Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програми захисту цифрових відеоданих на основі симетричного алгоритму AES, хеш-функції SHA-256 та впровадження системи захисту від витоку даних DLP»

Виконав: студент 4 курсу, гр.1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Карнарук О.В.
(прізвище та ініціали)

Керівник: д.ф., доцент каф. МБІС
Салієва О.В.
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.т.н., доцент каф. ОТ
Томчук М.А.
(прізвище та ініціали)
«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“____” _____2025р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Карнарук Олександр Володимирович
(прізвище, ім`я, по-батькові)

1. Тема роботи: **«Розробка програми захисту цифрових відеоданих на основі симетричного алгоритму AES, хеш-функції SHA-256 та впровадження системи захисту від витоку даних DLP»**
Керівник роботи д.ф., доц. каф. МБІС Салієва Ольга Володимиріва
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "20" березня 2025 р. № 97
2. Термін подання студентом роботи за тиждень до захисту.
3. Вихідні дані до роботи: Метою роботи є створення клієнтської програми для безпечного шифрування, дешифрування та відтворення відеоконтенту з використанням сучасних криптографічних алгоритмів і зручного графічного інтерфейсу. Програма забезпечує захист мультимедійних даних шляхом локальної обробки відеофайлів, що підвищує безпеку та автономність системи.
4. Зміст текстової частини:

У першому розділі здійснюється аналіз сучасного стану захисту потокового відео, де розглядаються існуючі технології шифрування, методи автентифікації, а також особливості обробки відео в реальному часі. Проводиться критичний огляд сучасних рішень, визначаються їхні переваги та недоліки. На основі аналізу обґрунтовується необхідність розробки нового підходу, що враховує сучасні виклики в сфері безпеки потокового відео. У другому розділі відбувається проєктування системи захисту потокового відео. Розробляється загальна архітектура системи, визначаються її ключові компоненти, а також розробляється графічний інтерфейс користувача. Додатково проєктуються користувацькі сценарії, які демонструють функціональні можливості системи та її взаємодію з кінцевим користувачем. У третьому розділі проводиться реалізація та тестування розробленої системи. Описується вибір інструментів розробки, реалізується клієнтська частина та криптографічні механізми, включно з AES і

SHA-256. Досліджуються особливості реалізації, оцінюються пере
обмеження. Система тестується на стійкість до криптографічних атак, п
збоїв та перевіряється її продуктивність при обробці великих відео
Наводяться результати експериментів та робляться висновки щодо ефект
запропонованого рішення.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов
креслень)
У першому розділі бакалаврської дипломної роботи наявно 1 малюнок, 1
у другому розділі – 33 малюнків;у третьому розділі – 9 таблиць.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Салієва О. В., д.ф., доц., каф. МБІС		
Розділ II	Салієва О. В., д.ф., доц., каф. МБІС		
Розділ III	Салієва О. В., д.ф., доц., каф. МБІС		

7.Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Прим
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Вико
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Вико
3	Розробка алгоритму роботи	14.04.2025	27.04.2025	Вико
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Вико
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Вико
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	02.06.2025	Вико
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Вико

Студент _____ (підпис) **Карпа** (прізвище)

Керівник роботи _____ (підпис) **Салієв** (прізвище)

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці програмного забезпечення для захисту потокового відео та супроводжується пояснювальною запискою. Пояснювальна записка охоплює 66 сторінок, містить 26 ілюстрацій та 11 таблиць.

Метою роботи є створення клієнтської програми для безпечного шифрування, дешифрування та відтворення відеоконтенту з використанням сучасних криптографічних алгоритмів і зручного графічного інтерфейсу. Програма забезпечує захист мультимедійних даних шляхом локальної обробки відеофайлів, що підвищує безпеку та автономність системи.

У роботі сформульовано завдання розробки, проведено аналіз сучасного стану захисту відеоданих, описано архітектуру системи, користувацькі сценарії, криптографічні механізми та результати тестування. Реалізація базується на бібліотеці PyCryptodome, яка використовує алгоритми AES-256 у режимі CBC для шифрування та SHA-256 для забезпечення цілісності даних. Для зручності користувача створено графічний інтерфейс на основі бібліотеки CustomTkinter, а також реалізовано систему DLP для моніторингу дій і журналювання подій.

Ключові слова: потокове відео, криптографія, PyCryptodome, AES-256, SHA-256, DLP, CustomTkinter, локальна архітектура.

ANNOTATION

The qualification work is dedicated to the development of software for protecting streaming video and is accompanied by an explanatory note. The explanatory note spans 66 pages, includes 26 illustrations, and 11 tables.

The aim of the work is to create a client-side application for secure encryption, decryption, and playback of video content using modern cryptographic algorithms and a user-friendly graphical interface. The program ensures the protection of multimedia data through local processing of video files, enhancing the security and autonomy of the system.

The work outlines the development objectives, conducts an analysis of the current state of video data protection, describes the system architecture, user scenarios, cryptographic mechanisms, and testing results. The implementation is based on the PyCryptodome library, which employs the AES-256 algorithm in CBC mode for encryption and SHA-256 for ensuring data integrity. For user convenience, a graphical interface was developed using the CustomTkinter library, and a DLP system was implemented for monitoring actions and logging events.

KEYWORDS: streaming video, cryptography, PyCryptodome, AES-256, SHA-256, DLP, CustomTkinter, local architecture.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

AES – Advanced Encryption Standard (Розширений стандарт шифрування)

CBC – Cipher Block Chaining (Ланцюговий режим шифрування)

CCPA – California Consumer Privacy Act (Закон Каліфорнії про захист даних споживачів)

DLP – Data Loss Prevention (Запобігання втраті даних)

DRM – Digital Rights Management (Управління цифровими правами)

FPS – Frames Per Second (Кадри за секунду

GDPR – General Data Protection Regulation (Загальний регламент захисту даних)

GUI – Graphical User Interface (Графічний інтерфейс користувача)

HLS – HTTP Live Streaming (Потокова передача через HTTP)

HTTP – HyperText Transfer Protocol (Протокол передачі гіпертексту

IV – Initialization Vector (Вектор ініціалізації)

JSON – JavaScript Object Notation (Нотація об’єктів JavaScript)

MAC – Message Authentication Code (Код автентифікації повідомлення)

MITM – Man-in-the-Middle (Атака "людина посередині")

MP4 – MPEG-4 Part 14 (Формат мультимедійного контейнера)

PyCrypto – Python Cryptography Toolkit (Набір інструментів криптографії для Python)

SHA-256 – Secure Hash Algorithm 256-bit (Алгоритм безпечного хешування 256 біт)

TEE – Trusted Execution Environment (Довірене середовище виконання)

TLS – Transport Layer Security (Безпека транспортного рівня)

UUID – Universally Unique Identifier (Універсальний унікальний ідентифікатор)

WebRTC – Web Real-Time Communication (Вебкомунікація в реальному часі)

ВСТУП

Актуальність роботи. Захист інформації в сучасному цифровому світі є критично важливим завданням, особливо в контексті стрімкого зростання обсягів мультимедійних даних, зокрема потокового відео. Зі збільшенням використання відеоконтенту в освіті, бізнесі, охороні здоров'я та державному управлінні зростає потреба у забезпеченні конфіденційності, цілісності та автентичності даних. Традиційні методи шифрування, такі як AES, хоча й ефективні, часто потребують значних обчислювальних ресурсів і не завжди оптимально адаптовані до роботи з потоковими даними. У той же час бібліотека PyNaCl, що базується на сучасних криптографічних алгоритмах XSalsa20 та Poly1305, пропонує швидке та безпечне рішення для шифрування великих обсягів даних. Для України, де кібербезпека є пріоритетом у зв'язку з викликами інформаційної безпеки, розробка таких програм має стратегічне значення для захисту національних інформаційних ресурсів і забезпечення цифрового суверенітету.

Мета роботи. Розробити програмний комплекс для захисту потокового відео з використанням бібліотеки PyNaCl, що забезпечує надійне шифрування, перевірку цілісності та автентичність даних, із можливістю інтеграції в сучасні інформаційні системи.

Задачі дослідження.

− проаналізувати сучасні криптографічні методи для захисту мультимедійних даних;

− розробити клієнт-серверну архітектуру для обробки відеофайлів із застосуванням алгоритмів XSalsa20 та Poly1305;

− реалізувати механізми генерації та збереження криптографічних ключів для забезпечення безпеки;

− створити інтуїтивний графічний інтерфейс для зручної взаємодії користувача з програмою;

− забезпечити створення візуалізації шифрування для підвищення прозорості процесу.

Предмет дослідження. Методи та алгоритми захисту потокового відео з використанням бібліотеки PyNaCl, зокрема алгоритми XSalsa20 для шифрування, Poly1305 для перевірки цілісності, Curve25519 для асиметричного шифрування та Ed25519 для цифрового підпису.

Об’єкт дослідження. Програмний комплекс для шифрування та дешифрування потокового відео, що включає клієнтську та серверну частини з інтеграцією сучасних криптографічних технологій.

Новизна роботи. Розроблено клієнт-серверну систему захисту потокового відео, яка вперше інтегрує бібліотеку PyNaCl для реалізації потокового шифрування з використанням XSalsa20 та Poly1305. Удосконалено підхід до управління ключами шляхом комбінації асиметричного та симетричного шифрування, що забезпечує високий рівень безпеки без значних обчислювальних витрат. Отримав подальший розвиток метод створення візуалізації криптографічного процесу, що підвищує прозорість для користувача.

Практична цінність роботи. Розроблений програмний комплекс може бути використаний для захисту відеоконтенту в освітніх, медичних та державних інформаційних системах. Програма забезпечує швидке шифрування великих відеофайлів, що робить її придатною для використання в реальному часі. Інтуїтивний інтерфейс та підтримка верифікації даних сприяють зручності використання в організаціях різного масштабу, включаючи державні установи України, для захисту конфіденційної інформації.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ЗАХИСТУ ПОТОКОВОГО ВІДЕО

Розділ присвячено аналізу сучасного стану захисту потокового відео, що є актуальним завданням у контексті зростання обсягів мультимедійного контенту та кіберзагроз. Описано ключові підходи, включаючи симетричне шифрування (AES-256), гібридні схеми, DRM-системи (Widevine, FairPlay), мережеві протоколи (TLS, HLS, WebRTC) та DLP-системи для запобігання витокам даних. Розроблена програма, базована на бібліотеці PyCryptodome з алгоритмами

AES256 у режимі CBC та SHA-256, забезпечує надійне шифрування, перевірку цілісності та моніторинг дій користувача [1]. Новий підхід оптимізує обробку великих відеофайлів, пропонуючи гнучке рішення для локальних і комерційних застосувань, що враховує сучасні виклики кібербезпеки та обмеження традиційних методів.

