 демонструє швидкість порядку 70–80 МБ/с, тоді як Blowfish – близько 60–65 МБ/с. Це означає, що в сценаріях, де AES з якихось причин недоступний або недоцільний, саме ці алгоритми можуть розглядатися як продуктивна альтернатива з помірним завантаженням процесора.

Щоб наочно показати різницю в продуктивності, результати таблиці 3.1 доцільно доповнити графічним представленням.

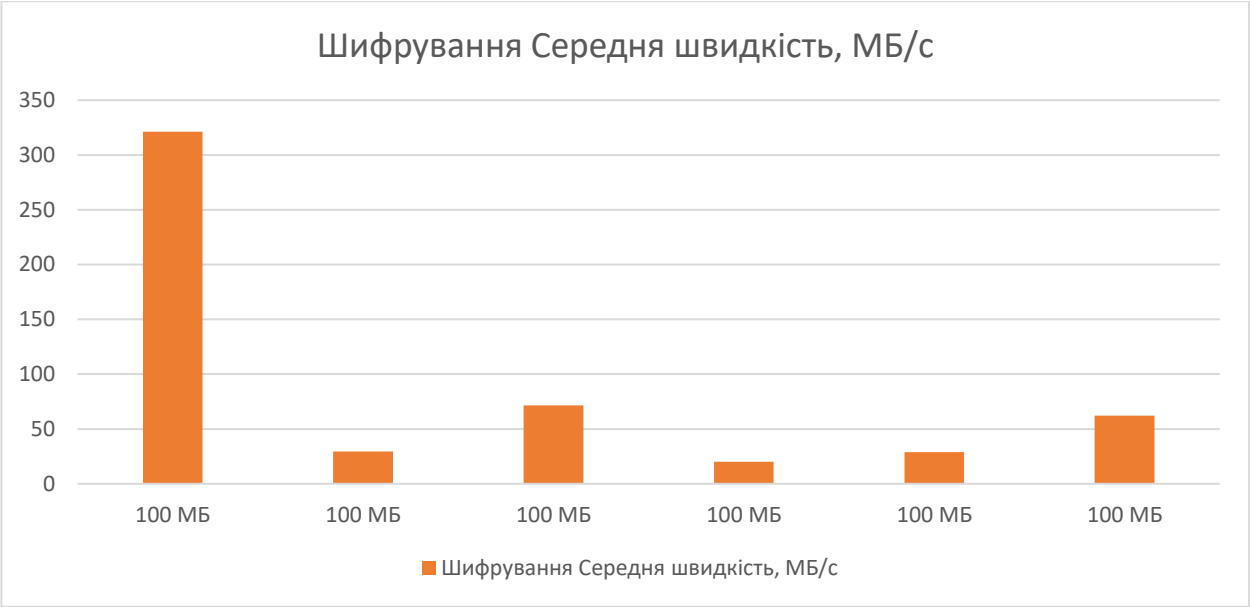

Рисунок 4.1 – Середня швидкість шифрування файлів обсягом 100 МБ на M.2

Аналіз рисунку 4.1 підтверджує домінування AES за продуктивністю: його стовпчик суттєво перевищує значення для інших алгоритмів. ChaCha20 та Blowfish показують приблизно вдвічі–утричі меншу швидкість, але все одно залишаються помітно швидшими, ніж DES, 3DES і Serpent. Останні, своєю чергою, демонструють подібний рівень продуктивності, що робить їх менш привабливими для задач, де критичною є швидкість обробки великих обсягів даних.

Для оцінки впливу типу носія на роботу алгоритмів доцільно порівняти поведінку одного обраного шифру (наприклад, AES) на M.2, SSD та HDD.

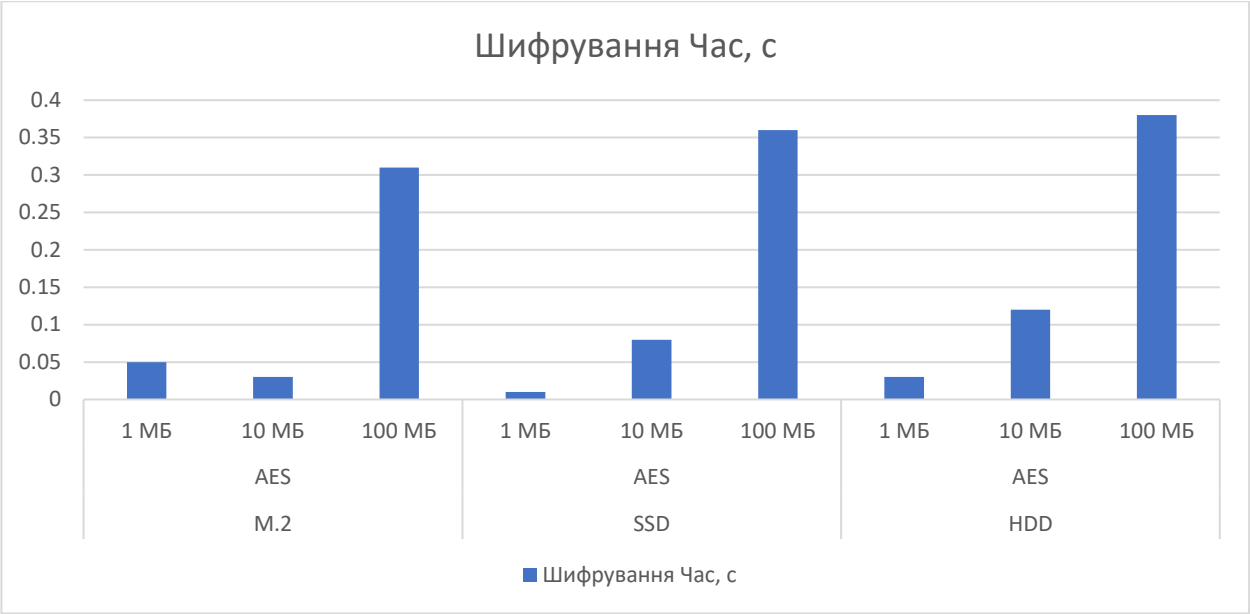

Рисунок 4.2 – Залежність середньої швидкості шифрування AES від типу носія для файлів обсягом 1, 10 та 100 МБ

З рисунку 4.2 видно, що при малих обсягах даних (1 МБ) різниця між типами носіїв практично не впливає на результат: час виконання настільки малий, що домінують накладні витрати на ініціалізацію та роботу алгоритму. Однак для файлів обсягом 100 МБ різниця стає більш відчутною: на M.2 та SSD AES демонструє максимально можливу для платформи швидкість, тоді як на HDD спостерігається певне просідання, пов’язане з обмеженнями підсистеми читання/запису та механічною природою жорсткого диска.

Дослідження симетричних алгоритмів на реальних наборах даних. Другий блок експериментів був спрямований на дослідження поведінки симетричних алгоритмів у реалістичних умовах роботи з великими наборами файлів. Для цього було сформовано три типові набори: набір фотографій, набір відеофайлів та набір текстових документів. Загальний обсяг кожного набору становив приблизно 500 МБ, що відповідає типовим задачам резервного копіювання, архівування користувацьких колекцій мультимедіа чи документів.

Для кожного набору виконувалося шифрування та розшифрування алгоритмами AES, DES, 3DES, Serpent, ChaCha20 і Blowfish на трьох типах носіїв: M.2, SSD та HDD. На відміну від синтетичних файлів фіксованого

розміру, реальні дані характеризуються різною внутрішньою структурою та ступенем фрагментації, що дозволяє оцінити інтегральний вплив файлової підсистеми, кешування та файлових систем на загальну продуктивність. Узагальнені результати для цього блоку наведено в таблицях 4.3 – 4.4.

Таблиця 4.3 – Шифрування різних наборів даних на різних носіях

Тип носія	Алгоритм	Об'єм даних	Шифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
M.2	AES	Фото	18,56	27,06	17,6	41,4
		Відео	2,03	248,73	9,7	61,5
		Текст	2,38	210,34	8,5	157,1
	DES	Фото	29,65	16,94	19,3	45,4
		Відео	17,21	29,41	16,3	61,7
		Текст	16,8	29,78	16,6	61,3
	ChaCha20	Фото	13,02	38,58	16	62,6
		Відео	6,6	76,7	16,7	146,9
		Текст	7,28	68,73	16,1	77,2
	Serpent	Фото	30,64	16,39	16,1	46,6
		Відео	26,78	18,9	16,1	147,1
		Текст	25,24	19,82	16,7	77,2
	3Des	Фото	32,01	15,69	16	46,6
		Відео	16,94	29,88	16,7	41
		Текст	16,73	29,89	16,8	41,3
	Blowfish	Фото	13,02	38,57	17,2	42,4
		Відео	7,95	63,69	16,8	41
		Текст	7,7	64,95	16,7	41,2
SSD	AES	Фото	28,88	17,39	6,7	41,9
		Відео	2,26	223,59	8,1	62,1
		Текст	1,29	391,96	17,3	61,5
	DES	Фото	22,92	21,91	20,2	43
		Відео	16,94	29,88	16,7	61,8
		Текст	19,72	25,67	14,3	61,4

Продовження таблиці 4.3

Тип носія	Алгоритм	Об’єм даних	Шифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
SSD	ChaCha20	Фото	9,38	53,53	17,6	59,4
		Відео	6,37	79,48	16,8	77,9
		Текст	7,21	70,2	14,7	77,4
	Serpent	Фото	28,05	17,9	17,3	43,1
		Відео	25,19	20,1	16,6	77,5
		Текст	27,12	18,66	15,5	77,4
	3Des	Фото	26,71	18,81	16,9	43,2
		Відео	17,09	29,61	16,5	40,5
		Текст	17,6	28,76	16,2	39,8
	Blowfish	Фото	11,49	43,72	17,6	44,4
		Відео	9,93	50,95	13,4	40,5
		Текст	7,84	64,59	16,5	39,9
HDD	AES	Фото	58,57	8,57	4,1	45,2
		Відео	5,84	86,61	3,5	66,1
		Текст	6,13	81,62	3,4	157,2
	DES	Фото	31,58	15,9	15	45,5
		Відео	17,62	28,73	16,3	66,2
		Текст	17,12	29,22	16,4	125,2
HDD	ChaCha20	Фото	11,32	44,36	14,6	62,6
		Відео	6,45	78,53	16,5	151,2
		Текст	6,53	76,56	16,3	77,1
	Serpent	Фото	28,77	17,46	16,9	45,5
		Відео	25,02	20,23	16,7	151,3
		Текст	26,32	19	16	77,1
	3Des	Фото	27,32	18,38	16,5	47
		Відео	17,06	29,66	16,6	45,1
		Текст	16,91	29,58	16,6	44,7
	Blowfish	Фото	13,28	37,83	16	45,8
		Відео	7,92	63,91	16,5	45,1
		Текст	8,62	57,99	15,2	44,7

Таблиця 4.4 – Дешифрування різних наборів даних на різних носіях

Тип носія	Алгоритм	Об’єм даних	Дешифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
M.2	AES	Фото	68,4	7,34	18,4	60,2
		Відео	2,47	204,61	12,7	77,5
		Текст	2,6	192,04	13,2	77
	DES	Фото	80,91	6,21	18,3	61,6
		Відео	18,29	27,67	16,2	146,8
		Текст	18,77	26,64	15,9	77,3
	ChaCha20	Фото	13,16	38,15	16,5	46,9
		Відео	6,49	78,03	16,7	147
		Текст	7,09	70,51	16,6	77,1
	Serpent	Фото	28,69	17,51	17	46,9
		Відео	24,95	20,29	16,7	147,1
		Текст	24,88	20,1	16,7	77,2
	3Des	Фото	28,46	17,65	17,65	39,1
		Відео	17,61	28,74	16,7	41
		Текст	17,47	28,63	16,7	41,3
	Blowfish	Фото	13,12	38,27	17,4	41,2
		Відео	9,45	53,55	15,2	41
		Текст	8,03	62,31	16,7	41,2
SSD	AES	Фото	63,27	7,94	17,4	58,9
		Відео	5,85	86,45	5,5	78,1
		Текст	6,72	75,32	5,7	77,5
	DES	Фото	73,15	6,87	18	59
		Відео	20,29	24,95	15	78
		Текст	19,63	25,79	15,2	77,6
	ChaCha20	Фото	8,97	56	18,1	43,9
		Відео	6,26	80,85	16,7	77,9
		Текст	7,39	68,45	14,5	77,4
	Serpent	Фото	28,53	17,6	16,8	43,3
		Відео	25	20,25	16,6	77,5
		Текст	30,25	16,73	14,1	77,4

Продовження таблиці 4.4

Тип носія	Алгоритм	Об’єм даних	Шифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
SSD	3Des	Фото	27,3	18,4	17	43,3
		Відео	19,63	25,78	15,4	40,5
		Текст	18,21	27,8	16,4	39,8
	Blowfish	Фото	10,96	45,82	18,7	45,4
		Відео	9,33	54,28	14,8	40,5
		Текст	8,3	60,95	16,4	39,9
HDD	AES	Фото	172,14	2,92	7,1	61,5
		Відео	6,34	79,79	4,6	82,2
		Текст	6,3	79,35	5,2	77,2
	DES	Фото	172,36	2,91	7,9	61,6
		Відео	22,75	22,25	13,6	151,4
		Текст	22,37	22,36	13,4	77,3
	ChaCha20	Фото	10,21	49,21	17,4	47,1
		Відео	6,41	79,02	16,7	151,2
		Текст	6,29	79,54	16,6	77,1
	Serpent	Фото	32,73	15,35	15,3	46
		Відео	26	19,47	16,1	151,3
		Текст	24,63	20,31	16,6	76,6
	3Des	Фото	27,88	18,01	16,6	46,8
		Відео	17,61	28,74	16,6	45,1
		Текст	17,62	28,39	16,5	44,6
	Blowfish	Фото	13	38,63	16,6	46,8
		Відео	8,33	60,77	16,4	45,1
		Текст	8,05	62,09	16,5	44,6

Дані таблиць 4.3 – 4.4 показують, що коефіцієнт розширення для більшості симетричних алгоритмів залишається близьким до одиниці: розмір зашифрованих наборів практично не відрізняється від вихідного обсягу, що є очікуваною властивістю блочних шифрів у типових режимах. Натомість найбільші відмінності спостерігаються у часових показниках та середніх швидкостях. Для відеофайлів, які, як правило, мають велику тривалість

послідовного читання, AES демонструє дуже високі швидкості, особливо на M.2 та SSD. Для наборів фотографій та текстових документів швидкість дещо знижується через велику кількість окремих файлів і збільшення накладних витрат на файлові операції, але загальна картина зберігається: AES, ChaCha20 та Blowfish є найпродуктивнішими, тоді як DES, 3DES і Serpent помітно відстають.

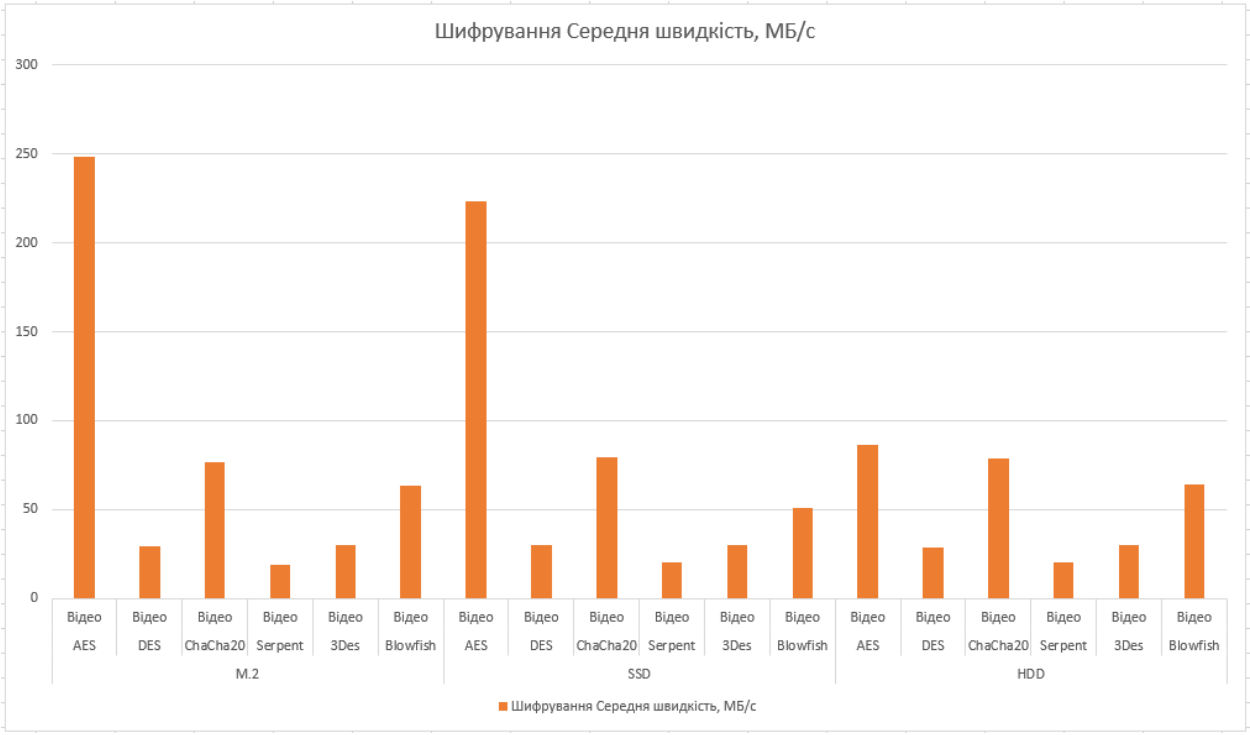

Рисунок 4.3 – Швидкість шифрування відео на різних носіях

Рисунок 4.3 підтверджує результати таблиць: на M.2 та SSD AES та ChaCha20 забезпечують мінімальний час шифрування великих послідовних файлів, тоді як на HDD різниця між алгоритмами частково «розмивається» через обмеження самої дискової підсистеми, але все одно зберігається тенденція до лідерства AES.

Рисунок 4.4 – Час шифрування на SSD

За рисунок 4.4 видно, що при роботі з великою кількістю дрібних текстових файлів абсолютні значення швидкості зменшуються порівняно з відео, однак співвідношення між алгоритмами залишається сталим: AES і ChaCha20 знову демонструють найкращі показники, Blowfish займає проміжну позицію, тоді як DES, 3DES та Serpent є найменш продуктивними. Це свідчить про стабільність поведінки алгоритмів у різних сценаріях.

Дослідження однопотокової та багатопотокової реалізацій AES. Наступним кроком стала спеціальна серія експериментів, присвячена оцінюванню впливу багатопотокової обробки на продуктивність алгоритму AES. Мета цієї серії полягала в тому, щоб з’ясувати, чи дає використання кількох ядер процесора реальний виграш у швидкодії, і за яких умов цей виграш є максимальним або, навпаки, відсутнім.

Було сформовано два набори файлів: невеликий набір із 84 файлів (сценарій обробки обмеженої кількості відносно великих об’єктів) та великий набір із 3154 файлів (сценарій масової обробки великої кількості дрібніших

об’єктів). Обидва набори шифрувалися та розшифровувалися двома реалізаціями: однопотоковою (умовно позначеною як AES) та багатопотоковою, що використовує до шести ядер процесора (AES_6Core). Вимірювання проводилися на носіях M.2, SSD та HDD.

Таблиця 4.5 – Порівняння однопотокового та багатопотокового шифрування

Алгоритм	Носій інформації	Кількість файлів	Шифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
AES	M2	84	1,29	395,51	12,2	51,2
	SSD	84	4,36	116,8	8,3	54,2
	HDD	84	1,39	366,81	12,6	44,9
AES_6Core	M2	84	1,58	322,43	16,3	49,5
	SSD	84	1,35	376,46	16,1	51
	HDD	84	6,31	80,68	16,5	52,7
AES	M2	3154	18,56	27,06	17,6	41,4
	SSD	3154	28,88	17,39	6,7	41,9
	HDD	3154	58,57	8,57	4,1	45,2
AES_6Core	M2	3154	7,11	70,66	17,1	49,5
	SSD	3154	6,68	75,16	17	49,4
	HDD	3154	80,17	6,26	5,5	51,3

Таблиця 4.6 – Порівняння однопотокового та багатопотокового дешифрування

Алгоритм	Носій інформації	Кількість файлів	Дешифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
AES	M2	84	2,69	189,07	11,2	67,4
	SSD	84	4,54	112,33	8,1	70,3
	HDD	84	9,17	55,53	12,9	64,7

Продовження таблиці 4.6

Алгоритм	Носій інформації	Кількість файлів	Дешифрування			
			Час, с	Середня швидкість, МБ/с	Пікове завантаження CPU, %	Пікове використання RAM, МБ
AES_6Core	M2	84	2,03	250,7	16,1	50,5
	SSD	84	3,72	137,04	16,1	52,3
	HDD	84	8,65	58,91	9,4	55
AES	M2	3154	68,4	7,34	18,4	60,2
	SSD	3154	63,27	7,94	17,4	58,9
	HDD	3154	172,14	2,92	7,1	61,5
AES_6Core	M2	3154	13,88	36,19	16,3	51,1
	SSD	3154	16,58	30,29	16,8	51,6
	HDD	3154	96,84	5,19	4,3	52,4

Аналіз таблиць 4.5 – 4.6 показує, що ефект від багатопотоковості істотно залежить від поєднання трьох чинників: типу носія, обсягу даних (кількості файлів) та характеру операції (шифрування чи розшифрування. Для невеликого набору з 84 файлів на швидкому накопичувачі M.2 однопотокова реалізація навіть виявляється швидшою, оскільки накладні витрати на створення та синхронізацію потоків переважають потенційний виграш. Натомість для SSD у тому самому сценарії багатопотокова реалізація дає суттєве скорочення часу шифрування.

Найяскравіші відмінності спостерігаються для великого набору з 3154 файлів. На M.2 та SSD багатопотоковий AES_6Core забезпечує скорочення часу шифрування в кілька разів і суттєве зростання середньої швидкості. Водночас на HDD паралельна обробка призводить до різкого погіршення продуктивності, оскільки головним «вузьким місцем» стає повільний носій, а не процесор: збільшується кількість операцій позиціонування головок, а загальний час доступу до даних істотно зростає.

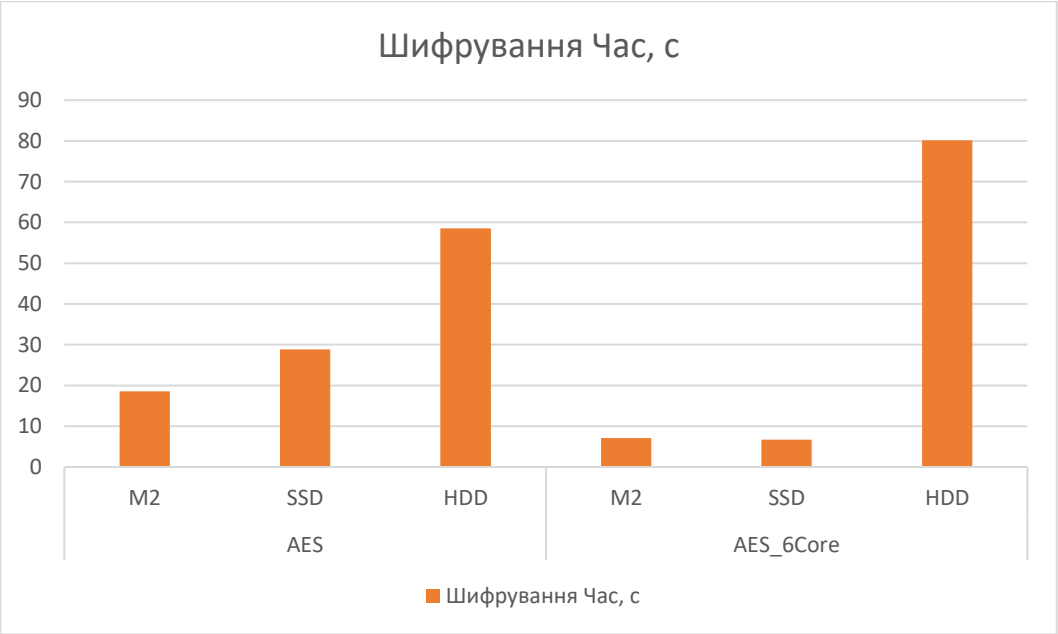

Рисунок 4.5 – Час шифрування набору з 3154 файлів алгоритмами AES та AES_6Core на різних носіях

Рисунок 4.5 наочно підтверджує, що багатопотокова реалізація є ефективною лише там, де дискова підсистема не є критичним обмеженням: на M.2 та SSD час обробки скорочується у 3–4 рази, тоді як на HDD він, навпаки, збільшується.

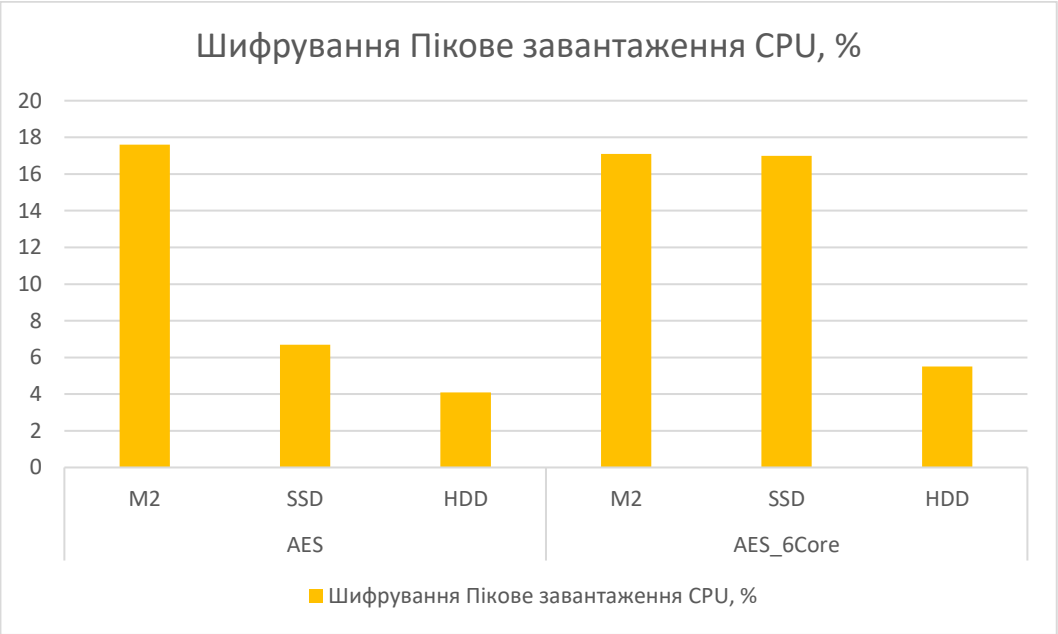

Рисунок 4.6 – Пікове завантаження CPU при шифруванні

З рисунку 4.6 видно, що багатопотокова реалізація значно повніше задіює процесор (особливо на M.2 та SSD), але при цьому навіть у піку завантаження не досягає 100 %, залишаючи резерв для інших процесів. На HDD загальне завантаження CPU залишається невисоким, що ще раз підтверджує домінуючу роль дискових затримок у цьому сценарії.

Для перевірки відтворюваності результатів та оцінки впливу конфігурації обладнання було проведено додаткову серію вимірювань на іншому персональному комп'ютері з більш потужним процесором:

1. Процесор: Intel Core i5-10400F, 6 ядер, 12 потоків, частота 3.2 ГГц;
2. Оперативна пам'ять: 16 ГБ, DDR4 – 3200 МГц;
3. Накопичувачі: M.2 Samsung 1ТБ, SSD Samsung 120 ГБ, HDD Seagate BarraCuda 1ТБ;
4. Операційна система: Windows 10;

У цій серії використовувалися реальні набори фотографій, відеофайлів та текстових документів із сумарним обсягом близько 500 МБ. Для кожного типу даних виконувалося шифрування й розшифрування алгоритмами AES, DES, 3DES, Serpent, ChaCha20 та Blowfish, з фіксацією часу, середньої швидкості, завантаження CPU та використання RAM.

Таблиця 4.7 – Результати Ш/Д часу та середньої швидкості на іншому ПК

Тип носія	Алгоритм	Тип даних	Час, с		Середня швидкість, МБ/с	
			Шифрування	Дешифрування	Шифрування	Дешифрування
M.2	AES	Фото	1,29	2,69	395,51	189,07
		Відео	1,29	1,79	393,71	282,26
		Текст	1,58	1,74	316,82	286,9
	DES	Фото	17,4	16,52	29,28	30,84
		Відео	16,4	17,34	30,87	29,19
		Текст	16,09	17,17	31,09	29,13
	ChaCha20	Фото	6,17	6,14	82,61	82,97
		Відео	5,98	6	84,57	84,32
		Текст	5,91	5,99	84,6	83,46

Продовження таблиці 4.7

Тип носія	Алгоритм	Тип даних	Час, с		Середня швидкість, МБ/с	
			Шифрування	Дешифрування	Шифрування	Дешифрування
M.2	Serpent	Фото	27,2	23,36	18,73	21,81
		Відео	22,64	23,08	22,36	21,93
		Текст	21,59	22,84	23,17	21,9
	3Des	Фото	16,65	17,3	30,61	29,45
		Відео	16,36	17,88	30,93	28,31
		Текст	16,82	16,68	29,74	29,98
	Blowfish	Фото	8,42	8,3	60,47	61,37
		Відео	8,09	7,74	62,59	65,41
		Текст	7,61	7,76	65,72	64,49
SSD	AES	Фото	2,12	1,76	240,07	289,67
		Відео	0,86	1,24	587,92	408,39
		Текст	1,34	1,27	372,02	392,93
	DES	Фото	3,67	4,52	138,68	112,63
		Відео	2,8	5,53	180,62	91,48
		Текст	3,05	2,68	161,4	186,48
	ChaCha20	Фото	18,81	21,88	27,08	23,29
		Відео	16,98	19,61	29,81	25,82
		Текст	16,35	18,77	30,59	26,64
	Serpent	Фото	6,43	7,48	79,22	68,11
		Відео	6,39	6,38	79,19	79,38
		Текст	6,38	5,97	78,43	83,82
	3Des	Фото	22,18	24,66	22,97	20,66
		Відео	23,14	23,54	21,87	21,5
		Текст	22,41	23,42	22,32	21,35
	Blowfish	Фото	17,8	17,61	28,62	28,92
		Відео	16,52	16,97	30,64	29,83
		Текст	16,38	17,22	30,54	29,05
HDD	AES	Фото	8,13	8,93	62,63	57,07
		Відео	7,48	7,96	67,67	63,62
		Текст	7,48	7,84	66,85	63,77
	DES	Фото	1,41	1,73	362,39	294,29
		Відео	0,81	1,22	624,4	414,7
		Текст	0,8	1,31	623,83	382,09

Продовження таблиці 4.7

Тип носія	Алгоритм	Тип даних	Час, с		Середня швидкість, МБ/с	
			Шифрування	Дешифрування	Шифрування	Дешифрування
HDD	ChaCha20	Фото	6,93	6,37	73,48	79,94
		Відео	6,22	6,09	81,38	83,13
		Текст	6,85	6,1	72,96	82
	Serpent	Фото	22,79	23,58	22,36	21,61
		Відео	22,29	22,95	22,71	22,06
		Текст	22,02	22,85	22,71	21,89
	3Des	Фото	16,71	17,56	30,48	29,06
		Відео	16,21	17,24	31,22	29,35
		Текст	16,12	16,78	31,02	29,8
	Blowfish	Фото	7,91	8,34	64,39	61,06
		Відео	7,66	8,04	66,09	62,98
		Текст	7,38	7,67	67,73	65,2

Таблиця 4.8 – Результати Ш/Д пікового завантаження CPU та RAM на іншому ПК

Тип носія	Алгоритм	Тип даних	Пікове завантаження CPU, %		Пікове використання RAM, МБ	
			Шифрування	Дешифрування	Шифрування	Дешифрування
M.2	AES	Фото	8,9	7,5	51,2	67,4
		Відео	6,6	7	68,1	84,2
		Текст	5,2	7,2	67	83,1
	DES	Фото	7,9	8,4	50,9	50,9
		Відео	8,1	8,2	82,6	82,7
		Текст	8,1	8,2	65,6	81,7
	ChaCha20	Фото	8,3	8,4	81,7	81,9
		Відео	8,3	8,3	81,9	82,2
		Текст	8,3	8,3	82,5	82,2
	Serpent	Фото	7,8	8,4	83,1	83,6
		Відео	8,3	8,3	83,5	83,6
		Текст	8,3	8,3	83,5	83,6

Продовження таблиці 4.8

Тип носія	Алгоритм	Тип даних	Пікове завантаження CPU, %		Пікове використання RAM, МБ	
			Шифрування	Дешифрування	Шифрування	Дешифрування
M.2	3Des	Фото	8,3	8,3	48	47,8
		Відео	8,3	8,3	40,5	41,4
		Текст	8,2	8,3	41,7	41,9
	Blowfish	Фото	7,6	8,3	43,9	44,1
		Відео	8,1	8,3	44,1	44,1
		Текст	8,3	8,4	44,1	44,1
SSD	AES	Фото	5,6	9,3	4,8	2,9
		Відео	8,3	9	2,7	1,6
		Текст	5,6	8,7	0,5	0,1
	DES	Фото	3,2	5,7	50,8	67,2
		Відео	2,9	2,8	66,6	151,7
		Текст	3,1	5,5	161,4	81,5
	ChaCha20	Фото	7,5	7,2	50,4	66,8
		Відео	8,1	7,2	66,1	151,4
		Текст	8,1	7,8	160,7	81
	Serpent	Фото	7,8	7,1	65	65
		Відео	8,1	7,9	64,8	64,8
		Текст	8	8,2	64,8	64,8
	3Des	Фото	8,3	8,1	64,9	64,9
		Відео	8,2	8,3	64,9	65,1
		Текст	8,2	8,2	65,1	65,1
	Blowfish	Фото	8	8,3	45,5	45,5
		Відео	8,3	8,3	45,4	45,4
		Текст	8,2	8,3	45,4	45,4
HDD	AES	Фото	8,1	7,5	45,4	45,3
		Відео	8,3	8,2	45,3	45,3
		Текст	8,3	8,4	45,3	45,3
	DES	Фото	8,2	8,7	0,6	2,8
		Відео	9,5	9,5	2,7	1,6
		Текст	9,6	9,5	0,5	0,1

Продовження таблиці 4.8

Тип носія	Алгоритм	Тип даних	Пікове завантаження CPU, %		Пікове використання RAM, МБ	
			Шифрування	Дешифрування	Шифрування	Дешифрування
HDD	ChaCha20	Фото	1,5	3	50,6	66,8
		Відео	2	3	66,2	151,5
		Текст	2	2,7	65,2	81,3
	Serpent	Фото	8,3	6,6	50,4	66,7
		Відео	8,4	7,2	66,2	151,4
		Текст	8,2	7,2	161,1	80,9
	3Des	Фото	7,7	8,2	80,8	80,5
		Відео	8,3	8,3	80,6	80,5
		Текст	7,5	8,3	80,5	80,5
	Blowfish	Фото	8,1	8,3	45	45
		Відео	8,4	8,3	45	45,1
		Текст	8,3	8,3	45,1	44,9

Порівняння таблиць 4.3 – 4.4 та 4.7 – 4.8 показує, що на більш потужному процесорі абсолютні значення часу шифрування та розшифрування зменшуються для всіх алгоритмів, однак їхнє відносне «ранжування» фактично не змінюється. AES залишається найшвидшим, ChaCha20 і Blowfish демонструють близькі до нього результати або займають проміжну позицію, тоді як DES, 3DES та Serpent стабільно відстають.

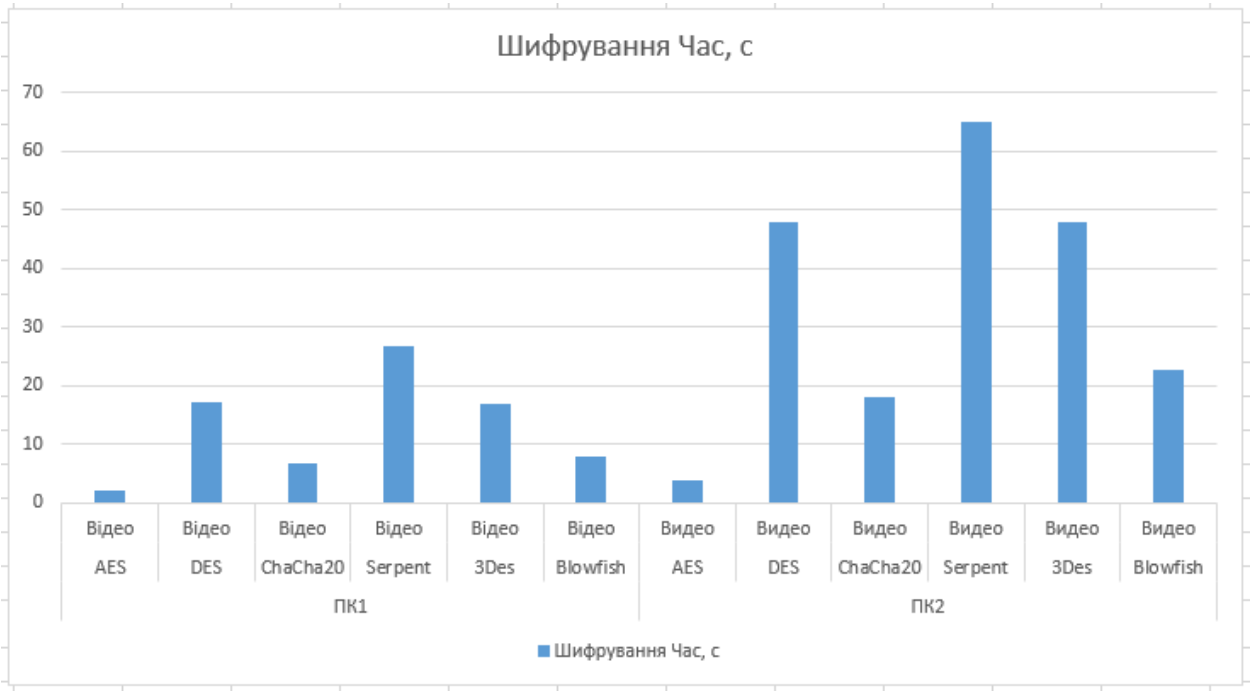

Рисунок 4.7 – Порівняння часу шифрування набору відеофайлів на першому та другому ПК

Рисунок 4.7 підтверджує, що при переході на потужніший процесор усі алгоритми прискорюються, але відносні співвідношення між ними залишаються незмінними. Це означає, що виявлені закономірності є стійкими до зміни апаратної платформи, а зроблені висновки можуть бути перенесені на інші системи з подібною архітектурою.