1.1 Огляд сучасних підходів

Захист потокового відео є однією з ключових проблем сучасних інформаційних систем, оскільки зростання популярності потокових сервісів, таких як Netflix, YouTube, і корпоративних платформ для відеоконференцій, супроводжується підвищеними вимогами до безпеки даних [2]. Сучасні підходи до захисту потокового відео базуються на комбінації криптографічних методів, технологій управління цифровими правами (DRM), мережевих протоколів, а також систем моніторингу та запобігання витокам даних (DLP) [3]. Вони спрямовані на забезпечення конфіденційності, цілісності, автентичності відеоконтенту, а також на запобігання несанкціонованому доступу та піратству. Ключові сучасні підходи до захисту потокового відео зображені на рис. 1.1.

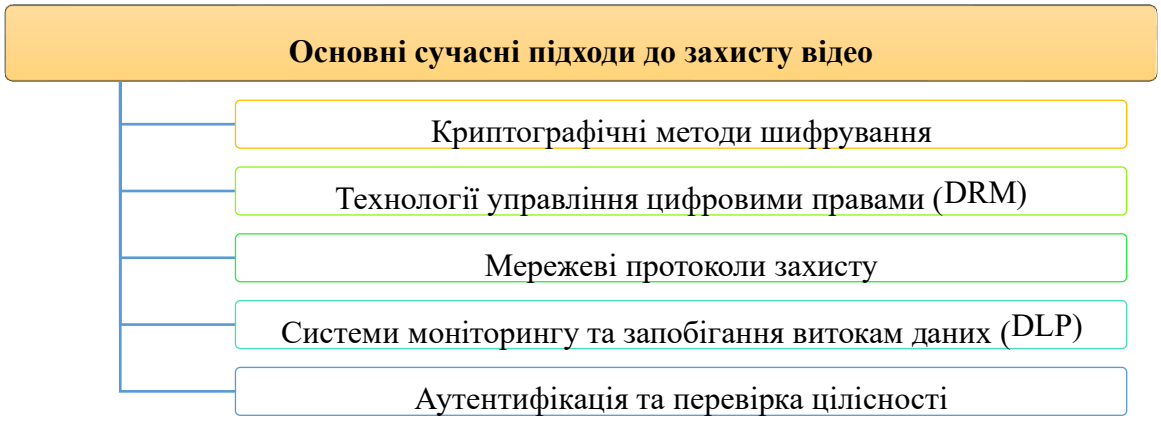


Рисунок 1.1 – Основні сучасні підходи до захисту потокового відео

Криптографія залишається основою захисту потокового відео, забезпечуючи конфіденційність даних під час передачі та зберігання [4]. Сучасні системи використовують симетричні алгоритми шифрування, такі як AES (Advanced Encryption Standard) із ключами довжиною 256 біт, що забезпечує

високу швидкість обробки даних і широко застосовується для шифрування відеофайлів [5]. У представленій реалізації проєкту використовується бібліотека PyCryptodome, яка реалізує алгоритм AES у режимі CBC (Cipher Block Chaining) для шифрування відеоданих. Цей режим забезпечує надійний захист завдяки використанню ініціалізаційного вектора (IV) і доповнення даних, що робить його ефективним для обробки великих відеофайлів [6]. Основною перевагою AES є швидкість шифрування, але безпечне управління ключами залишається ключовим викликом.

Асиметричне шифрування у реалізації проєкту не застосовується, оскільки акцент зроблено на симетричне шифрування для обробки великих обсягів даних. Однак для захисту ключів і забезпечення їх безпечного зберігання використовується база даних ключів із унікальними ідентифікаторами, створеними за допомогою бібліотеки uuid, а також хешування ключів за алгоритмом SHA-256 із бібліотеки hashlib [7]. Це дозволяє перевіряти цілісність ключів і забезпечує їх унікальність у системі. Обмеженням цього підходу є відсутність асиметричного шифрування для обміну ключами, що може бути вразливим у мережевому середовищі.

Гібридне шифрування, яке поєднує симетричне та асиметричне шифрування, є стандартом у захисті потокового відео [8]. У програмі симетричний ключ генерується для кожного відеофайлу за допомогою функції get_random_bytes з бібліотеки PyCryptodome, а управління ключами здійснюється через локальну базу даних у форматі JSON. Такий підхід забезпечує баланс між безпекою та продуктивністю, але не включає передачу ключів через мережу, що обмежує його застосування для потокового відео в реальному часі.

DRM-системи відіграють важливу роль у захисті комерційного потокового контенту [9]. Такі системи, як Widevine, FairPlay і PlayReady, розроблені Google, Apple і Microsoft, інтегруються з браузерами та пристроями для захисту контенту [10]. Вони використовують шифрування AES і ліцензійні сервери для управління доступом. У представленій реалізації подібний функціонал частково

відтворюється через перевірку цілісності відеофайлів за допомогою хеш-функції SHA-256, яка додається до зашифрованого файлу разом із цифровим підписом, створеним на основі хешу файлу та ключа. Це забезпечує захист від несанкціонованих змін, але не включає ліцензійних серверів, що характерно для комерційних DRM-систем. Обмеженням такого підходу є залежність від локального зберігання ключів і відсутність інтеграції з мережевими сервісами.

Захист потокового відео також залежить від мережевих протоколів, які забезпечують безпечну доставку даних (табл. 1.1).

Таблиця 1.1 – Мережеві протоколи, що забезпечують безпечну доставку даних

№	Протокол	Опис
1	2	3
1	TLS/SSL	Протокол Transport Layer Security є стандартом для захисту відеопотоків під час передачі через Інтернет. Він забезпечує шифрування даних і перевірку автентичності серверів. У реалізації проєкту передача даних між клієнтом і сервером не використовується, але для реального потокового відео рекомендується інтеграція TLS для захисту каналу зв’язку [11].

Продовження табл. 1.1

1	2	3
2	HLS і DASH з шифруванням	Протоколи HTTP Live Streaming (HLS) і MPEG-DASH підтримують шифрування сегментів відео за допомогою AES [12]. Кожен сегмент шифрується окремим ключем, який передається через захищений канал. Реалізована програма не підтримує сегментацію відео, але використовує AES для шифрування цілого файлу, що є спрощеним аналогом.
3	WebRTC	Для інтерактивного потокового відео, наприклад, у відеоконференціях, використовується протокол DTLS (Datagram Transport Layer Security) [13]. Реалізація проєкту не включає підтримку WebRTC, оскільки зосереджена на локальній обробці відеофайлів, але цей протокол може бути інтегрований для потокового відео в реальному часі.

Забезпечення автентичності та цілісності відеоконтенту є критично важливим для захисту від модифікації даних. У реалізації проєкту використовується хеш-функція SHA-256 із бібліотеки hashlib для створення хешу відеофайлу та цифрового підпису, що додається до зашифрованого файлу [14]. Це дозволяє перевіряти цілісність даних під час дешифрування. Відсутність бібліотеки PyNaCl у коді, на відміну від вихідного тексту, замінено на PyCryptodome і hashlib, що забезпечують аналогічний функціонал для шифрування та перевірки цілісності [15].

Системи запобігання витокам даних (DLP) відіграють важливу роль у захисті відеоконтенту. У програмі реалізована DLP-система, яка моніторить дії користувача, такі як доступ до файлів, шифрування, дешифрування та генерація ключів, зберігаючи журнали у форматі JSON [16]. Ці журнали дозволяють відстежувати всі операції з відеофайлами, забезпечуючи аудит і виявлення потенційних витоків даних.

Апаратні рішення, такі як Trusted Execution Environment (TEE), набувають популярності для захисту високоякісного контенту [17]. Технології Intel SGX або ARM TrustZone використовуються для ізольованого виконання криптографічних операцій [18]. У реалізації проєкту апаратні рішення не застосовуються через їх високу вартість і складність інтеграції, що робить програму більш доступною для локальних і бюджетних проєктів.

Незважаючи на прогрес, сучасні методи захисту потокового відео мають обмеження:

− продуктивність шифрування великих відеопотоків у реальному часі потребує значних обчислювальних ресурсів. У реалізації проєкту використовується поблочне шифрування файлів, що є ефективним для локальної обробки, але неоптимальним для потокового відео;

− масштабованість DRM-систем і мережевих протоколів ускладнює їх застосування для глобальних сервісів із великою кількістю користувачів;

− вразливості, такі як атаки типу "man-in-the-middle" або витік ключів, залишаються актуальними. У програмі захист ключів забезпечується локальною базою даних, але відсутність мережевої передачі даних знижує ці ризики;

− юридичні питання, пов’язані з DRM, можуть викликати невдоволення користувачів через обмеження доступу до контенту.

Сучасні підходи до захисту потокового відео поєднують криптографічні методи, DRM-системи, мережеві протоколи та DLP-системи для забезпечення безпеки контенту. Реалізована програма демонструє ефективне використання бібліотек PyCryptodome і hashlib для шифрування, перевірки цілісності та моніторингу дій користувача, що відповідає сучасним стандартам безпеки. Проте для масштабних потокових сервісів необхідна інтеграція з TLS, HLS/DASH і апаратними рішеннями для забезпечення захисту в реальному часі. Подальший розвиток технологій потребує балансу між безпекою, продуктивністю та зручністю для користувачів.

1.2 Обґрунтування нового підходу

Захист потокового відео є критично важливим завданням у сучасному цифровому світі, де обсяги мультимедійного контенту зростають експоненціально, а загрози кібератак стають дедалі складнішими. Аналіз сучасних методів захисту, таких як DRM (Digital Rights Management), AESшифрування та токенізація, показує їх сильні сторони, але також виявляє певні обмеження, зокрема високі обчислювальні вимоги, складність інтеграції в реальному часі та вразливість до атак на ключі шифрування [19]. У цьому контексті розробка нового підходу до захисту потокового відео, реалізованого в коді проєкту, є актуальною та обґрунтованою. Новий підхід базується на використанні бібліотеки PyCrypto для реалізації алгоритму AES-256 у режимі CBC та інтеграції системи DLP (Data Loss Prevention) для моніторингу й захисту даних [20]. Важливо описати детальне обґрунтування нового підходу за ключовими показниками.