Експериментальне дослідження асиметричних алгоритмів. Завершальний блок експериментів був присвячений асиметричним алгоритмам RSA, ElGamal та ECC. Оскільки такі алгоритми не призначені для шифрування великих файлів, дослідження проводилися на фіксованому текстовому блоці довжиною 100 байт, який послідовно шифрувався та розшифровувався багаторазово. Для кожного алгоритму було виконано 1000 повторів операцій шифрування та розшифрування, що дозволило усереднити випадкові коливання та отримати стабільні оцінки середнього часу.

Для RSA використовувався модуль довжиною 4096 біт, для ElGamal – параметри групи з модулем 1024 біти, для ECC – стандартна крива з довжиною ключа 256 біт. У ході експериментів вимірювалися час генерації ключів,

загальний та середній час шифрування й розшифрування, коефіцієнт розширення (відношення довжини шифротексту до довжини вхідного повідомлення), а також завантаження процесора та додаткове використання оперативної пам’яті.

Таблиця 4.9 – Порівняння асиметричних алгоритмів за загальним і середнім часом Ш/Д

Алгоритм	Довжина ключа / крива	Довжина шифротексту, байт	Час створення ключа, мс	Загальний час, мс		Середній час, мс	
				Шифрування	Дешифрування	Шифрування	Дешифрування
RSA	4096 біт	512	6132,33	580,35	31163,22	0,5803	31,1632
ElGamal	1024 біт	256	21794,77	13637	6986,63	13,637	6,9866
ECC	256 біт	193	57,43	5571,39	3991,46	5,5714	3,9915

Таблиця 4.10 – Порівняння асиметричних алгоритмів за завантаженням CPU і використанням додатковою пам’ятью RUM Ш/Д

Алгоритм	Довжина ключа / крива	Довжина шифротексту, байт	Час створення ключа, мс	Завантаження CPU, %			Додаткова пам’ять, МБ		
				Ключ	Шифрування	Дешифрування	Ключ	Шифрування	Дешифрування
RSA	4096 біт	512	6132,33	8,3	8,3	8,3	0,82	0,64	0,37
ElGamal	1024 біт	256	21794,77	8,3	8,3	8,2	2,89	0,05	0,04
ECC	256 біт	193	57,43	6,8	8,3	8,3	5,81	2,17	0,06

Дані таблиць 4.9 – 4.10 показують, що ECC забезпечує найменший час генерації ключів та найменший середній час розшифрування, тоді як RSA

потребує значно більше часу на генерацію ключової пари й має більший коефіцієнт розширення (довжина шифротексту в кілька разів перевищує довжину вихідного повідомлення). ElGamal демонструє ще більші витрати на генерацію ключів та формування шифротексту, що обмежує його практичне застосування в системах із жорсткими вимогами до продуктивності та обсягу передаваних даних.

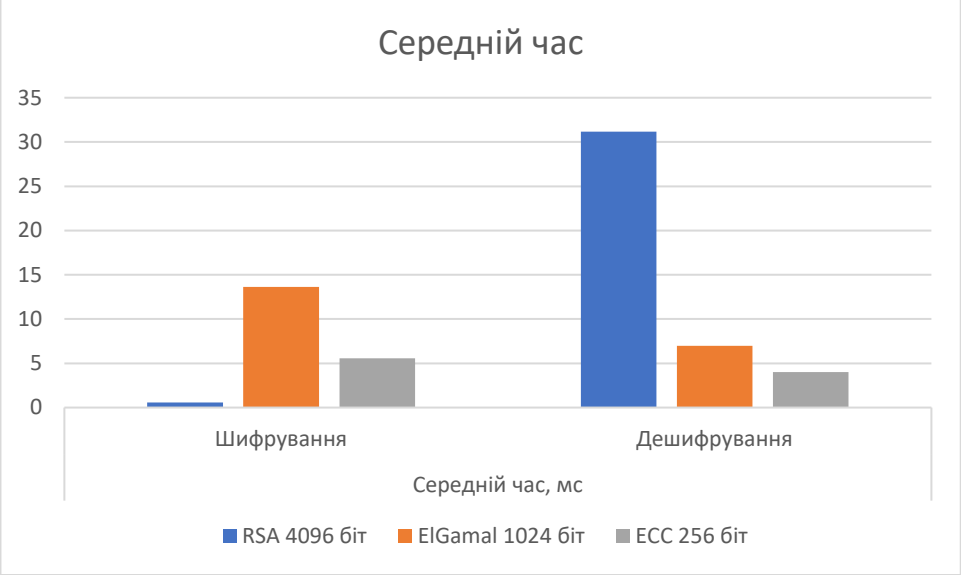

Рисунок 4.8 – Середній час шифрування та розшифрування текстового блоку

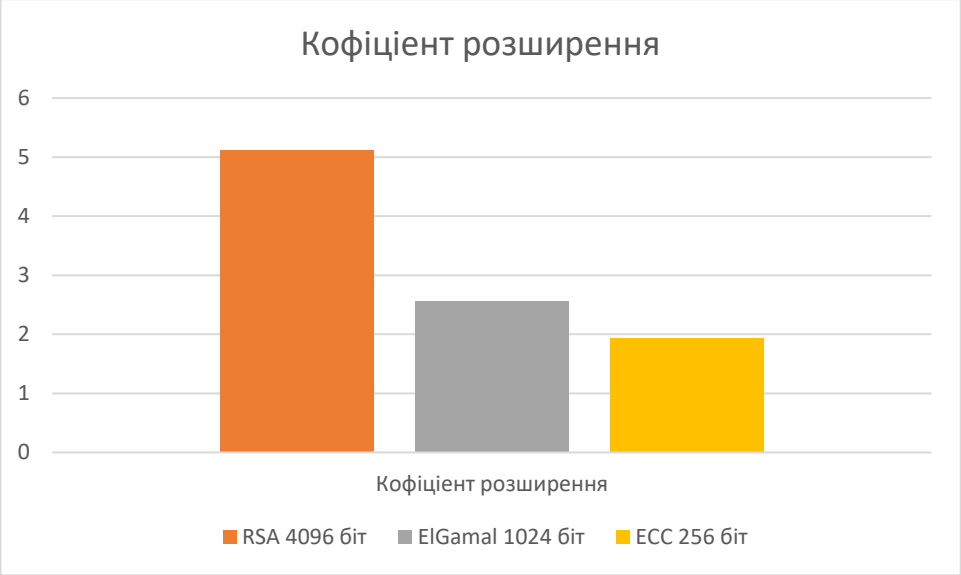

Рисунок 4.9 – Коефіцієнт розширення

За рисунках 4.8 та 4.9 видно, що ECC найкраще підходить для сценаріїв, де важлива висока продуктивність при обмежених ресурсах, тоді як RSA зберігає актуальність у традиційних інфраструктурах із менш жорсткими обмеженнями, а ElGamal виявляється менш вигідним як за часом, так і за коефіцієнтом розширення. Саме ці експериментальні результати лягають в основу подальшого порівняльного аналізу симетричних та асиметричних підходів.

4.4 Порівняльний аналіз отриманих результатів

Порівняння насамперед доцільно виконувати всередині однорідних груп алгоритмів, а вже потім переходити до міжгрупового аналізу. У межах даного дослідження такими групами є симетричні блочні шифри (AES, DES, 3DES, Serpent, Blowfish, ChaCha20), багатопотокова модифікація AES (AES_6Core) та асиметричні алгоритми (RSA, ElGamal, ECC). На основі числових даних, узагальнених у таблицях розділу 3, можна зробити низку узагальнених висновків щодо продуктивності, ресурсоспоживання й доцільності застосування кожного з підходів.

У групі симетричних алгоритмів найвиразнішим лідером за швидкістю шифрування та розшифрування є AES. Для файлів фіксованого розміру 100 МБ на носіях типу M.2 та SSD середня швидкість шифрування перевищує 300 МБ/с, тоді як для DES і 3DES вона зазвичай не виходить за межі 25–30 МБ/с, а для Serpent становить близько 20 МБ/с. Алгоритм ChaCha20 демонструє проміжні результати: його швидкість суттєво вища, ніж у DES/3DES/Serpent, але поступається AES на швидких носіях. Blowfish за багатьма показниками близький до ChaCha20 і також помітно відстає від AES при роботі з великими послідовними файлами. Таким чином, у «чистому» порівнянні за продуктивністю AES можна вважати базовим еталоном, до якого доцільно відносити швидкодію інших симетричних алгоритмів.

Якщо розглядати навантаження на центральний процесор, картина дещо змінюється. У більшості експериментів пікове завантаження CPU для AES, ChaCha20, Serpent, 3DES та Blowfish знаходиться в діапазоні 15–20 %, причому

для різних носіїв та типів даних ці значення відрізняються незначно. Це свідчить про те, що для симетричних шифрів основним обмежувальним фактором стає не стільки процесор, скільки підсистема введення-виведення. Інакше кажучи, навіть при відносно невеликому завантаженні CPU швидкість зчитування й запису на диск часто визначає загальний час шифрування, особливо на HDD. У цьому контексті ChaCha20 та Blowfish можна розглядати як компромісні варіанти: вони дещо менше навантажують процесор у порівнянні з AES, але й програють йому в абсолютній швидкості.

Коефіцієнт розширення для симетричних алгоритмів у більшості випадків близький до одиниці: розмір зашифрованого файла практично збігається з розміром вихідних даних. Це пояснюється тим, що шифрування здійснюється на рівні байтів чи блоків без додаткових структурних накладних витрат, окрім невеликого паддингу в останньому блоці. Тому за критерієм економії дискового простору в межах групи симетричних алгоритмів суттєвих відмінностей не виявлено, і вибір методу тут визначається передусім часом виконання та навантаженням на ресурси.

Окремого аналізу потребує багатопотокова реалізація AES (AES_6Core), результати для якої наведено в узагальненій таблиці з 84 та 3154 файлами. Якщо порівнювати AES та AES_6Core при невеликій кількості файлів (84 об’єкти), то на високошвидкісному накопичувачі M.2 однопотокова реалізація демонструє кращий результат: час шифрування 1,29 с проти 1,58 с для багатопотокового варіанту. Тобто при відносно невеликому обсязі даних накладні витрати на створення та синхронізацію потоків переважають потенційний виграш від паралелізму. Схожа ситуація спостерігається на HDD, де для 84 файлів час шифрування зростає з 1,39 с (AES) до 6,31 с (AES_6Core), тобто більш ніж у чотири рази. Лише на SSD багатопотокова реалізація дає очевидну перевагу: час шифрування зменшується з 4,36 с до 1,35 с, що відповідає скороченню приблизно на дві третини.

При переході до великого набору з 3154 файлів картина стає кардинально іншою на швидких носіях. На M.2 час шифрування скорочується з 27,69 с для

однопотокового AES до 7,11 с для AES_6Core; на SSD – з 27,97 с до 6,68 с. Середня швидкість обробки даних при цьому зростає приблизно у 3,5–4 рази. Пікове завантаження процесора підвищується з 3–4 % до близько 12 %, що свідчить про більш повне використання багатоядерної архітектури. Отже, для великих обсягів даних на швидких накопичувачах багатопотокова реалізація є доцільною і забезпечує суттєве прискорення обробки.

Натомість на HDD для 3154 файлів спостерігається зворотна ситуація: час шифрування однопотоковим AES становить 9,03 с, тоді як багатопотоковий варіант потребує 80,17 с. Середня швидкість знижується з 55,64 до 6,26 МБ/с, а завантаження CPU залишається на низькому рівні (приблизно 1–2 %). Це означає, що головним «вузьким місцем» тут є саме жорсткий диск, який не здатен ефективно обробляти численні паралельні запити від кількох потоків. У таких умовах багатопотоковість виявляється не просто неефективною, а й шкідливою, оскільки призводить до значного зростання часу виконання. З практичної точки зору це означає, що використання AES_6Core виправдане лише за наявності достатньо швидкої підсистеми зберігання.

Перейшовши до асиметричних алгоритмів, слід зазначити, що їхня поведінка суттєво відрізняється від симетричних шифрів. Результати, подані в таблиці з RSA, ElGamal та ECC, показують, що час генерації ключів для RSA (4096 біт) та ElGamal (1024 біти) на порядок перевищує аналогічний показник для ECC із 256-бітним ключем. Наприклад, якщо час створення ключа для RSA становить кілька секунд, то для ECC він вимірюється десятками мілісекунд. Це особливо важливо для сценаріїв, де ключі створюються динамічно, наприклад у протоколах короткочасної аутентифікації або в IoT-пристроях з обмеженими ресурсами.

Щодо операцій шифрування та розшифрування, то середній час однієї операції для ECC істотно менший, ніж для RSA та ElGamal. При однаковій кількості повторів (1000) сумарний час розшифрування для ECC виявляється найнижчим, а для RSA – найвищим. Це означає, що в задачах, де необхідно виконувати велику кількість криптографічних операцій на одиницю часу, ECC

забезпечує кращий баланс між швидкодією та рівнем безпеки. Додатковою перевагою ECC є менший коефіцієнт розширення: довжина шифротексту зростає відносно вихідного повідомлення менше, ніж у випадку RSA чи ElGamal. Для RSA, навпаки, спостерігається найбільше розширення, що прямо впливає на обсяг каналів зв'язку та сховищ.

У межах порівняння ресурсоспоживання асиметричні алгоритми загалом демонструють помірне завантаження процесора (близько 8–9 % у піку) та невеликі додаткові витрати оперативної пам'яті. Водночас різниця між ними не настільки значна, як відмінності у часових показниках і коефіцієнтах розширення. Це дозволяє зробити висновок, що вибір конкретної асиметричної схеми в більшості випадків визначається саме співвідношенням «швидкість / розмір шифротексту / довжина ключа», а не поточним навантаженням на CPU чи RAM.

Якщо перейти до міжгрупового порівняння, стає очевидним, що симетричні алгоритми й асиметричні схеми виконують різні ролі в криптографічних системах. Симетричні шифри, зокрема AES та ChaCha20, на кілька порядків швидші при шифруванні великих масивів даних: операції вимірюються мілісекундами або десятками мілісекунд для мегабайтів інформації. Асиметричні алгоритми, навпаки, мають значні накладні витрати навіть для шифрування коротких повідомлень розміром у десятки байтів. Звідси випливає практичний висновок: симетричні методи є незамінними для захисту файлів, потоків даних і резервних копій, тоді як асиметричні використовуються переважно для розподілу ключів, автентифікації й цифрових підписів.

Отримані у ході дослідження закономірності мають безпосереднє практичне значення. Для серверних систем і хмарних сервісів, які працюють зі швидкими дисковими масивами та великими обсягами інформації, доцільно використовувати симетричні алгоритми з підтримкою багатопотоковості (наприклад, AES_6Core), що дозволяє суттєво скоротити час обробки запитів. Для робочих станцій та мобільних пристроїв із твердотільними накопичувачами багатопотокова обробка також може бути корисною, але її ефект залежить від

конкретної конфігурації обладнання та характеру навантаження. Натомість для систем, що все ще використовують HDD, паралельне шифрування великої кількості дрібних файлів часто є неефективним, і в таких випадках доцільніше обмежитися однопотоковою реалізацією.

Що стосується асиметричних алгоритмів, то результати експериментів підтверджують переваги ECC у сценаріях, де важливі висока швидкість, компактні ключі та обмежені обчислювальні ресурси, зокрема в мобільних і вбудованих пристроях, IoT-системах, а також у сучасних протоколах захисту транспортного рівня. RSA з великими ключами зберігає свою актуальність у традиційних інфраструктурах та системах з розвиненою підтримкою цього стандарту, але програє ECC за швидкодією та ефективністю використання ресурсів. ElGamal, попри криптографічну стійкість, демонструє менш вигідне поєднання часу виконання та коефіцієнта розширення й тому в більшості практичних випадків поступається конкурентам.

У підсумку порівняльний аналіз показує, що жоден алгоритм не є «універсально найкращим» у всіх можливих умовах. Симетричні методи оптимальні для масового шифрування даних, асиметричні – для управління ключами та підпису, а багатопотокові реалізації виявляють свою ефективність лише за наявності достатньо швидкої підсистеми зберігання. Інтелектуальна модель, розроблена в межах даної роботи, дозволяє формалізувати ці спостереження, перетворити числові результати на рейтинги й рекомендації та тим самим підтримати прийняття обґрунтованих рішень щодо вибору криптографічних алгоритмів для конкретних практичних сценаріїв.

4.5 Висновки щодо ефективності моделей

Реалізована система оцінювання ефективності криптографічних алгоритмів показала придатність як для формалізованого порівняння окремих методів, так і для аналізу впливу зовнішніх факторів на їхню продуктивність. Система побудована на основі реальних експериментальних даних, що автоматично збираються програмним комплексом. Кожний запуск алгоритму

супроводжується реєстрацією часу шифрування та розшифрування, середньої швидкості обробки, пікового завантаження процесора, використання оперативної пам’яті та коефіцієнта зміни обсягу даних. Завдяки цьому оцінювання спирається не на одиничні спостереження, а на повну базу метрик, яка відображає фактичну поведінку реалізацій на конкретній апаратно-програмній платформі.

Важливою перевагою є формалізований характер подальшої обробки цих даних. Нормалізація метрик, введення інтегральних показників, побудова рейтингів для окремих сценаріїв дозволяють перейти від «сирих» чисел до структурованих висновків. Користувач системи може не лише подивитися, скільки секунд тривав той чи інший експеримент, а й побачити порівняльну таблицю, у якій алгоритми впорядковано за швидкістю, економністю використання ресурсів або комплексним індексом ефективності. Таким чином, модель виконує роль «аналітичного шару», що інтерпретує результати й робить їх придатними для прийняття інженерних рішень.

Окремо слід підкреслити здатність моделі аналізувати вплив зовнішніх факторів. Наявність у записах параметрів типу носія (M.2, SSD, HDD), кількості файлів, загального обсягу даних та режиму роботи (однопотоковий чи багатопотоковий) дозволяє будувати зрізи й порівняння не лише «алгоритм проти алгоритму», а й «алгоритм в різних умовах». З цієї точки зору модель виступає інструментом дослідження не абстрактних криптографічних примітивів, а конкретних реалізацій у реальному середовищі експлуатації.

За результатами аналізу симетричних алгоритмів можна сформулювати кілька ключових висновків. По-перше, багатопотокова обробка (реалізація AES_6Core) є доцільною лише за наявності швидких носіїв (M.2 та SSD) і достатнього обсягу даних. У сценаріях із тисячами файлів та сотнями мегабайт інформації перехід від однопотокового AES до багатопотокової реалізації забезпечує скорочення часу обробки у 3–4 рази та суттєве зростання середньої швидкості. Натомість на повільних накопичувачах типу HDD, особливо при роботі з великою кількістю дрібних файлів, багатопотоковість не дає виграшу і

навіть призводить до деградації продуктивності через обмеження підсистеми введення-виведення.

По-друге, для великих масивів даних (файли 100 МБ, набори фото й відео загальним обсягом понад 1 ГБ) оптимальним з точки зору співвідношення «швидкість / ресурсоспоживання» є AES. Він забезпечує найвищу середню швидкість шифрування й розшифрування на всіх типах носіїв при помірному завантаженні процесора й мінімальних накладних на пам’ять. ChaCha20 та Blowfish можна розглядати як альтернативні варіанти для середовищ, орієнтованих на програмну реалізацію без апаратної підтримки AES, але за швидкодією вони помітно поступаються еталонному алгоритму. DES, 3DES та Serpent демонструють значно гірші часові характеристики й у сучасних системах доцільні хіба що для порівняльних або навчальних цілей.

По-третє, якщо система працює в умовах обмежених ресурсів (невисока продуктивність процесора, обмежений обсяг оперативної пам’яті, повільний носій), вирішальним фактором стає не стільки вибір «найшвидшого» алгоритму, скільки відповідність режиму обробки можливостям платформи. В таких сценаріях доцільно використовувати однопотокові реалізації AES або ChaCha20, уникати надмірної паралельності й орієнтуватися на мінімізацію дискових операцій, оскільки саме підсистема зберігання даних стає головним «вузьким місцем».

Щодо асиметричних алгоритмів, інтелектуальна модель також дозволила сформулювати низку важливих висновків. RSA, попри усталену репутацію та широку підтримку, демонструє найбільший час генерації ключів і суттєвий коефіцієнт розширення шифротексту. Це робить його менш придатним для сценаріїв, де ключі формуються динамічно або де обмежені ресурси каналів зв’язку. Натомість RSA залишається доцільним вибором для класичних серверних інфраструктур, де ключі створюються відносно рідко, а апаратні ресурси не є критичним обмеженням.

ECC, навпаки, виявився найбільш збалансованим варіантом за сукупністю показників. Час генерації ключів і середній час розшифрування для алгоритмів

на еліптичних кривих істотно нижчі, ніж для RSA та ElGamal, при цьому рівень криптографічної стійкості зберігається на сучасному прийнятному рівні завдяки більш компактним ключам (256 біт проти тисяч біт у класичних схемах). Менший коефіцієнт розширення додатково зменшує вимоги до каналів передачі й сховищ. Це робить ECC природним кандидатом для застосування в мобільних пристроях, IoT-системах та інших середовищах із жорсткими обмеженнями за енергоспоживанням і продуктивністю.

Алгоритм ElGamal, за результатами моделі, програє конкурентам за сукупністю характеристик. Хоча він забезпечує достатню криптостійкість, час генерації ключів та операцій розшифрування є значно вищим, ніж у ECC, а коефіцієнт розширення шифротексту – один з найбільших. У типових сценаріях використання (захист ключів, організація обміну секретами) це означає більший трафік і повільнішу роботу без очевидних переваг у стійкості. Тому ElGamal доцільно розглядати переважно в академічних дослідженнях або спеціалізованих протоколах, де його особливі властивості критично важливі.

Практичне значення розробленої моделі полягає в тому, що вона може використовуватися як готовий інструмент підтримки рішень у сфері інформаційної безпеки. Для адміністратора безпеки система може виконувати роль «тестового стенду», на якому швидко перевіряються різні конфігурації криптографічних параметрів на конкретному обладнанні підприємства. Замість того, щоб орієнтуватися лише на теоретичні оцінки або загальні рекомендації стандартів, фахівець отримує числові показники саме для своєї інфраструктури й може обґрунтовано обрати оптимальний алгоритм та режим його роботи.

Як модуль у складі криптографічного програмного забезпечення інтелектуальна модель може використовуватися для періодичної переоцінки ефективності алгоритмів у міру зміни апаратної платформи або характеру навантаження. Наприклад, при переході з HDD на SSD чи M.2, при оновленні процесорів, збільшенні обсягів даних тощо система може автоматично перезапустити серію тестів і запропонувати нові рекомендовані налаштування.

Такий підхід наближає програмний продукт до концепції «адаптивної криптосистеми», здатної підлаштовуватися під поточні умови.

Крім того, розроблена модель є корисним навчальним інструментом. Її можна використовувати як лабораторний стенд для студентів спеціальностей, пов’язаних з кібербезпекою та програмною інженерією. Студенти мають змогу не лише побачити формули та теоретичні описи алгоритмів, а й експериментально переконатися, як змінюється час шифрування при збільшенні розміру файлу, чому асиметричні методи повільніші і як впливає тип носія на швидкодію. Такий досвід формує більш глибоке розуміння практичних аспектів криптографії.

Разом із тим поточна версія моделі має низку обмежень. По-перше, усі експерименти виконувалися в середовищі однієї операційної системи, що накладає відбиток на результат (особливості планувальника потоків, реалізація файлової системи, системних таймерів тощо). Для повної універсальності модель необхідно перевірити на інших платформах. По-друге, набір алгоритмів, що досліджувалися, хоч і включає найбільш уживані сучасні схеми, все ж таки є обмеженим і не охоплює, зокрема, постквантові алгоритми та нові режими шифрування, орієнтовані на високопродуктивні мережеві сервіси. По-третє, інструментальна система реалізована як настільний застосунок і не має веб-інтерфейсу чи віддаленого доступу, що ускладнює її використання в розподілених інфраструктурах. Нарешті, модель поки що не містить механізмів автоматичного підбору параметрів: вибір ваг метрик та сценаріїв здійснюється користувачем вручну.

Перспективними напрямами розвитку є, по-перше, розширення алгоритмічної бази за рахунок включення постквантових криптографічних схем та сучасних режимів шифрування потокових і блочних алгоритмів. Це дозволить оцінити їхню практичну ефективність на реальному обладнанні й порівняти з традиційними підходами. По-друге, доцільно перенести інструментальний комплекс у формат веб-сервісу (наприклад, на базі ASP.NET), що забезпечить доступ до моделі з різних робочих станцій, спростить централізоване зберігання

результатів та інтеграцію з корпоративними системами моніторингу. По-третє, цікавою є інтеграція методів машинного навчання, які на основі накопичених історичних даних зможуть автоматично пропонувати оптимальні конфігурації алгоритмів для заданого профілю навантаження та обмежень по ресурсах.

Узагальнюючи, можна зробити висновок, що розроблена інтелектуальна модель ефективності криптографічних алгоритмів є практично корисним, гнучким та розширюваним інструментом. Вона дозволяє системно оцінювати поведінку алгоритмів у реальних умовах, формувати обґрунтовані рекомендації щодо їхнього вибору та створює ґрунт для подальших досліджень у напрямі адаптивних і постквантових криптосистем.

Висновки до розділу

У розділі виконано комплексне експериментальне дослідження симетричних, асиметричних і багатопотокових реалізацій криптографічних алгоритмів у прикладних сценаріях оброблення даних. Експерименти проводилися за єдиним регламентом із контролем середовища, фіксацією метрик (час, середня швидкість, пікове завантаження CPU, використання RAM, коефіцієнт зміни обсягу даних) та обов’язковою валідацією коректності результатів, що забезпечило порівнюваність і відтворюваність отриманих висновків.

Для симетричних алгоритмів підтверджено стійке лідерство AES за швидкістю шифрування/розшифрування на всіх носіях, особливо на M.2 та SSD, де алгоритм демонструє найкраще співвідношення «продуктивність / ресурсоспоживання». ChaCha20 і Blowfish показали проміжні результати та можуть розглядатися як практичні альтернативи в середовищах без апаратної підтримки AES, тоді як DES, 3DES і Serpent істотно поступаються за часовими характеристиками й придатні переважно для порівняльних або навчальних задач. Встановлено, що для великих обсягів даних ключовим обмежувальним фактором часто стає підсистема введення-виведення: на HDD різниця між алгоритмами

частково нівелюється через дискові затримки, тоді як на швидких накопичувачах повною мірою проявляються обчислювальні відмінності реалізацій.

Окремо показано, що ефективність багатопотокового AES (AES_6Core) залежить від типу носія та структури навантаження: на M.2/SSD при великій кількості файлів паралелізація забезпечує прискорення в кілька разів, але на HDD може спричиняти деградацію продуктивності через конкуренцію за дискові ресурси. Для асиметричних алгоритмів експерименти підтвердили переваги ECC за сукупністю показників (швидша генерація ключів, кращі часові метрики та менший коефіцієнт розширення), тоді як RSA демонструє значні накладні витрати, а ElGamal є найменш вигідним у типових умовах. Загалом результати розділу сформували практичні рекомендації щодо вибору алгоритмів і режимів їх застосування залежно від сценарію, обсягів даних та характеристик платформи.

ВИСНОВКИ

У магістерській кваліфікаційній роботі розв’язано науково-практичну задачу порівняльного оцінювання ефективності сучасних криптографічних алгоритмів у прикладних сценаріях оброблення даних шляхом розроблення інструментальної системи вимірювання та подальшого аналізу експериментальних результатів. Дослідження спирається на поєднання теоретичного огляду предметної області, формалізованої методики проведення контрольованих серій запусків та узгоджених процедур аналітичного узагальнення, що дозволяє отримувати відтворювані результати і формувати практично релевантні рекомендації.

1. Проведено огляд предметної області криптографічного захисту в сучасних інформаційних системах та обґрунтовано актуальність оцінювання ефективності криптографічних механізмів у контексті експлуатаційних вимог. Показано, що інженерний вибір криптоалгоритму в прикладних рішеннях визначається не лише вимогами до криптографічної стійкості, але й особливостями обчислювального середовища, характером навантаження, організацією даних і взаємодією із підсистемою зберігання.

2. Сформовано методику дослідження, яка задає сценарії експлуатації, правила однаковості умов, процедури валідації коректності та принципи документування результатів. Методика передбачає стандартизоване виконання операцій шифрування і розшифрування для одиничних файлів і для наборів даних, а також окрему постановку для коротких операцій, характерних для асиметричних перетворень. Визначено групи показників, які відображають часову поведінку операцій, ресурсний профіль виконання та супутні ефекти, важливі для практик зберігання й передавання даних.

3. Розроблено інструментальну систему на платформі .NET з графічним інтерфейсом, яка забезпечує кероване виконання експериментів, уніфікований збір показників, валідацію коректності перетворень, журналювання контексту запусків і формування вихідних артефактів, придатних для подальшого аналізу.

Система реалізує єдиний регламент запусків і зменшує вплив ручних дій на результати, що підвищує відтворюваність експериментів та прозорість отриманих висновків.

4. За результатами експериментальних серій із симетричними алгоритмами встановлено, що сучасні алгоритми демонструють суттєво кращі експлуатаційні характеристики у типових сценаріях оброблення даних порівняно з застарілими рішеннями. У досліджених умовах AES загалом проявляє себе як найбільш збалансований варіант для практичного застосування в задачах масового шифрування даних, тоді як ChaCha20 є важливим представником потокових шифрів і доцільною альтернативою в програмних реалізаціях, зокрема у середовищах, де переваги спеціалізованих оптимізацій для блочних шифрів не є визначальними. Алгоритми DES і 3DES у контексті сучасних вимог до продуктивності та безпеки мають обмежену практичну доцільність і можуть розглядатися переважно як навчальні або історично значущі приклади. Serpent демонструє коректну криптографічну поведінку, однак у прикладних сценаріях зазвичай поступається більш продуктивним альтернативам, що знижує його привабливість для масового шифрування даних за типових умов експлуатації.

5. Дослідження впливу паралельного виконання на прикладі реалізацій AES показало, що ефективність багатопотокової обробки є контекстно залежною. За умов, коли обмежувальним фактором не виступає підсистема введення-виведення, паралелізація здатна підвищувати загальну ефективність оброблення наборів даних. Водночас у сценаріях, де домінують затримки доступу до носія або значну роль відіграють накладні витрати файлових операцій, потенційна перевага паралельності може нівелюватися або трансформуватися у погіршення результатів. Це підкреслює необхідність підбору конфігурації виконання з урахуванням конкретного профілю навантаження та характеристик середовища зберігання.

6. Експериментальне дослідження асиметричних алгоритмів підтвердило їх специфічну прикладну роль і відмінну природу обчислювальних витрат у порівнянні із симетричними перетвореннями. ECC у рамках виконаних серій

демонструє більш сприятливу поведінку в типових сценаріях, пов’язаних із роботою з ключовим матеріалом та короткими перетвореннями, що робить цей підхід привабливим для сучасних систем керування ключами та протоколів встановлення захищених каналів. RSA зберігає високу практичну поширеність і сумісність із широким спектром інфраструктур, однак характеризується вищими витратами виконання в задачах, чутливих до обчислювальної складності. ElGamal у розглянутих умовах виявляє менш вигідний профіль застосування, що обмежує його доцільність у сценаріях, де важливими є компактність представлення та оперативність виконання.

7. Узагальнення результатів у межах єдиного аналітичного підходу підтвердило практичну доцільність гібридних схем, у яких симетричні алгоритми використовуються для оброблення основного масиву даних, а асиметричні – для задач розподілу ключів, автентифікації та інших допоміжних криптографічних процедур. Така організація дозволяє поєднувати ефективність симетричних перетворень із функціональними можливостями асиметричної криптографії, зберігаючи придатність рішення для реальних інформаційних систем.

8. Практичне значення отриманих результатів полягає у тому, що розроблена інструментальна система та методика можуть використовуватися як прикладний засіб для порівняльного оцінювання криптографічних механізмів у контрольованих умовах, як навчальний стенд для демонстрації впливу алгоритмічних і середовищних факторів на ефективність, а також як базис для подальшого розширення досліджень і підключення додаткових алгоритмів, режимів та сценаріїв.

Поточна реалізація має обмеження, зумовлені фокусом на визначених сценаріях, конкретній платформі виконання та наявному наборі реалізацій алгоритмів. Перспективними напрямами розвитку є розширення переліку досліджуваних примітивів і режимів, поглиблення сценарного аналізу, інтеграція інструментальної системи з автоматизованими процесами тестування, а також розширення способів взаємодії з системою через командний інтерфейс.

ПЕРЕЛІК ПОСИЛАНЬ

1. Aumasson J.-P. Serious Cryptography: A Practical Introduction to Modern Encryption. San Francisco: No Starch Press, 2018. 312 с.

2. Katz J., Lindell Y. Introduction to Modern Cryptography, 3rd ed. Boca Raton: CRC Press, 2020. 648 с.

3. Paar C., Pelzl J., Güneysu T. Understanding Cryptography: From Established Symmetric and Asymmetric Ciphers to Post-Quantum Algorithms, 2nd ed. Berlin: Springer, 2024. 543 с.

4. Stallings W. Cryptography and Network Security: Principles and Practice, 8th ed. Boston: Pearson, 2020. 832 с.

5. Menezes A. J., van Oorschot P. C., Vanstone S. A. Handbook of Applied Cryptography. Boca Raton: CRC Press, 1996. 816 с.

6. NIST. Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography (SP 800-56A Rev. 3) [Електронний ресурс]. 2018. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final (дата звернення: 08.09.2025).

7. NIST. Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography (SP 800-56B Rev. 2) [Електронний ресурс]. 2019. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-56b/rev-2/final (дата звернення: 14.09.2025).

8. NIST. Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters (SP 800-186) [Електронний ресурс]. 2019. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-186/final (дата звернення: 21.09.2025).

9. NIST. Transitioning the Use of Cryptographic Algorithms and Key Lengths (SP 800-131A Rev. 2) [Електронний ресурс]. 2019. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-131a/rev-2/final (дата звернення: 02.10.2025).

10. 1NIST. Recommendation for Key Management: Part 1 – General (SP 800-57 Part 1 Rev. 5) [Електронний ресурс]. 2020. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final (дата звернення: 05.10.2025).

11. NIST. Guidelines for the Selection, Configuration, and Use of TLS Implementations (SP 800-52 Rev. 2) [Електронний ресурс]. 2019. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final (дата звернення: 06.10.2025).

12. NIST. Recommendation for the Entropy Sources Used for Random Bit Generation (SP 800-90B) [Електронний ресурс]. 2018. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-90b/final (дата звернення: 15.10.2025).

13. NIST. Recommendation for Cryptographic Key Generation (SP 800-133 Rev. 2) [Електронний ресурс]. 2020. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-133/rev-2/final (дата звернення: 16.10.2025).

14. NIST. Recommendation for Key Derivation Using Pseudorandom Functions (SP 800-108 Rev. 1) [Електронний ресурс]. 2022. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-108/rev-1/final (дата звернення: 20.10.2025).

15. NIST. Security Requirements for Cryptographic Modules (FIPS 140-3) [Електронний ресурс]. 2019. Режим доступу: https://csrc.nist.gov/publications/detail/fips/140/3/final (дата звернення: 29.10.2025).

16. NIST. Digital Signature Standard (FIPS 186-5) [Електронний ресурс]. 2023. Режим доступу: https://csrc.nist.gov/publications/detail/fips/186/5/final (дата звернення: 03.11.2025).

17. Nir Y., Langley A. ChaCha20 and Poly1305 for IETF Protocols [Електронний ресурс] // RFC 8439. IETF, 2018. Режим доступу: https://www.rfc-editor.org/rfc/rfc8439.html (дата звернення: 08.11.2025).

18. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3 [Електронний ресурс] // RFC 8446. IETF, 2018. Режим доступу: https://www.rfc-editor.org/rfc/rfc8446.html (дата звернення: 10.11.2025).

19. Thomson M. et al. Using TLS to Secure QUIC [Електронний ресурс] // RFC 9001. IETF, 2021. Режим доступу: https://www.rfc-editor.org/rfc/rfc9001.html (дата звернення: 17.11.2025).

20. Sheffer Y., Saint-Andre P., Fossati T. Recommendations for Secure Use of TLS and DTLS [Електронний ресурс] // RFC 9325. IETF, 2022. Режим доступу: https://www.rfc-editor.org/rfc/rfc9325.html (дата звернення: 19.01.2025).

21. Rescorla E., Tschofenig H. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3 [Електронний ресурс] // RFC 9147. IETF, 2022. Режим доступу: https://www.rfc-editor.org/rfc/rfc9147.html (дата звернення: 27.11.2025).

22. Barnes R. et al. Hybrid Public Key Encryption (HPKE) [Електронний ресурс] // RFC 9180. IETF, 2022. Режим доступу: https://www.rfc-editor.org/rfc/rfc9180.html (дата звернення: 28.11.2025).

23. NSA. Commercial National Security Algorithm Suite 2.0 (CNSA 2.0) [Електронний ресурс]. 2022. Режим доступу: https://media.defense.gov/ (дата звернення: 02.12.2025).

24. ENISA. Algorithms, Key Size and Parameters Report [Електронний ресурс]. 2023. Режим доступу: https://www.enisa.europa.eu/ (дата звернення: 07.12.2025).

25. BSI. Cryptographic Mechanisms: Recommendations and Key Lengths (TR-02102-1) [Електронний ресурс]. 2024. Режим доступу: https://www.bsi.bund.de/ (дата звернення: 11.12.2025).

26. Hankerson D., Menezes A., Vanstone S. Guide to Elliptic Curve Cryptography. New York: Springer, 2004. 332 с.

27. Ferguson N., Schneier B., Kohno T. Cryptography Engineering: Design Principles and Practical Applications. Indianapolis: Wiley, 2010. 384 с.

28. Wong D. Real-World Cryptography. Shelter Island, NY: Manning, 2021. 400 с.

29. ISO/IEC. Information security, cybersecurity and privacy protection – Information security management systems (ISO/IEC 27001:2022). Geneva: ISO, 2022. 19 с.

30. ISO/IEC. Information security, cybersecurity and privacy protection – Verification of cryptographic protocol specifications – Part 1: General (ISO/IEC 29128-1:2023). Geneva: ISO, 2023. 15 с.

31. Dworkin M. (NIST). Guidance on Authenticated Encryption and Use Cases in Systems [Електронний ресурс]. 2020. Режим доступу: https://csrc.nist.gov/ (дата звернення: 17.12.2025).