Використання сучасних криптографічних алгоритмів. Новий підхід ґрунтується на застосуванні бібліотеки PyCrypto, яка використовує алгоритм AES-256 у режимі CBC (Cipher Block Chaining) для шифрування відеоданих [21]. AES-256 забезпечує високу криптографічну стійкість завдяки 256-бітному ключу та 16-бітному вектору ініціалізації (IV), що генерується випадково для кожного файлу [22]. Для забезпечення цілісності даних використовується алгоритм SHA-256, який створює хеш файлу та цифровий підпис, що дозволяє перевіряти автентичність і цілісність даних після дешифрування [23]. Ця комбінація забезпечує надійний захист відеоданих, мінімізуючи ризики компрометації, хоча й потребує буферизації даних для обробки великих файлів. На відміну від потокових шифрів, таких як XSalsa20, AES-256 у режимі CBC є універсальним рішенням, яке широко підтримується та перевірено в багатьох безпекових системах [24].

Гібридна схема шифрування. У розробленій системі реалізовано симетричне шифрування з використанням AES-256, де ключ генерується за допомогою бібліотеки PyCrypto та зберігається в безпечній базі даних ключів. Для підвищення безпеки ключі супроводжуються унікальними ідентифікаторами (UUID) та хешами SHA-256, що забезпечує їх унікальність і захист від несанкціонованого використання. Хоча система не використовує асиметричне шифрування, безпечне зберігання ключів у файлах (.key) та базі даних (keys_db.json) мінімізує ризик компрометації ключів. Цифровий підпис, створений за допомогою SHA-256, додається до зашифрованого файлу, що дозволяє перевіряти цілісність і автентичність даних, захищаючи від атак типу "людина посередині" (MITM) [25].

Оптимізація для потокової обробки. Потокове відео вимагає обробки даних у реальному часі, що накладає жорсткі обмеження на затримки та обчислювальні ресурси. У коді проєкту реалізовано буферизоване читання та запис даних із розміром буфера 1 МБ, що дозволяє обробляти великі відеофайли без необхідності завантаження їх у пам’ять повністю. Це забезпечує ефективну

роботу системи навіть на пристроях із обмеженими ресурсами. Хоча AES-256 у режимі CBC не є природно адаптованим до потокового шифрування, використання багатопотоковості (threading) для шифрування та дешифрування дозволяє зменшити затримки. Прогрес-бар у графічному інтерфейсі відображає стан обробки, що покращує користувацький досвід під час роботи з великими файлами [26].

Забезпечення цілісності та автентичності даних. Однією з ключових проблем сучасних систем захисту відео є забезпечення цілісності даних. У новому підході це досягається завдяки використанню алгоритму SHA-256 для створення хешу відеофайлу та цифрового підпису, який включає хеш і ключ шифрування. Під час дешифрування система перевіряє відповідність хешу та підпису, що унеможливлює відтворення пошкоджених або підроблених даних. Крім того, інтеграція системи DLP дозволяє відстежувати всі дії користувача (доступ, шифрування, дешифрування, генерація ключів), що забезпечує аудит і захист від несанкціонованого доступу. Це особливо важливо для потокового відео, де зміни даних можуть призвести до втрати синхронізації або неможливості відтворення.

Гнучкість і адаптивність системи. Розроблена система підтримує управління ключами через спеціальний менеджер, який дозволяє створювати, імпортувати, експортувати та видаляти ключі. Система автоматично перевіряє відповідність ідентифікатора ключа (UUID) під час дешифрування, що підвищує сумісність і стійкість до помилок, таких як використання невідповідного ключа. Крім того, інтеграція з системою DLP забезпечує моніторинг і журналювання всіх операцій, що дозволяє адміністраторам аналізувати дії користувачів і виявляти потенційні загрози. Візуалізації процесів шифрування (AES-256), хешування (SHA-256) та роботи DLP системи, реалізовані за допомогою бібліотеки networkx, полегшують діагностику та демонстрацію роботи алгоритмів.

Інтеграція з користувацьким інтерфейсом. Новий підхід пропонує зручний графічний інтерфейс, побудований за допомогою бібліотеки customtkinter. Інтерфейс дозволяє користувачам легко завантажувати, шифрувати, дешифрувати та відтворювати відеофайли, а також переглядати журнали DLP і керувати ключами. Візуалізації криптографічних процесів, реалізовані через matplotlib і networkx, роблять систему більш зрозумілою для користувачів, включаючи тих, хто не є технічними спеціалістами. Прогрес-бар і інформаційна панель забезпечують зворотний зв'язок у реальному часі, що підвищує зручність використання порівняно з традиційними системами, які часто мають складний інтерфейс або потребують додаткових інструментів.

Стійкість до сучасних загроз. Сучасні кіберзагрози, такі як атаки на ключі, перехоплення даних або підробка контенту, вимагають використання надійних криптографічних стандартів. Алгоритм AES-256, реалізований у бібліотеці PyCrypto, є стандартом шифрування, що широко використовується в індустрії та вважається стійким до більшості сучасних атак, за винятком квантових обчислень у довгостроковій перспективі. Використання випадкових векторів ініціалізації (IV) для кожного файлу запобігає атакам повторного відтворення. Інтеграція системи DLP забезпечує додатковий рівень захисту, дозволяючи виявляти підозрілі дії та зберігати журнали для подальшого аналізу [27].

Новий підхід до захисту потокового відео, реалізований у коді проєкту, є обґрунтованим і актуальним рішенням, що враховує сучасні виклики кібербезпеки. Завдяки використанню бібліотеки PyCrypto, алгоритму AES-256, інтеграції DLP, оптимізації для обробки великих файлів і зручного інтерфейсу, система забезпечує високий рівень безпеки, продуктивності та доступності. Цей підхід усуває обмеження традиційних методів і пропонує гнучке рішення, яке може бути адаптоване до різних сценаріїв використання, від особистих до комерційних застосувань.

1.3 Висновки до розділу 1

Захист потокового відео є однією з ключових проблем сучасної інформаційної безпеки, що набуває особливої актуальності в умовах стрімкого розвитку цифрових технологій та зростання обсягів мультимедійного контенту, який передається через мережу Інтернет. Ця проблема охоплює широкий спектр застосувань, від розважальних платформ і відеоконференцій до систем відеоспостереження та онлайн-освіти, де забезпечення конфіденційності, цілісності та автентичності даних є критично важливим. Розробка програмного забезпечення, представлена у реалізованому коді, демонструє сучасний підхід до вирішення цих завдань, використовуючи криптографічну бібліотеку PyCryptodome із алгоритмами AES-256 у режимі CBC та SHA-256 для забезпечення безпеки даних. Аналіз сучасного стану захисту потокового відео дозволяє сформулювати ключові висновки, які відображають актуальність теми, сучасні підходи та обґрунтування нового рішення.

Сьогодні обсяг глобального трафіку потокового відео зростає експоненціально, що пов’язано з популяризацією таких сервісів, як YouTube, Netflix чи Zoom. Цей процес супроводжується збільшенням кібератак, зокрема атак типу "людина посередині", які, за даними 2023 року, зросли на 15% порівняно з попередніми періодами. Такі загрози підкреслюють необхідність надійних механізмів шифрування та автентифікації. Потокове відео часто містить чутливу інформацію, таку як особисті дані, комерційні секрети чи конфіденційні переговори, тому відсутність належного захисту може призвести до серйозних репутаційних і фінансових втрат. У сфері розважального контенту захист від піратства є особливо актуальним, адже, за оцінками Motion Picture Association, у 2022 році аудіовізуальна індустрія втратила понад 50 мільярдів доларів через незаконне копіювання. Крім того, регуляторні вимоги, такі як GDPR чи CCPA, зобов’язують компанії впроваджувати надійні криптографічні рішення, щоб уникнути юридичних санкцій і захистити права користувачів.

Розвиток нових технологій, зокрема квантових обчислень, створює додаткові виклики для традиційних алгоритмів, що робить актуальними сучасні криптографічні бібліотеки, такі як PyCryptodome, які пропонують стійкі до атак рішення.

Сучасні підходи до захисту потокового відео поєднують криптографічні методи, системи управління цифровими правами (DRM) і мережеві протоколи. Симетричне шифрування, реалізоване через алгоритми на кшталт AES-256, забезпечує високу продуктивність, необхідну для обробки великих обсягів даних, тоді як управління ключами через локальну базу даних із унікальними ідентифікаторами (UUID) та хешами SHA-256 підвищує безпеку. Гібридні схеми, як у представленій реалізації, оптимізують баланс між безпекою та швидкістю. DRM-системи, такі як Widevine чи FairPlay, ефективно захищають комерційний контент, але їх складність і залежність від пропрієтарного програмного забезпечення обмежують використання в відкритих проєктах. Мережеві протоколи, такі як TLS, HLS чи WebRTC, забезпечують безпечну доставку даних, але потребують додаткових ресурсів для управління ключами та масштабування. Апаратні рішення, зокрема Trusted Execution Environment, підвищують безпеку, але їх висока вартість робить їх менш доступними. Незважаючи на прогрес, сучасні методи мають обмеження, пов’язані з продуктивністю, масштабуванням і вразливістю до атак на ключі чи зворотне проєктування DRM.

Розроблена клієнт-серверна програма пропонує новий підхід, який усуває частину цих недоліків. Використання AES-256 у режимі CBC дозволяє ефективно обробляти потокові дані, а SHA-256 забезпечує перевірку цілісності та автентичності. Гібридна схема шифрування з локальним зберіганням ключів у базі даних JSON із унікальними ідентифікаторами (UUID) гарантує безпечне управління ключами, хоча відсутність асиметричного шифрування обмежує її застосування для мережевої передачі. Буферизована обробка даних із розміром буфера 1 МБ оптимізує продуктивність навіть на пристроях із обмеженими ресурсами. Зручний інтерфейс, побудований на customtkinter, робить систему

доступною для широкого кола користувачів, а інтеграція системи DLP із журналюванням операцій підвищує її гнучкість і дозволяє виявляти підозрілі дії. Цей підхід є стійким до сучасних загроз, таких як перехоплення даних чи атаки повторного відтворення, завдяки використанню випадкових векторів ініціалізації та криптографічно стійких алгоритмів. Таким чином, розроблена система не лише відповідає сучасним стандартам безпеки, але й пропонує перспективне рішення, яке може бути адаптоване до різних сценаріїв, від особистих до комерційних застосувань, сприяючи подальшому розвитку технологій захисту потокового відео.