32. IETF CFRG. Recommendations for AEAD and modern symmetric primitives in protocols [Електронний ресурс]. 2020. Режим доступу: https://datatracker.ietf.org/rg/cfrg/ (дата звернення: 10.01.2026).

33. Microsoft. Cryptography API: Next Generation (CNG) – Documentation [Електронний ресурс]. 2021. Режим доступу: https://learn.microsoft.com/ (дата звернення: 21.12.2025).

34. OpenSSL Project. OpenSSL 3.x Documentation (EVP, RSA, EC, AEAD) [Електронний ресурс]. 2021. Режим доступу: https://www.openssl.org/docs/ (дата звернення: 24.12.2025).

35. Bouncy Castle. Bouncy Castle Cryptography APIs (Java/C#) – Documentation [Електронний ресурс]. 2020. Режим доступу: https://www.bouncycastle.org/ (дата звернення: 29.12.2025).

ДОДАТОК А
Технічне завдання

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ

CRYPTOBENCH

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01529 – 01 – ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олена КУРОП’ЯТНИК
Виконавець
_____Олексій НОР
Нормоконтролер
_____Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО

44165850.01529 – 01

CRYPTOBENCH

Технічне завдання

Листів 13

2025

ЗМІСТ

ВВЕДЕННЯ 4
1 ПІДСТАВИ ДЛЯ РОЗРОБКИ 5
2 ПРИЗНАЧЕННЯ РОЗРОБКИ 6
3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ 7
3.1. Вимоги до функціональних характеристик 7
3.2. Вимоги до надійності 7
3.3. Вимоги до експлуатації 8
3.4. Вимоги до технічних засобів 8
3.5. Вимоги до інформаційної та програмної сумісності 9
3.6. Вимоги до маркування і упаковки 9
3.7. Вимоги до транспортування та зберігання 9
4. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ 11
5. ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ 12
БІБЛІОГРАФІЧНИЙ СПИСОК 13

ВВЕДЕННЯ

Назва програми – CryptoBench.

Програмне забезпечення призначене для порівняльного дослідження криптографічних алгоритмів у прикладних сценаріях оброблення даних. Основною метою є виконання серійних запусків і отримання кількісних показників ефективності для симетричних, асиметричних та гібридних підходів шифрування. У межах роботи оцінюються алгоритми AES, DES, 3DES, Serpent, Blowfish, ChaCha20, а також асиметричні схеми RSA, ElGamal та ECC (у задачах, характерних для роботи з ключовим матеріалом і короткими повідомленнями).

Застосунок забезпечує виконання операцій шифрування/розшифрування файлів і наборів файлів, а також тестування коротких криптографічних операцій для асиметричних алгоритмів. Під час запусків фіксуються ключові метрики: час виконання, середня швидкість оброблення даних, пікове завантаження CPU, пікове використання RAM і зміна обсягу даних після перетворення. Отримані результати документуються (у вигляді виводу/логів) та надалі узагальнюються у зведених таблицях і графіках.

Причиною розробки (та використання) програмного продукту є потреба в об’єктивному порівнянні реалізацій за однакових умов виконання й у виявленні практичних відмінностей, які не завжди очевидні з теоретичних характеристик. Окрема увага приділяється впливу типу носія інформації (M.2/SSD/HDD) і режиму оброблення (одиночні файли, пакетні набори, сценарії з великою кількістю об’єктів) на підсумкову продуктивність.

Область застосування програмного забезпечення охоплює задачі захисту даних у файлових сховищах і резервному копіюванні, у прикладних інформаційних системах, а також у середовищах із різними обмеженнями ресурсів (робочі станції, сервери, вбудовані пристрої), де вибір алгоритму та режиму шифрування має безпосередній вплив на швидкодію й експлуатаційні витрати.

1. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 02.10.2025 №1401ст ректора Українського державного університету науки і технологій «Про призначення наукових керівників та затвердження тем магістерських робіт» за спеціальністю 121 «Інженерія програмного забезпечення» факультету «Комп’ютерних технологій і систем» по кафедрі «Комп’ютерні інформаційні технології».

Тема дипломної роботи – «Дослідження ефективності алгоритмів шифрування та дешифрування даних». Керівник – Куроп’ятник О.С.

2. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення. Розроблений програмний продукт призначений для експериментального дослідження та порівняльного аналізу ефективності симетричних, асиметричних і гібридних алгоритмів шифрування в прикладних умовах оброблення даних. У межах роботи програмний комплекс забезпечує виконання стандартизованих серій випробувань для алгоритмів AES, DES, 3DES, Serpent, Blowfish, ChaCha20, а також асиметричних схем RSA, ElGamal, ECC і гібридних режимів (використання асиметричних алгоритмів для захисту ключового матеріалу та симетричних – для масового шифрування даних). Основною метою є отримання кількісних показників (час шифрування/дешифрування, середня швидкість, завантаження CPU, використання RAM, коефіцієнт зміни обсягу даних) та формування на їх основі обґрунтованих висновків щодо доцільності застосування алгоритмів у різних сценаріях.

Експлуатаційне призначення. Програмний продукт орієнтований на використання студентами, дослідниками та фахівцями з інформаційної безпеки як тестовий стенд для оцінювання продуктивності криптографічних перетворень на конкретній апаратно-програмній платформі з урахуванням типу носія: M.2 / SSD / HDD, розміру та структури даних, режиму виконання. Результати експериментів оформлюються у вигляді підсумкових таблиць і графічних матеріалів, що дозволяє застосовувати їх для підготовки звітів, порівняння реалізацій та вибору алгоритму під задані практичні обмеження (швидкодія, ресурсні витрати, характер даних, частота асиметричних операцій).

3. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1. Вимоги до функціональних характеристик

Програмний продукт має надавати користувачеві інструменти для дослідження ефективності алгоритмів шифрування та дешифрування даних у прикладних сценаріях оброблення даних. Додаток повинен забезпечувати запуск і порівняння таких криптографічних алгоритмів:

– симетричні: AES, DES, 3DES, Blowfish, Serpent, ChaCha20;

– асиметричні: RSA, ElGamal;

– гібридні схеми: ECC + AES (на основі ECDH для узгодження ключа та AES для шифрування даних).

Програмний продукт має підтримувати проведення експериментів у типових режимах: шифрування/дешифрування одиничного файлу та наборів файлів, а також короткі операції, характерні для асиметричних перетворень (генерація ключів, шифрування/розшифрування малих блоків, операції з ключовим матеріалом). Для кожного запуску система повинна фіксувати результати вимірювань основних показників продуктивності, зокрема часу виконання, середньої швидкості обробки, завантаження CPU, використання RAM та обсягу вхідних/вихідних даних. За результатами запуску формуються підсумкові зведення у текстовому форматі (журнал/звіт) для подальшого аналізу та порівняння.

3.2. Вимоги до надійності

Програмний продукт повинен забезпечувати стабільне виконання багаторазових серій експериментів із шифрування та дешифрування даних, у тому числі під час оброблення великих файлів і наборів файлів. Робота системи має бути захищена від неконтрольованого зависання та втрати результатів у разі збоїв підсистеми введення-виведення або нестачі доступних ресурсів під час виконання вимірювань.

Для забезпечення достовірності результатів програма повинна виконувати перевірку коректності перетворень (зокрема, підтвердження можливості відновлення даних після дешифрування). У процесі експлуатації не допускаються критичні

помилки, аварійні зупинки або неконтрольоване завершення роботи програми; у разі помилки система повинна коректно завершувати поточну операцію, зберігати вже отримані дані та дозволяти повторний запуск експерименту.

3.3. Вимоги до експлуатації

Програмний продукт має експлуатуватися в стандартних офісних умовах у приміщенні, без впливу підвищеної вологості, пилу та різких температурних коливань. Спеціальне технічне обслуговування програмного продукту не передбачається.

Працювати з програмним забезпеченням може користувач із базовими навичками роботи з персональним комп’ютером. Основною цільовою аудиторією є студенти, дослідники та розробники, які виконують експериментальні вимірювання ефективності алгоритмів шифрування та дешифрування.

3.4. Вимоги до технічних засобів

Для використання програмного продукту, призначеного для дослідження ефективності алгоритмів шифрування та дешифрування даних, необхідний наступний склад технічних засобів:

– персональний комп’ютер/ноутбук з процесором архітектури x86-64 не нижче 4 ядер, тактовою частотою не нижче 2,5 ГГц (рекомендовано 6 – 8 ядер для паралельних запусків і порівняльних серій);

– оперативна пам’ять обсягом не менше 8 ГБ (рекомендовано 16 ГБ для стабільної роботи під час експериментів);

– накопичувач для встановлення ПЗ та збереження результатів: SSD не менше 256 ГБ (рекомендовано 512 ГБ);

– для відтворення порівняння впливу підсистеми зберігання на швидкодію (за потреби дослідження) – наявність накопичувачів різних типів: M.2, SSD, HDD;

– монітор для відображення інтерфейсу та результатів;

– клавіатура;

– маніпулятор типу «миша».

3.5. Вимоги до інформаційної та програмної сумісності

Програмний продукт для дослідження ефективності алгоритмів шифрування та дешифрування даних має коректно функціонувати в середовищі операційних систем Windows 10 або Windows 11. Для роботи додатка необхідна наявність встановленого середовища виконання .NET 10.

3.6. Вимоги до маркування і упаковки

Упаковка програмного продукту разом із супровідною документацією має забезпечувати збереження носія та матеріалів під час зберігання і транспортування, захищаючи їх від можливих механічних пошкоджень і впливу несприятливих умов.

На упаковці слід зазначити:

– назву програмного продукту: «CruptoBench»;

– номер версії: v1.0;

– призначення: дослідницька платформа для порівняльного аналізу ефективності алгоритмів шифрування/дешифрування;

– відомості для користувача: підтримувана ОС (Windows 10/11), вимоги до середовища (.NET), коротка інструкція запуску, попередження про необхідність збереження ключів;

– рік випуску: 2025.

На зворотному боці упаковки необхідно вказати:

– розробника: Нор О.С. (Олексій Нор);

– організацію: Український державний університет науки і технологій (УДУНТ), м. Дніпро;

– юридичну адресу розробника/організації: Україна, 49010, Дніпропетровська область, місто Дніпро, вулиця Лазаряна, 2;

3.7. Вимоги до транспортування та зберігання

Транспортування програмного продукту повинно здійснюватися в заводській упаковці та забезпечувати збереження його цілісності й працездатності.

4. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

– Специфікація;

– Керівництво користувача. Керівництво дослідника криптографічних алгоритмів;

– Текст програми.

Програмна документація повинна відповідати вимогам ДСТУ [1].

5. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ, формулювання мети та завдань дослідження	01.09.2025 – 06.09.2025	
2	Огляд літератури та аналіз симетричних, асиметричних та гібридних алгоритмів шифрування	07.09.2025 – 20.09.2025	
3	Аналіз методик експериментального оцінювання криптографічних алгоритмів та підходів до стандартизації сценаріїв запуску	21.09.2025 – 08.10.2025	
4	Формування вимог до програмного продукту та постановка задачі	09.10.2025 – 12.10.2025	
5	Розробка та узгодження технічного завдання	13.10.2025 – 21.10.2025	30 %
6	Проектування архітектури системи оцінювання криптографічних алгоритмів	22.10.2025 – 29.10.2025	
7	Розробка математичної моделі оцінювання криптографічних алгоритмів	30.10.2025 – 09.11.2025	
8	Реалізація симетричних, асиметричних та гібридних алгоритмів шифрування	10.11.2025 – 28.11.2025	
9	Програмна реалізація модуля вибору криптографічних алгоритмів	29.11.2025 – 08.12.2025	
10	Проведення експериментальних серій шифрування/дешифрування та фіксація результатів вимірювань	09.12.2025 – 20.12.2025	
11	Тестування та відлагодження програмного продукту	21.12.2025 – 30.12.2025	60 %
12	Аналіз результатів досліджень та формування висновків	31.12.2025 – 04.01.2026	
13	Оформлення пояснювальної записки та програмної документації	05.01.2026 – 13.01.2026	100 %
14	Розробка демонстраційних матеріалів та підготовка до захисту	14.01.2026 – 20.01.2026	
15	Подання та захист кваліфікаційної роботи	21.01.2026	

6. ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль виконання здійснює керівник розробки. Прийом здійснюється уповноваженою комісією.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009.

ДОДАТОК Б
Керівництво користувача. Керівництво дослідника криптографічних алгоритмів

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ

CRYPTOBENCH

Керівництво користувача. Керівництво дослідника криптографічних алгоритмів

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01529-01 І3 01-ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олена КУРОП’ЯТНИК
Виконавець
_____Олексій НОР
Нормоконтролер
_____Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО

44165850.01529-01 І3 01

CRYPTOBENCH

Керівництво користувача. Керівництво дослідника криптографічних алгоритмів

Листів 15

2025

АННОТАЦІЯ

Документ 44165850.1527-01 І3 01 «Програмне забезпечення для проведення досліджень ефективності алгоритмів шифрування та дешифрування даних. Керівництво користувача. Керівництво дослідника криптографічних алгоритмів» входить до складу програмної документації на програмний комплекс, призначений для експериментального порівняння криптографічних алгоритмів у прикладних сценаріях оброблення даних.

У даному документі представлено «Керівництво користувача. Керівництво дослідника криптографічних алгоритмів». Програми написані на мові C#. Об'єм пам'яті, що займає програма, складає 8 Мб. Конфігурація комп'ютера стандартна. Операційна система Windows 10 або вище.

ЗМІСТ

ВВЕДЕННЯ 5
1. ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ 6
2. ПІДГОТОВКА ДО РОБОТИ 7
3. ОПИС ОПЕРАЦІЙ 8
3.1. Вибір алгоритму 8
3.2. Шифрування та розшифровка файлу/папки симетричним алгоритмом 9
3.3. Шифрування та розшифрування тексту асиметричним алгоритмом 10
3.4. Перегляд звіту 11
4. АВАРІЙНІ СИТУАЦІЇ 13
5. РЕКОМЕНДАЦІЇ ЩОДО ЗАСВОЄННЯ 14
БІБЛІОГРАФІЧНИЙ СПИСОК 15

ВВЕДЕННЯ

Назва програми – CryptoBench.

Програмне забезпечення призначене для порівняльного дослідження криптографічних алгоритмів у прикладних сценаріях оброблення даних. Основною метою є виконання серійних запусків і отримання кількісних показників ефективності для симетричних, асиметричних та гібридних підходів шифрування. У межах роботи оцінюються алгоритми AES, DES, 3DES, Serpent, Blowfish, ChaCha20, а також асиметричні схеми RSA, ElGamal та ECC

Застосунок забезпечує виконання операцій шифрування/розшифрування файлів і наборів файлів, а також тестування коротких криптографічних операцій (генерація ключів, шифрування/розшифрування малих блоків) для асиметричних алгоритмів. Під час запусків фіксуються ключові метрики: час виконання, середня швидкість оброблення даних, пікове завантаження CPU, пікове використання RAM і зміна обсягу даних після перетворення. Отримані результати документуються (у вигляді виводу/логів) та надалі узагальнюються у зведених таблицях і графіках.

Причиною розробки (та використання) програмного продукту є потреба в об’єктивному порівнянні реалізацій за однакових умов виконання й у виявленні практичних відмінностей, які не завжди очевидні з теоретичних характеристик. Окрема увага приділяється впливу типу носія інформації (M.2/SSD/HDD) і режиму оброблення (одиночні файли, пакетні набори, сценарії з великою кількістю об’єктів) на підсумкову продуктивність.

Область застосування програмного забезпечення охоплює задачі захисту даних у файлових сховищах і резервному копіюванні, у прикладних інформаційних системах, а також у середовищах із різними обмеженнями ресурсів (робочі станції, сервери, вбудовані пристрої), де вибір алгоритму та режиму шифрування має безпосередній вплив на швидкодію й експлуатаційні витрати.

1. ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

1.1. Види діяльності, функції, для автоматизації яких призначено даний засіб автоматизації

Додаток призначений дослідження роботи алгоритмів стиснення на різних типах даних, знаходження залежностей, переваг, недоліків. Основні функції програми включають:

– стискання даних;

– відновлення стиснутих даних;

– генерація звітів;

– проведення тестів;

– сортування результатів.

1.2. Вимоги до складу і параметрів технічних засобів

– операційна система Windows 10 або вище;

– програмне забезпечення Microsoft .NET 10

– процесор рівня Intel Core i5 10-го покоління або аналогічний, з підтримкою багатопотокової обробки, тактовою частотою не нижче 3,5 ГГц та щонайменше 8 ядрами;

– твердотільний накопичувач (SSD) ємністю не менше 512 ГБ;

– оперативна пам’ять обсягом не менше 16 ГБ стандарту DDR4 або DDR5;

1.3. Вимоги до вхідних даних

– файли та дані будь яких типів;

– кількість генерованих даних для тестових наборів;

– виконання в багатопотоковому режимі

1.4. Вимоги до вихідних даних

– файли з результатами досліджень;

– обрана комбінація алгоритму та препроцесингу;

– графіки результатів та порівнянь у форматі JPEG;

2. ПІДГОТОВКА ДО РОБОТИ

2.1. Склад і зміст дистрибутивного носія даних

До складу дистрибутиву програмного продукту, призначеного для дослідження ефективності алгоритмів шифрування та дешифрування даних, входять:

– .docx файл із керівництвом користувача;

– .exe файл із програмним застосунком;

– папка «Data» (за наявності) з прикладами вхідних файлів для тестування;

– файл налаштувань з параметрами запуску/значеннями за замовчуванням;

– приклад файлу результатів (.json / .csv, опціонально) для демонстрації формату збереження та подальшої обробки даних.

2.2. Порядок завантаження даних та програм

Перед початком роботи підготуйте програмне середовище та тестові дані.

– перевірте, що на комп’ютері встановлено операційну систему Windows 10 або вище;

– перевірте наявність установленого середовища виконання Microsoft .NET (відповідної версії);

– розархівуйте отриманий архів із програмою у вибрану папку;

– ознайомтеся з керівництвом користувача та описом можливостей програми;

– підготуйте тестові дані (файл або набір файлів), які планується використати для експериментів;

– запустіть програму, відкривши виконуваний файл (.exe) із папки встановлення.

2.3. Порядок перевірки працездатності

– інтерфейс відображається коректно та не закривається з помилкою;

– доступні основні функції (вибір алгоритму/режиму, завантаження або вибір даних, запуск операцій шифрування/дешифрування, перегляд/збереження результатів);

– програма реагує на дії користувача (кнопки, поля, списки) без зависань і аварійного завершення.

3. ОПИС ОПИРАЦІЙ

3.1. Вибір алгоритму

Умови виконання: запустіть додаток і відкрийте вікно керування (вибору алгоритмів).

Підготовчі дії: не виконуйте.

Основні дії:

– оберіть потрібний алгоритм у вікні керування (AES, DES, 3DES, Serpent, Blowfish, ChaCha20, RSA, ECC);

– дочекайтеся відкриття відповідного вікна алгоритму;

Заключні дії: перейдіть до виконання операцій шифрування/розшифрування у відкритому модулі.

У вікні керування користувач обирає криптографічний алгоритм для подальшого шифрування/розшифрування. Загальний вигляд вікна вибору алгоритмів наведено на рис. 1.

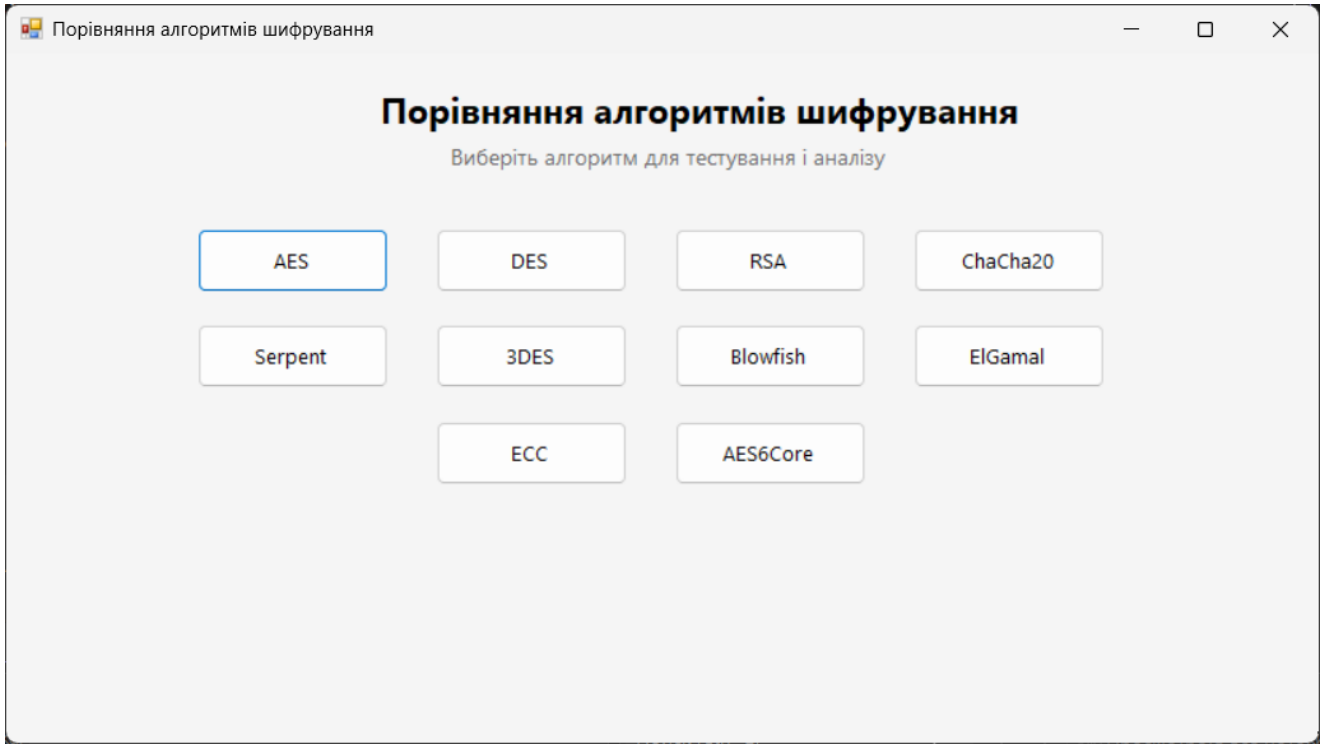

Рис. 1. Вікновибору алгоритму

3.2. Шифрування та розшифрування файлу/папки симетричним алгоритмом

Умови виконання: оберіть симетричний алгоритм; задайте ключ; вкажіть файл або папку.

Підготовчі дії:

– введіть ключ вручну або натисніть «Згенерувати ключ»;

– вкажіть шлях до файлу/папки: введіть його вручну або перетягніть об’єкт у поле шляху.

Основні дії:

– переконайтеся, що поле «Ключ» заповнене;

– оберіть файл або папку в полі «Шлях до файлу або папки»;

– натисніть «Зашифрувати» або «Розшифрувати»;

– дочекайтеся завершення операції (орієнтуйтеся на індикатор прогресу та повідомлення у журналі).

Заключні дії:

– перевірте, що у каталозі з’явився результат (зашифрований/розшифрований файл(и) з відповідним суфіксом/назвою);

– перегляньте службові повідомлення та підсумок у полі «Журнал / результати».

Після вибору симетричного алгоритму відкривається робоче вікно для шифрування або розшифрування файлу/папки з введенням (або генерацією) ключа. Приклад такого вікна наведено на рис. 2.

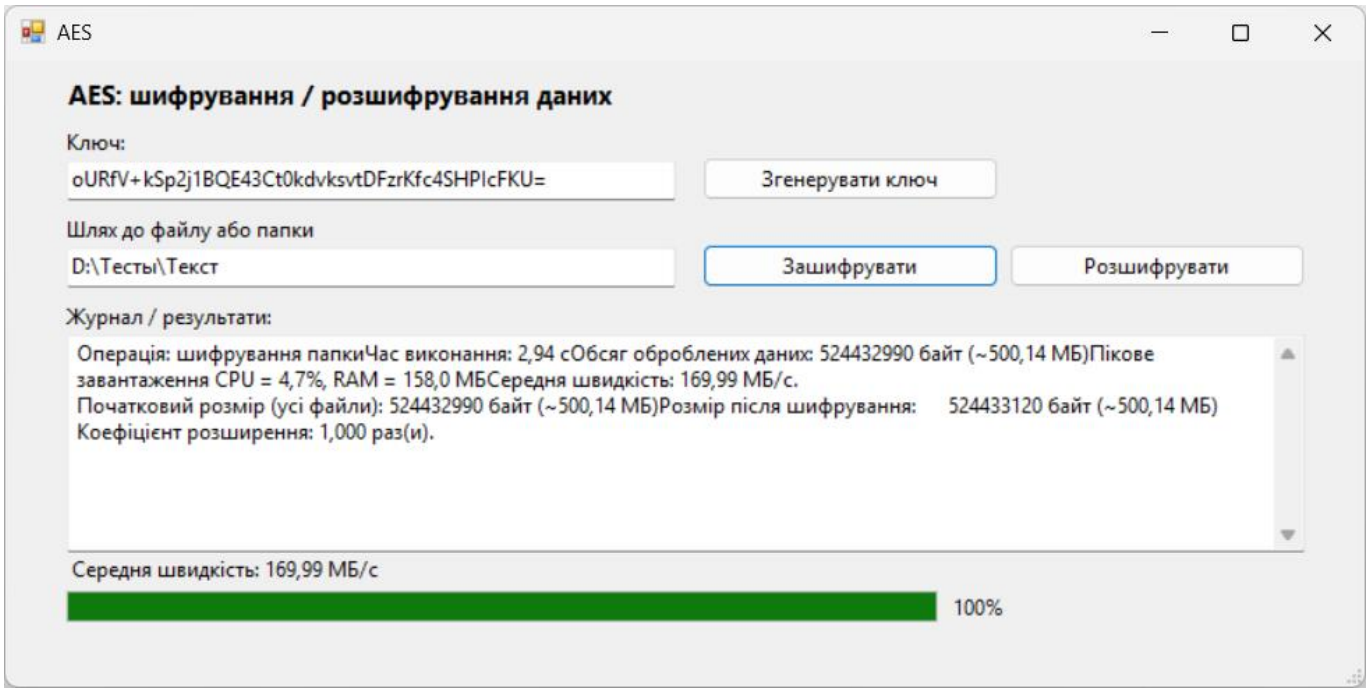

Рис. 2. Вікно симетричного алгоритмому

3.3. Шифрування та розшифрування тексту асиметричним алгоритмом

Умови виконання: відкрийте модуль RSA; підготуйте пару ключів (згенеруйте або завантажте); введіть текст.

Підготовчі дії:

– натисніть «Згенерувати ключі RSA» або завантажте наявні ключі (якщо це передбачено інтерфейсом);

– введіть текст у поле для обробки або завантажте його з файлу (якщо доступно).

Основні дії:

– для шифрування переконайтеся, що доступний відкритий ключ;

– для розшифрування переконайтеся, що доступний закритий ключ;

– введіть або вставте дані у поле «Текст для шифрування/розшифрування»;

– натисніть «Зашифрувати» або «Розшифрувати»;

– перегляньте результат у відповідному полі («Зашифрований текст» / «Розшифрований текст»).

Заключні дії: за потреби скопіюйте результат або збережіть його.

У модулі асиметричного шифрування виконуються операції шифрування та розшифрування тексту із застосуванням ключової пари. Інтерфейс відповідного модуля наведено на рис. 3.

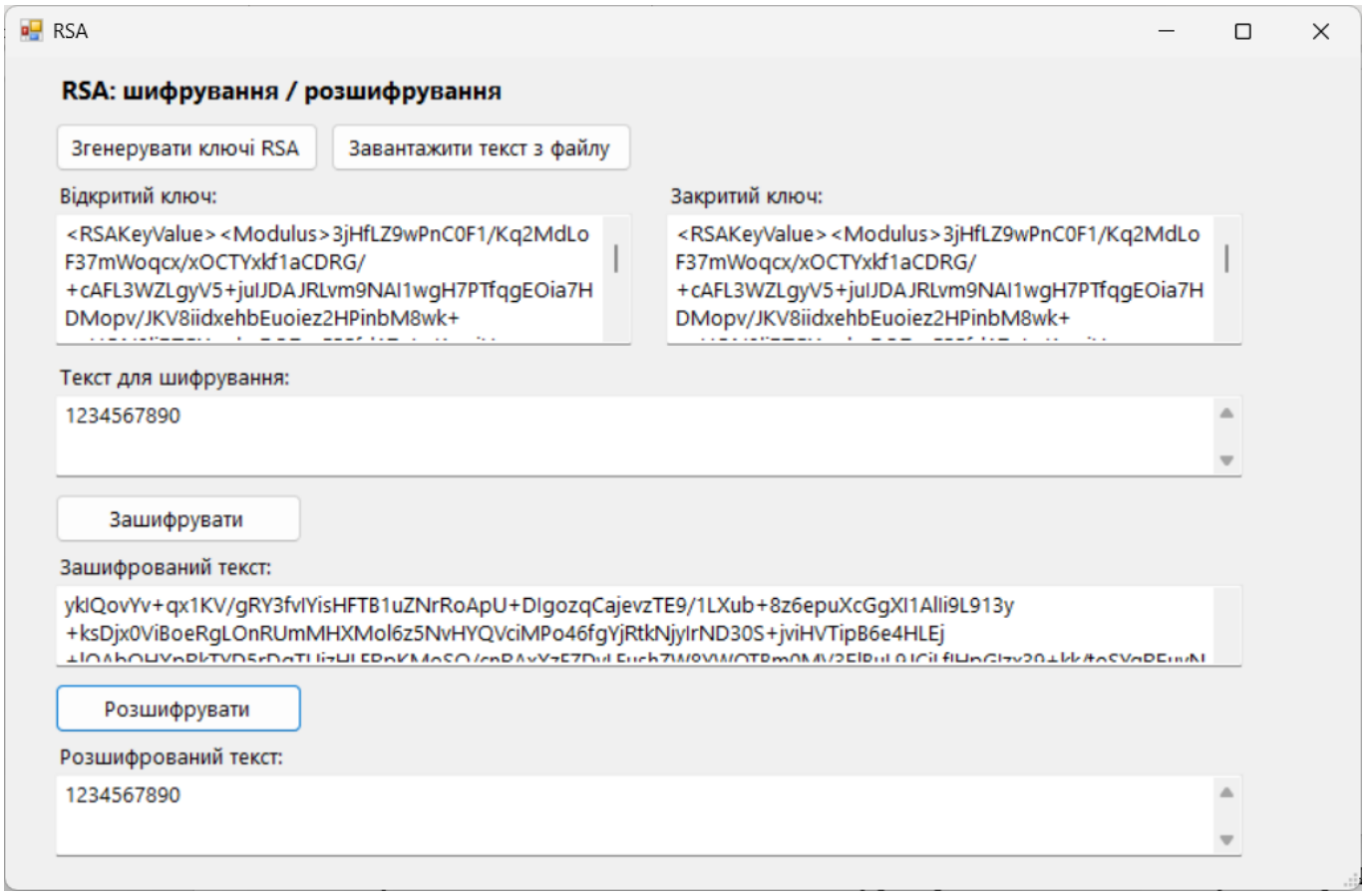

Рис. 3. Вікно асиметричного алгоритмому

3.4. Перегляд звіту

Умови виконання: запустіть операцію шифрування або розшифрування та дочекайтеся її завершення.

Підготовчі дії:

– оберіть потрібний алгоритм/модуль;

– задайте вхідні дані;

– натисніть «Зашифрувати» або «Розшифрувати».

Основні дії:

– дочекайтеся завершення операції;

– перегляньте звіт, який автоматично з’явиться після завершення обробки (інформаційне вікно з підсумками).

Заключні дії:

– натисніть «ОК», щоб закрити вікно звіту;

– за потреби перегляньте детальні повідомлення у журналі відповідного модуля.

Після завершення операції шифрування або розшифрування система автоматично відображає підсумковий звіт у вигляді інформаційного вікна. Приклад вікна звіту наведено на рис. 4.

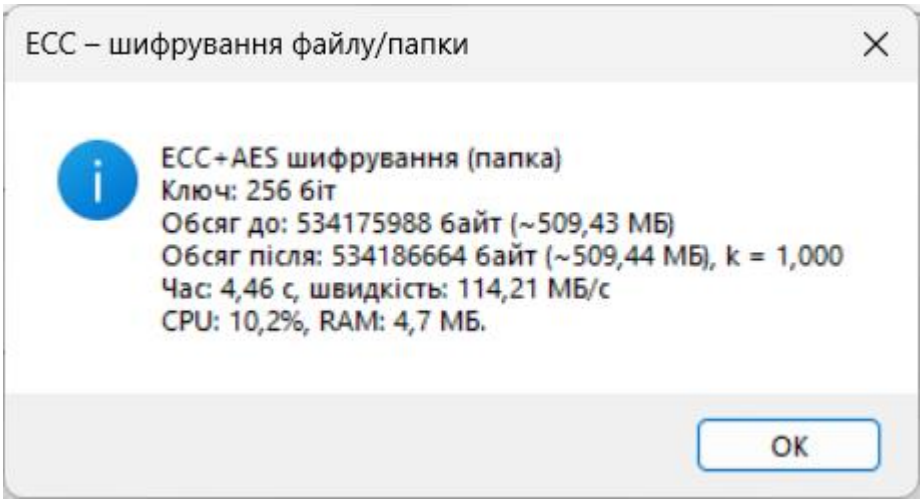

Рис. 4. Вікно звіту

4. АВАРІЙНІ СИТУАЦІЇ

Під час роботи програмного продукту з дослідження ефективності алгоритмів шифрування/дешифрування можуть виникати такі аварійні ситуації:

1. Зависання або дуже тривале виконання операції під час оброблення великих файлів, серійних запусків тестів або генерації ключів (особливо для асиметричних алгоритмів). Запобігання/усунення: зменшить обсяг даних або кількість повторів, запускатйте операції послідовно, закрийте зайві програми, дочекайтеся завершення процесу або перезапустити програму.

2. Помилка через некоректні параметри шифрування (порожній/невірний ключ, неправильний формат ключа, невідповідність ключа алгоритму, відсутній IV/nonce там, де він потрібен). Запобігання/усунення: використовувати генерацію ключів у програмі, перевіряти коректність введених значень, не змішувати ключі між різними алгоритмами, для дешифрування застосовувати ті самі параметри, що й під час шифрування.

3. Неможливість прочитати або записати файл (файл зайнятий іншою програмою, немає прав доступу, шлях некоректний, антивірус/захист блокує операцію). Запобігання/усунення: закрити програми, що використовують файл, обрати іншу папку збереження, запустити від імені користувача з потрібними правами, перевірити дозволи та налаштування захисту.

4. Некоректне дешифрування або пошкоджений результат (використано інший ключ/алгоритм/режим, вхідні дані змінені або файл частково пошкоджено). Запобігання/усунення: перевірити, що обрано правильний алгоритм і ключ, не змінювати зашифровані файли вручну, зберігати оригінальні параметри шифрування.

5. Аварійне завершення через помилку середовища виконання (відсутні необхідні компоненти платформи, збій бібліотеки криптографії, системна помилка). Запобігання/усунення: перевірити встановлення необхідного середовища виконання, перезапустити програму, повторити операцію; за потреби – перевстановити додаток або компоненти платформи.

5. РЕКОМЕНДАЦІЇ ЩОДО ЗАСВОЄННЯ

Для успішного засвоєння роботи з програмою необхідно уважно ознайомитися з керівництвом користувача та виконати підготовчі дії, наведені в розділі 3. Після запуску рекомендується спочатку протестувати роботу одного симетричного алгоритму (наприклад, AES) на невеликому файлі, перевіривши коректність результату шляхом зворотного дешифрування, а вже потім переходити до інших алгоритмів і більших обсягів даних. Також слід уважно дотримуватися правил роботи з ключами (генерація, збереження, введення без помилок), оскільки від цього залежить коректність шифрування/дешифрування та достовірність отриманих результатів.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009.

ДОДАТОК В

Текст програми

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ

CRYPTOBENCH

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01529 – 01 12 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олена КУРОП’ЯТНИК
Виконавець
_____Олексій НОР
Нормоконтролер
_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО

44165850.01529 – 01 12 01

CRYPTOBENCH

Текст програми

Листів 71

2026

АНОТАЦІЯ

Документ 44165850.1527-01 І3 01 «Програмне забезпечення для проведення досліджень ефективності алгоритмів шифрування та дешифрування даних. Керівництво користувача. Керівництво дослідника криптографічних алгоритмів» входить до складу програмної документації на програмний комплекс, призначений для експериментального порівняння криптографічних алгоритмів у прикладних сценаріях оброблення даних.

У даному документі представлено «Керівництво користувача. Керівництво дослідника криптографічних алгоритмів». Програми написані на мові C#. Об'єм пам'яті, що займає програма, складає 8 Мб. Конфігурація комп'ютера стандартна. Операційна система Windows 10 або вище.

ЗМІСТ

1. ОПИС СТРУКТУРИ ПРОГРАМИ 5
2. ТЕКСТ ПРОГРАМИ 7
2.1 3DESForm.cs 7
2.2 AES12Form.cs 12
2.3 AESForm.cs 18
2.4 BlowfishForm.cs 25
2.5 ChaCha20Form.cs 32
2.6 DESForm.cs 37
2.7 ECCForm.cs 43
2.8 ElGamalForm.cs 55
2.9 MainForm.cs 61
2.10 RSAForm.cs 62
2.11 SerpentForm.cs 65

1. ОПИС СТРУКТУРИ ПРОГРАМИ

1. Main (головний модуль)

Назначение: точка входа и запуск всей программы.

Как работает: запускает UI, принимает выбор алгоритма и режим (шифр/дешифр), передаёт параметры (путь, ключ, настройки) в выбранный криптомодуль и показывает результат/ошибки пользователю.

2. AES (модуль AES)

Назначение: шифрование/дешифрование файлов алгоритмом AES (блочный).

Как работает:

– Шифрование: из ключа формируется рабочий ключ AES → генерируется IV → файл читается блоками → данные шифруются → IV сохраняется вместе с результатом (в заголовке/начале файла).

– Дешифрование: из файла читается IV → тем же ключом выполняется обратное преобразование → исходные данные восстанавливаются и записываются в файл.

3. AES12 (модуль AES12 – 12 потоков)

Назначение: ускоренная обработка AES за счёт параллельности.

Как работает: это тот же AES, но операции запускаются в 12 потоков (обычно параллельно по файлам: несколько файлов шифруются/расшифровываются одновременно). Логика AES та же: ключ + IV + блочная обработка, просто задачи распределяются между 12 рабочими потоками.

4. DES (модуль DES)

Назначение: шифрование/дешифрование файлов алгоритмом DES (блочный).

Как работает:

– Шифрование: формируется ключ DES → генерируется IV → файл шифруется блоками → IV записывается в начало/заголовок результата.

– Дешифрование: читается IV → выполняется обратная операция DES тем же ключом → получаются исходные данные.

5. 3DES / TripleDES (модуль потрійного DES)

Назначение: шифрование/дешифрование TripleDES (усиленный вариант DES).

Как работает:

– Шифрование: проверяется длина ключа (обычно 16 или 24 байта) → генерируется IV → данные шифруются блочно алгоритмом 3DES → IV сохраняется в выходном файле.

– Дешифрование: читается IV → выполняется расшифрование тем же ключом → восстанавливается оригинальный файл.

6. Twofish (модуль Twofish)

Назначение: шифрование/дешифрование блочным алгоритмом Twofish.

Как работает:

– Шифрование: по ключу настраивается Twofish → создаётся IV (если используется блочный режим) → файл обрабатывается блоками → служебные параметры (например IV) сохраняются в начале результата.

– Дешифрование: извлекаются служебные параметры → выполняется обратное преобразование Twofish → данные возвращаются в исходный вид.

7. Blowfish (модуль Blowfish)

Назначение: шифрование/дешифрование блочным алгоритмом Blowfish.

Как работает: аналогично другим блочным: ключ → (при необходимости) IV → блочная обработка файла → сохранение IV/служебных данных → обратное восстановление при дешифровании.

8. ChaCha20 (модуль ChaCha20)

Назначение: шифрование/дешифрование потоковым алгоритмом ChaCha20.

Как работает:

– Шифрование: используется ключ и nonce (одноразовый параметр) → генерируется поток ключа → данные файла “складываются” с потоком → nonce сохраняется вместе с результатом.

– Дешифрование: читается nonce → с тем же ключом генерируется такой же поток → выполняется обратная операция → получаются исходные байты.

2. ТЕКСТ ПРОГРАМИ

2.1 3DESForm.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CruptoBench
{
    public partial class TripleDESForm : Form
    {
        private const int IoBufferSize = 81920;

        private byte[] _encryptionKey;

        private Stopwatch _operationStopwatch;
        private TimeSpan _previousCpuTime;
        private DateTime _previousSampleUtc;
        private double _peakCpuUsagePercent;
        private double _peakMemoryMb;

        private long _totalInputBytes;
        private long _totalOutputBytes;

        private bool _wrongKeyShown = false;
        private bool _wrongKeyDetected = false;

        public TripleDESForm()
        {
            InitializeComponent();

            txtPath.AllowDrop = true;
            txtPath.DragEnter += TxtPath_DragEnter;
            txtPath.DragDrop += TxtPath_DragDrop;
        }

        private void TxtPath_DragEnter(object sender, DragEventArgs e)
        {
            if (e.Data.GetDataPresent(DataFormats.FileDrop))
            {
                e.Effect = DragDropEffects.Copy;
            }
        }

        private void TxtPath_DragDrop(object sender, DragEventArgs e)
        {
            var droppedItems = e.Data.GetData(DataFormats.FileDrop) as string[];
            if (droppedItems is { Length: > 0 })
            {
                txtPath.Text = droppedItems[0];
            }
        }

        private void btnGenerateKey_Click(object sender, EventArgs e)
        {
            using var tripleDes = TripleDES.Create();
            tripleDes.KeySize = 192;
            tripleDes.GenerateKey();

            _encryptionKey = tripleDes.Key;
            txtKey.Text = Convert.ToBase64String(_encryptionKey);
        }

        private async void btnEncrypt_Click(object sender, EventArgs e)
        {
            await RunWithMonitoringAsync(encrypt: true);
        }

        private async void btnDecrypt_Click(object sender, EventArgs e)
        {
            await RunWithMonitoringAsync(encrypt: false);
        }

        private async Task RunWithMonitoringAsync(bool encrypt)
        {
            _wrongKeyShown = false;
            _wrongKeyDetected = false;

            string inputPath = txtPath.Text?.Trim() ?? string.Empty;

            if (!IsExistingFileOrDirectory(inputPath))
            {
                MessageBox.Show("Виберіть файл або папку.");
                return;
            }
```

```
            if (!TryReadKey(out
_encryptionKey))
            {
                MessageBox.Show("Невірний
формат ключа.");
                return;
            }

            Ui(() => {
                txtLog.Clear();
                progressBar.Value = 0;
                lblProgress.Text = "0%";
            });

            ResetStats();

            using var currentProcess =
Process.GetCurrentProcess();
            currentProcess.Refresh();

            _previousCpuTime =
currentProcess.TotalProcessorTime;
            _previousSampleUtc =
DateTime.UtcNow;

            _operationStopwatch =
Stopwatch.StartNew();

            try
            {
                if
(Directory.Exists(inputPath))
                {
                    await
ProcessDirectoryAsync(inputPath, encrypt,
currentProcess);
                }
                else
                {
                    await
ProcessSingleFileAsync(inputPath,
encrypt, currentProcess);
                }


_operationStopwatch.Stop();

SamplePerformance(currentProcess);

                if (_wrongKeyDetected &&
!encrypt)
                {
                    return;
                }

                ShowFinalResult();
            }
            catch (Exception ex)
            {

_operationStopwatch?.Stop();
                AppendLog($"Помилка:
{ex.Message}{Environment.NewLine}");

MessageBox.Show(ex.Message);
            }
        }

        private async Task
ProcessDirectoryAsync(string folderPath,
bool encrypt, Process currentProcess)
        {
            string outputSuffix = encrypt
? "_3des_encrypted" : "_3des_decrypted";
            string outputFolderPath =
folderPath + outputSuffix;


Directory.CreateDirectory(outputFolderPat
h);

            string[] inputFiles =
Directory.GetFiles(folderPath);
            if (inputFiles.Length == 0)
            {
                MessageBox.Show("У
вибраній папці немає файлів.");
                return;
            }

            Ui(() => {
                progressBar.Minimum = 0;
                progressBar.Maximum =
inputFiles.Length;
                progressBar.Value = 0;
                lblProgress.Text = "0%";
            });

            for (int fileIndex = 0;
fileIndex < inputFiles.Length;
fileIndex++)
            {
                if (_wrongKeyDetected &&
!encrypt)
                {
                    break;
                }

                string inputFilePath =
inputFiles[fileIndex];
                string outputFilePath =
Path.Combine(outputFolderPath,
Path.GetFileName(inputFilePath));

                await
ProcessOneFileAsync(inputFilePath,
outputFilePath, encrypt, currentProcess);

                if (_wrongKeyDetected &&
!encrypt)
                {
                    break;
                }

                UpdateProgress(fileIndex
+ 1, inputFiles.Length);
            }
        }

        private async Task
ProcessSingleFileAsync(string
inputFilePath, bool encrypt, Process
currentProcess)
        {
```

```
            string outputFilePath =
Path.Combine(

Path.GetDirectoryName(inputFilePath) ??
string.Empty,

Path.GetFileNameWithoutExtension(inputFil
ePath) +
                (encrypt ?
"_3des_encrypted" : "_3des_decrypted") +

Path.GetExtension(inputFilePath));

            Ui(() => {
                progressBar.Minimum = 0;
                progressBar.Maximum = 1;
                progressBar.Value = 0;
                lblProgress.Text = "0%";
            });

            await
ProcessOneFileAsync(inputFilePath,
outputFilePath, encrypt, currentProcess);

            if (_wrongKeyDetected &&
!encrypt)
            {
                return;
            }

            UpdateProgress(1, 1);
        }

        private async Task
ProcessOneFileAsync(string inputFilePath,
string outputFilePath, bool encrypt,
Process currentProcess)
        {
            long inputFileBytes = 0;
            long outputFileBytes = 0;

            var fileStopwatch =
Stopwatch.StartNew();

            try
            {
                inputFileBytes = new
FileInfo(inputFilePath).Length;

                var result = await
Task.Run(() => {
                    bool wk;
                    string err;
                    bool ok =
TryProcessFile3DES(inputFilePath,
outputFilePath, encrypt, out wk, out
err);
                    return (Ok: ok,
WrongKey: wk, ErrorText: err);
                });

                if (!result.Ok)
                {
                    if (!encrypt &&
result.WrongKey)
                    {
                        _wrongKeyDetected
= true;


AppendLog("Невірний ключ" +
Environment.NewLine);

                        if
(!_wrongKeyShown)
                        {

_wrongKeyShown = true;

MessageBox.Show("Невірний ключ");
                        }

                        return;
                    }

                    AppendLog($"Файл:
{Path.GetFileName(inputFilePath)} |
Помилка:
{result.ErrorText}{Environment.NewLine}")
;
                    return;
                }

                outputFileBytes = new
FileInfo(outputFilePath).Length;

                _totalInputBytes +=
inputFileBytes;
                _totalOutputBytes +=
outputFileBytes;

                AppendLog(
                    $"Файл:
{Path.GetFileName(inputFilePath)} ->
{Path.GetFileName(outputFilePath)} | " +
                    $"Вхід:
{inputFileBytes / 1024.0 / 1024.0:F2} МБ
| " +
                    $"Вихід:
{outputFileBytes / 1024.0 / 1024.0:F2} МБ
| " +
                    $"Час:
{fileStopwatch.Elapsed.TotalSeconds:F2}
c{Environment.NewLine}");
            }
            catch (Exception ex)
            {
                AppendLog($"Файл:
{Path.GetFileName(inputFilePath)} |
Помилка:
{ex.Message}{Environment.NewLine}");
            }
            finally
            {
                fileStopwatch.Stop();

SamplePerformance(currentProcess);
            }
        }

        private bool TryProcessFile3DES(
            string inputFilePath,
            string outputFilePath,
            bool encrypt,
```

```
            out bool wrongKey,
            out string errorText
        )
        {
            wrongKey = false;
            errorText = string.Empty;

            try
            {
                using var inputStream =
new FileStream(
                    inputFilePath,
                    FileMode.Open,
                    FileAccess.Read,
                    FileShare.Read,
                    IoBufferSize,

FileOptions.SequentialScan
                );

                using var outputStream =
new FileStream(
                    outputFilePath,
                    FileMode.Create,
                    FileAccess.Write,
                    FileShare.None,
                    IoBufferSize,

FileOptions.SequentialScan
                );

                using var tripleDes =
TripleDES.Create();
                tripleDes.Key =
_encryptionKey;
                tripleDes.Mode =
CipherMode.CBC;
                tripleDes.Padding =
PaddingMode.PKCS7;

                if (encrypt)
                {

tripleDes.GenerateIV();
                    byte[]
initializationVector = tripleDes.IV;


outputStream.Write(initializationVector,
0, initializationVector.Length);

                    using var encryptor =
tripleDes.CreateEncryptor();
                    using var
cryptoStream = new
CryptoStream(outputStream, encryptor,
CryptoStreamMode.Write);


inputStream.CopyTo(cryptoStream,
IoBufferSize);

cryptoStream.FlushFinalBlock();

                    return true;
                }
                else
                {
                    int ivSizeBytes =
tripleDes.BlockSize / 8;
                    byte[]
initializationVector2 = new
byte[ivSizeBytes];

                    int totalRead = 0;
                    while (totalRead <
ivSizeBytes)
                    {
                        int chunkRead =
inputStream.Read(initializationVector2,
totalRead, ivSizeBytes - totalRead);
                        if (chunkRead <= 0)
                        {
                            break;
                        }
                        totalRead +=
chunkRead;
                    }

                    if (totalRead !=
ivSizeBytes)
                    {
                        errorText =
"Некоректний формат файлу (IV не
знайдено).";

TryDeleteFile(outputFilePath);
                        return false;
                    }

                    tripleDes.IV =
initializationVector2;

                    using var decryptor =
tripleDes.CreateDecryptor();
                    using var cryptoStream2 =
new CryptoStream(inputStream, decryptor,
CryptoStreamMode.Read);

                    try
                    {

cryptoStream2.CopyTo(outputStream,
IoBufferSize);
                        outputStream.Flush();
                        return true;
                    }
                    catch
(CryptographicException)
                    {
                        wrongKey = true;
                        errorText = "Невірний
ключ";

TryDeleteFile(outputFilePath);
                        return false;
                    }
                }
                catch (Exception ex)
                {
                    errorText = ex.Message;

TryDeleteFile(outputFilePath);
                    return false;
                }
```

```
        }

        private void TryDeleteFile(string
path)
        {
            try
            {
                if (File.Exists(path))
                {
                    File.Delete(path);
                }
            }
            catch { }
        }

        private void ResetStats()
        {
            _totalInputBytes = 0;
            _totalOutputBytes = 0;
            _peakCpuUsagePercent = 0.0;
            _peakMemoryMb = 0.0;
            _operationStopwatch = null;
        }

        private void
SamplePerformance(Process currentProcess)
        {
            currentProcess.Refresh();

            var sampleUtc =
DateTime.UtcNow;
            var cpuTimeNow =
currentProcess.TotalProcessorTime;

            double wallTimeMs =
(sampleUtc -
_previousSampleUtc).TotalMilliseconds;
            if (wallTimeMs > 1)
            {
                double cpuTimeMs =
(cpuTimeNow -
_previousCpuTime).TotalMilliseconds;
                double cpuUsagePercent =
cpuTimeMs / (Environment.ProcessorCount *
wallTimeMs) * 100.0;

                if
(!double.IsNaN(cpuUsagePercent) &&
!double.IsInfinity(cpuUsagePercent))
                {
                    _peakCpuUsagePercent
= Math.Max(_peakCpuUsagePercent,
cpuUsagePercent);
                }
            }

            double memoryMb =
currentProcess.WorkingSet64 / (1024.0 *
1024.0);
            _peakMemoryMb =
Math.Max(_peakMemoryMb, memoryMb);

            _previousCpuTime =
cpuTimeNow;
            _previousSampleUtc =
sampleUtc;
        }

        private void ShowFinalResult()
        {
            double elapsedSeconds =
_operationStopwatch?.Elapsed.TotalSeconds
?? 0.0;
            if (elapsedSeconds <= 0)
            {
                elapsedSeconds = 0.0001;
            }

            double inputSizeMb =
_totalInputBytes / 1024.0 / 1024.0;
            double outputSizeMb =
_totalOutputBytes / 1024.0 / 1024.0;
            double expansionFactor =
_totalInputBytes > 0 ?
(double)_totalOutputBytes /
_totalInputBytes : 0.0;
            double averageSpeedMbPerSec =
inputSizeMb / elapsedSeconds;

            string sizeInfo =
string.Format(
                "Початковий обсяг даних:
{0:F2} МБ\n" +
                "Обсяг після обробки:
{1:F2} МБ\n" +
                "Коефіцієнт розширення:
{2:F3}x",
                inputSizeMb,
outputSizeMb, expansionFactor);

            string resultMsg =
string.Format(
                "Операцію завершено за
{0:F2} с.\n" +
                "Пікове завантаження CPU
= {1:F1}%, RAM = {2:F1} MB\n" +
                "Середня швидкість:
{3:F2} МБ/с\n\n{4}",
                elapsedSeconds,
_peakCpuUsagePercent, _peakMemoryMb,
averageSpeedMbPerSec, sizeInfo);

            AppendLog(resultMsg +
Environment.NewLine +
Environment.NewLine);
            MessageBox.Show(resultMsg);
        }

        private bool TryReadKey(out
byte[] decodedKey)
        {
            decodedKey = null;

            try
            {
                byte[] rawKey =
Convert.FromBase64String(txtKey.Text?.Tri
m() ?? string.Empty);
                if (rawKey.Length != 16
&& rawKey.Length != 24)
                {
                    return false;
                }
```

```
                decodedKey = rawKey;
                return true;
            }
            catch
            {
                return false;
            }A
        }

        private static bool
IsExistingFileOrDirectory(string path)
        {
            if
(string.IsNullOrWhiteSpace(path))
            {
                return false;
            }
            return File.Exists(path) ||
Directory.Exists(path);
        }

        private void UpdateProgress(int
processedCount, int totalCount)
        {
            Ui(() => {
                if (totalCount <= 0)
                {
                    totalCount = 1;
                }

                progressBar.Maximum =
totalCount;
                progressBar.Value =
Math.Min(Math.Max(processedCount,
progressBar.Minimum),
progressBar.Maximum);
                int percent =
(int)Math.Round((double)progressBar.Value
/ totalCount * 100.0);
                lblProgress.Text =
$"{percent}%";
            });
        }

        private void AppendLog(string
text)
        {
            Ui(() =>
txtLog.AppendText(text));
        }

        private void Ui(Action action)
        {
            if (IsDisposed)
            {
                return;
            }
            if (InvokeRequired)
            {
                Invoke(action);
            }
            else
            {
                action();
            }
        }
    }
}
```

2.2 AES12Form.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CruptoBench
{
    public partial class AES12Form : Form
    {
        private Stopwatch
_operationStopwatch;
        private TimeSpan
_previousCpuTime;
        private DateTime
_previousSampleUtc;
        private double
_peakCpuUsagePercent = 0.0;
        private double _peakMemoryMb =
0.0;

        private long _lastUiUpdateBytes =
0;

        private bool _wrongKeyShown =
false;
        private bool _wrongKeyDetected =
false;

        public AES12Form()
        {
            InitializeComponent();
            txtInput.AllowDrop = true;
            txtInput.DragEnter +=
TxtInput_DragEnter;
            txtInput.DragDrop +=
TxtInput_DragDrop;
        }

        private void
btnGenerateKey_Click(object sender,
EventArgs e)
        {
            using (Aes aes =
Aes.Create())
            {
                txtKey.Text =
Convert.ToBase64String(aes.Key);
            }
        }

        private async void
btnEncryptFile_Click(object sender,
EventArgs e)
        {
```

```
            if (!TryReadAesKey(out byte[]
aesKey))
            {
                MessageBox.Show("Спочатку
згенеруйте або введіть ключ AES.");
                return;
            }

            string inputPath =
txtInput.Text?.Trim() ?? string.Empty;
            if
(!IsExistingFileOrDirectory(inputPath))
            {
                MessageBox.Show("Файл або
папка не знайдені.");
                return;
            }

            if
(Directory.Exists(inputPath))
            {
                string folderPath =
inputPath;
                long plainTotalBytes =
GetDirectoryTotalBytes(folderPath);

                if (plainTotalBytes == 0)
                {

MessageBox.Show("Папка порожня, немає
файлів для шифрування.");
                    return;
                }

                await
MonitorPerformanceAsync(
                    () =>
EncryptFolderAsync(folderPath, aesKey,
plainTotalBytes),
                    plainTotalBytes,
                    "шифрування папки"
                );

                string outputDir =
folderPath + "_encrypted";
                long cipherTotalBytes =
Directory.Exists(outputDir) ?
GetDirectoryTotalBytes(outputDir) : 0;

                string sizeMsg =
BuildSizeMessageForFolder(plainTotalBytes
, cipherTotalBytes);
                txtLog.AppendText(sizeMsg
+ Environment.NewLine);
                MessageBox.Show(sizeMsg,
"Розмір даних (AES)");
                return;
            }

            if (File.Exists(inputPath))
            {
                string filePath =
inputPath;
                long plainBytes = new
FileInfo(filePath).Length;

                if (plainBytes == 0)
                {
                    MessageBox.Show("Файл
порожній, немає даних для шифрування.");
                    return;
                }

                string outputPath =
BuildOutputPath(filePath, "_encrypted");

                await
MonitorPerformanceAsync(
                    () =>
EncryptFileAsync(filePath, outputPath,
aesKey, plainBytes, 0),
                    plainBytes,
                    "шифрування файлу"
                );

                long cipherBytes =
File.Exists(outputPath) ? new
FileInfo(outputPath).Length : 0;

                string sizeMsg =
BuildSizeMessageForFile(plainBytes,
cipherBytes);
                txtLog.AppendText(sizeMsg
+ Environment.NewLine);
                MessageBox.Show(sizeMsg,
"Розмір даних (AES)");
                return;
            }

            MessageBox.Show("Файл або
папка не знайдені.");
        }

        private async void
btnDecryptFile_Click(object sender,
EventArgs e)
        {
            _wrongKeyShown = false;
            _wrongKeyDetected = false;

            if (!TryReadAesKey(out byte[]
aesKey))
            {
                MessageBox.Show("Спочатку
введіть ключ AES для розшифрування.");
                return;
            }

            string inputPath =
txtInput.Text?.Trim() ?? string.Empty;
            if
(!IsExistingFileOrDirectory(inputPath))
            {
                MessageBox.Show("Файл або
папка не знайдені.");
                return;
            }

            if
(Directory.Exists(inputPath))
            {
                string folderPath =
inputPath;
                long totalBytes =
```

```
GetDirectoryTotalBytes(folderPath);

                if (totalBytes == 0)
                {

MessageBox.Show("Папка порожня, немає
файлів для розшифрування.");
                    return;
                }

                await
MonitorPerformanceAsync(
                    () =>
DecryptFolderAsync(folderPath, aesKey,
totalBytes),
                    totalBytes,
                    "розшифрування папки"
                );

                return;
            }

            if (File.Exists(inputPath))
            {
                string filePath =
inputPath;
                long fileBytes = new
FileInfo(filePath).Length;

                if (fileBytes == 0)
                {
                    MessageBox.Show("Файл
порожній, немає даних для
розшифрування.");
                    return;
                }

                string outputPath =
BuildOutputPath(filePath, "_decrypted");

                await
MonitorPerformanceAsync(
                    () =>
DecryptFileAsync(filePath, outputPath,
aesKey, fileBytes, 0),
                    fileBytes,
                    "розшифрування файлу"
                );

                return;
            }

            MessageBox.Show("Файл або
папка не знайдені.");
        }

        private async Task
MonitorPerformanceAsync(Func<Task<long>>
operation, long totalBytes, string
operationName)
        {
            txtLog.Clear();
            _lastUiUpdateBytes = 0;

            _peakCpuUsagePercent = 0.0;
            _peakMemoryMb = 0.0;

            Process proc =
Process.GetCurrentProcess();
            proc.Refresh();

            _previousCpuTime =
proc.TotalProcessorTime;
            _previousSampleUtc =
DateTime.UtcNow;

            _operationStopwatch =
Stopwatch.StartNew();

            long processedBytes;
            try
            {
                processedBytes = await
operation.Invoke();

                if (_wrongKeyDetected)
                {

_operationStopwatch.Stop();
                    lblSpeed.Text =
string.Empty;
                    return;
                }
            }
            catch (Exception ex)
            {

_operationStopwatch.Stop();

MessageBox.Show(ex.Message);

txtLog.AppendText("Помилка: " +
ex.Message + Environment.NewLine);
                lblSpeed.Text =
string.Empty;
                return;
            }

            _operationStopwatch.Stop();
            proc.Refresh();

            double elapsedSeconds =
_operationStopwatch.Elapsed.TotalSeconds;
            if (elapsedSeconds <= 0)
            {
                elapsedSeconds = 0.0001;
            }

            SamplePerformance(proc);

            double processedMb =
processedBytes / 1024.0 / 1024.0;
            double avgSpeedMbPerSec =
processedMb / elapsedSeconds;

            string resultMsg =
string.Format(
                "Операція: {0}\n" +
                "Час виконання: {1:F2}
c\n" +
                "Обсяг оброблених даних:
{2} байт (~{3:F2} МБ)\n" +
                "Пікове завантаження CPU
= {4:F1}%, RAM = {5:F1} МБ\n" +
```

```
                "Середня швидкість:
{6:F2} МБ/с.",
                operationName,
                elapsedSeconds,
                processedBytes,
                processedMb,
                _peakCpuUsagePercent,
                _peakMemoryMb,
                avgSpeedMbPerSec
            );

            txtLog.AppendText(resultMsg +
Environment.NewLine);
            lblSpeed.Text = $"Середня
швидкість: {avgSpeedMbPerSec:F2} МБ/с";
            MessageBox.Show(resultMsg,
"Статистика AES");
        }

        private async Task<long>
EncryptFileAsync(string inputPath, string
outputPath, byte[] key, long totalBytes,
long offset)
        {
            using Aes aes = Aes.Create();
            aes.Key = key;
            aes.Mode = CipherMode.CBC;
            aes.Padding =
PaddingMode.PKCS7;
            aes.GenerateIV();
            byte[] iv = aes.IV;

            using FileStream fsIn =
new(inputPath, FileMode.Open,
FileAccess.Read, FileShare.Read, 4096,
true);
            using FileStream fsOut =
new(outputPath, FileMode.Create,
FileAccess.Write, FileShare.None, 4096,
true);

            await fsOut.WriteAsync(iv, 0,
iv.Length);

            using CryptoStream cs =
new(fsOut, aes.CreateEncryptor(),
CryptoStreamMode.Write);

            byte[] buffer = new byte[1024
* 1024];

            long processedBytes = 0;
            int len;
            var watch =
Stopwatch.StartNew();

            while ((len = await
fsIn.ReadAsync(buffer, 0, buffer.Length))
> 0)
            {
                await
cs.WriteAsync(buffer, 0, len);
                processedBytes += len;

UpdateProgress(totalBytes, offset +
processedBytes,
watch.Elapsed.TotalSeconds);
            }

            cs.FlushFinalBlock();
            await fsOut.FlushAsync();

            return processedBytes;
        }

        private async Task<long>
DecryptFileAsync(string inputPath, string
outputPath, byte[] key, long totalBytes,
long offset)
        {
            try
            {
                using FileStream fsIn =
new(inputPath, FileMode.Open,
FileAccess.Read, FileShare.Read, 4096,
true);

                byte[] iv = new byte[16];
                int ivRead = 0;

                while (ivRead <
iv.Length)
                {
                    int chunk = await
fsIn.ReadAsync(iv, ivRead, iv.Length -
ivRead);
                    if (chunk <= 0)
                    {
                        break;
                    }
                    ivRead += chunk;
                }

                if (ivRead != iv.Length)
                {
                    throw new
InvalidDataException("Некоректний формат
файлу (IV не знайдено).");
                }

                using Aes aes =
Aes.Create();
                aes.Key = key;
                aes.IV = iv;
                aes.Mode =
CipherMode.CBC;
                aes.Padding =
PaddingMode.PKCS7;

                using CryptoStream cs =
new(fsIn, aes.CreateDecryptor(),
CryptoStreamMode.Read);
                using FileStream fsOut =
new(outputPath, FileMode.Create,
FileAccess.Write, FileShare.None, 4096,
true);

                byte[] buffer = new
byte[1024 * 1024];

                long processedBytes = 0;
                int len;
                var watch =
Stopwatch.StartNew();
```

```
                while ((len = await
cs.ReadAsync(buffer, 0, buffer.Length)) >
0)
                {
                    await
fsOut.WriteAsync(buffer, 0, len);
                    processedBytes +=
len;

UpdateProgress(totalBytes, offset +
processedBytes,
watch.Elapsed.TotalSeconds);
                }

                await fsOut.FlushAsync();
                return processedBytes;
            }
            catch
(CryptographicException)
            {
                _wrongKeyDetected = true;

                try
                {
                    if
(File.Exists(outputPath))
                    {

File.Delete(outputPath);
                    }
                }
                catch { }

                if (!_wrongKeyShown)
                {
                    _wrongKeyShown =
true;

MessageBox.Show("Неправельний ключ");
                }


txtLog.AppendText("Неправельний ключ" +
Environment.NewLine);
                return 0;
            }
        }

        private async Task<long>
EncryptFolderAsync(string folderPath,
byte[] key, long totalBytes)
        {
            string outDir = folderPath +
"_encrypted";

Directory.CreateDirectory(outDir);

            long processedBytes = 0;

            foreach (var file in
Directory.GetFiles(folderPath))
            {
                string outFile =
Path.Combine(
                    outDir,
Path.GetFileNameWithoutExtension(file) +
"_encrypted" + Path.GetExtension(file)
                );

                processedBytes += await
EncryptFileAsync(file, outFile, key,
totalBytes, processedBytes);
            }

            return processedBytes;
        }

        private async Task<long>
DecryptFolderAsync(string folderPath,
byte[] key, long totalBytes)
        {
            string outDir = folderPath +
"_decrypted";

Directory.CreateDirectory(outDir);

            long processedBytes = 0;

            foreach (var file in
Directory.GetFiles(folderPath))
            {
                if (_wrongKeyDetected)
                {
                    break;
                }

                string outFile =
Path.Combine(
                    outDir,
Path.GetFileNameWithoutExtension(file) +
"_decrypted" + Path.GetExtension(file)
                );

                try
                {
                    processedBytes +=
await DecryptFileAsync(file, outFile,
key, totalBytes, processedBytes);
                }
                catch (Exception ex)
                {

txtLog.AppendText($"Файл:
{Path.GetFileName(file)} | Помилка:
{ex.Message}{Environment.NewLine}");
                }
            }

            return processedBytes;
        }

        private void UpdateProgress(long
totalBytes, long currentBytes, double
elapsedSeconds)
        {
            if (totalBytes <= 0)
            {
                return;
            }

            if (currentBytes -
```

```
_lastUiUpdateBytes >= 64 * 1024 * 1024 ||
currentBytes >= totalBytes)
            {
                double percent =
(double)currentBytes / totalBytes *
100.0;
                if (percent < 0)
                {
                    percent = 0;
                }
                if (percent > 100)
                {
                    percent = 100;
                }

                double speed =
elapsedSeconds > 0
                    ? (currentBytes /
1024.0 / 1024.0) / elapsedSeconds
                    : 0.0;


this.Invoke((MethodInvoker)(() => {
                    progressBar.Value =
(int)percent;
                    lblProgress.Text =
$"{(int)percent}%";
                    lblSpeed.Text =
$"Поточна швидкість: {speed:F2} МБ/с";
                }));

                _lastUiUpdateBytes =
currentBytes;
            }
        }

        private void
SamplePerformance(Process proc)
        {
            var nowUtc = DateTime.UtcNow;
            var cpuNow =
proc.TotalProcessorTime;

            double wallMs = (nowUtc -
_previousSampleUtc).TotalMilliseconds;
            if (wallMs > 1)
            {
                double cpuMs = (cpuNow -
_previousCpuTime).TotalMilliseconds;
                double cpuPercent = cpuMs
/ (Environment.ProcessorCount * wallMs) *
100.0;

                if
(!double.IsNaN(cpuPercent) &&
!double.IsInfinity(cpuPercent))
                {
                    _peakCpuUsagePercent
= Math.Max(_peakCpuUsagePercent,
cpuPercent);
                }
            }

            double memMb =
proc.WorkingSet64 / (1024.0 * 1024.0);
            _peakMemoryMb =
Math.Max(_peakMemoryMb, memMb);
            _previousCpuTime = cpuNow;
            _previousSampleUtc = nowUtc;
        }

        private long
GetDirectoryTotalBytes(string folderPath)
        {
            long total = 0;
            foreach (var file in
Directory.GetFiles(folderPath))
            {
                total += new
FileInfo(file).Length;
            }
            return total;
        }

        private void
TxtInput_DragEnter(object sender,
DragEventArgs e)
        {
            if
(e.Data.GetDataPresent(DataFormats.FileDr
op))
            {
                e.Effect =
DragDropEffects.Copy;
            }
        }

        private void
TxtInput_DragDrop(object sender,
DragEventArgs e)
        {
            var files =
(string[])e.Data.GetData(DataFormats.File
Drop);
            if (files.Length > 0)
            {
                txtInput.Text = files[0];
            }
        }

        private bool TryReadAesKey(out
byte[] key)
        {
            key = null;

            string keyText =
txtKey.Text?.Trim() ?? string.Empty;
            if
(string.IsNullOrWhiteSpace(keyText))
            {
                return false;
            }

            try
            {
                key =
Convert.FromBase64String(keyText);
                return key.Length == 16
|| key.Length == 24 || key.Length == 32;
            }
            catch
            {
                return false;
```

```
            }
        }

        private static bool
IsExistingFileOrDirectory(string path)
        {
            if
(string.IsNullOrWhiteSpace(path))
            {
                return false;
            }
            return File.Exists(path) ||
Directory.Exists(path);
        }

        private static string
BuildOutputPath(string inputPath, string
suffix)
        {
            return Path.Combine(

Path.GetDirectoryName(inputPath) ??
string.Empty,

Path.GetFileNameWithoutExtension(inputPat
h) + suffix +
Path.GetExtension(inputPath)
            );
        }

        private static string
BuildSizeMessageForFolder(long
plainBytes, long cipherBytes)
        {
            double plainMb = plainBytes /
1024.0 / 1024.0;
            double cipherMb = cipherBytes
/ 1024.0 / 1024.0;
            double expansion = plainBytes
> 0 ? (double)cipherBytes / plainBytes :
0.0;

            return
                $"Початковий розмір (усі
файли): {plainBytes} байт (~{plainMb:F2}
МБ)\n" +
                $"Розмір після
шифрування:      {cipherBytes} байт
(~{cipherMb:F2} МБ)\n" +
                $"Коефіцієнт розширення:
{expansion:F3} раз(и).";
        }

        private static string
BuildSizeMessageForFile(long plainBytes,
long cipherBytes)
        {
            double plainMb = plainBytes /
1024.0 / 1024.0;
            double cipherMb = cipherBytes
/ 1024.0 / 1024.0;
            double expansion = plainBytes
> 0 ? (double)cipherBytes / plainBytes :
0.0;

            return
                $"Початковий розмір
файлу: {plainBytes} байт (~{plainMb:F2}
МБ)\n" +
                $"Розмір після
шифрування: {cipherBytes} байт
(~{cipherMb:F2} МБ)\n" +
                $"Коефіцієнт розширення:
{expansion:F3} раз(и).";
        }
    }
}
```

2.3 AESForm.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CruptoBench
{
    public partial class AESForm : Form
    {
        private const int
StreamBufferSize = 4096;
        private const int
CryptoBufferSize = 16 * 1024 * 1024;
        private const long
UiUpdateStepBytes = 64L * 1024 * 1024;

        private Stopwatch
_operationStopwatch;
        private TimeSpan
_previousCpuTime;
        private DateTime
_previousSampleUtc;
        private double
_peakCpuUsagePercent;
        private double _peakMemoryMb;

        private long _lastUiUpdateBytes;

        private bool _wrongKeyShown;
        private bool _wrongKeyDetected;

        public AESForm()
        {
            InitializeComponent();

            txtInput.AllowDrop = true;
            txtInput.DragEnter +=
TxtInput_DragEnter;
            txtInput.DragDrop +=
TxtInput_DragDrop;
        }

        private void
btnGenerateKey_Click(object sender,
EventArgs e)
        {
```

```
            using var aes = Aes.Create();
            txtKey.Text =
Convert.ToBase64String(aes.Key);
        }

        private async void
btnEncryptFile_Click(object sender,
EventArgs e)
        {
            if (!TryReadAesKey(out byte[]
aesKey))
            {
                MessageBox.Show("Спочатку
згенеруйте або введіть ключ AES.");
                return;
            }

            string inputPath =
txtInput.Text?.Trim() ?? string.Empty;
            if
(!IsExistingFileOrDirectory(inputPath))
            {
                MessageBox.Show("Файл або
папка не знайдені.");
                return;
            }

            if
(Directory.Exists(inputPath))
            {
                string folderPath =
inputPath;
                long plainTotalBytes =
GetDirectoryTotalBytes(folderPath);

                if (plainTotalBytes == 0)
                {

MessageBox.Show("Папка порожня, немає
файлів для шифрування.");
                    return;
                }

                await
RunWithMonitoringAsync(
                    () =>
EncryptFolderAsync(folderPath, aesKey,
plainTotalBytes),
                    plainTotalBytes,
                    "шифрування папки"
                );

                string outputDir =
folderPath + "_encrypted";
                long cipherTotalBytes =
Directory.Exists(outputDir) ?
GetDirectoryTotalBytes(outputDir) : 0;

                string sizeMsg =
BuildSizeMessageForFolder(plainTotalBytes
, cipherTotalBytes);
                AppendLog(sizeMsg +
Environment.NewLine);
                MessageBox.Show(sizeMsg,
"Розмір даних (AES)");
                return;
            }
            if (File.Exists(inputPath))
            {
                string filePath =
inputPath;
                long plainBytes = new
FileInfo(filePath).Length;

                if (plainBytes == 0)
                {
                    MessageBox.Show("Файл
порожній, немає даних для шифрування.");
                    return;
                }

                string outputPath =
BuildOutputPath(filePath, "_encrypted");

                await
RunWithMonitoringAsync(
                    () =>
EncryptFileAsync(filePath, outputPath,
aesKey, plainBytes, 0),
                    plainBytes,
                    "шифрування файлу"
                );

                long cipherBytes =
File.Exists(outputPath) ? new
FileInfo(outputPath).Length : 0;

                string sizeMsg =
BuildSizeMessageForFile(plainBytes,
cipherBytes);
                AppendLog(sizeMsg +
Environment.NewLine);
                MessageBox.Show(sizeMsg,
"Розмір даних (AES)");
                return;
            }

            MessageBox.Show("Файл або
папка не знайдені.");
        }

        private async void
btnDecryptFile_Click(object sender,
EventArgs e)
        {
            _wrongKeyShown = false;
            _wrongKeyDetected = false;

            if (!TryReadAesKey(out byte[]
aesKey))
            {
                MessageBox.Show("Спочатку
введіть ключ AES для розшифрування.");
                return;
            }

            string inputPath =
txtInput.Text?.Trim() ?? string.Empty;
            if
(!IsExistingFileOrDirectory(inputPath))
            {
                MessageBox.Show("Файл або
папка не знайдені.");
```

```
                return;
            }

            if
(Directory.Exists(inputPath))
            {
                string folderPath =
inputPath;
                long totalBytes =
GetDirectoryTotalBytes(folderPath);

                if (totalBytes == 0)
                {

MessageBox.Show("Папка порожня, немає
файлів для розшифрування.");
                    return;
                }

                await
RunWithMonitoringAsync(
                    () =>
DecryptFolderAsync(folderPath, aesKey,
totalBytes),
                    totalBytes,
                    "розшифрування папки"
                );
                return;
            }

            if (File.Exists(inputPath))
            {
                string filePath =
inputPath;
                long fileBytes = new
FileInfo(filePath).Length;

                if (fileBytes == 0)
                {
                    MessageBox.Show("Файл
порожній, немає даних для
розшифрування.");
                    return;
                }

                string outputPath =
BuildOutputPath(filePath, "_decrypted");

                await
RunWithMonitoringAsync(
                    () =>
DecryptFileAsync(filePath, outputPath,
aesKey, fileBytes, 0),
                    fileBytes,
                    "розшифрування файлу"
                );
                return;
            }

            MessageBox.Show("Файл або
папка не знайдені.");
        }

        private async Task
RunWithMonitoringAsync(Func<Task<long>>
operation, long totalBytes, string
operationName)
        {
            Ui(() => {
                txtLog.Clear();
                _lastUiUpdateBytes = 0;
                lblSpeed.Text =
string.Empty;
                lblProgress.Text = "0%";
                progressBar.Value = 0;
            });