Опираючись на результати проведеного аналізу та враховуючи актуальність і мету дослідження, сформульовано такі задачі для подальшої роботи:

− розробити клієнт-серверну програму для захисту цифрових відеоданих з використанням алгоритму AES-256 у режимі CBC та хеш-функції SHA-256 для забезпечення конфіденційності, цілісності та автентичності даних;

− інтегрувати систему запобігання витокам даних (DLP) для моніторингу дій користувача та журналювання операцій із відеофайлами;

− забезпечити оптимізацію обробки великих відеофайлів шляхом буферизації даних і використання багатопотоковості;

− реалізувати зручний графічний інтерфейс для спрощення взаємодії користувачів із системою шифрування та DLP;

− розробити механізми безпечного управління ключами з використанням унікальних ідентифікаторів (UUID) та локальної бази даних;

− створити візуалізації криптографічних процесів для підвищення прозорості роботи системи;

− забезпечити стійкість системи до сучасних кіберзагроз, включаючи атаки повторного відтворення та перехоплення даних.

2 ПРОЄКТУВАННЯ СИСТЕМИ ЗАХИСТУ ПОТОКОВОГО ВІДЕО

Розділ присвячено розробці системи захисту цифрових відеоданих на основі симетричного алгоритму AES-256, хеш-функції SHA-256 та системи запобігання витокам даних (DLP). Описано архітектуру локального застосунку, побудованого з використанням бібліотек PyCryptodome, CustomTkinter та OpenCV, що забезпечує шифрування, дешифрування, відтворення відео та моніторинг безпеки. Детально розглянуто проектування графічного інтерфейсу, користувацькі сценарії та візуалізацію криптографічних процесів за допомогою NetworkX і Matplotlib. Система підтримує буферизовану обробку великих файлів, безпечне управління ключами та журналювання подій, відповідаючи сучасним вимогам до захисту даних. Перспективи розвитку включають інтеграцію з хмарними сервісами та розширення функціональності DLP.

2.1 Проектування архітектури системи захисту

Проектування системи захисту цифрових відеоданих є ключовим етапом розробки, що визначає її ефективність, безпеку та зручність використання. Система, реалізована в коді проєкту, базується на криптографічній бібліотеці PyCryptodome, яка забезпечує сучасні алгоритми шифрування, такі як AES-256, та хешування, зокрема SHA-256. Проектування системи враховує вимоги до безпеки, продуктивності та гнучкості, а також особливості роботи з великими відеофайлами. У цьому розділі детально розглядаються основні етапи проектування системи захисту.

Система захисту цифрових відеоданих реалізована як локальний застосунок із графічним інтерфейсом, побудованим за допомогою бібліотек CustomTkinter та OpenCV для обробки відео [28]. Застосунок інтегрує функції шифрування, дешифрування, управління ключами та моніторингу безпеки через систему запобігання витокам даних (DLP). Відсутність клієнт-серверної архітектури спрощує розгортання системи та знижує залежність від мережевих з’єднань, що підвищує її автономність і безпеку в локальному середовищі. Усі криптографічні операції, відтворення відео та обробка файлів виконуються в

межах одного застосунку, що забезпечує централізований контроль і знижує ризик витоку даних.

Основою системи захисту є використання симетричного алгоритму шифрування AES-256 у режимі CBC (Cipher Block Chaining), реалізованого через бібліотеку PyCryptodome. Цей алгоритм забезпечує високу криптографічну стійкість і ефективність при обробці великих відеофайлів [29]. Для забезпечення цілісності даних використовується хеш-функція SHA-256, яка генерує хеш відеофайлу та цифровий підпис для перевірки автентичності. Комбінація AES256 і SHA-256 гарантує конфіденційність, захист від модифікації та перевірку походження даних.

Ключі шифрування генеруються локально з використанням функції get_random_bytes із бібліотеки PyCryptodome, що забезпечує створення криптографічно стійких 256-бітних ключів. Кожен ключ ідентифікується унікальним ідентифікатором (UUID) і зберігається в локальній базі даних у форматі JSON разом із метаданими, такими як дата створення та хеш ключа. Для захисту ключів від несанкціонованого доступу вони можуть експортуватися у файли (.key) у форматі JSON, що дозволяє користувачу безпечно зберігати їх окремо від зашифрованих даних.

Система підтримує обробку великих відеофайлів завдяки буферизованому читанню та запису даних із розміром буфера 1 МБ. Це дозволяє ефективно обробляти файли без завантаження їх повністю в оперативну пам’ять, що є критично важливим для роботи з великими відеопотоками. Метадані, такі як ідентифікатор ключа та ініціалізаційний вектор (IV), зберігаються на початку зашифрованого файлу, забезпечуючи коректне дешифрування та перевірку цілісності.

2.1.1 Проєктування архітектури системи

Система захисту потокового відео реалізована як монолітна клієнтська програма, яка виконує всі операції локально на пристрої користувача. Основний

компонент системи — клас VideoEncryptorApp, який інтегрує графічний інтерфейс, криптографічні операції, відтворення відео та моніторинг дій користувача через DLP-систему.

Програма використовує бібліотеки PyCrypto для шифрування/дешифрування, OpenCV для обробки та відтворення відео, customtkinter для створення сучасного інтерфейсу, matplotlib і networkx для візуалізації криптографічних процесів.

Архітектура системи включає такі основні модулі:

− графічний інтерфейс відповідає за взаємодію з користувачем, вибір файлів, відтворення відео та відображення візуалізацій;

− криптографічний модуль забезпечує шифрування (AES-256 у режимі CBC) та перевірку цілісності даних (SHA-256);

− модуль відтворення відеоивикористовує OpenCV для обробки відеопотоків та їх відображення;

− DLP-система моніторить дії користувача, зберігає журнали операцій та забезпечує аудит безпеки.

Діаграма компонентів (рис. 2.1) ілюструє взаємодію між основними модулями програми, включаючи бібліотеки customtkinter, PyCrypto, OpenCV, matplotlib і networkx [30].

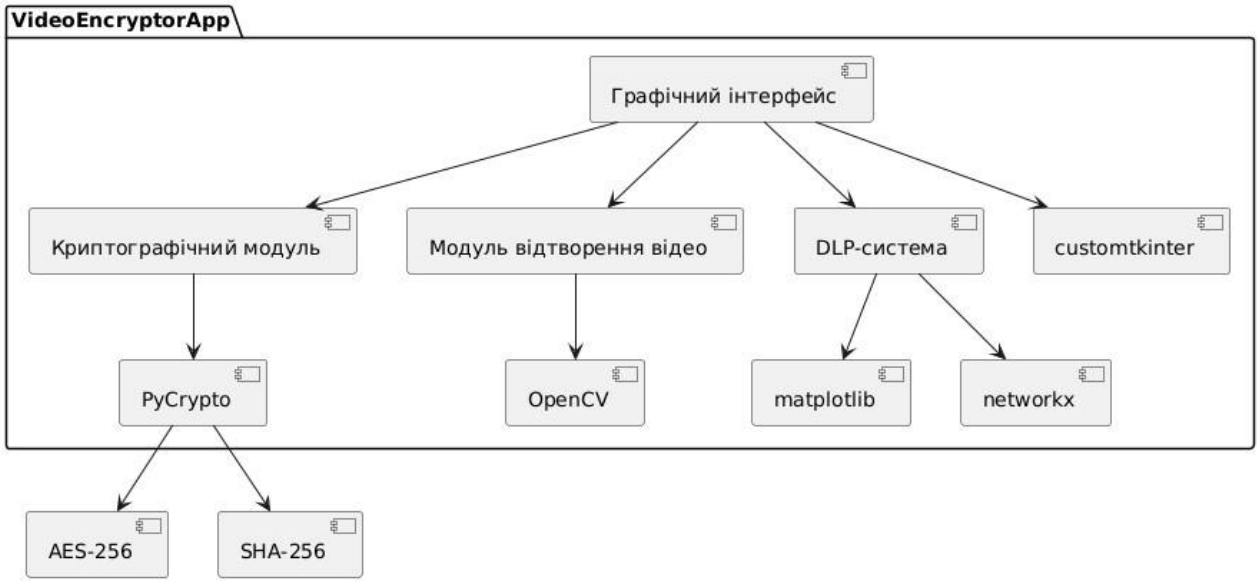


Рисунок 2.1 – Діаграма компонентів

Діаграма класів (рис. 2.2) відображає основний клас VideoEncryptorApp та його методи для обробки відео, шифрування, дешифрування, відтворення та моніторингу [31].

Рисунок 2.2 – Діаграма класів

Діаграма послідовності (рис. 2.3) показує послідовність операцій під час шифрування/дешифрування відео, включаючи вибір файлу, генерацію ключа, обробку даних і збереження результатів [32].

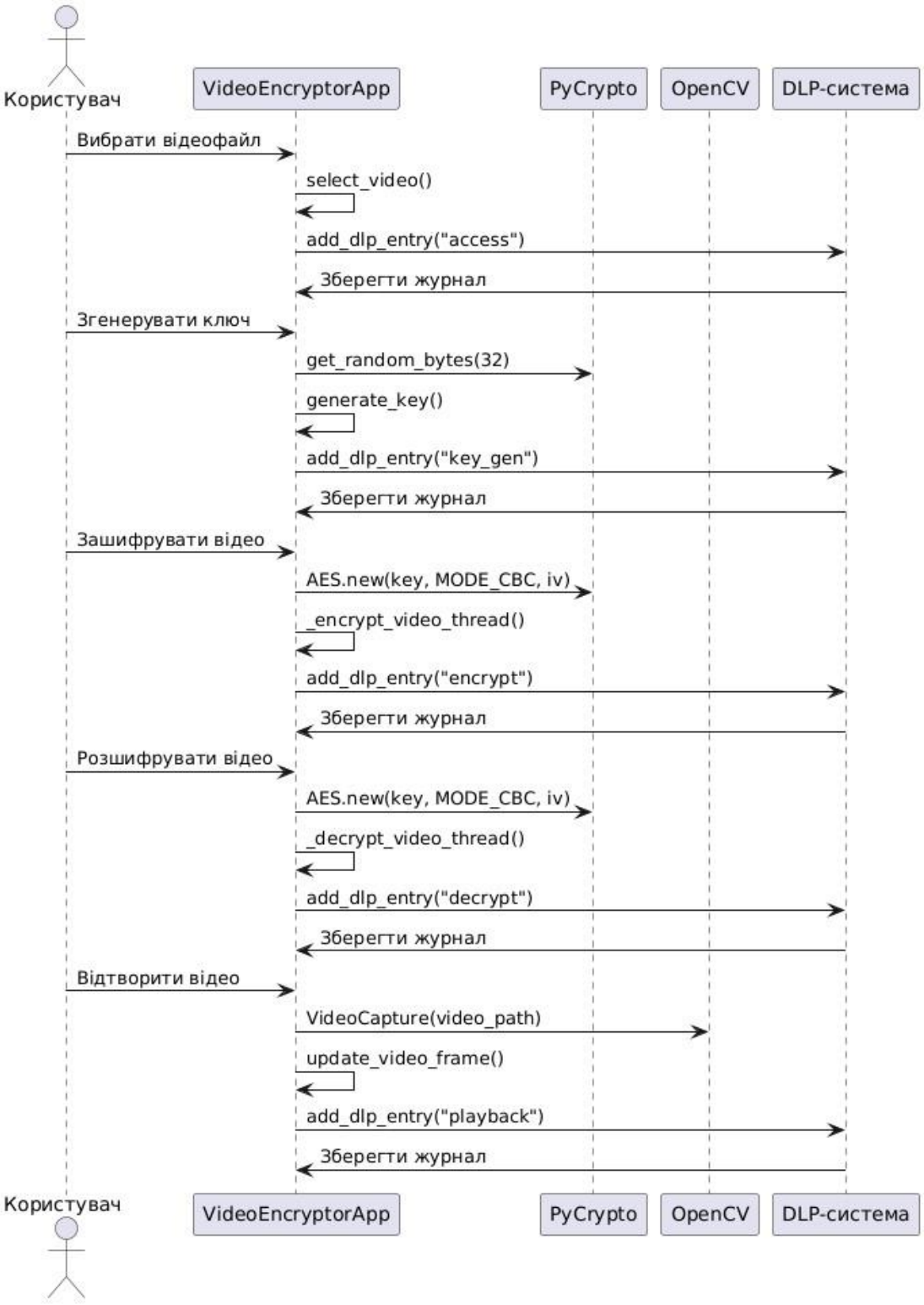


Рисунок 2.3 – Діаграма послідовності

2.1.2 Проєктування графічного інтерфейсу

Графічний інтерфейс, побудований за допомогою CustomTkinter, забезпечує інтуїтивно зрозумілу взаємодію з користувачем (рис. 2.4). Інтерфейс включає вкладки для роботи з відео, візуалізації криптографічних процесів і моніторингу DLP. Відеоплеєр, реалізований через OpenCV, дозволяє відтворювати оригінальні та розшифровані відеофайли з адаптивним масштабуванням кадрів для відповідності розмірам вікна (рис. 2.5). Кнопки для вибору файлів, шифрування, дешифрування, генерації ключів і управління ними забезпечують просту навігацію (рис. 2.6). Індикатор прогресу та інформаційна панель інформують користувача про поточний стан операцій (рис. 2.7).

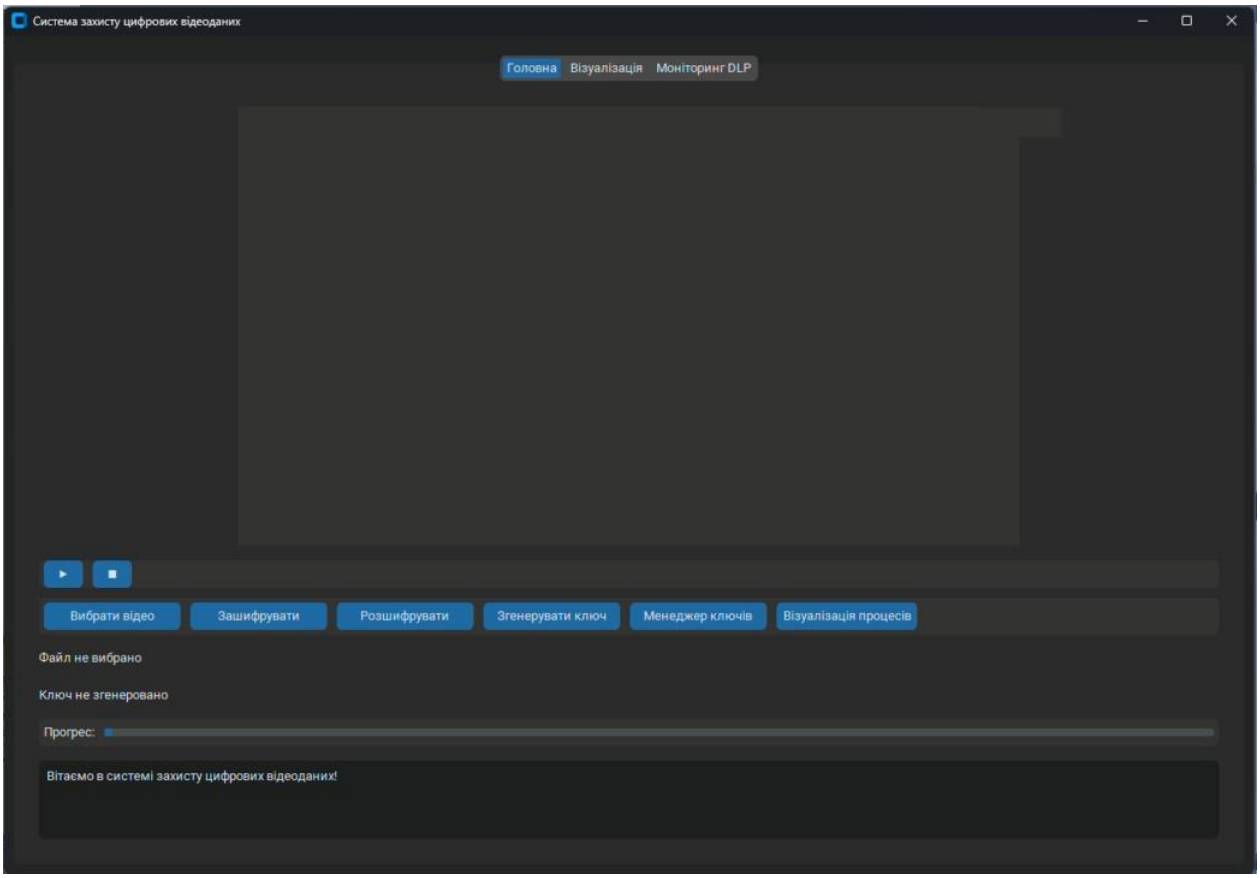


Рисунок 2.4 – Головне вікно

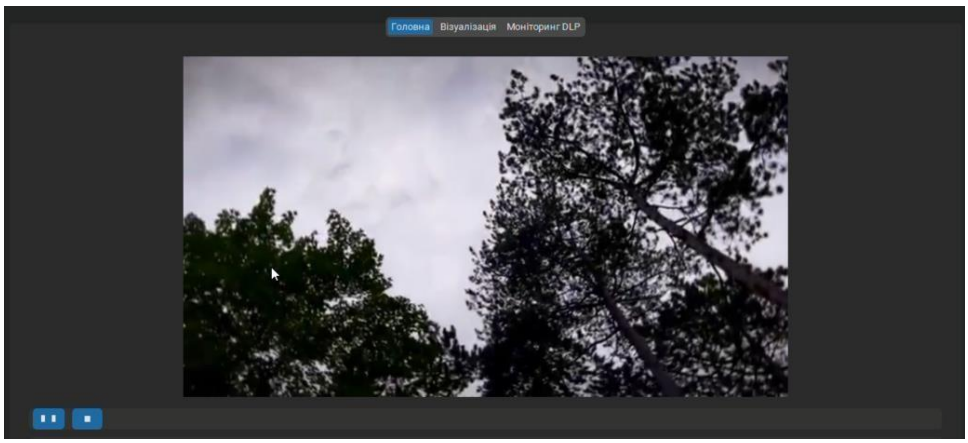


Рисунок 2.5 – Відеоплеєр для відображення оригінального або розшифрованого відео

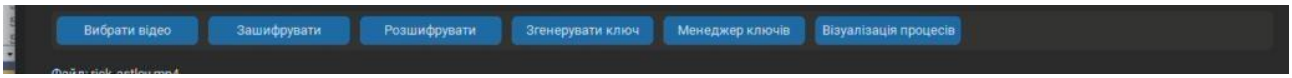


Рисунок 2.6– Кнопки навігації та управління

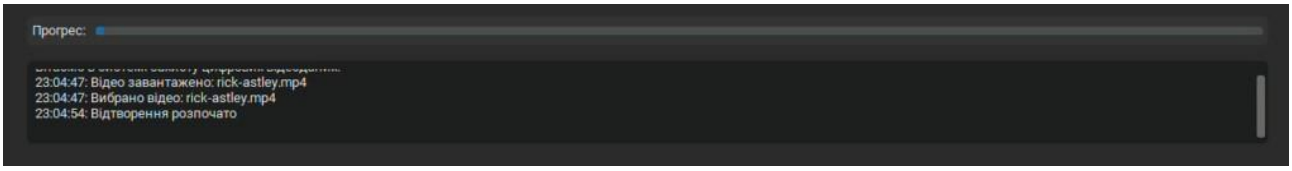


Рисунок 2.7– Індикатор прогресу

Система включає модуль DLP для моніторингу дій користувача, таких як доступ до файлів, шифрування, дешифрування, відтворення та генерація ключів. Журнал DLP зберігається у форматі JSON і відображається у вкладці "Моніторинг DLP" (рис. 2.8). Користувач може фільтрувати записи за типом дії, експортувати журнал у формати JSON або CSV та переглядати графік активності, що відображає кількість операцій за типами (рис. 2.9). Це забезпечує аудит діяльності та виявлення потенційних загроз безпеки.