            _peakCpuUsagePercent = 0.0;
            _peakMemoryMb = 0.0;

            using var proc =
Process.GetCurrentProcess();
            proc.Refresh();

            _previousCpuTime =
proc.TotalProcessorTime;
            _previousSampleUtc =
DateTime.UtcNow;

            _operationStopwatch =
Stopwatch.StartNew();

            long processedBytes;
            try
            {
                processedBytes = await
operation.Invoke();

                if (_wrongKeyDetected)
                {

_operationStopwatch.Stop();
                    Ui(() =>
lblSpeed.Text = string.Empty);
                    return;
                }
            }
            catch (Exception ex)
            {

_operationStopwatch.Stop();
                AppendLog("Помилка: " +
ex.Message + Environment.NewLine);

MessageBox.Show(ex.Message);
                Ui(() => lblSpeed.Text =
string.Empty);
                return;
            }

            _operationStopwatch.Stop();
            proc.Refresh();
            SamplePerformance(proc);

            double elapsedSeconds =
_operationStopwatch.Elapsed.TotalSeconds;
            if (elapsedSeconds <= 0)
            {
                elapsedSeconds = 0.0001;
            }

            double processedMb =
processedBytes / 1024.0 / 1024.0;
            double avgSpeedMbPerSec =
```

```
processedMb / elapsedSeconds;

            string resultMsg =
string.Format(
                "Операція: {0}\n" +
                "Час виконання: {1:F2}
c\n" +
                "Обсяг оброблених даних:
{2} байт (~{3:F2} МБ)\n" +
                "Пікове завантаження CPU
= {4:F1}%, RAM = {5:F1} МБ\n" +
                "Середня швидкість:
{6:F2} МБ/с.",
                operationName,
                elapsedSeconds,
                processedBytes,
                processedMb,
                _peakCpuUsagePercent,
                _peakMemoryMb,
                avgSpeedMbPerSec
            );

            AppendLog(resultMsg +
Environment.NewLine);
            Ui(() => lblSpeed.Text =
$"Середня швидкість:
{avgSpeedMbPerSec:F2} МБ/с");
            MessageBox.Show(resultMsg,
"Статистика AES");
        }

        private async Task<long>
EncryptFileAsync(
            string inputPath,
            string outputPath,
            byte[] key,
            long totalBytes,
            long offset
        )
        {
            using var aes = Aes.Create();
            aes.Key = key;
            aes.Mode = CipherMode.CBC;
            aes.Padding =
PaddingMode.PKCS7;
            aes.GenerateIV();
            byte[] iv = aes.IV;

            using var inputStream = new
FileStream(inputPath, FileMode.Open,
FileAccess.Read, FileShare.Read,
StreamBufferSize, true);
            using var outputStream = new
FileStream(outputPath, FileMode.Create,
FileAccess.Write, FileShare.None,
StreamBufferSize, true);

            await
outputStream.WriteAsync(iv, 0,
iv.Length);
            using var cryptoStream = new
CryptoStream(outputStream,
aes.CreateEncryptor(),
CryptoStreamMode.Write);

            byte[] buffer = new
byte[CryptoBufferSize];
            long processedBytes = 0;

            int read;
            var watch =
Stopwatch.StartNew();

            while ((read = await
inputStream.ReadAsync(buffer, 0,
buffer.Length)) > 0)
            {
                await
cryptoStream.WriteAsync(buffer, 0, read);
                processedBytes += read;

UpdateProgress(totalBytes, offset +
processedBytes,
watch.Elapsed.TotalSeconds);
            }

            await
cryptoStream.FlushAsync();
            return processedBytes;
        }

        private async Task<long>
DecryptFileAsync(
            string inputPath,
            string outputPath,
            byte[] key,
            long totalBytes,
            long offset
        )
        {
            try
            {
                using var inputStream =
new FileStream(inputPath, FileMode.Open,
FileAccess.Read, FileShare.Read,
StreamBufferSize, true);

                byte[] iv = new byte[16];
                int ivRead = 0;

                while (ivRead <
iv.Length)
                {
                    int chunk = await
inputStream.ReadAsync(iv, ivRead,
iv.Length - ivRead);
                    if (chunk <= 0)
                    {
                        break;
                    }
                    ivRead += chunk;
                }

                if (ivRead != iv.Length)
                {
                    throw new
InvalidDataException("Некоректний формат
файлу (IV не знайдено).");
                }

                using var aes =
Aes.Create();
                aes.Key = key;
                aes.IV = iv;
```

```
                aes.Mode =
CipherMode.CBC;
                aes.Padding =
PaddingMode.PKCS7;

                using var cryptoStream =
new CryptoStream(inputStream,
aes.CreateDecryptor(),
CryptoStreamMode.Read);
                using var outputStream =
new FileStream(outputPath,
FileMode.Create, FileAccess.Write,
FileShare.None, StreamBufferSize, true);

                byte[] buffer = new
byte[CryptoBufferSize];
                long processedBytes = 0;

                int read;
                var watch =
Stopwatch.StartNew();

                while ((read = await
cryptoStream.ReadAsync(buffer, 0,
buffer.Length)) > 0)
                {
                    await
outputStream.WriteAsync(buffer, 0, read);
                    processedBytes +=
read;

UpdateProgress(totalBytes, offset +
processedBytes,
watch.Elapsed.TotalSeconds);
                }

                await
outputStream.FlushAsync();
                return processedBytes;
            }
            catch
(CryptographicException)
            {
                _wrongKeyDetected = true;


TryDeleteFile(outputPath);

                if (!_wrongKeyShown)
                {
                    _wrongKeyShown =
true;

MessageBox.Show("Невірний ключ");
                }

                AppendLog("Невірний ключ"
+ Environment.NewLine);
                return 0;
            }
        }

        private async Task<long>
EncryptFolderAsync(string folderPath,
byte[] key, long totalBytes)
        {
            string outDir = folderPath +
"_encrypted";

Directory.CreateDirectory(outDir);

            long processedBytes = 0;

            foreach (var file in
Directory.GetFiles(folderPath))
            {
                string outFile =
Path.Combine(
                    outDir,

Path.GetFileNameWithoutExtension(file) +
"_encrypted" + Path.GetExtension(file)
                );

                processedBytes += await
EncryptFileAsync(file, outFile, key,
totalBytes, processedBytes);
            }

            return processedBytes;
        }

        private async Task<long>
DecryptFolderAsync(string folderPath,
byte[] key, long totalBytes)
        {
            string outDir = folderPath +
"_decrypted";

Directory.CreateDirectory(outDir);

            long processedBytes = 0;

            foreach (var file in
Directory.GetFiles(folderPath))
            {
                if (_wrongKeyDetected)
                {
                    break;
                }

                string outFile =
Path.Combine(
                    outDir,

Path.GetFileNameWithoutExtension(file) +
"_decrypted" + Path.GetExtension(file)
                );

                processedBytes += await
DecryptFileAsync(file, outFile, key,
totalBytes, processedBytes);
            }

            return processedBytes;
        }

        private void UpdateProgress(long
totalBytes, long currentBytes, double
elapsedSeconds)
        {
            if (totalBytes <= 0)
            {
                return;
```

```
            }

            if (currentBytes -
_lastUiUpdateBytes < UiUpdateStepBytes &&
currentBytes < totalBytes)
            {
                return;
            }

            double percent =
(double)currentBytes / totalBytes *
100.0;
            if (percent < 0)
            {
                percent = 0;
            }
            if (percent > 100)
            {
                percent = 100;
            }

            double speed = elapsedSeconds
> 0
                ? (currentBytes / 1024.0
/ 1024.0) / elapsedSeconds
                : 0.0;

            Ui(() => {
                progressBar.Value =
(int)percent;
                lblProgress.Text =
$"{(int)percent}%";
                lblSpeed.Text = $"Поточна
швидкість: {speed:F2} МБ/с";
            });

            _lastUiUpdateBytes =
currentBytes;
        }

        private void
SamplePerformance(Process proc)
        {
            var nowUtc = DateTime.UtcNow;
            var cpuNow =
proc.TotalProcessorTime;

            double wallMs = (nowUtc -
_previousSampleUtc).TotalMilliseconds;
            if (wallMs > 1)
            {
                double cpuMs = (cpuNow -
_previousCpuTime).TotalMilliseconds;
                double cpuPercent = cpuMs
/ (Environment.ProcessorCount * wallMs) *
100.0;

                if
(!double.IsNaN(cpuPercent) &&
!double.IsInfinity(cpuPercent))
                {
                    _peakCpuUsagePercent
= Math.Max(_peakCpuUsagePercent,
cpuPercent);
                }
            }
            double memMb =
proc.WorkingSet64 / (1024.0 * 1024.0);
            _peakMemoryMb =
Math.Max(_peakMemoryMb, memMb);

            _previousCpuTime = cpuNow;
            _previousSampleUtc = nowUtc;
        }

        private long
GetDirectoryTotalBytes(string folderPath)
        {
            long total = 0;
            foreach (var file in
Directory.GetFiles(folderPath))
            {
                total += new
FileInfo(file).Length;
            }
            return total;
        }

        private void
TxtInput_DragEnter(object sender,
DragEventArgs e)
        {
            if
(e.Data.GetDataPresent(DataFormats.FileDr
op))
            {
                e.Effect =
DragDropEffects.Copy;
            }
        }

        private void
TxtInput_DragDrop(object sender,
DragEventArgs e)
        {
            var files =
(string[])e.Data.GetData(DataFormats.File
Drop);
            if (files.Length > 0)
            {
                txtInput.Text = files[0];
            }
        }

        private bool TryReadAesKey(out
byte[] key)
        {
            key = null;

            string keyText =
txtKey.Text?.Trim() ?? string.Empty;
            if
(string.IsNullOrWhiteSpace(keyText))
            {
                return false;
            }

            try
            {
                key =
Convert.FromBase64String(keyText);
                return key.Length == 16
|| key.Length == 24 || key.Length == 32;
```

```
            }
            catch
            {
                return false;
            }
        }

        private static bool
IsExistingFileOrDirectory(string path)
        {
            if
(string.IsNullOrWhiteSpace(path))
            {
                return false;
            }
            return File.Exists(path) ||
Directory.Exists(path);
        }

        private static string
BuildOutputPath(string inputPath, string
suffix)
        {
            return Path.Combine(

Path.GetDirectoryName(inputPath) ??
string.Empty,

Path.GetFileNameWithoutExtension(inputPat
h) + suffix +
Path.GetExtension(inputPath)
            );
        }

        private static string
BuildSizeMessageForFolder(long
plainBytes, long cipherBytes)
        {
            double plainMb = plainBytes /
1024.0 / 1024.0;
            double cipherMb = cipherBytes
/ 1024.0 / 1024.0;
            double expansion = plainBytes
> 0 ? (double)cipherBytes / plainBytes :
0.0;

            return
                $"Початковий розмір (усі
файли): {plainBytes} байт (~{plainMb:F2}
МБ)\n" +
                $"Розмір після
шифрування:      {cipherBytes} байт
(~{cipherMb:F2} МБ)\n" +
                $"Коефіцієнт розширення:
{expansion:F3} раз(и).";
        }

        private static string
BuildSizeMessageForFile(long plainBytes,
long cipherBytes)
        {
            double plainMb = plainBytes /
1024.0 / 1024.0;
            double cipherMb = cipherBytes
/ 1024.0 / 1024.0;
            double expansion = plainBytes
> 0 ? (double)cipherBytes / plainBytes :
0.0;

            return
                $"Початковий розмір
файлу: {plainBytes} байт (~{plainMb:F2}
МБ)\n" +
                $"Розмір після
шифрування: {cipherBytes} байт
(~{cipherMb:F2} МБ)\n" +
                $"Коефіцієнт розширення:
{expansion:F3} раз(и).";
        }

        private void AppendLog(string
text)
        {
            Ui(() =>
txtLog.AppendText(text));
        }

        private void Ui(Action action)
        {
            if (IsDisposed)
            {
                return;
            }
            if (InvokeRequired)
            {
                Invoke(action);
            }
            else
            {
                action();
            }
        }

        private void TryDeleteFile(string
path)
        {
            try
            {
                if (File.Exists(path))
                {
                    File.Delete(path);
                }
            }
            catch { }
        }
    }
}
```

2.4 BlowfishForm.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System.Windows.Forms;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Engines;
```

```
using Org.BouncyCastle.Crypto.Modes;
using Org.BouncyCastle.Crypto.Paddings;
using Org.BouncyCastle.Crypto.Parameters;

namespace CruptoBench
{
    public partial class BlowfishForm :
Form
    {
        private const int IoBufferSize =
8192;

        private static readonly byte[]
MagicHeader = new byte[] {
            0x43, 0x42, 0x45, 0x4E, 0x43,
0x48, 0x2D, 0x42, 0x46, 0x2D, 0x31
        };

        private byte[] _encryptionKey;

        private Stopwatch
_operationStopwatch;
        private TimeSpan
_previousCpuTime;
        private DateTime
_previousSampleUtc;
        private double
_peakCpuUsagePercent;
        private double _peakMemoryMb;

        private long _totalInputBytes;
        private long _totalOutputBytes;

        private bool _wrongKeyShown;
        private bool _wrongKeyDetected;

        private string
_decryptOutputFolderPath;

        public BlowfishForm()
        {
            InitializeComponent();

            txtPath.AllowDrop = true;
            txtPath.DragEnter +=
TxtPath_DragEnter;
            txtPath.DragDrop +=
TxtPath_DragDrop;
        }

        private void
TxtPath_DragEnter(object sender,
DragEventArgs e)
        {
            if
(e.Data.GetDataPresent(DataFormats.FileDr
op))
            {
                e.Effect =
DragDropEffects.Copy;
            }
        }

        private void
TxtPath_DragDrop(object sender,
DragEventArgs e)
        {
            var droppedItems =
e.Data.GetData(DataFormats.FileDrop) as
string[];
            if (droppedItems is { Length:
> 0 })
            {
                txtPath.Text =
droppedItems[0];
            }
        }

        private void
btnGenerateKey_Click(object sender,
EventArgs e)
        {
            _encryptionKey = new
byte[32];
            using var rng =
RandomNumberGenerator.Create();
            rng.GetBytes(_encryptionKey);

            txtKey.Text =
Convert.ToBase64String(_encryptionKey);
        }

        private async void
btnEncrypt_Click(object sender, EventArgs
e)
        {
            await
RunWithMonitoringAsync(encrypt: true);
        }

        private async void
btnDecrypt_Click(object sender, EventArgs
e)
        {
            await
RunWithMonitoringAsync(encrypt: false);
        }

        private async Task
RunWithMonitoringAsync(bool encrypt)
        {
            _wrongKeyShown = false;
            _wrongKeyDetected = false;
            _decryptOutputFolderPath =
null;

            string inputPath =
txtPath.Text?.Trim() ?? string.Empty;
            if
(!IsExistingFileOrDirectory(inputPath))
            {
                MessageBox.Show("Выберите
файл или папку.");
                return;
            }

            if (!TryReadBlowfishKey(out
_encryptionKey))
            {
                MessageBox.Show("Неверный
формат ключа.");
                return;
            }
```

```
            Ui(() => {
                txtLog.Clear();
                progressBar.Value = 0;
                lblProgress.Text = "0%";
            });

            ResetStats();

            using var currentProcess =
Process.GetCurrentProcess();
            currentProcess.Refresh();

            _previousCpuTime =
currentProcess.TotalProcessorTime;
            _previousSampleUtc =
DateTime.UtcNow;

            _operationStopwatch =
Stopwatch.StartNew();

            try
            {
                if
(Directory.Exists(inputPath))
                {
                    await
ProcessDirectoryAsync(inputPath, encrypt,
currentProcess);

                    if (_wrongKeyDetected
&& !encrypt)
                    {

TryDeleteDirectoryRecursive(_decryptOutpu
tFolderPath);
                        return;
                    }
                }
                else
                {
                    await
ProcessSingleFileAsync(inputPath,
encrypt, currentProcess);

                    if (_wrongKeyDetected
&& !encrypt)
                    {
                        return;
                    }
                }


_operationStopwatch.Stop();

SamplePerformance(currentProcess);

                if (_wrongKeyDetected &&
!encrypt)
                {

TryDeleteDirectoryRecursive(_decryptOutpu
tFolderPath);
                    return;
                }

                ShowFinalResult();
            }
            catch (Exception ex)
            {

_operationStopwatch?.Stop();
                AppendLog($"Ошибка:
{ex.Message}{Environment.NewLine}");

MessageBox.Show(ex.Message);
            }
        }

        private async Task
ProcessDirectoryAsync(string folderPath,
bool encrypt, Process currentProcess)
        {
            string suffix = encrypt ?
"_blowfish_encrypted" :
"_blowfish_decrypted";
            string outputFolderPath =
folderPath + suffix;

            if (!encrypt)
            {
                _decryptOutputFolderPath
= outputFolderPath;
            }


Directory.CreateDirectory(outputFolderPat
h);

            string[] inputFiles =
Directory.GetFiles(folderPath);
            if (inputFiles.Length == 0)
            {
                MessageBox.Show("В
выбранной папке нет файлов.");
                return;
            }

            Ui(() => {
                progressBar.Minimum = 0;
                progressBar.Maximum =
inputFiles.Length;
                progressBar.Value = 0;
                lblProgress.Text = "0%";
            });

            for (int fileIndex = 0;
fileIndex < inputFiles.Length;
fileIndex++)
            {
                if (_wrongKeyDetected &&
!encrypt)
                {
                    break;
                }

                string inputFilePath =
inputFiles[fileIndex];
                string outputFilePath =
Path.Combine(outputFolderPath,
Path.GetFileName(inputFilePath));

                await
ProcessOneFileAsync(inputFilePath,
outputFilePath, encrypt, currentProcess);
```

```
                if (_wrongKeyDetected &&
!encrypt)
                {
                    break;
                }

                UpdateProgress(fileIndex
+ 1, inputFiles.Length);
            }
        }

        private async Task
ProcessSingleFileAsync(string
inputFilePath, bool encrypt, Process
currentProcess)
        {
            string outputFilePath =
Path.Combine(

Path.GetDirectoryName(inputFilePath) ??
string.Empty,

Path.GetFileNameWithoutExtension(inputFil
ePath) +
                (encrypt ?
"_blowfish_encrypted" :
"_blowfish_decrypted") +

Path.GetExtension(inputFilePath)
            );

            Ui(() => {
                progressBar.Minimum = 0;
                progressBar.Maximum = 1;
                progressBar.Value = 0;
                lblProgress.Text = "0%";
            });

            await
ProcessOneFileAsync(inputFilePath,
outputFilePath, encrypt, currentProcess);

            if (_wrongKeyDetected &&
!encrypt)
            {
                return;
            }

            UpdateProgress(1, 1);
        }

        private async Task
ProcessOneFileAsync(string inputFilePath,
string outputFilePath, bool encrypt,
Process currentProcess)
        {
            var fileStopwatch =
Stopwatch.StartNew();

            try
            {
                long inputBytes = new
FileInfo(inputFilePath).Length;

                var result = await
Task.Run(() => {
                    bool wk;
                    string err;
                    bool ok =
TryProcessFileBlowfish(inputFilePath,
outputFilePath, encrypt, out wk, out
err);
                    return (Ok: ok,
WrongKey: wk, ErrorText: err);
                });

                if (!result.Ok)
                {
                    if (!encrypt &&
result.WrongKey)
                    {
                        _wrongKeyDetected
= true;


AppendLog("Неверный ключ" +
Environment.NewLine);

                        if
(!_wrongKeyShown)
                        {

_wrongKeyShown = true;

MessageBox.Show("Неверный ключ");
                        }

                        return;
                    }

                    AppendLog($"Файл:
{Path.GetFileName(inputFilePath)} |
Ошибка:
{result.ErrorText}{Environment.NewLine}")
;
                    return;
                }

                long outputBytes = new
FileInfo(outputFilePath).Length;

                _totalInputBytes +=
inputBytes;
                _totalOutputBytes +=
outputBytes;

                AppendLog(
                    $"Файл:
{Path.GetFileName(inputFilePath)} ->
{Path.GetFileName(outputFilePath)} | " +
                    $"Вход: {inputBytes /
1024.0 / 1024.0:F2} МБ | " +
                    $"Выход: {outputBytes
/ 1024.0 / 1024.0:F2} МБ | " +
                    $"Время:
{fileStopwatch.Elapsed.TotalSeconds:F2}
c{Environment.NewLine}"
                );
            }
            finally
            {
                fileStopwatch.Stop();
```

```
SamplePerformance(currentProcess);
            }
        }

        private bool
TryProcessFileBlowfish(
            string inputFilePath,
            string outputFilePath,
            bool encrypt,
            out bool wrongKey,
            out string errorText
        )
        {
            wrongKey = false;
            errorText = string.Empty;

            try
            {
                using var inputStream =
new FileStream(
                    inputFilePath,
                    FileMode.Open,
                    FileAccess.Read,
                    FileShare.Read,
                    IoBufferSize,

FileOptions.SequentialScan
                );

                using var outputStream =
new FileStream(
                    outputFilePath,
                    FileMode.Create,
                    FileAccess.Write,
                    FileShare.None,
                    IoBufferSize,

FileOptions.SequentialScan
                );

                IBlockCipher engine = new
BlowfishEngine();
                int blockSize =
engine.GetBlockSize();
                byte[] iv = new
byte[blockSize];

                if (encrypt)
                {
                    using var rng =
RandomNumberGenerator.Create();
                    rng.GetBytes(iv);

outputStream.Write(iv, 0, iv.Length);
                }
                else
                {
                    int readIv =
ReadExactly(inputStream, iv, 0,
iv.Length);
                    if (readIv !=
iv.Length)
                    {
                        errorText = "Файл
повреждён: не удалось прочитать IV.";

TryDeleteFile(outputFilePath);
                        return false;
                    }
                }

                var cipher = new
PaddedBufferedBlockCipher(new
CbcBlockCipher(engine));
                cipher.Init(encrypt, new
ParametersWithIV(new
KeyParameter(_encryptionKey), iv));

                byte[] inputBuffer = new
byte[IoBufferSize];
                byte[] outputBuffer = new
byte[cipher.GetOutputSize(inputBuffer.Len
gth)];

                if (encrypt)
                {
                    int headerOutLen =
cipher.ProcessBytes(MagicHeader, 0,
MagicHeader.Length, outputBuffer, 0);
                    if (headerOutLen > 0)
                    {

outputStream.Write(outputBuffer, 0,
headerOutLen);
                    }

                    int bytesReadEnc;
                    while ((bytesReadEnc
= inputStream.Read(inputBuffer, 0,
inputBuffer.Length)) > 0)
                    {
                        int outLenEnc =
cipher.ProcessBytes(inputBuffer, 0,
bytesReadEnc, outputBuffer, 0);
                        if (outLenEnc >
0)
                        {

outputStream.Write(outputBuffer, 0,
outLenEnc);
                        }
                    }

                    int finalLenEnc =
cipher.DoFinal(outputBuffer, 0);
                    if (finalLenEnc > 0)
                    {

outputStream.Write(outputBuffer, 0,
finalLenEnc);
                    }

                    outputStream.Flush();
                    return true;
                }

                int headerMatchedBytes =
0;

                int bytesRead;
                while ((bytesRead =
inputStream.Read(inputBuffer, 0,
inputBuffer.Length)) > 0)
                {
```

```
                    int outLen =
cipher.ProcessBytes(inputBuffer, 0,
bytesRead, outputBuffer, 0);
                    if (outLen > 0)
                    {
                        if
(!TryConsumeHeaderAndWrite(outputStream,
outputBuffer, outLen, ref
headerMatchedBytes))
                        {
                            wrongKey =
true;
                            errorText =
"Неверный ключ";

TryDeleteFile(outputFilePath);
                            return false;
                        }
                    }
                }

                int finalLen;
                try
                {
                    finalLen =
cipher.DoFinal(outputBuffer, 0);
                }
                catch (CryptoException)
                {
                    wrongKey = true;
                    errorText = "Неверный
ключ";

TryDeleteFile(outputFilePath);
                    return false;
                }

                if (finalLen > 0)
                {
                    if
(!TryConsumeHeaderAndWrite(outputStream,
outputBuffer, finalLen, ref
headerMatchedBytes))
                    {
                        wrongKey = true;
                        errorText =
"Неверный ключ";

TryDeleteFile(outputFilePath);
                        return false;
                    }
                }

                if (headerMatchedBytes <
MagicHeader.Length)
                {
                    wrongKey = true;
                    errorText = "Неверный
ключ";

TryDeleteFile(outputFilePath);
                    return false;
                }

                outputStream.Flush();
                return true;
            }
            catch (Exception ex)
            {
                errorText = ex.Message;

TryDeleteFile(outputFilePath);
                return false;
            }
        }

        private static bool
TryConsumeHeaderAndWrite(
            Stream outputStream,
            byte[] buffer,
            int length,
            ref int headerMatchedBytes
        )
        {
            int offset = 0;

            if (headerMatchedBytes <
MagicHeader.Length)
            {
                int need =
MagicHeader.Length - headerMatchedBytes;
                int take = Math.Min(need,
length);

                for (int i = 0; i < take;
i++)
                {
                    if (buffer[offset +
i] != MagicHeader[headerMatchedBytes +
i])
                    {
                        return false;
                    }
                }

                headerMatchedBytes +=
take;
                offset += take;
                length -= take;

                if (length <= 0)
                {
                    return true;
                }
            }

            outputStream.Write(buffer,
offset, length);
            return true;
        }

        private static int
ReadExactly(Stream stream, byte[] buffer,
int offset, int count)
        {
            int totalRead = 0;
            while (totalRead < count)
            {
                int read =
stream.Read(buffer, offset + totalRead,
count - totalRead);
                if (read <= 0)
                {
                    break;
```

```
            }
            totalRead += read;
        }
        return totalRead;
    }

    private void ResetStats()
    {
        _totalInputBytes = 0;
        _totalOutputBytes = 0;
        _peakCpuUsagePercent = 0.0;
        _peakMemoryMb = 0.0;
        _operationStopwatch = null;
    }

    private void
SamplePerformance(Process currentProcess)
    {
        currentProcess.Refresh();

        var sampleUtc =
DateTime.UtcNow;
        var cpuTimeNow =
currentProcess.TotalProcessorTime;

        double wallTimeMs =
(sampleUtc -
_previousSampleUtc).TotalMilliseconds;
        if (wallTimeMs > 1)
        {
            double cpuTimeMs =
(cpuTimeNow -
_previousCpuTime).TotalMilliseconds;
            double cpuUsagePercent =
cpuTimeMs / (Environment.ProcessorCount *
wallTimeMs) * 100.0;

            if
(!double.IsNaN(cpuUsagePercent) &&
!double.IsInfinity(cpuUsagePercent))
            {
                _peakCpuUsagePercent
= Math.Max(_peakCpuUsagePercent,
cpuUsagePercent);
            }
        }

        double memoryMb =
currentProcess.WorkingSet64 / (1024.0 *
1024.0);
        _peakMemoryMb =
Math.Max(_peakMemoryMb, memoryMb);

        _previousCpuTime =
cpuTimeNow;
        _previousSampleUtc =
sampleUtc;
    }

    private void ShowFinalResult()
    {
        double elapsedSeconds =
_operationStopwatch?.Elapsed.TotalSeconds
?? 0.0;
        if (elapsedSeconds <= 0)
        {
            elapsedSeconds = 0.0001;
        }

        double inputMb =
_totalInputBytes / 1024.0 / 1024.0;
        double outputMb =
_totalOutputBytes / 1024.0 / 1024.0;
        double avgSpeedMbPerSec =
inputMb / elapsedSeconds;

        double deltaMb = outputMb -
inputMb;
        double deltaPercent = inputMb
> 0 ? (outputMb / inputMb - 1.0) * 100.0
: 0.0;

        string sizeInfo =
            $"Исходный объём данных:
{inputMb:F2} МБ\n" +
            $"Объём после обработки:
{outputMb:F2} МБ\n" +
            $"Изменение размера:
{deltaMb:+0.00;-0.00;0.00} МБ
({deltaPercent:+0.00;-0.00;0.00}%)";

        string resultMsg =
string.Format(
            "Операция завершена за
{0:F2} с.\n" +
            "Пиковая загрузка CPU =
{1:F1}%, RAM = {2:F1} МВ\n" +
            "Средняя скорость: {3:F2}
МБ/с\n\n{4}",
            elapsedSeconds,
_peakCpuUsagePercent, _peakMemoryMb,
avgSpeedMbPerSec, sizeInfo
        );

        AppendLog(resultMsg +
Environment.NewLine +
Environment.NewLine);
        MessageBox.Show(resultMsg);
    }

    private bool
TryReadBlowfishKey(out byte[] decodedKey)
    {
        decodedKey = null;

        try
        {
            byte[] rawKey =
Convert.FromBase64String(txtKey.Text?.Tri
m() ?? string.Empty);
            if (rawKey.Length < 4 ||
rawKey.Length > 56)
            {
                return false;
            }

            decodedKey = rawKey;
            return true;
        }
        catch
        {
            return false;
        }
    }
```

```
        private static bool
IsExistingFileOrDirectory(string path)
        {
            if
(string.IsNullOrWhiteSpace(path))
            {
                return false;
            }
            return File.Exists(path) ||
Directory.Exists(path);
        }

        private void UpdateProgress(int
processedCount, int totalCount)
        {
            Ui(() => {
                if (totalCount <= 0)
                {
                    totalCount = 1;
                }

                progressBar.Maximum =
totalCount;
                progressBar.Value =
Math.Min(Math.Max(processedCount,
progressBar.Minimum),
progressBar.Maximum);

                int percent =
(int)Math.Round((double)progressBar.Value
/ totalCount * 100.0);
                lblProgress.Text =
$"{percent}%";
            });
        }

        private void AppendLog(string
text)
        {
            Ui(() =>
txtLog.AppendText(text));
        }

        private void Ui(Action action)
        {
            if (IsDisposed)
            {
                return;
            }
            if (InvokeRequired)
            {
                Invoke(action);
            }
            else
            {
                action();
            }
        }

        private void TryDeleteFile(string
path)
        {
            try
            {
                if (File.Exists(path))
                {
                    File.Delete(path);
                }
            }
            catch { }
        }

        private void
TryDeleteDirectoryRecursive(string path)
        {
            if
(string.IsNullOrWhiteSpace(path))
            {
                return;
            }

            try
            {
                if
(Directory.Exists(path))
                {

Directory.Delete(path, recursive: true);
                }
            }
            catch { }
        }
    }
}
```

2.5 ChaCha20Form.cs

```
using Org.BouncyCastle.Crypto.Engines;
using Org.BouncyCastle.Crypto.Parameters;
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CruptoBench
{
    public partial class ChaCha20Form :
Form
    {
        private byte[] _key32;
        private byte[] _nonce12;

        private Stopwatch _stopwatch;
        private TimeSpan _startCpuTime;
        private DateTime _startWallUtc;
        private double _peakCpuPercent;
        private double _peakMemoryMb;
        private long _totalInputBytes;

        private static readonly byte[]
Magic = { (byte)'C', (byte)'C',
(byte)'2', (byte)'0' };
        private const byte Version = 1;
```

```
        public ChaCha20Form()
        {
            InitializeComponent();

            txtFolder.AllowDrop = true;
            txtFolder.DragEnter +=
TxtFolder_DragEnter;
            txtFolder.DragDrop +=
TxtFolder_DragDrop;
        }

        private void
TxtFolder_DragEnter(object sender,
DragEventArgs e)
        {
            if (e.Data != null &&
e.Data.GetDataPresent(DataFormats.FileDro
p))
                e.Effect =
DragDropEffects.Copy;
        }

        private void
TxtFolder_DragDrop(object sender,
DragEventArgs e)
        {
            var items =
(string[])e.Data.GetData(DataFormats.File
Drop);
            if (items != null &&
items.Length > 0)
                txtFolder.Text =
items[0];
        }

        private void
btnGenerateKey_Click(object sender,
EventArgs e)
        {
            _key32 = new byte[32];
            _nonce12 = new byte[12];

            using (var rng =
RandomNumberGenerator.Create())
            {
                rng.GetBytes(_key32);
                rng.GetBytes(_nonce12);
            }

            txtKey.Text =
Convert.ToBase64String(_key32);
            txtNonce.Text =
Convert.ToBase64String(_nonce12);
        }

        private async void
btnEncrypt_Click(object sender, EventArgs
e)
        {
            await
RunOperationAsync(true);
        }

        private async void
btnDecrypt_Click(object sender, EventArgs
e)
        {
            await
RunOperationAsync(false);
        }

        private async Task
RunOperationAsync(bool isEncrypt)
        {
            string inputPath =
txtFolder.Text;

            if
(string.IsNullOrWhiteSpace(inputPath) ||

(!Directory.Exists(inputPath) &&
!File.Exists(inputPath)))
            {
                MessageBox.Show("Выберите
файл или папку.");
                return;
            }

            if (!TryReadKeyAndNonce(out
var key32, out var nonce12, out var
error))
            {
                MessageBox.Show(error);
                return;
            }

            _key32 = key32;
            _nonce12 = nonce12;

            _stopwatch =
Stopwatch.StartNew();
            _totalInputBytes = 0;
            _peakCpuPercent = 0.0;
            _peakMemoryMb = 0.0;

            var process =
Process.GetCurrentProcess();
            process.Refresh();
            _startCpuTime =
process.TotalProcessorTime;
            _startWallUtc =
DateTime.UtcNow;

            SafeUi(() =>
            {
                progressBar.Minimum = 0;
                progressBar.Maximum =
100;
                progressBar.Value = 0;
                lblProgress.Text = "0%";
            });

            long inputTotalBytes = 0;
            long outputTotalBytes = 0;

            try
            {
                if
(Directory.Exists(inputPath))
                {
                    string outputRoot =
GetOutputFolderPath(inputPath,
isEncrypt);
```

```
Directory.CreateDirectory(outputRoot);

                    var files =
Directory.GetFiles(inputPath, "*",
SearchOption.AllDirectories);

                    foreach (var f in
files)
                        inputTotalBytes
+= new FileInfo(f).Length;

                    long processedBytes =
0;

                    foreach (string file
in files)
                    {
                        string relative =
GetRelativePathCompat(inputPath, file);
                        string outFile =
Path.Combine(outputRoot, relative);

                        string outDir =
Path.GetDirectoryName(outFile);
                        if
(!string.IsNullOrEmpty(outDir))

Directory.CreateDirectory(outDir);

                        long inSize = new
FileInfo(file).Length;

                        bool ok = await
Task.Run(() => ProcessFileWithAuth(file,
outFile, isEncrypt));
                        if (!ok &&
!isEncrypt)
                        {

SafeDeleteFile(outFile);

MessageBox.Show("Неверний ключ");
                            return;
                        }

                        long outSize =
new FileInfo(outFile).Length;

                        processedBytes +=
inSize;
                        _totalInputBytes
+= inSize;
                        outputTotalBytes
+= outSize;


UpdateProgress(processedBytes,
inputTotalBytes);
                    }
                }
                else
                {
                    string outFile =
GetOutputFilePath(inputPath, isEncrypt);

                    long inSize = new
FileInfo(inputPath).Length;
                    inputTotalBytes =
inSize;

                    bool ok = await
Task.Run(() =>
ProcessFileWithAuth(inputPath, outFile,
isEncrypt));
                    if (!ok &&
!isEncrypt)
                    {

SafeDeleteFile(outFile);

MessageBox.Show("Неверний ключ");
                        return;
                    }

                    long outSize = new
FileInfo(outFile).Length;
                    outputTotalBytes =
outSize;

                    _totalInputBytes =
inSize;

UpdateProgress(inSize, inSize);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Ошибка:
" + ex.Message);
                return;
            }
            finally
            {
                _stopwatch.Stop();
            }

            process.Refresh();

            double elapsedSeconds =
_stopwatch.Elapsed.TotalSeconds;
            double avgSpeedMbSec =
elapsedSeconds > 0
                ? (_totalInputBytes /
1024.0 / 1024.0) / elapsedSeconds
                : 0.0;

            double cpuPercent =
(process.TotalProcessorTime -
_startCpuTime).TotalMilliseconds /

(Environment.ProcessorCount *
(DateTime.UtcNow -
_startWallUtc).TotalMilliseconds) *
100.0;

            _peakCpuPercent =
Math.Max(_peakCpuPercent, cpuPercent);
            double memMb =
process.WorkingSet64 / (1024.0 * 1024.0);
            _peakMemoryMb =
Math.Max(_peakMemoryMb, memMb);

            double inputMb =
inputTotalBytes / (1024.0 * 1024.0);
```

```
            double outputMb =
outputTotalBytes / (1024.0 * 1024.0);
            double deltaMb = outputMb -
inputMb;
            double ratio =
inputTotalBytes > 0 ?
(double)outputTotalBytes /
inputTotalBytes : 0.0;

            string resultMessage =
string.Format(
                "Операция завершена за
{0:F2} с.\n" +
                "Пиковая загрузка CPU =
{1:F1}%, RAM = {2:F1} МВ\n" +
                "Средняя скорость: {3:F2}
МБ/с\n\n" +
                "Исходный объём данных:
{4:F2} МБ\n" +
                "После операции:
{5:F2} МБ\n" +
                "Изменение размера:
{6:+0.00;-0.00;0.00} МБ (коэфф.
{7:F2}×)",
                elapsedSeconds,
_peakCpuPercent, _peakMemoryMb,
avgSpeedMbSec,
                inputMb, outputMb,
deltaMb, ratio);

            AppendLog(resultMessage);

MessageBox.Show(resultMessage);
        }

        private bool
TryReadKeyAndNonce(out byte[] key32, out
byte[] nonce12, out string error)
        {
            key32 = null;
            nonce12 = null;
            error = null;

            try
            {
                key32 =
Convert.FromBase64String(txtKey.Text.Trim
());
                nonce12 =
Convert.FromBase64String(txtNonce.Text.Tr
im());
            }
            catch
            {
                error = "Неверный формат
ключа или nonce (ожидается Base64).";
                return false;
            }

            if (key32.Length != 32)
            {
                error = "Неверная длина
ключа. Нужен ключ 32 байта (Base64).";
                return false;
            }

            if (nonce12.Length != 12)
            {
                error = "Неверная длина
nonce. Нужен nonce 12 байт (Base64).";
                return false;
            }

            return true;
        }

        private static string
GetOutputFolderPath(string inputFolder,
bool isEncrypt)
        {
            string trimmed =
inputFolder.TrimEnd(Path.DirectorySeparat
orChar, Path.AltDirectorySeparatorChar);
            return trimmed + (isEncrypt ?
"_encrypted" : "_decrypted");
        }

        private static string
GetOutputFilePath(string inputFile, bool
isEncrypt)
        {
            string dir =
Path.GetDirectoryName(inputFile);
            string name =
Path.GetFileNameWithoutExtension(inputFil
e);
            string ext =
Path.GetExtension(inputFile);
            string suffix = isEncrypt ?
"_encrypted" : "_decrypted";
            return Path.Combine(dir ??
"", name + suffix + ext);
        }

        private void UpdateProgress(long
processedBytes, long totalBytes)
        {
            if (totalBytes <= 0)
            {
                SafeUi(() =>
                {
                    progressBar.Value =
0;
                    lblProgress.Text =
"0%";
                });
                return;
            }

            int percent =
(int)(processedBytes * 100 / totalBytes);
            if (percent < 0) percent = 0;
            if (percent > 100) percent =
100;

            SafeUi(() =>
            {
                progressBar.Value =
percent;
                lblProgress.Text =
percent + "%";
            });
        }
```

```
        private bool
ProcessFileWithAuth(string inputPath,
string outputPath, bool isEncrypt)
        {
            byte[] macKey =
DeriveMacKey(_key32);

            if (isEncrypt)
            {
                EncryptFile(inputPath,
outputPath, _key32, _nonce12, macKey);
                return true;
            }

            return DecryptFile(inputPath,
outputPath, _key32, macKey);
        }

        private static void
EncryptFile(string inputPath, string
outputPath, byte[] key32, byte[] nonce12,
byte[] macKey)
        {
            using var fsIn = new
FileStream(inputPath, FileMode.Open,
FileAccess.Read, FileShare.Read);
            using var fsOut = new
FileStream(outputPath, FileMode.Create,
FileAccess.Write, FileShare.None);

            fsOut.Write(Magic, 0,
Magic.Length);
            fsOut.WriteByte(Version);
            fsOut.Write(nonce12, 0,
nonce12.Length);

            var engine = new
ChaCha7539Engine();
            engine.Init(true, new
ParametersWithIV(new KeyParameter(key32),
nonce12));

            using var hmac = new
HMACSHA256(macKey);
            hmac.TransformBlock(Magic, 0,
Magic.Length, null, 0);
            hmac.TransformBlock(new[] {
Version }, 0, 1, null, 0);
            hmac.TransformBlock(nonce12,
0, nonce12.Length, null, 0);

            byte[] inBuffer = new byte[64
* 1024];
            byte[] outBuffer = new
byte[64 * 1024];

            int read;
            while ((read =
fsIn.Read(inBuffer, 0, inBuffer.Length))
> 0)
            {

engine.ProcessBytes(inBuffer, 0, read,
outBuffer, 0);
                fsOut.Write(outBuffer, 0,
read);
hmac.TransformBlock(outBuffer, 0, read,
null, 0);
            }


hmac.TransformFinalBlock(Array.Empty<byte
>(), 0, 0);
            byte[] tag = hmac.Hash;

            fsOut.Write(tag, 0,
tag.Length);
        }

        private static bool
DecryptFile(string inputPath, string
outputPath, byte[] key32, byte[] macKey)
        {
            using var fsIn = new
FileStream(inputPath, FileMode.Open,
FileAccess.Read, FileShare.Read);

            if (fsIn.Length < 4 + 1 + 12
+ 32)
                throw new Exception("Файл
слишком маленький или не в ожидаемом
формате.");

            byte[] magic = new byte[4];
            fsIn.Read(magic, 0, 4);
            if
(!magic.SequenceEqual(Magic))
                throw new
Exception("Некорректный формат файла.");

            int ver = fsIn.ReadByte();
            if (ver != Version)
                throw new
Exception("Некорректная версия
формата.");

            byte[] nonce12 = new
byte[12];
            fsIn.Read(nonce12, 0, 12);

            long cipherLen = fsIn.Length
- (4 + 1 + 12 + 32);
            if (cipherLen < 0)
                throw new
Exception("Некорректный размер файла.");

            fsIn.Position = fsIn.Length -
32;
            byte[] expectedTag = new
byte[32];
            fsIn.Read(expectedTag, 0,
32);

            using var hmac = new
HMACSHA256(macKey);
            hmac.TransformBlock(Magic, 0,
Magic.Length, null, 0);
            hmac.TransformBlock(new[] {
(byte)ver }, 0, 1, null, 0);
            hmac.TransformBlock(nonce12,
0, nonce12.Length, null, 0);

            fsIn.Position = 4 + 1 + 12;
```

```
            byte[] buffer = new byte[64 *
1024];
            long remaining = cipherLen;

            while (remaining > 0)
            {
                int toRead =
(int)Math.Min(buffer.Length, remaining);
                int read =
fsIn.Read(buffer, 0, toRead);
                if (read <= 0) break;


hmac.TransformBlock(buffer, 0, read,
null, 0);
                remaining -= read;
            }


hmac.TransformFinalBlock(Array.Empty<byte
>(), 0, 0);
            byte[] actualTag = hmac.Hash;

            if
(!FixedTimeEquals(expectedTag,
actualTag))
                return false;

            fsIn.Position = 4 + 1 + 12;

            using var fsOut = new
FileStream(outputPath, FileMode.Create,
FileAccess.Write, FileShare.None);

            var engine = new
ChaCha7539Engine();
            engine.Init(false, new
ParametersWithIV(new KeyParameter(key32),
nonce12));

            byte[] inBuffer = new byte[64
* 1024];
            byte[] outBuffer = new
byte[64 * 1024];
            remaining = cipherLen;

            while (remaining > 0)
            {
                int toRead =
(int)Math.Min(inBuffer.Length,
remaining);
                int read =
fsIn.Read(inBuffer, 0, toRead);
                if (read <= 0) break;


engine.ProcessBytes(inBuffer, 0, read,
outBuffer, 0);
                fsOut.Write(outBuffer, 0,
read);

                remaining -= read;
            }

            return true;
        }
        private static byte[]
DeriveMacKey(byte[] key32)
        {
            using var sha =
SHA256.Create();
            byte[] prefix = { (byte)'M',
(byte)'A', (byte)'C', (byte)':' };
            byte[] data = new
byte[prefix.Length + key32.Length];

            Buffer.BlockCopy(prefix, 0,
data, 0, prefix.Length);
            Buffer.BlockCopy(key32, 0,
data, prefix.Length, key32.Length);

            return sha.ComputeHash(data);
        }

        private static bool
FixedTimeEquals(byte[] a, byte[] b)
        {
            if (a == null || b == null ||
a.Length != b.Length)
                return false;

            int diff = 0;
            for (int i = 0; i < a.Length;
i++)
                diff |= (a[i] ^ b[i]);

            return diff == 0;
        }

        private static string
GetRelativePathCompat(string basePath,
string fullPath)
        {
            basePath =
Path.GetFullPath(basePath);
            fullPath =
Path.GetFullPath(fullPath);

            if
(!basePath.EndsWith(Path.DirectorySeparat
orChar.ToString()) &&

!basePath.EndsWith(Path.AltDirectorySepar
atorChar.ToString()))
            {
                basePath +=
Path.DirectorySeparatorChar;
            }

            var baseUri = new
Uri(basePath, UriKind.Absolute);
            var fullUri = new
Uri(fullPath, UriKind.Absolute);

            if (baseUri.Scheme !=
fullUri.Scheme)
                return fullPath;

            string rel =
Uri.UnescapeDataString(baseUri.MakeRelati
veUri(fullUri).ToString());
            return rel.Replace('/',
```

```
Path.DirectorySeparatorChar);
        }

        private void SafeUi(Action
action)
        {
            if (IsDisposed) return;

            if (InvokeRequired)
Invoke(action);
            else action();
        }

        private void AppendLog(string
text)
        {
            SafeUi(() =>
txtLog.AppendText(text +
Environment.NewLine));
        }

        private static void
SafeDeleteFile(string path)
        {
            try
            {
                if (File.Exists(path))
                    File.Delete(path);
            }
            catch
            {
            }
        }
    }
}
```