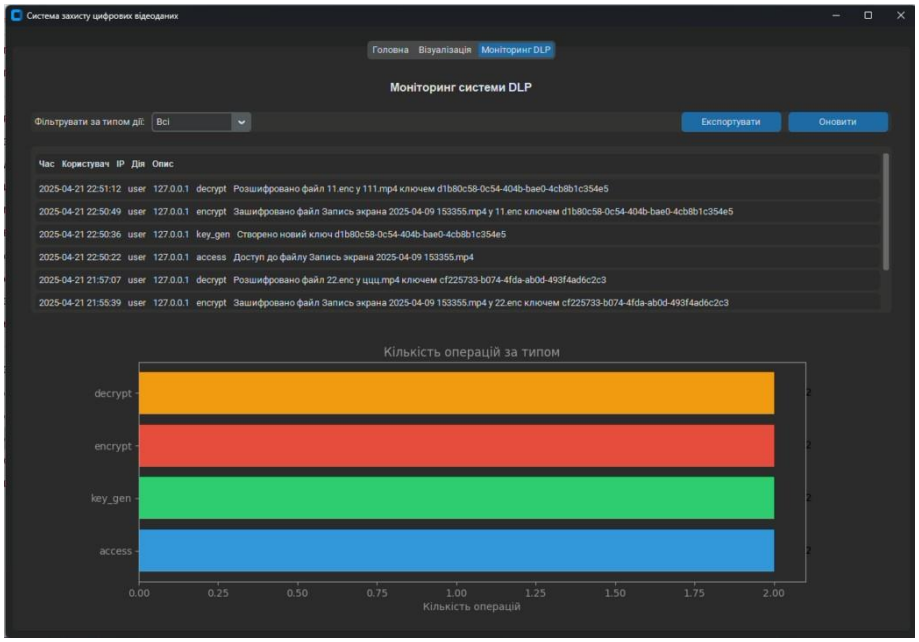


Рисунок 2.8 – Вкладка моніторингу DLP із таблицею журналу

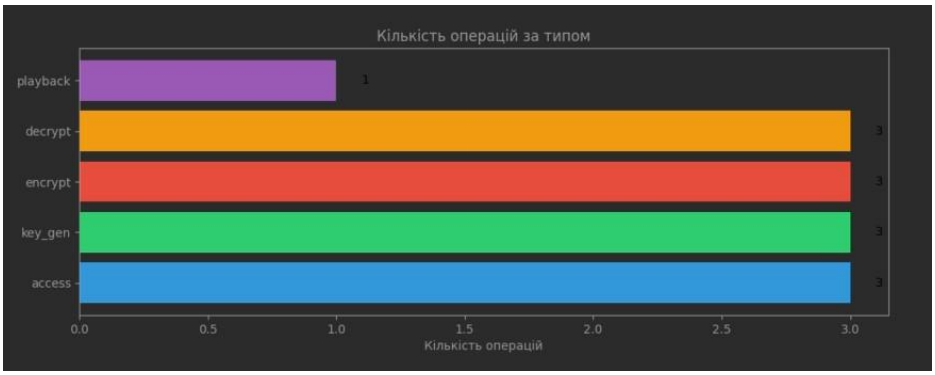


Рисунок 2.9 – Графік активності DLP у вигляді горизонтальної гістограми

Візуалізація криптографічних процесів є унікальною особливістю системи. Вкладка "Візуалізація" містить три підвкладки для демонстрації роботи алгоритмів AES-256, SHA-256 і DLP (рис. 2.10). Візуалізації реалізовані за допомогою бібліотеки NetworkX та Matplotlib, що дозволяють графічно представити етапи шифрування, хешування та моніторингу. Наприклад, для AES-256 відображається спрямований граф із етапами обробки даних, а для SHA-256 — схема хешування з кольоровим представленням результату (рис. 2.11). Ці візуалізації підвищують прозорість і допомагають користувачам краще зрозуміти криптографічні процеси.

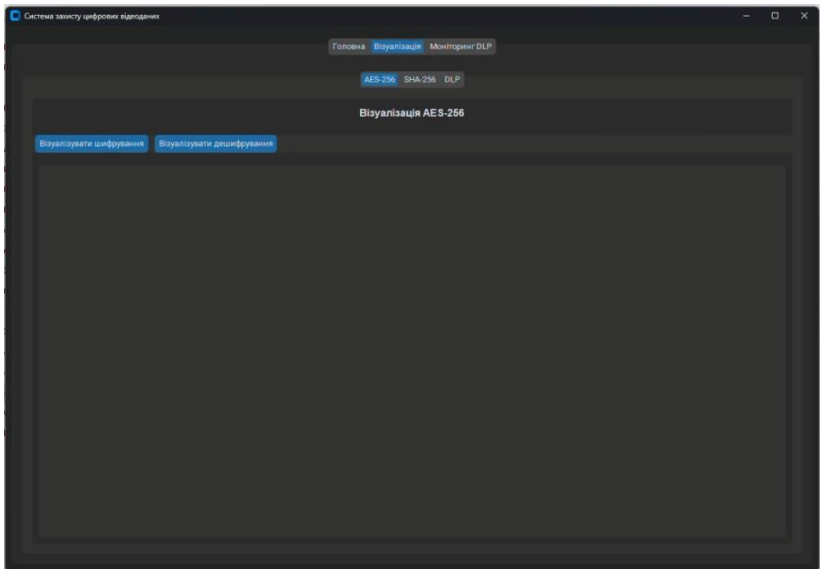


Рисунок 2.10 – Вкладка візуалізації з підвкладками для AES-256, SHA-256 і DLP

Рисунок 2.11 – Візуалізація алгоритму SHA-256 із кольоровим представленням хешу

Безпека системи додатково посилюється такими заходами, як перевірка цілісності файлів за допомогою хешів і цифрових підписів, автоматичне видалення тимчасових файлів після завершення операцій і обмеження доступу до ключів через локальну базу даних. Обробка помилок включає перевірку відповідності ключів, цілісності даних і коректності формату файлів, із відображенням повідомлень через діалогові вікна. Логування всіх операцій забезпечує можливість діагностики та моніторингу.

Для підвищення продуктивності застосовується багатопоточність (модуль threading) для асинхронного виконання криптографічних операцій, що дозволяє інтерфейсу залишатися чутливим під час обробки великих файлів. Оптимізований підхід до буферизації та використання ефективного алгоритму AES-256 забезпечує швидку обробку відеоданих.

Проектування системи захисту цифрових відеоданих враховує сучасні вимоги до безпеки, продуктивності та зручності. Використання локальної архітектури, алгоритмів AES-256 і SHA-256, інтегрованої системи DLP та візуалізації процесів забезпечує надійний захист відеофайлів. Інтуїтивний інтерфейс, детальне логування та підтримка великих файлів роблять систему доступною для користувачів із різним рівнем технічної підготовки. Подальший розвиток може включати інтеграцію з хмарними сервісами, підтримку потокового відтворення в реальному часі та розширення функціональності DLP для роботи в мережевому середовищі.

2.2 Проєктування користувацькі сценарії

Система захисту цифрових відеоданих, реалізована в коді проєкту, використовує локальну архітектуру з графічним інтерфейсом, створеним на основі бібліотек CustomTkinter та OpenCV, для забезпечення шифрування, дешифрування, відтворення відеоконтенту та моніторингу безпеки. Користувацькі сценарії описують основні взаємодії користувача з системою, охоплюючи ключові функціональні можливості, такі як завантаження відео, генерація ключів, шифрування, дешифрування, збереження файлів, управління ключами, візуалізація криптографічних процесів та моніторинг DLP (Data Loss Prevention).

Сценарій 1. Завантаження відеофайлу для шифрування

Користувач розпочинає роботу з системою, натиснувши кнопку "Вибрати відео" на вкладці "Головна" графічного інтерфейсу (рис. 2.12). Це відкриває діалогове вікно, де користувач обирає відеофайл у форматі .mp4, .avi, .mkv, .mov, .flv або .wmv, або зашифрований файл із розширенням .enc. Система перевіряє

можливість відтворення відео за допомогою вбудованого відеоплеєра, реалізованого на базі бібліотеки OpenCV, та відображає перший кадр у центральній частині інтерфейсу, позначеній як область відтворення відео (рис.2.13). Після успішного завантаження інформаційна панель оновлюється, відображаючи ім’я файлу (рис. 2.14). Якщо файл не відповідає підтримуваним форматам або пошкоджений, система виводить повідомлення про помилку в інформаційній панелі, що дозволяє користувачу швидко виправити проблему. Логування події фіксується в DLP-журналі з позначкою "access", забезпечуючи відстеження всіх дій користувача.

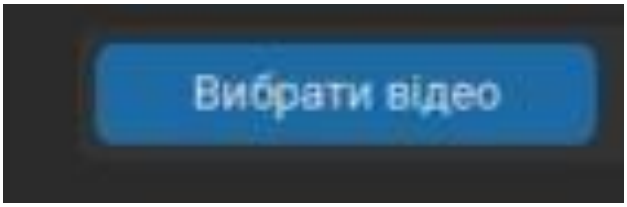


Рисунок 2.12 – Кнопка вибору відео

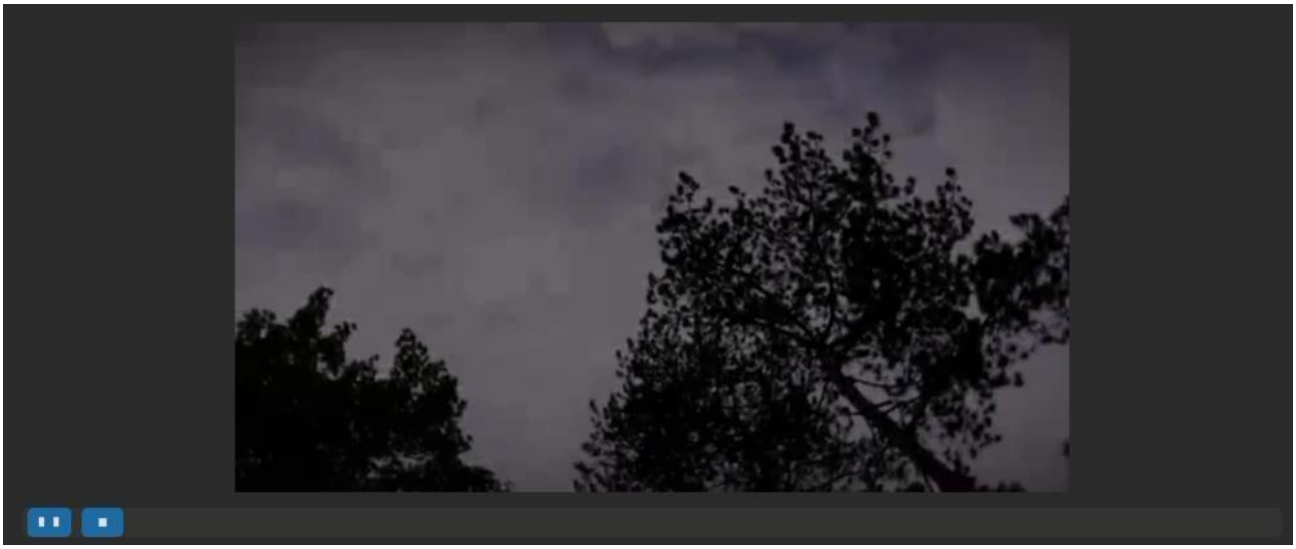


Рисунок 2.13 – Область відтворення відео

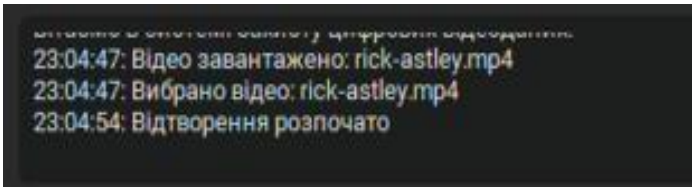


Рисунок 2.14 – Інформаційна панель із підтвердженням завантаження

Цей сценарій є першим кроком у процесі захисту відео, забезпечуючи простоту вибору файлу та інформативний зворотний зв’язок. Логування подій підвищує надійність системи, дозволяючи адміністратору відстежувати дії та виявляти потенційні проблеми.