2.6 DESForm.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CruptoBench
{
    public partial class DESForm : Form
    {
        private Stopwatch _stopwatch;
        private TimeSpan _cpuStart;
        private DateTime _wallStartUtc;
        private double _peakCpuPercent;
        private double _peakMemoryMb;

        private long _lastUiBytes;
        private readonly Stopwatch
_speedStopwatch = new Stopwatch();

        public DESForm()
        {
            InitializeComponent();
            txtInput.AllowDrop = true;
            txtInput.DragEnter +=
TxtInput_DragEnter;
            txtInput.DragDrop +=
TxtInput_DragDrop;
        }

        private void
btnGenerateKey_Click(object sender,
EventArgs e)
        {
            using (var des =
DES.Create())
                txtKey.Text =
Convert.ToBase64String(des.Key);
        }

        private async void
btnEncryptFile_Click(object sender,
EventArgs e)
        {
            if (!TryReadKey(out var
desKey, out var keyError))
            {

MessageBox.Show(keyError);
                return;
            }

            string inputPath =
txtInput.Text;

            if
(!Directory.Exists(inputPath) &&
!File.Exists(inputPath))
            {
                MessageBox.Show("Файл або
папка не знайдені.");
                return;
            }

            if
(Directory.Exists(inputPath))
            {
                string sourceFolder =
inputPath;
                long totalPlainBytes =
GetTotalSizeRecursive(sourceFolder);

                if (totalPlainBytes <= 0)
                {

MessageBox.Show("Папка порожня, немає
файлів для шифрування.");
                    return;
                }

                string outputFolder =
sourceFolder.TrimEnd(Path.DirectorySepara
torChar, Path.AltDirectorySeparatorChar)
+ "_encrypted";
                await
RunWithStatsAsync(() =>
EncryptFolderAsync(sourceFolder,
outputFolder, desKey, totalPlainBytes),
totalPlainBytes);
```

```
                long totalCipherBytes =
Directory.Exists(outputFolder) ?
GetTotalSizeRecursive(outputFolder) : 0;


ShowSizeInfo(totalPlainBytes,
totalCipherBytes, "Розмір даних (DES)");
                return;
            }

            {
                string sourceFile =
inputPath;
                long plainBytes = new
FileInfo(sourceFile).Length;

                if (plainBytes <= 0)
                {
                    MessageBox.Show("Файл
порожній, немає даних для шифрування.");
                    return;
                }

                string outFile =
Path.Combine(

Path.GetDirectoryName(sourceFile) ?? "",

Path.GetFileNameWithoutExtension(sourceFi
le) + "_encrypted" +
Path.GetExtension(sourceFile)
                );

                await
RunWithStatsAsync(() =>
EncryptFileAsync(sourceFile, outFile,
desKey, plainBytes, 0), plainBytes);

                long cipherBytes =
File.Exists(outFile) ? new
FileInfo(outFile).Length : 0;
                ShowSizeInfo(plainBytes,
cipherBytes, "Розмір даних (DES)");
            }
        }

        private async void
btnDecryptFile_Click(object sender,
EventArgs e)
        {
            if (!TryReadKey(out var
desKey, out var keyError))
            {

MessageBox.Show(keyError);
                return;
            }

            string inputPath =
txtInput.Text;

            if
(!Directory.Exists(inputPath) &&
!File.Exists(inputPath))
            {
                MessageBox.Show("Файл або
папка не знайдені.");
                return;
            }

            if
(Directory.Exists(inputPath))
            {
                string sourceFolder =
inputPath;
                long totalCipherBytes =
GetTotalSizeRecursive(sourceFolder);

                if (totalCipherBytes <=
0)
                {

MessageBox.Show("Папка порожня, немає
файлів для розшифрування.");
                    return;
                }

                string outputFolder =
sourceFolder.TrimEnd(Path.DirectorySepara
torChar, Path.AltDirectorySeparatorChar)
+ "_decrypted";
                await
RunWithStatsAsync(() =>
DecryptFolderAsync(sourceFolder,
outputFolder, desKey, totalCipherBytes),
totalCipherBytes);
                return;
            }

            {
                string sourceFile =
inputPath;
                long cipherBytes = new
FileInfo(sourceFile).Length;

                if (cipherBytes <= 0)
                {
                    MessageBox.Show("Файл
порожній, немає даних для
розшифрування.");
                    return;
                }

                string outFile =
Path.Combine(

Path.GetDirectoryName(sourceFile) ?? "",

Path.GetFileNameWithoutExtension(sourceFi
le) + "_decrypted" +
Path.GetExtension(sourceFile)
                );

                try
                {
                    await
RunWithStatsAsync(() =>
DecryptFileAsync(sourceFile, outFile,
desKey, cipherBytes, 0), cipherBytes);
                }
                catch
(CryptographicException)
                {
```

```
SafeDeleteFile(outFile);

MessageBox.Show("Неверний ключ");
                }
            }
        }

        private bool TryReadKey(out
byte[] desKey, out string error)
        {
            desKey = null;
            error = null;

            if
(string.IsNullOrWhiteSpace(txtKey.Text))
            {
                error = "Спочатку
згенеруйте або введіть ключ DES.";
                return false;
            }

            try
            {
                desKey =
Convert.FromBase64String(txtKey.Text.Trim
());
            }
            catch
            {
                error = "Невірний формат
ключа (очікується Base64).";
                return false;
            }

            if (desKey.Length != 8)
            {
                error = "Невірна довжина
ключа DES. Потрібно 8 байт (64 біти).";
                return false;
            }

            return true;
        }

        private async Task
RunWithStatsAsync(Func<Task<long>>
operation, long totalBytes)
        {
            txtLog.Clear();

            _stopwatch =
Stopwatch.StartNew();
            _speedStopwatch.Restart();
            _lastUiBytes = 0;

            _peakCpuPercent = 0.0;
            _peakMemoryMb = 0.0;

            var proc =
Process.GetCurrentProcess();
            proc.Refresh();
            _cpuStart =
proc.TotalProcessorTime;
            _wallStartUtc =
DateTime.UtcNow;

            SafeUi(() =>
            {
                progressBar.Minimum = 0;
                progressBar.Maximum =
100;
                progressBar.Value = 0;
                lblProgress.Text = "0%";
            });

            long processedBytes = await
operation.Invoke();

            _stopwatch.Stop();
            proc.Refresh();

            double cpuPercent =
(proc.TotalProcessorTime -
_cpuStart).TotalMilliseconds /

(Environment.ProcessorCount *
(DateTime.UtcNow -
_wallStartUtc).TotalMilliseconds) *
100.0;
            _peakCpuPercent =
Math.Max(_peakCpuPercent, cpuPercent);

            double memMb =
proc.WorkingSet64 / (1024.0 * 1024.0);
            _peakMemoryMb =
Math.Max(_peakMemoryMb, memMb);

            double elapsedSeconds =
Math.Max(0.000001,
_stopwatch.Elapsed.TotalSeconds);
            double avgSpeedMbSec =
(processedBytes / 1024.0 / 1024.0) /
elapsedSeconds;
            double totalMb =
processedBytes / 1024.0 / 1024.0;

            string resultMsg =
string.Format(
                "Операцію завершено за
{0:F2} с.\n" +
                "Обсяг оброблених даних:
{1} байт (~{2:F2} МБ)\n" +
                "Пікове завантаження CPU
= {3:F1}%, RAM = {4:F1} МБ\n" +
                "Середня швидкість:
{5:F2} МБ/с.",

_stopwatch.Elapsed.TotalSeconds,
                processedBytes,
                totalMb,
                _peakCpuPercent,
                _peakMemoryMb,
                avgSpeedMbSec);

            txtLog.AppendText(resultMsg +
Environment.NewLine);
            MessageBox.Show(resultMsg,
"Статистика DES");
        }

        private async Task<long>
EncryptFileAsync(
            string inputFile,
```

```
            string outputFile,
            byte[] desKey,
            long totalBytes,
            long alreadyProcessedBytes)
        {
            byte[] iv = new byte[8];
            using (var rng =
RandomNumberGenerator.Create())
                rng.GetBytes(iv);

            using (var des =
DES.Create())
            {
                des.Key = desKey;
                des.IV = iv;

                using (var fsIn = new
FileStream(inputFile, FileMode.Open,
FileAccess.Read, FileShare.Read, 4096,
true))
                using (var fsOut = new
FileStream(outputFile, FileMode.Create,
FileAccess.Write, FileShare.None, 4096,
true))
                {
                    await
fsOut.WriteAsync(iv, 0,
iv.Length).ConfigureAwait(false);

                    using (var cryptoOut
= new CryptoStream(fsOut,
des.CreateEncryptor(),
CryptoStreamMode.Write))
                    {
                        byte[] buffer =
new byte[1024 * 1024];
                        long readBytes =
0;

                        int read;
                        while ((read =
await fsIn.ReadAsync(buffer, 0,
buffer.Length).ConfigureAwait(false)) >
0)
                        {
                            await
cryptoOut.WriteAsync(buffer, 0,
read).ConfigureAwait(false);
                            readBytes +=
read;

UpdateProgress(totalBytes,
alreadyProcessedBytes + readBytes);
                        }

                        await
cryptoOut.FlushAsync().ConfigureAwait(fal
se);
                        return readBytes;
                    }
                }
            }
        }

        private async Task<long>
DecryptFileAsync(
            string inputFile,
            string outputFile,
            byte[] desKey,
            long totalBytes,
            long alreadyProcessedBytes)
        {
            using (var fsIn = new
FileStream(inputFile, FileMode.Open,
FileAccess.Read, FileShare.Read, 4096,
true))
            {
                byte[] iv = new byte[8];
                int ivRead = await
fsIn.ReadAsync(iv, 0,
iv.Length).ConfigureAwait(false);
                if (ivRead != 8)
                    throw new
Exception("Некоректний формат файла (не
знайдено IV).");

                using (var des =
DES.Create())
                {
                    des.Key = desKey;
                    des.IV = iv;

                    using (var cryptoIn =
new CryptoStream(fsIn,
des.CreateDecryptor(),
CryptoStreamMode.Read))
                    using (var fsOut =
new FileStream(outputFile,
FileMode.Create, FileAccess.Write,
FileShare.None, 4096, true))
                    {
                        byte[] buffer =
new byte[1024 * 1024];
                        long writtenBytes
= 0;

                        int read;
                        while ((read =
await cryptoIn.ReadAsync(buffer, 0,
buffer.Length).ConfigureAwait(false)) >
0)
                        {
                            await
fsOut.WriteAsync(buffer, 0,
read).ConfigureAwait(false);
                            writtenBytes
+= read;

UpdateProgress(totalBytes,
alreadyProcessedBytes +
Math.Min(alreadyProcessedBytes +
writtenBytes, totalBytes));
                        }

                        await
fsOut.FlushAsync().ConfigureAwait(false);
                        return
writtenBytes;
                    }
                }
            }
        }

        private async Task<long>
```

```
EncryptFolderAsync(string sourceFolder,
string outputFolder, byte[] desKey, long
totalBytes)
        {

Directory.CreateDirectory(outputFolder);

            var files =
Directory.GetFiles(sourceFolder, "*",
SearchOption.AllDirectories);
            long processed = 0;

            foreach (var file in files)
            {
                string relative =
GetRelativePathCompat(sourceFolder,
file);
                string outFile =
Path.Combine(outputFolder, relative);

                string outDir =
Path.GetDirectoryName(outFile);
                if
(!string.IsNullOrEmpty(outDir))

Directory.CreateDirectory(outDir);

                processed += await
EncryptFileAsync(file, outFile, desKey,
totalBytes,
processed).ConfigureAwait(false);
            }

            UpdateProgress(totalBytes,
totalBytes);
            return processed;
        }

        private async Task<long>
DecryptFolderAsync(string sourceFolder,
string outputFolder, byte[] desKey, long
totalBytes)
        {

Directory.CreateDirectory(outputFolder);

            var files =
Directory.GetFiles(sourceFolder, "*",
SearchOption.AllDirectories);
            long processed = 0;

            foreach (var file in files)
            {
                string relative =
GetRelativePathCompat(sourceFolder,
file);
                string outFile =
Path.Combine(outputFolder, relative);

                string outDir =
Path.GetDirectoryName(outFile);
                if
(!string.IsNullOrEmpty(outDir))

Directory.CreateDirectory(outDir);

                try
                {
                    processed += await
DecryptFileAsync(file, outFile, desKey,
totalBytes,
processed).ConfigureAwait(false);
                }
                catch
(CryptographicException)
                {

SafeDeleteFile(outFile);

MessageBox.Show("Неверний ключ");
                    return processed;
                }
                catch
                {

SafeDeleteFile(outFile);
                    throw;
                }
            }

            UpdateProgress(totalBytes,
totalBytes);
            return processed;
        }

        private void UpdateProgress(long
totalBytes, long currentBytes)
        {
            if (totalBytes <= 0) return;

            long uiStep = 4L * 1024 *
1024;
            if (currentBytes -
_lastUiBytes < uiStep && currentBytes <
totalBytes)
                return;

            double percent =
(double)currentBytes / totalBytes *
100.0;
            if (percent < 0) percent = 0;
            if (percent > 100) percent =
100;

            double elapsed =
Math.Max(0.000001,
_speedStopwatch.Elapsed.TotalSeconds);
            double speedMbSec =
(currentBytes / 1024.0 / 1024.0) /
elapsed;

            SafeUi(() =>
            {
                int p = (int)percent;
                if (p < 0) p = 0;
                if (p > 100) p = 100;
                progressBar.Value = p;
                lblProgress.Text = p +
"%";
            });

            _lastUiBytes = currentBytes;
        }
```

```
        private static long
GetTotalSizeRecursive(string folderPath)
        {
            long total = 0;
            foreach (var file in
Directory.GetFiles(folderPath, "*",
SearchOption.AllDirectories))
            {
                try { total += new
FileInfo(file).Length; }
                catch { }
            }
            return total;
        }

        private static string
GetRelativePathCompat(string basePath,
string fullPath)
        {
            basePath =
Path.GetFullPath(basePath);
            fullPath =
Path.GetFullPath(fullPath);

            if
(!basePath.EndsWith(Path.DirectorySeparat
orChar.ToString()) &&

!basePath.EndsWith(Path.AltDirectorySepar
atorChar.ToString()))
            {
                basePath +=
Path.DirectorySeparatorChar;
            }

            var baseUri = new
Uri(basePath, UriKind.Absolute);
            var fullUri = new
Uri(fullPath, UriKind.Absolute);

            if (baseUri.Scheme !=
fullUri.Scheme)
                return fullPath;

            string rel =
Uri.UnescapeDataString(baseUri.MakeRelati
veUri(fullUri).ToString());
            return rel.Replace('/',
Path.DirectorySeparatorChar);
        }

        private static void
SafeDeleteFile(string path)
        {
            try
            {
                if (File.Exists(path))
                    File.Delete(path);
            }
            catch
            {
            }
        }

        private void ShowSizeInfo(long
inputBytes, long outputBytes, string
title)
        {
            double inputMb = inputBytes /
1024.0 / 1024.0;
            double outputMb = outputBytes
/ 1024.0 / 1024.0;
            double ratio = inputBytes > 0
? (double)outputBytes / inputBytes : 0.0;

            string msg =
                $"Початковий розмір:
{inputBytes} байт (~{inputMb:F2} МБ)\n" +
                $"Розмір після операції:
{outputBytes} байт (~{outputMb:F2} МБ)\n"
+
                $"Коефіцієнт розширення:
{ratio:F3} раз(и).";

            txtLog.AppendText(msg +
Environment.NewLine);
            MessageBox.Show(msg, title);
        }

        private void
TxtInput_DragEnter(object sender,
DragEventArgs e)
        {
            if (e.Data != null &&
e.Data.GetDataPresent(DataFormats.FileDro
p))
                e.Effect =
DragDropEffects.Copy;
        }

        private void
TxtInput_DragDrop(object sender,
DragEventArgs e)
        {
            var items =
(string[])e.Data.GetData(DataFormats.File
Drop);
            if (items != null &&
items.Length > 0)
                txtInput.Text = items[0];
        }

        private void SafeUi(Action
action)
        {
            if (IsDisposed) return;
            if (InvokeRequired)
Invoke(action);
            else action();
        }
    }
}
```

2.7 ECCForm.cs

```
using Org.BouncyCastle.Asn1.Sec;
using Org.BouncyCastle.Asn1.X9;
using Org.BouncyCastle.Crypto.Agreement;
using Org.BouncyCastle.Crypto.Generators;
```

```
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Math;
using Org.BouncyCastle.Security;
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CruptoBench
{
    public partial class ECCForm : Form
    {
        private ECPublicKeyParameters
_publicKey;
        private ECPrivateKeyParameters
_privateKey;

        private const int IterationCount
= 1000;

        private static readonly byte[]
FileMagic = { (byte)'E', (byte)'C',
(byte)'C', (byte)'1' };
        private const byte FileVersion =
1;

        private static readonly
X9ECParameters CurveParams =
SecNamedCurves.GetByName("secp256r1");
        private static readonly
ECDomainParameters Domain = new
ECDomainParameters(
            CurveParams.Curve,
CurveParams.G, CurveParams.N,
CurveParams.H, CurveParams.GetSeed()
        );

        public ECCForm()
        {
            InitializeComponent();

            txtFilePath.AllowDrop = true;
            txtFilePath.DragEnter +=
TxtFilePath_DragEnter;
            txtFilePath.DragDrop +=
TxtFilePath_DragDrop;
        }

        private void
TxtFilePath_DragEnter(object sender,
DragEventArgs e)
        {
            if (e.Data != null &&
e.Data.GetDataPresent(DataFormats.FileDro
p))
                e.Effect =
DragDropEffects.Copy;
        }

        private void
TxtFilePath_DragDrop(object sender,
DragEventArgs e)
        {
            var items =
(string[])e.Data.GetData(DataFormats.File
Drop);
            if (items != null &&
items.Length > 0)
                txtFilePath.Text =
items[0];
        }

        private void
btnGenerateKeys_Click(object sender,
EventArgs e)
        {
            try
            {
                var process =
Process.GetCurrentProcess();
                process.Refresh();

                var sw =
Stopwatch.StartNew();
                var cpuStart =
process.TotalProcessorTime;
                var memStart =
process.WorkingSet64;

                int keyBits =
Domain.Curve.FieldSize;

                var rng = new
SecureRandom();
                var keyGen = new
ECKeyPairGenerator();
                keyGen.Init(new
ECKeyGenerationParameters(Domain, rng));
                var keyPair =
keyGen.GenerateKeyPair();

                _publicKey =
(ECPublicKeyParameters)keyPair.Public;
                _privateKey =
(ECPrivateKeyParameters)keyPair.Private;

                sw.Stop();
                process.Refresh();

                var cpuEnd =
process.TotalProcessorTime;
                var memEnd =
process.WorkingSet64;

                double totalMs =
sw.Elapsed.TotalMilliseconds;
                double cpuPercent =
totalMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * totalMs) *
100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                txtPublicKey.Text =
ExportPublicKey(_publicKey);
```

```
                txtPrivateKey.Text =
ExportPrivateKey(_privateKey);

                string info =
                    $"Ключі ECC
згенеровано за {totalMs:F2} мс; " +
                    $"довжина ключа:
{keyBits} біт; " +
                    $"CPU:
{cpuPercent:F2}%; " +
                    $"RAM: {memMb:F2}
МБ.";

                txtInfo.Text = info;

                MessageBox.Show(
                    "Генерація ключів
ECC\n\n" +
                    $"Час: {totalMs:F2}
мс\n" +
                    $"Довжина ключа:
{keyBits} біт\n" +
                    $"CPU:
{cpuPercent:F2}%\n" +
                    $"RAM: {memMb:F2}
МБ",
                    "ECC – генерація
ключів",
                    MessageBoxButtons.OK,

MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
під час генерації ключів: " +
ex.Message);
            }
        }

        private void
btnEncrypt_Click(object sender, EventArgs
e)
        {
            try
            {
                if
(string.IsNullOrWhiteSpace(txtPublicKey.T
ext))
                {

MessageBox.Show("Спочатку вставте або
згенеруйте відкритий ключ ECC.");
                    return;
                }

                if
(string.IsNullOrWhiteSpace(txtPlainText.T
ext))
                {

MessageBox.Show("Введіть текст для
шифрування.");
                    return;
                }

                _publicKey =
ImportPublicKey(txtPublicKey.Text);
                int keyBits =
_publicKey.Parameters.Curve.FieldSize;

                byte[] plainBytes =
Encoding.UTF8.GetBytes(txtPlainText.Text)
;
                int plainLen =
plainBytes.Length;
                double plainKb = plainLen
/ 1024.0;

                var process =
Process.GetCurrentProcess();
                process.Refresh();
                var cpuStart =
process.TotalProcessorTime;
                var memStart =
process.WorkingSet64;

                var sw =
Stopwatch.StartNew();

                byte[] cipherBytes =
null;
                for (int i = 0; i <
IterationCount; i++)
                    cipherBytes =
EncryptHybridBytes(plainBytes,
_publicKey);

                sw.Stop();
                process.Refresh();

                var cpuEnd =
process.TotalProcessorTime;
                var memEnd =
process.WorkingSet64;

                int cipherLen =
cipherBytes.Length;
                double cipherKb =
cipherLen / 1024.0;

                double totalMs =
sw.Elapsed.TotalMilliseconds;
                double avgMs = totalMs /
IterationCount;

                double cpuPercent =
totalMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * totalMs) *
100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                txtEncrypted.Text =
Convert.ToBase64String(cipherBytes);

                string info =
                    $"Шифрування ECC
(ECDH + AES-CBC + HMAC, {IterationCount}
```

```
повторень)\n" +
                    $"Довжина ключа:
{keyBits} біт\n" +
                    $"Загальний час:
{totalMs:F2} мс; середній час: {avgMs:F4}
мс\n" +
                    $"Розмір відкритого
тексту: {plainLen} байт (~{plainKb:F2}
КБ)\n" +
                    $"Розмір шифротексту:
{cipherLen} байт (~{cipherKb:F2} КБ)\n" +
                    $"CPU:
{cpuPercent:F2}%; RAM: {memMb:F2} МБ.";

                txtInfo.Text = info;

                MessageBox.Show(
                    info,
                    "ECC – шифрування",
                    MessageBoxButtons.OK,

MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
під час шифрування: " + ex.Message);
            }
        }

        private void
btnDecrypt_Click(object sender, EventArgs
e)
        {
            try
            {
                if
(string.IsNullOrWhiteSpace(txtPrivateKey.
Text))
                {

MessageBox.Show("Спочатку вставте
закритий ключ ECC.");
                    return;
                }

                if
(string.IsNullOrWhiteSpace(txtEncrypted.T
ext))
                {

MessageBox.Show("Немає даних для
розшифрування.");
                    return;
                }

                _privateKey =
ImportPrivateKey(txtPrivateKey.Text);
                int keyBits =
_privateKey.Parameters.Curve.FieldSize;

                byte[] cipherBytes =
Convert.FromBase64String(txtEncrypted.Tex
t);
                int cipherLen =
cipherBytes.Length;
                double cipherKb =
cipherLen / 1024.0;

                var process =
Process.GetCurrentProcess();
                process.Refresh();
                var cpuStart =
process.TotalProcessorTime;
                var memStart =
process.WorkingSet64;

                var sw =
Stopwatch.StartNew();

                byte[] plainBytes = null;
                for (int i = 0; i <
IterationCount; i++)
                    plainBytes =
DecryptHybridBytes(cipherBytes,
_privateKey);

                sw.Stop();
                process.Refresh();

                var cpuEnd =
process.TotalProcessorTime;
                var memEnd =
process.WorkingSet64;

                int plainLen =
plainBytes.Length;
                double plainKb = plainLen
/ 1024.0;

                double totalMs =
sw.Elapsed.TotalMilliseconds;
                double avgMs = totalMs /
IterationCount;

                double cpuPercent =
totalMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * totalMs) *
100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                txtDecrypted.Text =
Encoding.UTF8.GetString(plainBytes);

                string info =
                    $"Розшифрування ECC
(ECDH + AES-CBC + HMAC, {IterationCount}
повторень)\n" +
                    $"Довжина ключа:
{keyBits} біт\n" +
                    $"Загальний час:
{totalMs:F2} мс; середній час: {avgMs:F4}
мс\n" +
                    $"Розмір шифротексту:
{cipherLen} байт (~{cipherKb:F2} КБ)\n" +
                    $"Розмір
розшифрованого тексту: {plainLen} байт
(~{plainKb:F2} КБ)\n" +
```

```
                    $"CPU:
{cpuPercent:F2}%; RAM: {memMb:F2} МБ.";

                txtInfo.Text = info;

                MessageBox.Show(
                    info,
                    "ECC -
розшифрування",
                    MessageBoxButtons.OK,

MessageBoxIcon.Information);
            }
            catch
(CryptographicException)
            {
                MessageBox.Show("Неверний
ключ");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
під час розшифрування: " + ex.Message);
            }
        }

        private void
btnBrowseFile_Click(object sender,
EventArgs e)
        {
            using (var ofd = new
OpenFileDialog())
            {
                if (ofd.ShowDialog() ==
DialogResult.OK)
                    txtFilePath.Text =
ofd.FileName;
            }
        }

        private async void
btnEncryptFile_Click(object sender,
EventArgs e)
        {
            try
            {
                if
(string.IsNullOrWhiteSpace(txtPublicKey.T
ext))
                {

MessageBox.Show("Спочатку вставте або
згенеруйте відкритий ключ ECC.");
                    return;
                }

                string inputPath =
txtFilePath.Text;
                if
(string.IsNullOrWhiteSpace(inputPath) ||
(!File.Exists(inputPath) &&
!Directory.Exists(inputPath)))
                {

MessageBox.Show("Оберіть файл або папку
для шифрування.");
                    return;
                }

                _publicKey =
ImportPublicKey(txtPublicKey.Text);
                int keyBits =
_publicKey.Parameters.Curve.FieldSize;

                var process =
Process.GetCurrentProcess();
                process.Refresh();
                var cpuStart =
process.TotalProcessorTime;
                var memStart =
process.WorkingSet64;
                var sw =
Stopwatch.StartNew();

                long inputBytes = 0;
                long outputBytes = 0;

                if
(File.Exists(inputPath))
                {
                    string outFile =
Path.Combine(

Path.GetDirectoryName(inputPath) ?? "",

Path.GetFileNameWithoutExtension(inputPat
h) + "_ecc_encrypted" +
Path.GetExtension(inputPath));

                    inputBytes = new
FileInfo(inputPath).Length;

                    await Task.Run(() =>
EncryptHybridFile(inputPath, outFile,
_publicKey));

                    outputBytes = new
FileInfo(outFile).Length;
                }
                else
                {
                    string sourceFolder =
inputPath;
                    string outFolder =
sourceFolder.TrimEnd(Path.DirectorySepara
torChar, Path.AltDirectorySeparatorChar)
+ "_ecc_encrypted";

Directory.CreateDirectory(outFolder);

                    var files =
Directory.GetFiles(sourceFolder, "*",
SearchOption.AllDirectories);
                    inputBytes =
GetTotalSizeRecursive(sourceFolder);

                    await Task.Run(() =>
                    {
                        foreach (var file
in files)
                        {
                            string
relative =
GetRelativePathCompat(sourceFolder,
```

```
file);
                                string
outFile = Path.Combine(outFolder,
relative);

                                string outDir
= Path.GetDirectoryName(outFile);
                                if
(!string.IsNullOrEmpty(outDir))

Directory.CreateDirectory(outDir);


EncryptHybridFile(file, outFile,
_publicKey);
                            }
                        });

                        outputBytes =
GetTotalSizeRecursive(outFolder);
                    }

                    sw.Stop();
                    process.Refresh();
                    var cpuEnd =
process.TotalProcessorTime;
                    var memEnd =
process.WorkingSet64;

                    double inMb = inputBytes
/ 1024.0 / 1024.0;
                    double outMb =
outputBytes / 1024.0 / 1024.0;
                    double ratio = inputBytes
> 0 ? (double)outputBytes / inputBytes :
0.0;

                    double totalSec =
sw.Elapsed.TotalSeconds;
                    double speed = totalSec >
0 ? inMb / totalSec : 0.0;

                    double cpuPercent =
sw.Elapsed.TotalMilliseconds > 0
                        ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount *
sw.Elapsed.TotalMilliseconds) * 100.0
                        : 0.0;

                    double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                    if (memMb < 0) memMb = 0;

                    string targetType =
File.Exists(inputPath) ? "файл" :
"папка";

                    string info =
                        $"ECC+AES шифрування
({targetType})\n" +
                        $"Ключ: {keyBits}
біт\n" +
                        $"Обсяг до:
{inputBytes} байт (~{inMb:F2} МБ)\n" +
                        $"Обсяг після:
{outputBytes} байт (~{outMb:F2} МБ), k =
{ratio:F3}\n" +
                        $"Час: {totalSec:F2}
с, швидкість: {speed:F2} МБ/с\n" +
                        $"CPU:
{cpuPercent:F1}%, RAM: {memMb:F1} МБ.";

                    txtInfo.Text = info;

                    MessageBox.Show(info,
"ECC – шифрування файлу/папки",
MessageBoxButtons.OK,
MessageBoxIcon.Information);
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Помилка
шифрування файлу/папки: " + ex.Message);
                }
            }

            private async void
btnDecryptFile_Click(object sender,
EventArgs e)
            {
                try
                {
                    if
(string.IsNullOrWhiteSpace(txtPrivateKey.
Text))
                    {

MessageBox.Show("Спочатку вставте
закритий ключ ECC.");
                        return;
                    }

                    string inputPath =
txtFilePath.Text;
                    if
(string.IsNullOrWhiteSpace(inputPath) ||
(!File.Exists(inputPath) &&
!Directory.Exists(inputPath)))
                    {

MessageBox.Show("Оберіть файл або папку
для розшифрування.");
                        return;
                    }

                    _privateKey =
ImportPrivateKey(txtPrivateKey.Text);
                    int keyBits =
_privateKey.Parameters.Curve.FieldSize;

                    var process =
Process.GetCurrentProcess();
                    process.Refresh();
                    var cpuStart =
process.TotalProcessorTime;
                    var memStart =
process.WorkingSet64;
                    var sw =
Stopwatch.StartNew();

                    long inputBytes = 0;
                    long outputBytes = 0;
```

```
                if
(File.Exists(inputPath))
                {
                    string outFile =
Path.Combine(

Path.GetDirectoryName(inputPath) ?? "",

Path.GetFileNameWithoutExtension(inputPat
h) + "_ecc_decrypted" +
Path.GetExtension(inputPath));

                    inputBytes = new
FileInfo(inputPath).Length;

                    bool ok = await
Task.Run(() =>
TryDecryptHybridFile(inputPath, outFile,
_privateKey));
                    if (!ok)
                    {

SafeDeleteFile(outFile);

MessageBox.Show("Неверний ключ");
                        return;
                    }

                    outputBytes = new
FileInfo(outFile).Length;
                }
                else
                {
                    string sourceFolder =
inputPath;
                    string outFolder =
sourceFolder.TrimEnd(Path.DirectorySepara
torChar, Path.AltDirectorySeparatorChar)
+ "_ecc_decrypted";