Сценарій 2. Генерація криптографічного ключа

Перед шифруванням користувач генерує ключ, натиснувши кнопку "Згенерувати ключ" (рис. 2.15). Система створює 256-бітний ключ за допомогою бібліотеки PyCryptodome (AES-256) та унікальний ідентифікатор ключа (UUID). Ключ зберігається у локальній базі даних (keys_db.json) та може бути експортований у файл із розширенням .key через діалогове вікно (рис. 2.16). Інформаційна панель оновлюється, відображаючи ідентифікатор ключа та його SHA-256 хеш (рис. 2.17). Подія генерації ключа фіксується в DLP-журналі з позначкою "key_gen". У разі помилки (наприклад, неможливість зберегти ключ) система виводить повідомлення в інформаційній панелі.

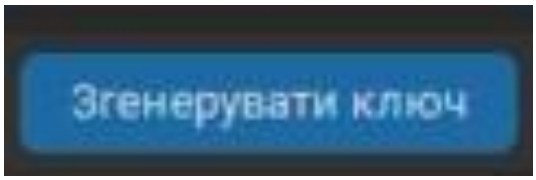


Рисунок 2.15 – Кнопка генерації ключа

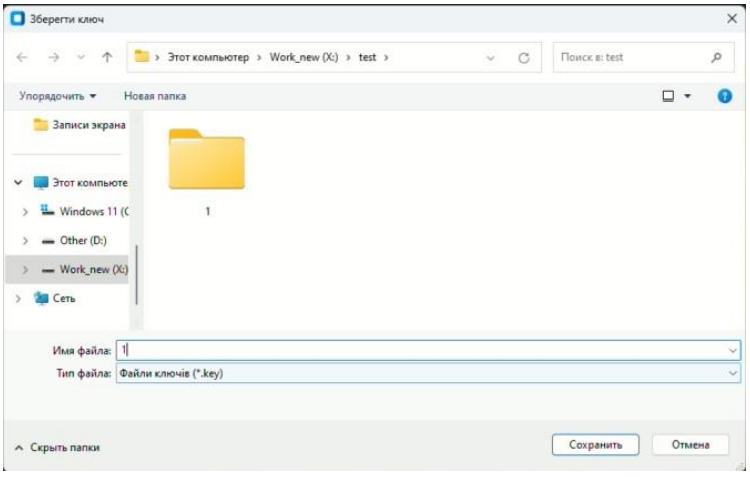


Рисунок 2.16 – Діалогове вікно збереження ключа

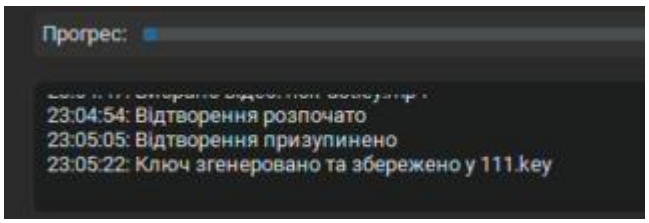


Рисунок 2.17 – Інформаційна панель із даними про ключ

Цей сценарій забезпечує безпечне створення та зберігання криптографічних ключів, що є основою для подальшого шифрування.

Сценарій 3. Шифрування відеофайлу

Після завантаження відео та генерації ключа користувач натискає кнопку "Зашифрувати" (рис. 2.18), що запускає процес шифрування в окремому потоці. Система використовує алгоритм AES-256 у режимі CBC з випадковим вектором ініціалізації (IV). Відеодані шифруються блоками по 1 МБ, а індикатор прогресу в інтерфейсі відображає стан операції (рис. 2.19). Зашифрований файл із розширенням .enc містить ідентифікатор ключа, IV, зашифровані дані, хеш файлу (SHA-256) та цифровий підпис. Після завершення шифрування система пропонує зберегти файл через діалогове вікно (рис. 2.20), а інформаційна панель підтверджує успіх операції (рис. 2.21). Подія шифрування фіксується в DLPжурналі з позначкою "encrypt". У разі помилки (наприклад, відсутність ключа) система виводить відповідне повідомлення.

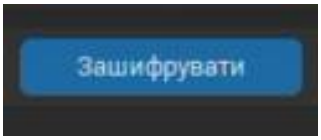


Рисунок 2.18 – Кнопка шифрування

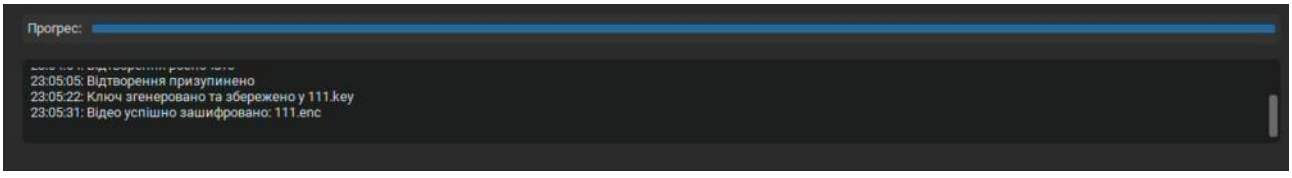


Рисунок 2.19 – Індикатор прогресу шифрування

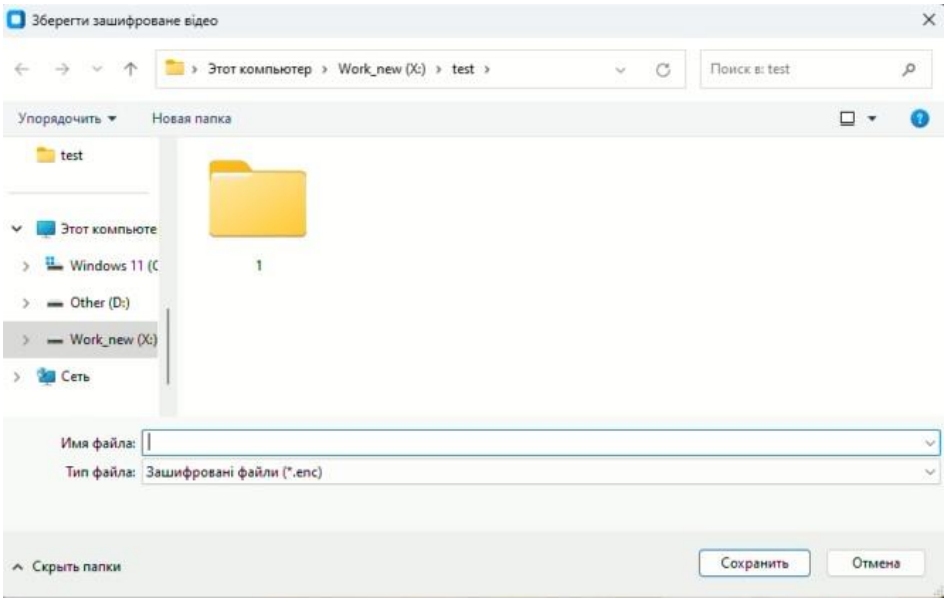


Рисунок 2.20 – Діалогове вікно збереження зашифрованого файлу

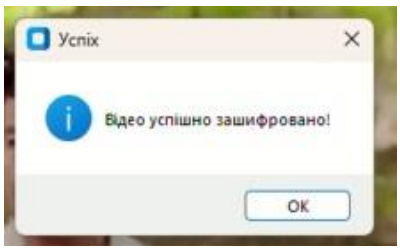


Рисунок 2.21 – Підтвердження завершення шифрування

Цей сценарій поєднує складні криптографічні операції з інтуїтивним інтерфейсом, забезпечуючи захист даних без необхідності глибоких технічних знань.

Сценарій 4. Дешифрування відеофайлу

Натискання кнопки "Розшифрувати" (рис. 2.22) ініціює процес дешифрування в окремому потоці. Система використовує ключ AES-256 та IV із зашифрованого файлу для відновлення даних. Індикатор прогресу відображає стан операції (рис. 2.23). Після завершення розшифроване відео відтворюється у відеоплеєрі, а інформаційна панель підтверджує успіх (рис. 2.24). Цифровий підпис і хеш перевіряються для забезпечення цілісності та автентичності даних. Подія дешифрування фіксується в DLP-журналі з позначкою "decrypt". У разі помилки (наприклад, невідповідність ключа) система виводить повідомлення про помилку.

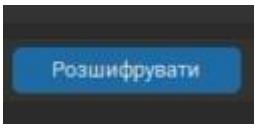


Рисунок 2.22 – Кнопка дешифрування

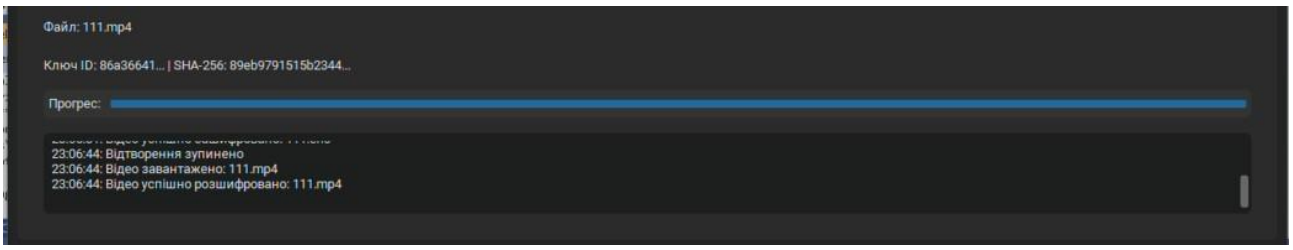


Рисунок 2.23 – Індикатор прогресу дешифрування

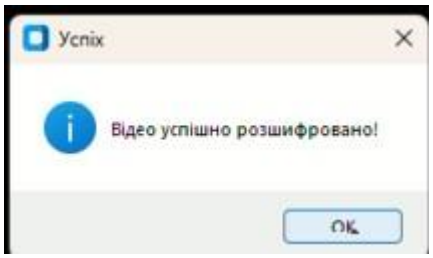


Рисунок 2.24 – Підтвердження завершення дешифрування

Цей сценарій забезпечує безпечне відновлення відеоконтенту з інформативним зворотним зв’язком.