Directory.CreateDirectory(outFolder);

                    var files =
Directory.GetFiles(sourceFolder, "*",
SearchOption.AllDirectories);
                    inputBytes =
GetTotalSizeRecursive(sourceFolder);

                    bool badKey = false;

                    await Task.Run(() =>
                    {
                        foreach (var file
in files)
                        {
                            string
relative =
GetRelativePathCompat(sourceFolder,
file);
                            string
outFile = Path.Combine(outFolder,
relative);

                            string outDir
= Path.GetDirectoryName(outFile);
                            if
(!string.IsNullOrEmpty(outDir))
Directory.CreateDirectory(outDir);

                            bool ok =
TryDecryptHybridFile(file, outFile,
_privateKey);
                            if (!ok)
                            {

SafeDeleteFile(outFile);
                                badKey =
true;
                                return;
                            }
                        }
                    });

                    if (badKey)
                    {

MessageBox.Show("Неверний ключ");
                        return;
                    }

                    outputBytes =
GetTotalSizeRecursive(outFolder);
                }

                sw.Stop();
                process.Refresh();
                var cpuEnd =
process.TotalProcessorTime;
                var memEnd =
process.WorkingSet64;

                double inMb = inputBytes
/ 1024.0 / 1024.0;
                double outMb =
outputBytes / 1024.0 / 1024.0;

                double totalSec =
sw.Elapsed.TotalSeconds;
                double speed = totalSec >
0 ? outMb / totalSec : 0.0;

                double cpuPercent =
sw.Elapsed.TotalMilliseconds > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount *
sw.Elapsed.TotalMilliseconds) * 100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                string targetType =
File.Exists(inputPath) ? "файл" :
"папка";

                string info =
                    $"ECC+AES
розшифрування ({targetType})\n" +
                    $"Ключ: {keyBits}
біт\n" +
                    $"Обсяг шифротексту:
```

```
{inputBytes} байт (~{inMb:F2} МБ)\n" +
                    $"Обсяг після
розшифрування: {outputBytes} байт
(~{outMb:F2} МБ)\n" +
                    $"Час: {totalSec:F2}
с, швидкість: {speed:F2} МБ/с\n" +
                    $"CPU:
{cpuPercent:F1}%, RAM: {memMb:F1} МБ.";

                txtInfo.Text = info;

                MessageBox.Show(info,
"ECC – розшифрування файлу/папки",
MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
розшифрування файлу/папки: " +
ex.Message);
            }
        }

        private static byte[]
EncryptHybridBytes(byte[] plainBytes,
ECPublicKeyParameters recipientPublicKey)
        {
            var rng = new SecureRandom();

            var keyGen = new
ECKeyPairGenerator();
            keyGen.Init(new
ECKeyGenerationParameters(Domain, rng));
            var ephPair =
keyGen.GenerateKeyPair();

            var ephPriv =
(ECPrivateKeyParameters)ephPair.Private;
            var ephPub =
(ECPublicKeyParameters)ephPair.Public;

            byte[] ephPubBytes =
ephPub.Q.GetEncoded(false);

            byte[] sharedSecret =
ComputeSharedSecret(ephPriv,
recipientPublicKey);
            DeriveKeys(sharedSecret, out
byte[] encKey16, out byte[] macKey32);

            using (var aes =
Aes.Create())
            {
                aes.KeySize = 128;
                aes.BlockSize = 128;
                aes.Mode =
CipherMode.CBC;
                aes.Padding =
PaddingMode.PKCS7;
                aes.Key = encKey16;
                aes.GenerateIV();

                byte[] cipherBytes;
                using (var ms = new
MemoryStream())
                using (var cs = new
CryptoStream(ms, aes.CreateEncryptor(),
CryptoStreamMode.Write))
                {
                    cs.Write(plainBytes,
0, plainBytes.Length);
                    cs.FlushFinalBlock();
                    cipherBytes =
ms.ToArray();
                }

                byte[] header = new
byte[ephPubBytes.Length + aes.IV.Length];

Buffer.BlockCopy(ephPubBytes, 0, header,
0, ephPubBytes.Length);
                Buffer.BlockCopy(aes.IV,
0, header, ephPubBytes.Length,
aes.IV.Length);

                byte[] tag =
ComputeHmac(macKey32, header,
cipherBytes);

                byte[] result = new
byte[header.Length + cipherBytes.Length +
tag.Length];
                Buffer.BlockCopy(header,
0, result, 0, header.Length);

Buffer.BlockCopy(cipherBytes, 0, result,
header.Length, cipherBytes.Length);
                Buffer.BlockCopy(tag, 0,
result, header.Length +
cipherBytes.Length, tag.Length);
                return result;
            }
        }

        private static byte[]
DecryptHybridBytes(byte[] data,
ECPrivateKeyParameters
recipientPrivateKey)
        {
            const int ephLen = 65;
            const int ivLen = 16;
            const int tagLen = 32;

            if (data == null ||
data.Length < ephLen + ivLen + tagLen +
1)
                throw new
Exception("Невірний формат
шифротексту.");

            int headerLen = ephLen +
ivLen;
            int cipherLen = data.Length -
headerLen - tagLen;

            byte[] ephPubBytes = new
byte[ephLen];
            byte[] iv = new byte[ivLen];
            byte[] cipherBytes = new
byte[cipherLen];
            byte[] expectedTag = new
byte[tagLen];
```

```
            Buffer.BlockCopy(data, 0,
ephPubBytes, 0, ephLen);
            Buffer.BlockCopy(data,
ephLen, iv, 0, ivLen);
            Buffer.BlockCopy(data,
headerLen, cipherBytes, 0, cipherLen);
            Buffer.BlockCopy(data,
headerLen + cipherLen, expectedTag, 0,
tagLen);

            var qPoint =
CurveParams.Curve.DecodePoint(ephPubBytes
);
            var ephPub = new
ECPublicKeyParameters(qPoint, Domain);

            byte[] sharedSecret =
ComputeSharedSecret(recipientPrivateKey,
ephPub);
            DeriveKeys(sharedSecret, out
byte[] encKey16, out byte[] macKey32);

            byte[] header = new
byte[headerLen];
            Buffer.BlockCopy(ephPubBytes,
0, header, 0, ephLen);
            Buffer.BlockCopy(iv, 0,
header, ephLen, ivLen);

            byte[] actualTag =
ComputeHmac(macKey32, header,
cipherBytes);
            if
(!FixedTimeEquals(expectedTag,
actualTag))
                throw new
CryptographicException("Bad MAC");

            using (var aes =
Aes.Create())
            {
                aes.KeySize = 128;
                aes.BlockSize = 128;
                aes.Mode =
CipherMode.CBC;
                aes.Padding =
PaddingMode.PKCS7;
                aes.Key = encKey16;
                aes.IV = iv;

                using (var ms = new
MemoryStream())
                using (var cs = new
CryptoStream(ms, aes.CreateDecryptor(),
CryptoStreamMode.Write))
                {
                    cs.Write(cipherBytes,
0, cipherBytes.Length);
                    cs.FlushFinalBlock();
                    return ms.ToArray();
                }
            }
        }

        private static void
EncryptHybridFile(string inputFile,
string outputFile, ECPublicKeyParameters
recipientPublicKey)
        {
            var rng = new SecureRandom();

            var keyGen = new
ECKeyPairGenerator();
            keyGen.Init(new
ECKeyGenerationParameters(Domain, rng));
            var ephPair =
keyGen.GenerateKeyPair();

            var ephPriv =
(ECPrivateKeyParameters)ephPair.Private;
            var ephPub =
(ECPublicKeyParameters)ephPair.Public;

            byte[] ephPubBytes =
ephPub.Q.GetEncoded(false);

            byte[] sharedSecret =
ComputeSharedSecret(ephPriv,
recipientPublicKey);
            DeriveKeys(sharedSecret, out
byte[] encKey16, out byte[] macKey32);

            using (var aes =
Aes.Create())
            {
                aes.KeySize = 128;
                aes.BlockSize = 128;
                aes.Mode =
CipherMode.CBC;
                aes.Padding =
PaddingMode.PKCS7;
                aes.Key = encKey16;
                aes.GenerateIV();

                using (var fsIn = new
FileStream(inputFile, FileMode.Open,
FileAccess.Read, FileShare.Read))
                using (var fsOut = new
FileStream(outputFile, FileMode.Create,
FileAccess.Write, FileShare.None))
                using (var hmac = new
HMACSHA256(macKey32))
                {

fsOut.Write(FileMagic, 0,
FileMagic.Length);

fsOut.WriteByte(FileVersion);

fsOut.Write(ephPubBytes, 0,
ephPubBytes.Length);
                    fsOut.Write(aes.IV,
0, aes.IV.Length);


hmac.TransformBlock(FileMagic, 0,
FileMagic.Length, null, 0);

hmac.TransformBlock(new[] { FileVersion
}, 0, 1, null, 0);

hmac.TransformBlock(ephPubBytes, 0,
ephPubBytes.Length, null, 0);
```

```
hmac.TransformBlock(aes.IV, 0,
aes.IV.Length, null, 0);

                    using (var macStream
= new HmacWriteStream(fsOut, hmac,
leaveOpen: true))
                    {
                        using (var cs =
new CryptoStream(macStream,
aes.CreateEncryptor(),
CryptoStreamMode.Write))
                        {
                            byte[] buffer
= new byte[1024 * 1024];
                            int read;
                            while ((read
= fsIn.Read(buffer, 0, buffer.Length)) >
0)

cs.Write(buffer, 0, read);


cs.FlushFinalBlock();
                        }


hmac.TransformFinalBlock(Array.Empty<byte
>(), 0, 0);
                        byte[] tag =
hmac.Hash;
                        fsOut.Write(tag,
0, tag.Length);
                    }
                }
            }
        }

        private static bool
TryDecryptHybridFile(string inputFile,
string outputFile, ECPrivateKeyParameters
recipientPrivateKey)
        {
            const int ephLen = 65;
            const int ivLen = 16;
            const int tagLen = 32;

            try
            {
                using (var fsIn = new
FileStream(inputFile, FileMode.Open,
FileAccess.Read, FileShare.Read))
                {
                    if (fsIn.Length <
FileMagic.Length + 1 + ephLen + ivLen +
tagLen + 1)
                        return false;

                    byte[] magic = new
byte[FileMagic.Length];
                    if (fsIn.Read(magic,
0, magic.Length) != magic.Length) return
false;
                    if
(!magic.SequenceEqual(FileMagic)) return
false;

                    int ver =
fsIn.ReadByte();
                    if (ver !=
FileVersion) return false;

                    byte[] ephPubBytes =
new byte[ephLen];
                    byte[] iv = new
byte[ivLen];

                    if
(fsIn.Read(ephPubBytes, 0, ephLen) !=
ephLen) return false;
                    if (fsIn.Read(iv, 0,
ivLen) != ivLen) return false;

                    long headerLen =
FileMagic.Length + 1 + ephLen + ivLen;
                    long cipherLen =
fsIn.Length - headerLen - tagLen;
                    if (cipherLen < 0)
return false;

                    fsIn.Position =
fsIn.Length - tagLen;
                    byte[] expectedTag =
new byte[tagLen];
                    if
(fsIn.Read(expectedTag, 0, tagLen) !=
tagLen) return false;

                    var qPoint =
CurveParams.Curve.DecodePoint(ephPubBytes
);
                    var ephPub = new
ECPublicKeyParameters(qPoint, Domain);

                    byte[] sharedSecret =
ComputeSharedSecret(recipientPrivateKey,
ephPub);

DeriveKeys(sharedSecret, out byte[]
encKey16, out byte[] macKey32);

                    byte[] actualTag;
                    using (var hmac = new
HMACSHA256(macKey32))
                    {

hmac.TransformBlock(FileMagic, 0,
FileMagic.Length, null, 0);

hmac.TransformBlock(new[] { (byte)ver },
0, 1, null, 0);

hmac.TransformBlock(ephPubBytes, 0,
ephPubBytes.Length, null, 0);

hmac.TransformBlock(iv, 0, iv.Length,
null, 0);

                        fsIn.Position =
headerLen;
                        byte[] buffer =
new byte[1024 * 1024];
                        long remaining =
cipherLen;
```

```
                        while (remaining
> 0)
                        {
                            int toRead =
(int)Math.Min(buffer.Length, remaining);
                            int read =
fsIn.Read(buffer, 0, toRead);
                            if (read <=
0) return false;

hmac.TransformBlock(buffer, 0, read,
null, 0);
                            remaining -=
read;
                        }

hmac.TransformFinalBlock(Array.Empty<byte
>(), 0, 0);
                        actualTag =
hmac.Hash;
                    }

                    if
(!FixedTimeEquals(expectedTag,
actualTag))
                        return false;

                    fsIn.Position =
headerLen;

                    using (var
limitedCipherStream = new
LimitedReadStream(fsIn, cipherLen,
leaveOpen: true))
                    using (var aes =
Aes.Create())
                    {
                        aes.KeySize =
128;
                        aes.BlockSize =
128;
                        aes.Mode =
CipherMode.CBC;
                        aes.Padding =
PaddingMode.PKCS7;
                        aes.Key =
encKey16;
                        aes.IV = iv;

                        using (var
cryptoIn = new
CryptoStream(limitedCipherStream,
aes.CreateDecryptor(),
CryptoStreamMode.Read))
                        using (var fsOut
= new FileStream(outputFile,
FileMode.Create, FileAccess.Write,
FileShare.None))
                        {
                            byte[] outBuf
= new byte[1024 * 1024];
                            int read;
                            while ((read
= cryptoIn.Read(outBuf, 0,
outBuf.Length)) > 0)
fsOut.Write(outBuf, 0, read);
                        }
                    }

                    return true;
                }
            }
            catch
(CryptographicException)
            {
                return false;
            }
            catch
            {

SafeDeleteFile(outputFile);
                throw;
            }
        }

        private static byte[]
ComputeSharedSecret(ECPrivateKeyParameter
s privateKey, ECPublicKeyParameters
publicKey)
        {
            var agree = new
ECDHBasicAgreement();
            agree.Init(privateKey);
            BigInteger shared =
agree.CalculateAgreement(publicKey);
            return
shared.ToByteArrayUnsigned();
        }

        private static void
DeriveKeys(byte[] sharedSecret, out
byte[] aesKey16, out byte[] macKey32)
        {
            aesKey16 =
Sha256Concat(sharedSecret,
Encoding.ASCII.GetBytes("enc")).Take(16).
ToArray();
            macKey32 =
Sha256Concat(sharedSecret,
Encoding.ASCII.GetBytes("mac"));
        }

        private static byte[]
Sha256Concat(byte[] a, byte[] b)
        {
            using (var sha =
SHA256.Create())
            {
                byte[] data = new
byte[a.Length + b.Length];
                Buffer.BlockCopy(a, 0,
data, 0, a.Length);
                Buffer.BlockCopy(b, 0,
data, a.Length, b.Length);
                return
sha.ComputeHash(data);
            }
        }

        private static byte[]
ComputeHmac(byte[] macKey, byte[] header,
```

```
byte[] data)
        {
            using (var hmac = new
HMACSHA256(macKey))
            {

hmac.TransformBlock(header, 0,
header.Length, null, 0);
                hmac.TransformBlock(data,
0, data.Length, null, 0);

hmac.TransformFinalBlock(Array.Empty<byte
>(), 0, 0);
                return hmac.Hash;
            }
        }

        private static bool
FixedTimeEquals(byte[] a, byte[] b)
        {
            if (a == null || b == null ||
a.Length != b.Length) return false;
            int diff = 0;
            for (int i = 0; i < a.Length;
i++) diff |= (a[i] ^ b[i]);
            return diff == 0;
        }

        private static long
GetTotalSizeRecursive(string folderPath)
        {
            long total = 0;
            foreach (var file in
Directory.GetFiles(folderPath, "*",
SearchOption.AllDirectories))
            {
                try { total += new
FileInfo(file).Length; } catch { }
            }
            return total;
        }

        private static string
GetRelativePathCompat(string basePath,
string fullPath)
        {
            basePath =
Path.GetFullPath(basePath);
            fullPath =
Path.GetFullPath(fullPath);

            if
(!basePath.EndsWith(Path.DirectorySeparat
orChar.ToString()) &&

!basePath.EndsWith(Path.AltDirectorySepar
atorChar.ToString()))
            {
                basePath +=
Path.DirectorySeparatorChar;
            }

            var baseUri = new
Uri(basePath, UriKind.Absolute);
            var fullUri = new
Uri(fullPath, UriKind.Absolute);
            if (baseUri.Scheme !=
fullUri.Scheme)
                return fullPath;

            string rel =
Uri.UnescapeDataString(baseUri.MakeRelati
veUri(fullUri).ToString());
            return rel.Replace('/',
Path.DirectorySeparatorChar);
        }

        private static void
SafeDeleteFile(string path)
        {
            try { if (File.Exists(path))
File.Delete(path); } catch { }
        }

        private string
ExportPublicKey(ECPublicKeyParameters
pub)
        {
            byte[] q =
pub.Q.GetEncoded(false);
            return
Convert.ToBase64String(q);
        }

        private string
ExportPrivateKey(ECPrivateKeyParameters
priv)
        {
            byte[] d =
priv.D.ToByteArrayUnsigned();
            return
Convert.ToBase64String(d);
        }

        private ECPublicKeyParameters
ImportPublicKey(string text)
        {
            byte[] qBytes =
Convert.FromBase64String(text);
            var qPoint =
CurveParams.Curve.DecodePoint(qBytes);
            return new
ECPublicKeyParameters(qPoint, Domain);
        }

        private ECPrivateKeyParameters
ImportPrivateKey(string text)
        {
            byte[] dBytes =
Convert.FromBase64String(text);
            var d = new BigInteger(1,
dBytes);
            return new
ECPrivateKeyParameters(d, Domain);
        }

        private sealed class
HmacWriteStream : Stream
        {
            private readonly Stream
_baseStream;
            private readonly HMAC _hmac;
            private readonly bool
```

```
_leaveOpen;

            public HmacWriteStream(Stream
baseStream, HMAC hmac, bool leaveOpen)
            {
                _baseStream = baseStream
?? throw new
ArgumentNullException(nameof(baseStream))
;
                _hmac = hmac ?? throw new
ArgumentNullException(nameof(hmac));
                _leaveOpen = leaveOpen;
            }

            public override bool CanRead
=> false;
            public override bool CanSeek
=> false;
            public override bool CanWrite
=> _baseStream.CanWrite;
            public override long Length
=> _baseStream.Length;
            public override long Position
{ get => _baseStream.Position; set =>
throw new NotSupportedException(); }

            public override void Flush()
=> _baseStream.Flush();
            public override int
Read(byte[] buffer, int offset, int
count) => throw new
NotSupportedException();
            public override long
Seek(long offset, SeekOrigin origin) =>
throw new NotSupportedException();
            public override void
SetLength(long value) =>
_baseStream.SetLength(value);

            public override void
Write(byte[] buffer, int offset, int
count)
            {
                if (count > 0)

_hmac.TransformBlock(buffer, offset,
count, null, 0);

                _baseStream.Write(buffer,
offset, count);
            }

            protected override void
Dispose(bool disposing)
            {
                if (disposing &&
!_leaveOpen)

_baseStream.Dispose();
                base.Dispose(disposing);
            }
        }

        private sealed class
LimitedReadStream : Stream
        {
            private readonly Stream
_baseStream;
            private long _remaining;
            private readonly bool
_leaveOpen;

            public
LimitedReadStream(Stream baseStream, long
length, bool leaveOpen)
            {
                _baseStream = baseStream
?? throw new
ArgumentNullException(nameof(baseStream))
;
                _remaining = length < 0 ?
0 : length;
                _leaveOpen = leaveOpen;
            }

            public override bool CanRead
=> true;
            public override bool CanSeek
=> false;
            public override bool CanWrite
=> false;
            public override long Length
=> _remaining;
            public override long Position
{ get => 0; set => throw new
NotSupportedException(); }

            public override void Flush()
{ }

            public override int
Read(byte[] buffer, int offset, int
count)
            {
                if (_remaining <= 0)
return 0;

                int toRead =
(int)Math.Min(count, _remaining);
                int read =
_baseStream.Read(buffer, offset, toRead);
                _remaining -= read;
                return read;
            }

            public override long
Seek(long offset, SeekOrigin origin) =>
throw new NotSupportedException();
            public override void
SetLength(long value) => throw new
NotSupportedException();
            public override void
Write(byte[] buffer, int offset, int
count) => throw new
NotSupportedException();

            protected override void
Dispose(bool disposing)
            {
                if (disposing &&
!_leaveOpen)

_baseStream.Dispose();
                base.Dispose(disposing);
```

```
            }
        }
    }
```

2.8 ElGamalForm.cs

```
using System;
using System.Diagnostics;
using System.Security.Cryptography;
using System.Text;
using System.Windows.Forms;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Engines;
using Org.BouncyCastle.Crypto.Generators;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Math;
using Org.BouncyCastle.Security;

namespace CruptoBench
{
    public partial class ElGamalForm :
Form
    {
        private
ElGamalPublicKeyParameters _publicKey;
        private
ElGamalPrivateKeyParameters _privateKey;

        private const int IterationCount
= 1000;
        private const int ParameterBits =
1024;
        private const int
ParameterCertainty = 20;

        private static readonly byte[]
PayloadMagic = { (byte)'E', (byte)'L',
(byte)'G', (byte)'1' };
        private const int
PayloadHeaderSize = 4 + 4;
        private const int PayloadTagSize
= 16;

        public ElGamalForm()
        {
            InitializeComponent();
        }

        private void
btnGenerateKeys_Click(object sender,
EventArgs e)
        {
            try
            {
                var rng = new
SecureRandom();

                var process =
Process.GetCurrentProcess();
                process.Refresh();

                var sw =
Stopwatch.StartNew();
                var cpuStart =
process.TotalProcessorTime;
                var memStart =
process.WorkingSet64;
            }
        }
```

```
                var paramGen = new
ElGamalParametersGenerator();

paramGen.Init(ParameterBits,
ParameterCertainty, rng);
                ElGamalParameters
parameters =
paramGen.GenerateParameters();

                var keyGen = new
ElGamalKeyPairGenerator();
                keyGen.Init(new
ElGamalKeyGenerationParameters(rng,
parameters));
                AsymmetricCipherKeyPair
keyPair = keyGen.GenerateKeyPair();

                _publicKey =
(ElGamalPublicKeyParameters)keyPair.Publi
c;
                _privateKey =
(ElGamalPrivateKeyParameters)keyPair.Priv
ate;

                sw.Stop();
                process.Refresh();

                var cpuEnd =
process.TotalProcessorTime;
                var memEnd =
process.WorkingSet64;

                double totalMs =
sw.Elapsed.TotalMilliseconds;
                double cpuPercent =
totalMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * totalMs) *
100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                txtPublicKey.Text =
ExportPublicKey(_publicKey);
                txtPrivateKey.Text =
ExportPrivateKey(_privateKey);

                string info =
                    $"Генерація ключів
ElGamal завершена.\n" +
                    $"Розмір модуля p:
{parameters.P.BitLength} біт\n" +
                    $"Час генерації:
{totalMs:F2} мс\n" +
                    $"Завантаження CPU:
{cpuPercent:F2}%\n" +
```

```
                $"Додаткова пам'ять:
{memMb:F2} МБ.";

                txtInfo.Text =
$"Генерація: {totalMs:F2} мс, p =
{parameters.P.BitLength} біт";
                MessageBox.Show(info,
"Генерація ключів ElGamal");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
під час генерації ключів: " +
ex.Message);
            }
        }

        private void
btnEncrypt_Click(object sender, EventArgs
e)
        {
            try
            {
                if
(string.IsNullOrWhiteSpace(txtPublicKey.T
ext))
                {

MessageBox.Show("Спочатку згенеруйте або
вставте відкритий ключ.");
                    return;
                }

                if
(string.IsNullOrWhiteSpace(txtPlainText.T
ext))
                {

MessageBox.Show("Введіть текст для
шифрування.");
                    return;
                }

                _publicKey =
ImportPublicKey(txtPublicKey.Text);

                byte[] plainBytes =
Encoding.UTF8.GetBytes(txtPlainText.Text)
;
                byte[] payload =
BuildPayload(plainBytes);

                var engine = new
ElGamalEngine();
                var rng = new
SecureRandom();
                engine.Init(true, new
ParametersWithRandom(_publicKey, rng));

                int maxInput =
engine.GetInputBlockSize();
                int overhead =
PayloadHeaderSize + PayloadTagSize;
                int maxTextBytes =
Math.Max(0, maxInput - overhead);

                if (payload.Length >
maxInput)
                {
                    MessageBox.Show(
                        "Текст занадто
довгий для прямого ElGamal.\n" +
                        $"Максимум для
цієї ключової пари: {maxTextBytes} байт
(UTF-8).\n" +
                        $"Зараз:
{plainBytes.Length} байт.",
                        "Помилка"
                    );
                    return;
                }

                var process =
Process.GetCurrentProcess();
                process.Refresh();

                var sw =
Stopwatch.StartNew();
                var cpuStart =
process.TotalProcessorTime;
                var memStart =
process.WorkingSet64;

                byte[] cipherBytes =
null;

                for (int i = 0; i <
IterationCount; i++)
                    cipherBytes =
engine.ProcessBlock(payload, 0,
payload.Length);

                sw.Stop();
                process.Refresh();

                var cpuEnd =
process.TotalProcessorTime;
                var memEnd =
process.WorkingSet64;

                txtEncrypted.Text =
Convert.ToBase64String(cipherBytes);

                double totalMs =
sw.Elapsed.TotalMilliseconds;
                double avgMs = totalMs /
IterationCount;

                double cpuPercent =
totalMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * totalMs) *
100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                long plainLen =
plainBytes.Length;
                long cipherLen =
cipherBytes.Length;
```

```
                double expansion =
plainLen > 0 ? (double)cipherLen /
plainLen : 0.0;

                string info =
                    $"Шифрування ElGamal
({IterationCount} повторень)\n\n" +
                    $"Загальний час:
{totalMs:F2} мс\n" +
                    $"Середній час на 1
операцію: {avgMs:F4} мс\n\n" +
                    $"Розмір відкритого
тексту: {plainLen} байт\n" +
                    $"Розмір шифротексту:
{cipherLen} байт\n" +
                    $"Коефіцієнт
розширення: {expansion:F3} раз(и)\n\n" +
                    $"Завантаження CPU:
{cpuPercent:F2}%\n" +
                    $"Додаткова пам'ять:
{memMb:F2} МБ.";

                txtInfo.Text =
$"Шифрування: {avgMs:F4} мс/оп.,
{plainLen} → {cipherLen} байт
(x{expansion:F3})";
                MessageBox.Show(info,
"Статистика ElGamal (шифрування)");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
під час шифрування: " + ex.Message);
            }
        }

        private void
btnDecrypt_Click(object sender, EventArgs
e)
        {
            try
            {
                if
(string.IsNullOrWhiteSpace(txtPrivateKey.
Text))
                {

MessageBox.Show("Спочатку вставте
закритий ключ.");
                    return;
                }

                if
(string.IsNullOrWhiteSpace(txtEncrypted.T
ext))
                {

MessageBox.Show("Немає даних для
розшифрування.");
                    return;
                }

                _privateKey =
ImportPrivateKey(txtPrivateKey.Text);

                byte[] cipherBytes;
                try
                {
                    cipherBytes =
Convert.FromBase64String(txtEncrypted.Tex
t.Trim());
                }
                catch
                {

MessageBox.Show("Невірний формат
шифротексту (очікується Base64).");
                    return;
                }

                var engine = new
ElGamalEngine();
                engine.Init(false,
_privateKey);

                byte[] firstPayload;
                try
                {
                    firstPayload =
engine.ProcessBlock(cipherBytes, 0,
cipherBytes.Length);
                }
                catch
                {

MessageBox.Show("Неверний ключ");
                    return;
                }

                if
(!TryParsePayload(firstPayload, out _))
                {

MessageBox.Show("Неверний ключ");
                    return;
                }

                var process =
Process.GetCurrentProcess();
                process.Refresh();

                var sw =
Stopwatch.StartNew();
                var cpuStart =
process.TotalProcessorTime;
                var memStart =
process.WorkingSet64;

                byte[] lastPayload =
null;

                for (int i = 0; i <
IterationCount; i++)
                    lastPayload =
engine.ProcessBlock(cipherBytes, 0,
cipherBytes.Length);

                sw.Stop();
                process.Refresh();

                var cpuEnd =
process.TotalProcessorTime;
                var memEnd =
process.WorkingSet64;
```

```
                if
(!TryParsePayload(lastPayload, out byte[]
plainBytes))
                {

MessageBox.Show("Неверний ключ");
                    return;
                }

                txtDecrypted.Text =
Encoding.UTF8.GetString(plainBytes);

                double totalMs =
sw.Elapsed.TotalMilliseconds;
                double avgMs = totalMs /
IterationCount;

                double cpuPercent =
totalMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * totalMs) *
100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                long plainLen =
plainBytes.Length;
                long cipherLen =
cipherBytes.Length;
                double expansion =
plainLen > 0 ? (double)cipherLen /
plainLen : 0.0;

                string info =
                    $"Розшифрування
ElGamal ({IterationCount} повторень)\n\n"
+
                    $"Загальний час:
{totalMs:F2} мс\n" +
                    $"Середній час на 1
операцію: {avgMs:F4} мс\n\n" +
                    $"Розмір шифротексту:
{cipherLen} байт\n" +
                    $"Розмір
розшифрованого тексту: {plainLen} байт\n"
+
                    $"Коефіцієнт
розширення: {expansion:F3} раз(и)\n\n" +
                    $"Завантаження CPU:
{cpuPercent:F2}%\n" +
                    $"Додаткова пам'ять:
{memMb:F2} МБ.";

                txtInfo.Text =
$"Розшифрування: {avgMs:F4} мс/оп.,
{cipherLen} → {plainLen} байт";
                MessageBox.Show(info,
"Статистика ElGamal (розшифрування)");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
під час розшифрування: " + ex.Message);
            }
        }

        private static byte[]
BuildPayload(byte[] plainBytes)
        {
            if (plainBytes == null)
plainBytes = Array.Empty<byte>();

            byte[] header = new
byte[PayloadHeaderSize];

Buffer.BlockCopy(PayloadMagic, 0, header,
0, 4);

            int len = plainBytes.Length;
            header[4] = (byte)((len >>
24) & 0xFF);
            header[5] = (byte)((len >>
16) & 0xFF);
            header[6] = (byte)((len >> 8)
& 0xFF);
            header[7] = (byte)(len &
0xFF);

            byte[] tag =
ComputeTag(header, plainBytes);

            byte[] payload = new
byte[header.Length + plainBytes.Length +
tag.Length];
            Buffer.BlockCopy(header, 0,
payload, 0, header.Length);
            Buffer.BlockCopy(plainBytes,
0, payload, header.Length,
plainBytes.Length);
            Buffer.BlockCopy(tag, 0,
payload, header.Length +
plainBytes.Length, tag.Length);

            return payload;
        }

        private static bool
TryParsePayload(byte[] payload, out
byte[] plainBytes)
        {
            plainBytes = null;

            if (payload == null ||
payload.Length < PayloadHeaderSize +
PayloadTagSize)
                return false;

            if (payload[0] !=
PayloadMagic[0] ||
                payload[1] !=
PayloadMagic[1] ||
                payload[2] !=
PayloadMagic[2] ||
                payload[3] !=
PayloadMagic[3])
                return false;

            int len =
                (payload[4] << 24) |
```

```
                (payload[5] << 16) |
                (payload[6] << 8) |
                payload[7];

            if (len < 0)
                return false;

            int expectedTotal =
PayloadHeaderSize + len + PayloadTagSize;
            if (payload.Length !=
expectedTotal)
                return false;

            plainBytes = new byte[len];
            Buffer.BlockCopy(payload,
PayloadHeaderSize, plainBytes, 0, len);

            byte[] header = new
byte[PayloadHeaderSize];
            Buffer.BlockCopy(payload, 0,
header, 0, PayloadHeaderSize);

            byte[] expectedTag =
ComputeTag(header, plainBytes);

            byte[] actualTag = new
byte[PayloadTagSize];
            Buffer.BlockCopy(payload,
PayloadHeaderSize + len, actualTag, 0,
PayloadTagSize);

            if
(!FixedTimeEquals(expectedTag,
actualTag))
                return false;

            return true;
        }

        private static byte[]
ComputeTag(byte[] header, byte[] data)
        {
            using (var sha =
SHA256.Create())
            {

sha.TransformBlock(header, 0,
header.Length, null, 0);

sha.TransformFinalBlock(data, 0,
data.Length);
                byte[] hash = sha.Hash;

                byte[] tag = new
byte[PayloadTagSize];
                Buffer.BlockCopy(hash, 0,
tag, 0, PayloadTagSize);
                return tag;
            }
        }

        private static bool
FixedTimeEquals(byte[] a, byte[] b)
        {
            if (a == null || b == null ||
a.Length != b.Length) return false;
            int diff = 0;
            for (int i = 0; i < a.Length;
i++) diff |= (a[i] ^ b[i]);
            return diff == 0;
        }

        private string
ExportPublicKey(ElGamalPublicKeyParameter
s pub)
        {
            string p =
ToB64(pub.Parameters.P);
            string g =
ToB64(pub.Parameters.G);
            string y = ToB64(pub.Y);
            return $"{p}|{g}|{y}";
        }

        private string
ExportPrivateKey(ElGamalPrivateKeyParamet
ers priv)
        {
            string p =
ToB64(priv.Parameters.P);
            string g =
ToB64(priv.Parameters.G);
            string x = ToB64(priv.X);
            return $"{p}|{g}|{x}";
        }

        private
ElGamalPublicKeyParameters
ImportPublicKey(string text)
        {
            var parts = SplitKey(text, 3,
"Невірний формат відкритого ключа.");
            var p = FromB64(parts[0],
"Невірний формат P у відкритому ключі.");
            var g = FromB64(parts[1],
"Невірний формат G у відкритому ключі.");
            var y = FromB64(parts[2],
"Невірний формат Y у відкритому ключі.");

            var parameters = new
ElGamalParameters(p, g);
            return new
ElGamalPublicKeyParameters(y,
parameters);
        }

        private
ElGamalPrivateKeyParameters
ImportPrivateKey(string text)
        {
            var parts = SplitKey(text, 3,
"Невірний формат закритого ключа.");
            var p = FromB64(parts[0],
"Невірний формат P у закритому ключі.");
            var g = FromB64(parts[1],
"Невірний формат G у закритому ключі.");
            var x = FromB64(parts[2],
"Невірний формат X у закритому ключі.");

            var parameters = new
ElGamalParameters(p, g);
            return new
ElGamalPrivateKeyParameters(x,
parameters);
```

```
        }

        private static string[]
SplitKey(string text, int expectedParts,
string errorMessage)
        {
            if
(string.IsNullOrWhiteSpace(text))
                throw new
Exception(errorMessage);

            var parts =
text.Trim().Split('|');
            if (parts.Length !=
expectedParts)
                throw new
Exception(errorMessage);

            return parts;
        }

        private static string
ToB64(BigInteger value)
        {
            return
Convert.ToBase64String(value.ToByteArrayU
nsigned());
        }

        private static BigInteger
FromB64(string b64, string errorMessage)
        {
            try
            {
                return new BigInteger(1,
Convert.FromBase64String(b64));
            }
            catch
            {
                throw new
Exception(errorMessage);
            }
        }
    }
}
```

2.9 MainForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using System.Windows.Forms;

namespace CruptoBench
{
    public partial class MainMenuForm :
Form
    {
        public MainMenuForm()
        {
            InitializeComponent();
        }

        private void button1_Click(object
sender, EventArgs e)
        {
            AESForm aesForm = new
AESForm();
            aesForm.ShowDialog();
        }

        private void button2_Click(object
sender, EventArgs e)
        {
            DESForm aesForm = new
DESForm();
            aesForm.ShowDialog();
        }

        private void button3_Click(object
sender, EventArgs e)
        {
            RSAForm aesForm = new
RSAForm();
            aesForm.ShowDialog();
        }

        private void button4_Click(object
sender, EventArgs e)
        {
            ChaCha20Form aesForm = new
ChaCha20Form();
            aesForm.ShowDialog();
        }

        private void button5_Click(object
sender, EventArgs e)
        {
            SerpentForm aesForm = new
SerpentForm();
            aesForm.ShowDialog();
        }

        private void button6_Click(object
sender, EventArgs e)
        {
            TripleDESForm aesForm = new
TripleDESForm();
            aesForm.ShowDialog();
        }

        private void button7_Click(object
sender, EventArgs e)
        {
            BlowfishForm aesForm = new
BlowfishForm();
            aesForm.ShowDialog();
        }

        private void button8_Click(object
sender, EventArgs e)
        {
```

```
            ElGamalForm aesForm = new
ElGamalForm();
            aesForm.ShowDialog();
        }

        private void button9_Click(object
sender, EventArgs e)
        {
            ECCForm aesForm = new
ECCForm();
            aesForm.ShowDialog();
        }

        private void
button10_Click(object sender, EventArgs
e)
        {
            AES12Form aesForm = new
AES12Form();
            aesForm.ShowDialog();
        }
    }
}
```

2.10 RSAForm.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using System.Windows.Forms;

namespace CruptoBench
{
    public partial class RSAForm : Form
    {
        private RSACryptoServiceProvider
_rsaProvider;

        private const int IterationCount
= 1000;
        private const int DefaultKeyBits
= 4096;

        public RSAForm()
        {
            InitializeComponent();

            _rsaProvider = new
RSACryptoServiceProvider(DefaultKeyBits);
            this.FormClosed +=
RSAForm_FormClosed;
        }

        private void
RSAForm_FormClosed(object sender,
FormClosedEventArgs e)
        {
            try {
_rsaProvider?.Dispose(); } catch { }
            _rsaProvider = null;
        }

        private void
btnGenerateKeys_Click(object sender,
EventArgs e)
        {
            try
            {
                var process =
Process.GetCurrentProcess();
                process.Refresh();

                var sw =
Stopwatch.StartNew();
                TimeSpan cpuStart =
process.TotalProcessorTime;
                long memStart =
process.WorkingSet64;

                try {
_rsaProvider?.Dispose(); } catch { }
                _rsaProvider = new
RSACryptoServiceProvider(DefaultKeyBits);

                string publicKeyXml =
_rsaProvider.ToXmlString(false);
                string privateKeyXml =
_rsaProvider.ToXmlString(true);
                int keyBits =
_rsaProvider.KeySize;

                sw.Stop();
                process.Refresh();

                TimeSpan cpuEnd =
process.TotalProcessorTime;
                long memEnd =
process.WorkingSet64;

                double elapsedMs =
sw.Elapsed.TotalMilliseconds;
                double cpuPercent =
elapsedMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * elapsedMs)
* 100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                txtPublicKey.Text =
publicKeyXml;
                txtPrivateKey.Text =
privateKeyXml;

                MessageBox.Show(
                    "Генерація ключів
RSA\n\n" +
                    $"Довжина ключа:
{keyBits} біт\n" +
                    $"Час генерації:
{elapsedMs:F2} мс\n" +
                    $"Завантаження CPU:
```

```
{cpuPercent:F2}%\n" +
                    $"Додаткова пам'ять:
{memMb:F2} МБ.",
                    "Генерація ключів
RSA",
                    MessageBoxButtons.OK,

MessageBoxIcon.Information
                );
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
під час генерації ключів: " +
ex.Message);
            }
        }

        private void
btnEncrypt_Click(object sender, EventArgs
e)
        {
            try
            {
                if
(string.IsNullOrWhiteSpace(txtPublicKey.T
ext))
                {

MessageBox.Show("Спочатку вставте або
згенеруйте відкритий ключ.");
                    return;
                }

                if
(string.IsNullOrWhiteSpace(txtPlainText.T
ext))
                {

MessageBox.Show("Введіть текст для
шифрування.");
                    return;
                }



_rsaProvider.FromXmlString(txtPublicKey.T
ext.Trim());
                int keyBits =
_rsaProvider.KeySize;

                byte[] plainBytes =
Encoding.UTF8.GetBytes(txtPlainText.Text)
;

                bool useOaep = true;
                int maxPlainBytes =
GetMaxRsaMessageLengthBytes(_rsaProvider.
KeySize, useOaep);

                if (plainBytes.Length >
maxPlainBytes)
                {
                    MessageBox.Show(
                        "Текст занадто
довгий для прямого RSA.\n" +
                        $"Максимум для
RSA-{keyBits} (OAEP SHA-1):
{maxPlainBytes} байт.\n" +
                        $"Зараз:
{plainBytes.Length} байт.\n\n" +
                        "Порада: для
великих даних використовуй гібридне
шифрування (AES + RSA).",
                        "Помилка"
                    );
                    return;
                }

                var process =
Process.GetCurrentProcess();
                process.Refresh();

                var sw =
Stopwatch.StartNew();
                TimeSpan cpuStart =
process.TotalProcessorTime;
                long memStart =
process.WorkingSet64;

                byte[] cipherBytes =
null;
                for (int i = 0; i <
IterationCount; i++)
                    cipherBytes =
_rsaProvider.Encrypt(plainBytes,
useOaep);

                sw.Stop();
                process.Refresh();

                TimeSpan cpuEnd =
process.TotalProcessorTime;
                long memEnd =
process.WorkingSet64;

                txtEncrypted.Text =
Convert.ToBase64String(cipherBytes);

                int plainLen =
plainBytes.Length;
                int cipherLen =
cipherBytes.Length;
                double expansion =
plainLen > 0 ? (double)cipherLen /
plainLen : 0.0;

                double elapsedMs =
sw.Elapsed.TotalMilliseconds;
                double avgMs = elapsedMs
/ IterationCount;

                double cpuPercent =
elapsedMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * elapsedMs)
* 100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
                if (memMb < 0) memMb = 0;

                MessageBox.Show(
```

```
                    $"Шифрування RSA
({IterationCount} повторень)\n\n" +
                    $"Довжина ключа:
{keyBits} біт\n" +
                    $"Padding: OAEP(SHA-
1)\n\n" +
                    $"Розмір відкритого
тексту: {plainLen} байт\n" +
                    $"Розмір
зашифрованого тексту: {cipherLen} байт\n"
+
                    $"Коефіцієнт
розширення: {expansion:F3} раз(и)\n\n" +
                    $"Загальний час:
{elapsedMs:F2} мс\n" +
                    $"Середній час на 1
операцію: {avgMs:F4} мс\n" +
                    $"Завантаження CPU:
{cpuPercent:F2}%\n" +
                    $"Додаткова пам'ять:
{memMb:F2} МБ.",
                    "Статистика
шифрування RSA",
                    MessageBoxButtons.OK,

MessageBoxIcon.Information
                );
            }
            catch (CryptographicException
ex)
            {
                MessageBox.Show("Помилка
шифрування: " + ex.Message);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
шифрування: " + ex.Message);
            }
        }

        private void
btnDecrypt_Click(object sender, EventArgs
e)
        {
            try
            {
                if
(string.IsNullOrWhiteSpace(txtPrivateKey.
Text))
                {

MessageBox.Show("Спочатку вставте
закритий ключ.");
                    return;
                }

                if
(string.IsNullOrWhiteSpace(txtEncrypted.T
ext))
                {

MessageBox.Show("Немає зашифрованого
тексту для розшифрування.");
                    return;
                }

_rsaProvider.FromXmlString(txtPrivateKey.
Text.Trim());
                int keyBits =
_rsaProvider.KeySize;

                byte[] cipherBytes;
                try
                {
                    cipherBytes =
Convert.FromBase64String(txtEncrypted.Tex
t.Trim());
                }
                catch
                {

MessageBox.Show("Невірний формат
шифротексту (очікується Base64).");
                    return;
                }

                bool useOaep = true;

                var process =
Process.GetCurrentProcess();
                process.Refresh();

                var sw =
Stopwatch.StartNew();
                TimeSpan cpuStart =
process.TotalProcessorTime;
                long memStart =
process.WorkingSet64;

                byte[] plainBytes = null;
                for (int i = 0; i <
IterationCount; i++)
                    plainBytes =
_rsaProvider.Decrypt(cipherBytes,
useOaep);

                sw.Stop();
                process.Refresh();

                TimeSpan cpuEnd =
process.TotalProcessorTime;
                long memEnd =
process.WorkingSet64;

                txtDecrypted.Text =
Encoding.UTF8.GetString(plainBytes);

                double elapsedMs =
sw.Elapsed.TotalMilliseconds;
                double avgMs = elapsedMs
/ IterationCount;

                double cpuPercent =
elapsedMs > 0
                    ? (cpuEnd -
cpuStart).TotalMilliseconds /
(Environment.ProcessorCount * elapsedMs)
* 100.0
                    : 0.0;

                double memMb = (memEnd -
memStart) / (1024.0 * 1024.0);
```

```
                if (memMb < 0) memMb = 0;

                MessageBox.Show(
                    $"Розшифрування RSA
({IterationCount} повторень)\n\n" +
                    $"Довжина ключа:
{keyBits} біт\n" +
                    $"Padding: OAEP(SHA-
1)\n\n" +
                    $"Загальний час:
{elapsedMs:F2} мс\n" +
                    $"Середній час на 1
операцію: {avgMs:F4} мс\n" +
                    $"Завантаження CPU:
{cpuPercent:F2}%\n" +
                    $"Додаткова пам'ять:
{memMb:F2} МБ.",
                    "Статистика
розшифрування RSA",
                    MessageBoxButtons.OK,

MessageBoxIcon.Information
                );
            }
            catch
(CryptographicException)
            {
                MessageBox.Show("Неверний
ключ");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка
розшифрування: " + ex.Message);
            }
        }

        private void
btnLoadFile_Click(object sender,
EventArgs e)
        {
            if
(openFileDialog.ShowDialog() !=
DialogResult.OK)
                return;

            string filePath =
openFileDialog.FileName;

            try
            {
                txtPlainText.Text =
File.ReadAllText(filePath,
Encoding.UTF8);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Не
вдалося прочитати файл: " + ex.Message);
            }
        }

        private static int
GetMaxRsaMessageLengthBytes(int keyBits,
bool oaepSha1)
        {
            int k = keyBits / 8;

            if (oaepSha1)
            {
                int hLen = 20;
                return k - 2 * hLen - 2;
            }

            return k - 11;
        }
    }
}
```

2.11 SerpentForm.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System.Windows.Forms;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Digests;
using Org.BouncyCastle.Crypto.Engines;
using Org.BouncyCastle.Crypto.Macs;
using Org.BouncyCastle.Crypto.Modes;
using Org.BouncyCastle.Crypto.Paddings;
using Org.BouncyCastle.Crypto.Parameters;

namespace CruptoBench
{
    public partial class SerpentForm :
Form
    {
        private byte[] _masterKey;

        private Stopwatch _stopwatch;
        private TimeSpan _cpuStart;
        private DateTime _timeStart;

        private double _peakCpu;
        private double _peakMemMb;
        private long _totalInputBytes;

        private const int KeyBytes = 32;
        private const int BlockBytes =
16;
        private const int TagBytes = 32;

        private static readonly byte[]
Magic = { (byte)'S', (byte)'E',
(byte)'R', (byte)'P', (byte)'1' };
        private const byte Version = 1;

        public SerpentForm()
        {
            InitializeComponent();

            txtFolder.AllowDrop = true;
            txtFolder.DragEnter +=
TxtFolder_DragEnter;
            txtFolder.DragDrop +=
TxtFolder_DragDrop;
```

```
        }

        private void
TxtFolder_DragEnter(object sender,
DragEventArgs e)
        {
            if (e.Data != null &&
e.Data.GetDataPresent(DataFormats.FileDro
p))
                e.Effect =
DragDropEffects.Copy;
        }

        private void
TxtFolder_DragDrop(object sender,
DragEventArgs e)
        {
            var items =
(string[])e.Data.GetData(DataFormats.File
Drop);
            if (items != null &&
items.Length > 0)
                txtFolder.Text =
items[0];
        }

        private void
btnGenerateKey_Click(object sender,
EventArgs e)
        {
            _masterKey = new
byte[KeyBytes];
            using (var rng =
RandomNumberGenerator.Create())
                rng.GetBytes(_masterKey);

            txtKey.Text =
Convert.ToBase64String(_masterKey);
        }

        private async void
btnEncrypt_Click(object sender, EventArgs
e)
        {
            await
RunOperationAsync(isEncrypt: true);
        }

        private async void
btnDecrypt_Click(object sender, EventArgs
e)
        {
            await
RunOperationAsync(isEncrypt: false);
        }

        private async Task
RunOperationAsync(bool isEncrypt)
        {
            string inputPath =
txtFolder.Text;

            if
(string.IsNullOrWhiteSpace(inputPath) ||
(!Directory.Exists(inputPath) &&
!File.Exists(inputPath)))
            {
                MessageBox.Show("Выберите
файл или папку.");
                return;
            }

            if (!TryReadKey(out byte[]
keyBytes))
            {
                MessageBox.Show("Неверный
формат ключа.");
                return;
            }

            _masterKey = keyBytes;

            _stopwatch =
Stopwatch.StartNew();
            _peakCpu = 0.0;
            _peakMemMb = 0.0;
            _totalInputBytes = 0;

            var process =
Process.GetCurrentProcess();
            process.Refresh();
            _cpuStart =
process.TotalProcessorTime;
            _timeStart = DateTime.UtcNow;

            long totalBytes = 0;
            if (File.Exists(inputPath))
            {
                totalBytes = new
FileInfo(inputPath).Length;
            }
            else
            {
                totalBytes =
GetTotalSizeRecursive(inputPath);
                if (totalBytes == 0)
                {

MessageBox.Show("Папка пустая, нет файлов
для обработки.");
                    return;
                }
            }

            progressBar.Minimum = 0;
            progressBar.Maximum = 100;
            progressBar.Value = 0;
            lblProgress.Text = "0%";

            try
            {
                if
(File.Exists(inputPath))
                {
                    string outputFile =
BuildOutputPathForFile(inputPath,
isEncrypt);
                    bool ok = await
Task.Run(() =>
ProcessSingleFile(inputPath, outputFile,
isEncrypt, totalBytes, 0));
                    if (!ok)
                    {
```

```
SafeDelete(outputFile);

MessageBox.Show("Неверний ключ");
                        return;
                    }

                    _totalInputBytes =
new FileInfo(inputPath).Length;

UpdateProgressBytes(totalBytes,
totalBytes);
                }
                else
                {
                    string inputFolder =
inputPath;
                    string outputFolder =
BuildOutputPathForFolder(inputFolder,
isEncrypt);

Directory.CreateDirectory(outputFolder);

                    var files =
Directory.GetFiles(inputFolder, "*",
SearchOption.AllDirectories);

                    long processed = 0;
                    foreach (var file in
files)
                    {
                        string relative =
GetRelativePathCompat(inputFolder, file);
                        string outFile =
Path.Combine(outputFolder, relative);

                        string outDir =
Path.GetDirectoryName(outFile);
                        if
(!string.IsNullOrEmpty(outDir))

Directory.CreateDirectory(outDir);

                        long fileSize =
0;
                        try { fileSize =
new FileInfo(file).Length; } catch { }

                        bool ok = await
Task.Run(() => ProcessSingleFile(file,
outFile, isEncrypt, totalBytes,
processed));
                        if (!ok)
                        {

SafeDelete(outFile);

MessageBox.Show("Неверний ключ");
                            return;
                        }

                        processed +=
fileSize;
                        _totalInputBytes
+= fileSize;


UpdateProgressBytes(processed,
totalBytes);
                    }


UpdateProgressBytes(totalBytes,
totalBytes);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Ошибка:
" + ex.Message);
                return;
            }
            finally
            {
                _stopwatch.Stop();
            }

            process.Refresh();

            double elapsedSec =
Math.Max(0.000001,
_stopwatch.Elapsed.TotalSeconds);
            double avgSpeed =
(_totalInputBytes / 1024.0 / 1024.0) /
elapsedSec;

            double cpuUsage =
(process.TotalProcessorTime -
_cpuStart).TotalMilliseconds /

(Environment.ProcessorCount *
(DateTime.UtcNow -
_timeStart).TotalMilliseconds) * 100.0;
            _peakCpu = Math.Max(_peakCpu,
cpuUsage);

            double memMb =
process.WorkingSet64 / (1024.0 * 1024.0);
            _peakMemMb =
Math.Max(_peakMemMb, memMb);

            string result =
string.Format(
                "Операция завершена за
{0:F2} с.\nПиковая загрузка CPU =
{1:F1}%, RAM = {2:F1} MB\nСредняя
скорость: {3:F2} МБ/с",
                elapsedSec, _peakCpu,
_peakMemMb, avgSpeed);

            txtLog.AppendText(result +
Environment.NewLine);
            MessageBox.Show(result);
        }

        private bool TryReadKey(out
byte[] keyBytes)
        {
            keyBytes = null;

            try
            {
                if
(string.IsNullOrWhiteSpace(txtKey.Text))
                    return false;
```

```
                byte[] raw =
Convert.FromBase64String(txtKey.Text.Trim
());
                if (raw.Length !=
KeyBytes)
                    return false;

                keyBytes = raw;
                return true;
            }
            catch
            {
                return false;
            }
        }

        private bool
ProcessSingleFile(string inputFile,
string outputFile, bool isEncrypt, long
totalBytes, long alreadyProcessedBytes)
        {
            if (isEncrypt)
            {

EncryptFileWithHeaderAndMac(inputFile,
outputFile, totalBytes,
alreadyProcessedBytes);
                return true;
            }

            return
TryDecryptFileWithHeaderAndMac(inputFile,
outputFile, totalBytes,
alreadyProcessedBytes);
        }

        private void
EncryptFileWithHeaderAndMac(string
inputFile, string outputFile, long
totalBytes, long alreadyProcessedBytes)
        {
            DeriveKeys(_masterKey, out
byte[] encKey, out byte[] macKey);

            byte[] iv = new
byte[BlockBytes];
            using (var rng =
RandomNumberGenerator.Create())
                rng.GetBytes(iv);

            IBlockCipher engine = new
SerpentEngine();
            var cipher = new
PaddedBufferedBlockCipher(new
CbcBlockCipher(engine));
            cipher.Init(true, new
ParametersWithIV(new
KeyParameter(encKey), iv));

            var hmac = new HMac(new
Sha256Digest());
            hmac.Init(new
KeyParameter(macKey));

            using (var fsIn = new
FileStream(inputFile, FileMode.Open,
FileAccess.Read, FileShare.Read))
                using (var fsOut = new
FileStream(outputFile, FileMode.Create,
FileAccess.Write, FileShare.None))
            {
                fsOut.Write(Magic, 0,
Magic.Length);
                fsOut.WriteByte(Version);
                fsOut.Write(iv, 0,
iv.Length);

                hmac.BlockUpdate(Magic,
0, Magic.Length);
                hmac.Update(Version);
                hmac.BlockUpdate(iv, 0,
iv.Length);

                byte[] inBuf = new
byte[1024 * 1024];
                byte[] outBuf = new
byte[cipher.GetOutputSize(inBuf.Length)];

                long processedLocal = 0;

                int read;
                while ((read =
fsIn.Read(inBuf, 0, inBuf.Length)) > 0)
                {
                    int outLen =
cipher.ProcessBytes(inBuf, 0, read,
outBuf, 0);
                    if (outLen > 0)
                    {

fsOut.Write(outBuf, 0, outLen);

hmac.BlockUpdate(outBuf, 0, outLen);
                    }

                    processedLocal +=
read;

UpdateProgressBytes(alreadyProcessedBytes
+ processedLocal, totalBytes);
                }

                int finalLen =
cipher.DoFinal(outBuf, 0);
                if (finalLen > 0)
                {
                    fsOut.Write(outBuf,
0, finalLen);

hmac.BlockUpdate(outBuf, 0, finalLen);
                }

                byte[] tag = new
byte[TagBytes];
                hmac.DoFinal(tag, 0);
                fsOut.Write(tag, 0,
tag.Length);
            }
        }

        private bool
TryDecryptFileWithHeaderAndMac(string
inputFile, string outputFile, long
```

```
totalBytes, long alreadyProcessedBytes)
        {
            DeriveKeys(_masterKey, out
byte[] encKey, out byte[] macKey);

            using (var fsIn = new
FileStream(inputFile, FileMode.Open,
FileAccess.Read, FileShare.Read))
            {
                if (fsIn.Length <
Magic.Length + 1 + BlockBytes + TagBytes
+ 1)
                    return false;

                byte[] magic = new
byte[Magic.Length];
                if (fsIn.Read(magic, 0,
magic.Length) != magic.Length) return
false;
                if
(!FixedTimeEquals(magic, Magic)) return
false;

                int ver =
fsIn.ReadByte();
                if (ver != Version)
return false;

                byte[] iv = new
byte[BlockBytes];
                if (fsIn.Read(iv, 0,
iv.Length) != iv.Length) return false;

                long headerLen =
Magic.Length + 1 + BlockBytes;
                long cipherLen =
fsIn.Length - headerLen - TagBytes;
                if (cipherLen <= 0)
return false;

                fsIn.Position =
fsIn.Length - TagBytes;
                byte[] expectedTag = new
byte[TagBytes];
                if
(fsIn.Read(expectedTag, 0,
expectedTag.Length) !=
expectedTag.Length) return false;

                var hmac = new HMac(new
Sha256Digest());
                hmac.Init(new
KeyParameter(macKey));

                hmac.BlockUpdate(Magic,
0, Magic.Length);
                hmac.Update((byte)ver);
                hmac.BlockUpdate(iv, 0,
iv.Length);

                fsIn.Position =
headerLen;

                byte[] buffer = new
byte[1024 * 1024];
                long remaining =
cipherLen;
                while (remaining > 0)
                {
                    int toRead =
(int)Math.Min(buffer.Length, remaining);
                    int read =
fsIn.Read(buffer, 0, toRead);
                    if (read <= 0) return
false;


hmac.BlockUpdate(buffer, 0, read);
                    remaining -= read;
                }

                byte[] actualTag = new
byte[TagBytes];
                hmac.DoFinal(actualTag,
0);

                if
(!FixedTimeEquals(expectedTag,
actualTag))
                    return false;

                fsIn.Position =
headerLen;

                IBlockCipher engine = new
SerpentEngine();
                var cipher = new
PaddedBufferedBlockCipher(new
CbcBlockCipher(engine));
                cipher.Init(false, new
ParametersWithIV(new
KeyParameter(encKey), iv));

                using (var fsOut = new
FileStream(outputFile, FileMode.Create,
FileAccess.Write, FileShare.None))
                using (var limited = new
LimitedReadStream(fsIn, cipherLen,
leaveOpen: true))
                {
                    byte[] inBuf = new
byte[1024 * 1024];
                    byte[] outBuf = new
byte[cipher.GetOutputSize(inBuf.Length)];

                    long processedLocal =
0;

                    int read;
                    while ((read =
limited.Read(inBuf, 0, inBuf.Length)) >
0)
                    {
                        int outLen =
cipher.ProcessBytes(inBuf, 0, read,
outBuf, 0);
                        if (outLen > 0)

fsOut.Write(outBuf, 0, outLen);

                        processedLocal +=
read;
```

```
UpdateProgressBytes(alreadyProcessedBytes
+ processedLocal, totalBytes);
                    }

                    int finalLen =
cipher.DoFinal(outBuf, 0);
                    if (finalLen > 0)

fsOut.Write(outBuf, 0, finalLen);
                }

                return true;
            }
        }

        private static void
DeriveKeys(byte[] masterKey, out byte[]
encKey, out byte[] macKey)
        {
            encKey =
Sha256Concat(masterKey, new byte[] {
(byte)'e', (byte)'n', (byte)'c' });
            macKey =
Sha256Concat(masterKey, new byte[] {
(byte)'m', (byte)'a', (byte)'c' });
        }

        private static byte[]
Sha256Concat(byte[] a, byte[] b)
        {
            using (var sha =
SHA256.Create())
            {
                byte[] data = new
byte[a.Length + b.Length];
                Buffer.BlockCopy(a, 0,
data, 0, a.Length);
                Buffer.BlockCopy(b, 0,
data, a.Length, b.Length);
                return
sha.ComputeHash(data);
            }
        }

        private void
UpdateProgressBytes(long processedBytes,
long totalBytes)
        {
            if (totalBytes <= 0)
totalBytes = 1;

            int percent =
(int)(processedBytes * 100 / totalBytes);
            if (percent < 0) percent = 0;
            if (percent > 100) percent =
100;

            if
(progressBar.IsHandleCreated)
            {

progressBar.Invoke((Action)(() =>
                {
                    progressBar.Value =
percent;
                    lblProgress.Text =
$"{percent}%";
                }));
            }
        }

        private static long
GetTotalSizeRecursive(string folderPath)
        {
            long total = 0;
            foreach (var file in
Directory.GetFiles(folderPath, "*",
SearchOption.AllDirectories))
            {
                try { total += new
FileInfo(file).Length; } catch { }
            }
            return total;
        }

        private static string
BuildOutputPathForFile(string inputFile,
bool isEncrypt)
        {
            string dir =
Path.GetDirectoryName(inputFile) ?? "";
            string name =
Path.GetFileNameWithoutExtension(inputFil
e);
            string ext =
Path.GetExtension(inputFile);
            string suffix = isEncrypt ?
"_encrypted" : "_decrypted";
            return Path.Combine(dir, name
+ suffix + ext);
        }

        private static string
BuildOutputPathForFolder(string
inputFolder, bool isEncrypt)
        {
            string suffix = isEncrypt ?
"_encrypted" : "_decrypted";
            return
inputFolder.TrimEnd(Path.DirectorySeparat
orChar, Path.AltDirectorySeparatorChar) +
suffix;
        }

        private static string
GetRelativePathCompat(string basePath,
string fullPath)
        {
            basePath =
Path.GetFullPath(basePath);
            fullPath =
Path.GetFullPath(fullPath);

            if
(!basePath.EndsWith(Path.DirectorySeparat
orChar.ToString()) &&

!basePath.EndsWith(Path.AltDirectorySepar
atorChar.ToString()))
            {
                basePath +=
Path.DirectorySeparatorChar;
            }
```

```
            var baseUri = new
Uri(basePath, UriKind.Absolute);
            var fullUri = new
Uri(fullPath, UriKind.Absolute);

            if (baseUri.Scheme !=
fullUri.Scheme)
                return fullPath;

            string rel =
Uri.UnescapeDataString(baseUri.MakeRelati
veUri(fullUri).ToString());
            return rel.Replace('/',
Path.DirectorySeparatorChar);
        }

        private static void
SafeDelete(string filePath)
        {
            try { if
(File.Exists(filePath))
File.Delete(filePath); } catch { }
        }

        private static bool
FixedTimeEquals(byte[] a, byte[] b)
        {
            if (a == null || b == null ||
a.Length != b.Length) return false;
            int diff = 0;
            for (int i = 0; i < a.Length;
i++) diff |= (a[i] ^ b[i]);
            return diff == 0;
        }

        private sealed class
LimitedReadStream : Stream
        {
            private readonly Stream
_baseStream;
            private long _remaining;
            private readonly bool
_leaveOpen;

            public
LimitedReadStream(Stream baseStream, long
length, bool leaveOpen)
            {
                _baseStream = baseStream
?? throw new
ArgumentNullException(nameof(baseStream))
;
                _remaining = length < 0 ?
0 : length;
                _leaveOpen = leaveOpen;
            }

            public override bool CanRead
=> true;
            public override bool CanSeek
=> false;
            public override bool CanWrite
=> false;
            public override long Length
=> _remaining;
            public override long Position
{ get => 0; set => throw new
NotSupportedException(); }

            public override void Flush()
{ }
            public override long
Seek(long offset, SeekOrigin origin) =>
throw new NotSupportedException();
            public override void
SetLength(long value) => throw new
NotSupportedException();
            public override void
Write(byte[] buffer, int offset, int
count) => throw new
NotSupportedException();

            public override int
Read(byte[] buffer, int offset, int
count)
            {
                if (_remaining <= 0)
return 0;

                int toRead =
(int)Math.Min(count, _remaining);
                int read =
_baseStream.Read(buffer, offset, toRead);
                _remaining -= read;
                return read;
            }

            protected override void
Dispose(bool disposing)
            {
                if (disposing &&
!_leaveOpen)

_baseStream.Dispose();
                base.Dispose(disposing);
            }
        }
    }
}
```

ДОДАТОК Г

Тези доповідей

Міністерство освіти і науки України

Український державний університет науки і технологій

ТЕЗИ
XIX Міжнародної науково-практичної конференції
«СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ»
Присвячено пам'яті Ігоря ЖУКОВИЦЬКОГО

ABSTRACTS
of the XIX International Conference
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
Dedicated to the memory of Igor ZHUKOVYTSKY

18.12.2025 – 19.12.2025

Дніпро
2025

Кіберзагрози операторам мобільного зв'язку під час надзвичайних подій.....158
Тимошенко Л.С., Український державний університет науки і технологій, Україна

Дослідження впливу способу управління станом Flutter додатку на його швидкодію та ресурсоємність 159
Кривоносов Ю. А., Шаравара В.В. Український державний університет науки і технологій. Україна.

Дослідження комбінованого варіанту виявлення DoS атак засобами нейронних мереж 160
Пахомова В.М., Метьолкін І.С., Український державний університет науки і технологій, Україна

Дослідження ефективності алгоритмів шифрування та дешифрування даних.....161
Нор О.С., Український державний університет науки і технологій, Україна

Застосування локальних мовних моделей для автоматизації формування функціональних профілів захищеності інформаційних систем.....162
Остапець Д. О., Дзюба В. В., Український державний університет науки і технологій, Україна

Розробка та дослідження менеджера паролів 163
Мілін Н. В., Остапець Д. О., Український державний університет науки і технологій, Україна

Огляд шкідливих програм та методів їх виявлення 164
Панін Д. В., Остапець Д. О., Український державний університет науки і технологій, Україна

Стегоаналіз при приховуванні інформації методом LSB з використанням графічних файлів–контейнерів.....165
Пірогов Д. А., Остапець Д. О. Український державний університет науки і технологій, Україна

Огляд алгоритмів постквантової криптографії.....166
Русакевич С.Р., Остапець Д.О., Український державний університет науки і технологій, Україна

Розробка та дослідження ефективності засобів стеганографічного аналізу інформації, прихованої в зображеннях.....167
Сасаров О.О., Остапець Д.О. Український державний університет науки і технологій, Україна

Огляд принципів використання одноразових паролів 168
Сливець О. Д., Остапець Д. О., Український державний університет науки і технологій, Україна

Огляд інтелектуальних методів для вирішення задачі вибору функціональних профілів захищеності інформації.....169
Сухомлин О.О., Остапець Д. О., Український державний університет науки і технологій, Україна

Захищена система формування дипломів університету 170
Панін Д. В., Остапець Д. О., Український державний університет науки і технологій, Україна

Дослідження ефективності алгоритмів шифрування та дешифрування даних

Нор О.С., Український державний університет науки і технологій, Україна

Інформаційні системи щоденно працюють із великими обсягами даних, що потребують захисту від несанкціонованого доступу та втрати. Тому криптографічні методи шифрування залишаються ключовим інструментом забезпечення конфіденційності та цілісності інформації. Однак у практичних умовах важливо не лише гарантувати безпеку, але й оцінити ефективність алгоритмів із погляду швидкодії, споживання ресурсів і стабільності роботи.

Метою дослідження є створення програмного комплексу для експериментальної оцінки ефективності криптографічних алгоритмів у середовищі C#. Робота передбачає аналіз часових характеристик шифрування та розшифрування, середньої швидкості обробки, завантаження центрального процесора та використання оперативної пам'яті. Об'єктом дослідження виступають процеси криптографічного перетворення даних у програмних системах, предметом – методи та метрики оцінювання ефективності.

У межах дослідження реалізовано програмну систему з настільним інтерфейсом, що підтримує типові сценарії експериментів: вибір алгоритму та параметрів, завантаження вхідних даних, запуск шифрування/розшифрування, а також перегляд узагальнених результатів у вигляді метрик і журналів.

Система забезпечує виконання криптографічних перетворень за допомогою різних алгоритмів – AES, Serpent, ChaCha20, RSA, ElGamal, ECC – та автоматичний збір показників ефективності. Для кожного запуску фіксуються умови тесту (алгоритм, параметри, тип носія, обсяг і кількість файлів), первинні метрики і похідні показники.

Методика експериментів побудована за принципом «однакові умови – різні алгоритми». Для підвищення надійності вимірювань застосовується серійність запусків і усереднення результатів, а також зіставлення роботи алгоритмів на різних типах носіїв (M.2, SSD, HDD) та за різної кількості потоків. Особливу увагу приділено впливу підсистеми введення-виведення на загальний час обробки: для швидких накопичувачів обмеженням часто виступає CPU (і тоді багатопотоковість дає приріст), тоді як для HDD зростає роль випадкових звернень до диска, що може нівелювати переваги паралельної обробки або навіть погіршувати підсумкову швидкодію.

Окремо враховано практичні аспекти коректності використання криптографії: для режимів автентифікованого шифрування оцінка ефективності інтерпретується разом із перевагами контролю цілісності, оскільки в реальних системах важливо не лише «зашифрувати швидко», а й забезпечити захист від непомітної модифікації даних. Для асиметричних алгоритмів акцент зроблено на типових для них задачах – операціях з ключами та короткими повідомленнями – оскільки пряме шифрування великих файлів асиметрією є ресурсно затратним і практично недоцільним.

Результати експериментів показали, що симетричні алгоритми мають найвищу швидкодію при роботі з великими масивами даних, тоді як асиметрично доцільно застосовувати для захисту ключів і коротких повідомлень у складі гібридних схем. Також встановлено, що багатопотокова обробка підвищує продуктивність на швидких носіях, але може знижувати ефективність при роботі з HDD через I/O-обмеження та конкуренцію потоків за ресурс диска.

Отримані результати можуть бути використані для побудови гібридних систем шифрування та оптимізації прикладних рішень у сфері інформаційної безпеки. Розроблений підхід дає змогу формувати практичні рекомендації щодо вибору алгоритмів з урахуванням обмежень продуктивності, характеристик носія та доступних ресурсів процесора й пам'яті, а також забезпечує основу для подальшого розширення експериментів (додавання нових режимів, алгоритмів і сценаріїв навантаження).

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ

МІЖНАРОДНА НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ І МОЛОДИХ УЧЕНИХ "МОЛОДА АКАДЕМІЯ - 25"

Дніпро
2025

	Селівьорстова Т.В.)	
3	Фатуроті В.О. (КН01-21і), Чуніхін А.С. (аспірант) «Програмна реалізація Agile Metrics Models для моніторингу командної роботи» (кер. доц. Селівьорстова Т.В.)	30
4	Сетькін Н.В.(КН01-21) «Програмна реалізація веб-сервісу для визначення психологічного підтипу людини з використанням нейронних мереж» (кер. доц. Островська К.Ю.)	32
5	Мінаєнко О.С. (КН01-21) «Розробка модулю ІС для фіксації переміщення робітників на виробництві на основі алгоритмів комп'ютерного зору» (кер. доц. Островська К.Ю.)	33
6	Спишник Д.Д. (КН01-21) «Розробка програмних компонентів для симулятора демонтажного робота на платформі unity» (кер. доц. Островська К.Ю.)	34
7	Окунєв М.В. (КН01-21) «Програмна реалізація компонентів для системи автономного запобігання зіткненню для великогабаритної техніки» (кер. доц. Островська К.Ю.)	35
8	Желєзний Д.М. (ПЗ01-21) «Програмна реалізація задачі сегментації зображень із застосуванням згорткових нейронних мереж» (кер. доц. Островська К.Ю.)	35
9	Бурдін Д.А. (КН01-21) «Прогнозування кредитний ризиків за допомогою нейронних мереж» (кер. доц. Островська К.Ю.)	36
10	Герасименко Г. А. (КН01-21) «Створення 3d-моделі аніме-персонажу засобами blender» (кер. доц. Дмитрієва І.С.)	36
11	Анісімов О.Д. (КН01-21) «Програмна реалізація та часовий аналіз ефективності методів шифрування та дешифрування» (кер. доц. Журба А.О.)	38
12	Леванович М. (КН901-20) «Розробка програмного модулю для обчислення статистичних характеристик часових рядів» (кер. доц. Журба А.О.)	39
13	Фурса Д.М. (КН01-21) «Розробка інформаційного тематичного веб-додатку для людей з вадами зору» (кер. доц. Журба А.О.)	40
14	Бондар М.Р. (КН01-21) «Розробка застосунку для обліку фінансів з використанням мови програмування Java» (кер. доц. Журба А.О.)	41
15	Пікус Б.В. (ПЗ01-21) «Розробка клієнтської частини веб-додатку «Персональний асистент» (кер. доц. Журба А.О.)	42
16	Легкоконець Д.С. (ПЗ2112) «Аналізатор алгоритмів виявлення програмних клонів» (кер. доц. Куроп'ятник О. С.)	43
17	Нор О. С. (ПЗ2421) «Аналіз методів генерації тестів» (кер. доц. Куроп'ятник О. С.)	44
3	***Підсекція «Комп'ютерні технології в бізнесі»***	45
1	Глущенко О. О. (КТ01-21) «Технологія оцінки ефективності маркетингових рішень на підприємстві» (кер. ст. викл. Савчук Р. В.)	45
2	Заворуєв О. П. (КТ01- 21) «Технологія формування інформаційних ресурсів у задачах моніторингу діяльності підприємства» (кер. проф. Савчук Л. М.)	46
3	Мелешко Д. С. (КТ01-21) «Інформаційні технології визначення потенціалу підприємства» (кер. ст. викл. Савчук Р. В.)	46
4	Сліпченко О. С. (КТ01-21) «Технології підтримки прийняття інвестиційних рішень на підприємстві» (кер. проф. Савчук Л. М.)	47
5	Явтушенко Є. Ю. (КТ01-21) «Компонентна модель оцінки конкурентноспроможності продукції підприємства» (кер. ст. викладач Савчук Р. В.)	48

АНАЛІЗ МЕТОДІВ ГЕНЕРАЦІЇ ТЕСТІВ

Нор О. С., керівник Куроп'ятник О. С.,
Український державний університет науки і технологій

У сучасному розробленні програмного забезпечення автоматизоване тестування відіграє важливу роль, оскільки дозволяє підвищити якість продукту та зменшити витрати на тестування. Існує чимало засобі автоматизованого тестування, зокрема Selenium, JUnit, TestNG, PyTest, Katalon Studio, Appium та Postman. Вони дозволяють автоматизувати запуск тестів та визначення успішності їх виконання. З метою зменшення витрат на тестування доцільним є впровадження автоматизованої генерації тестових випадків.

Розглянемо методи генерації тестових випадків. Методи на основі еволюційних алгоритмів базуються на принципах природного відбору та генетичних механізмах. Процес включає ініціалізацію популяції тестів, їх оцінку, відбір найкращих варіантів та їх модифікацію через мутацію та схрещування. Основна перевага цього методу – висока ефективність та здатність знаходити складні помилки, але він потребує значних обчислювальних ресурсів.

Метод випадкової генерації передбачає створення тестових сценаріїв без урахування структури коду чи логіки його роботи. Випадкові вхідні значення генеруються та подаються на вхід програмного модуля. Після цього аналізуються отримані вихідні дані: перевіряється їх відповідність очікуваним результатам або виявлення аномалій (наприклад, аварійне завершення роботи, некоректні вихідні значення чи порушення обмежень, заданих у специфікації). Головною перевагою є швидкість і простота реалізації, однак спостерігається слабке покриття коду.

Методи покриття коду, спрямовані на максимальне використання різних шляхів виконання програми, включають покриття операторів, умов, рішень і шляхів.

Фазове тестування використовує структурований підхід, коли тестові випадки створюються на основі специфікації програмного забезпечення та аналізу можливих сценаріїв його використання. Цей метод дозволяє знаходити логічні помилки, але потребує детального опрацювання тестових сценаріїв.

Комбінаторне тестування спрямоване на перевірку всіх можливих комбінацій вхідних параметрів, що дозволяє охопити більшу кількість сценаріїв виконання. Основна проблема – експоненційне зростання кількості комбінацій. В деяких випадках комбінаторне тестування може доповнювати фазове, особливо на етапі перевірки взаємодії параметрів.

Однією з головних проблем у генерації тестових сценаріїв є забезпечення достатнього покриття коду при мінімальних витратах ресурсів. Випадкові методи часто не гарантують якісного охоплення всіх можливих виконань програми, тоді як структурні методи потребують значних обчислювальних потужностей.

Для вирішення цієї проблеми доцільно використовувати комбіновані підходи, що поєднують ефективність еволюційних алгоритмів із точністю методів покриття коду. Також важливим напрямом є автоматизація процесу тестування за допомогою сучасних інструментів машинного навчання, які можуть покращувати генерацію тестів.

Кожен із методів має свої особливості та застосовується залежно від поставленої задачі. Методи, рекомендовані для подальшого дослідження, базуються на різних принципах: природному відборі та генетичних механізмах, аналізі структурних елементів програми, перевірці взаємодії різних комбінацій вхідних параметрів. В якості критерію для їх подальшого аналізу пропонується оцінювати рівень покриття коду, ефективність виявлення помилок, вимоги до обчислювальних ресурсів та адаптивність до змін у програмному забезпеченні.

Розвиток цих методів сприятиме створенню ефективних стратегій тестування програмного забезпечення.