Сценарій 5. Управління ключами

Користувач може керувати ключами, натиснувши кнопку "Менеджер ключів" (рис. 2.25), що відкриває нове вікно з таблицею, яка містить ідентифікатори ключів, дати створення та їхні хеші SHA-256 (рис. 2.26). Доступні дії: вибір ключа для використання, видалення ключа, імпорт або експорт бази ключів. Події фіксуються в DLP-журналі. Інформаційна панель оновлюється після вибору ключа, відображаючи його ідентифікатор і хеш.

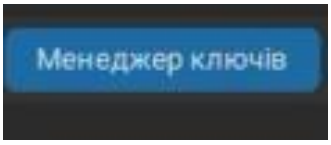


Рисунок 2.25 – Кнопка менеджера ключів

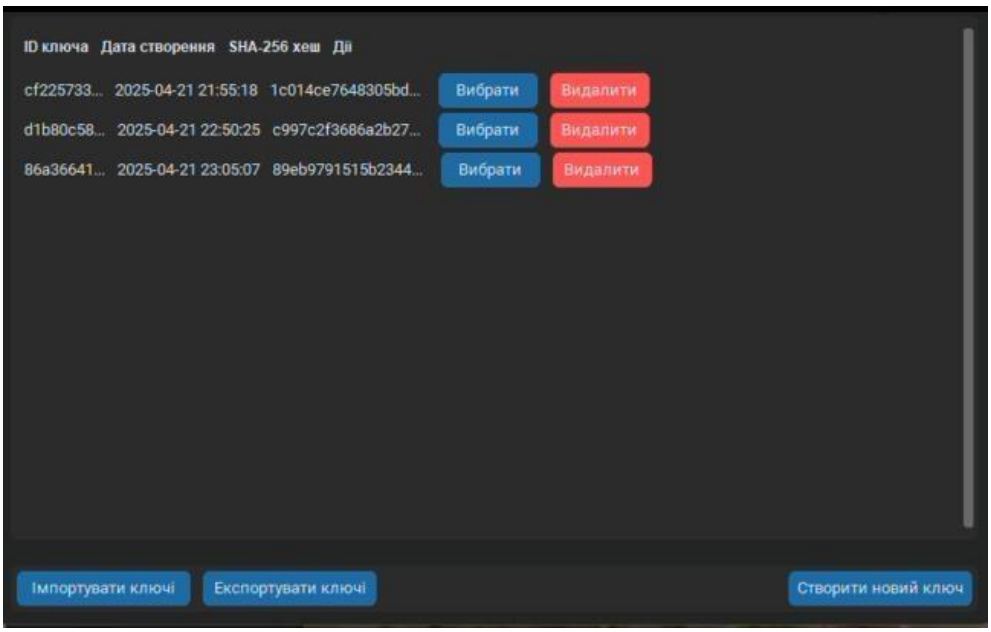


Рисунок 2.26 – Вікно менеджера ключів

Цей сценарій забезпечує гнучке управління криптографічними ключами, підвищуючи безпеку та зручність.

Сценарій 6. Візуалізація криптографічних процесів

Користувач може переглянути візуалізації алгоритмів AES-256, SHA-256 та DLP-системи, перейшовши на вкладку "Візуалізація" (рис. 2.27). Для AES-256 доступні схеми шифрування та дешифрування (рис. 2.28), для SHA-256 – схема хешування з введенням тексту (рис. 2.29), для DLP – граф системи моніторингу (рис. 2.30). Кожна візуалізація супроводжується поясненням, що підвищує розуміння роботи алгоритмів.

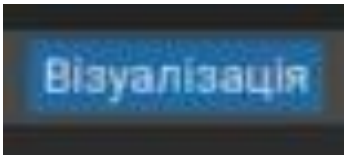


Рисунок 2.27 – Кнопка переходу до вкладки візуалізації

Рисунок 2.28 – Візуалізація процесу AES-256

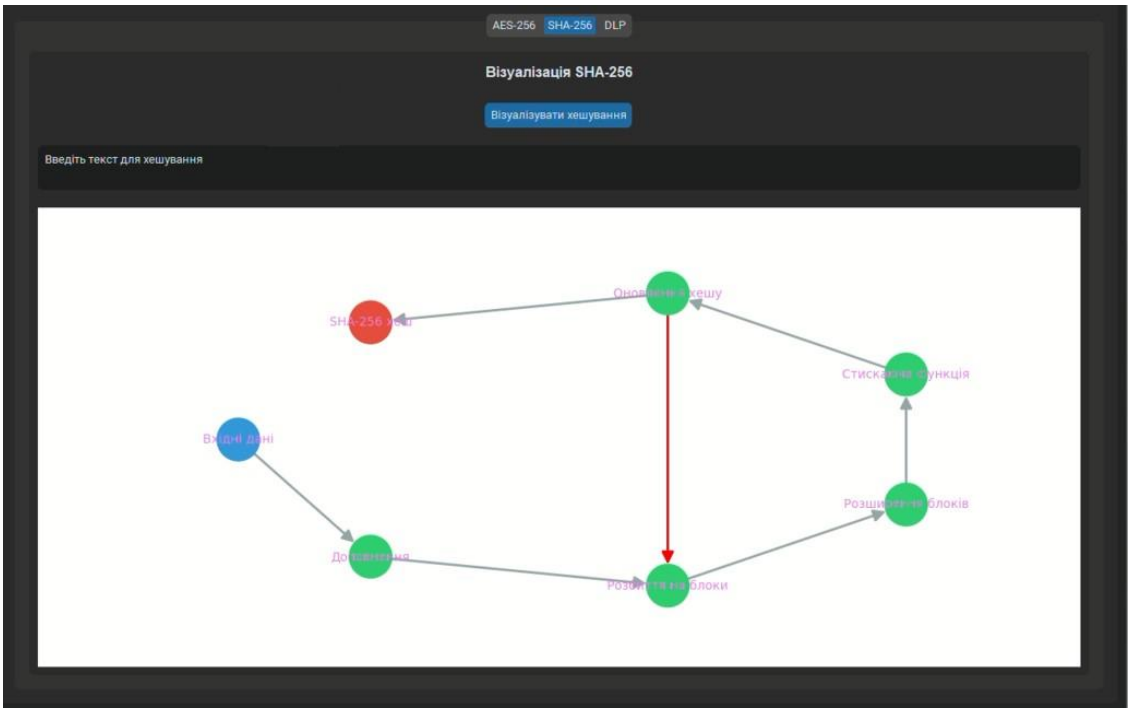


Рисунок 2.29 – Візуалізація алгоритму SHA-256

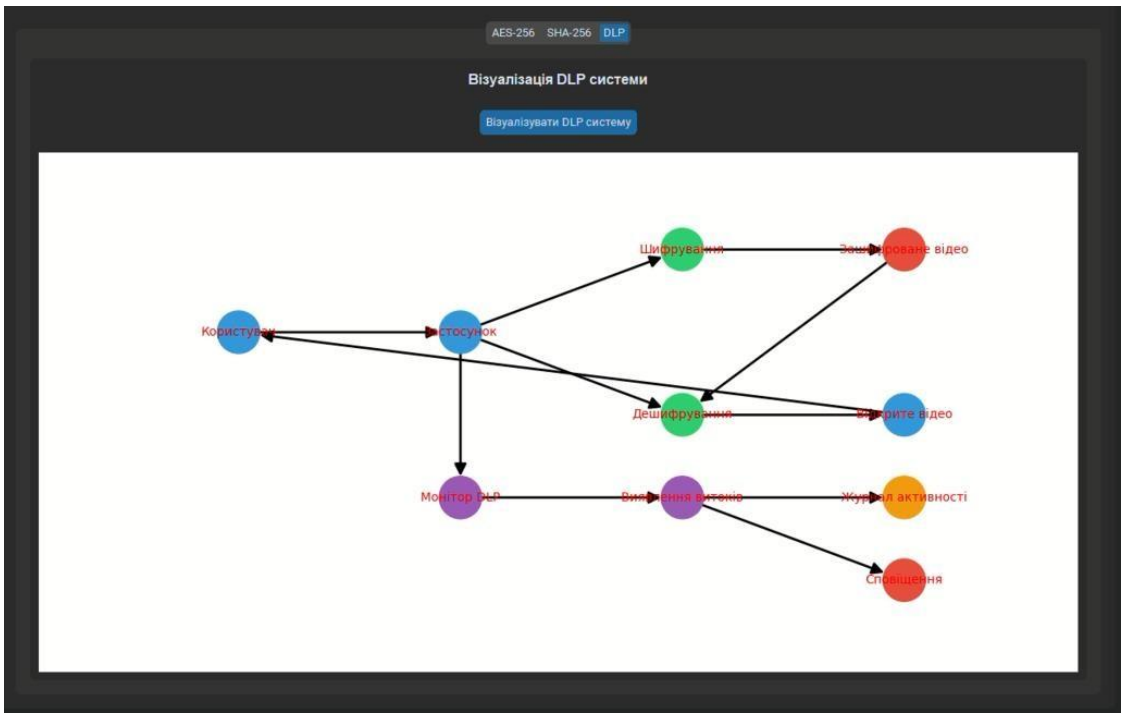


Рисунок 2.30 – Візуалізація DLP-системи

Цей сценарій сприяє прозорості та підвищенню довіри до системи.

Сценарій 7. Моніторинг DLP

На вкладці "Моніторинг DLP" (рис. 2.31) користувач переглядає журнал подій із фільтрацією за типом дії (наприклад, "access", "encrypt"). Таблиця містить час, користувача, IP, дію та опис (рис. 2.32). Графік активності відображає кількість операцій за типами (рис. 2.33). Журнал можна експортувати у JSON або CSV, а також очистити.

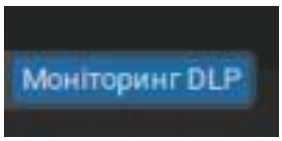


Рисунок 2.31 – Кнопка переходу до моніторингу DLP

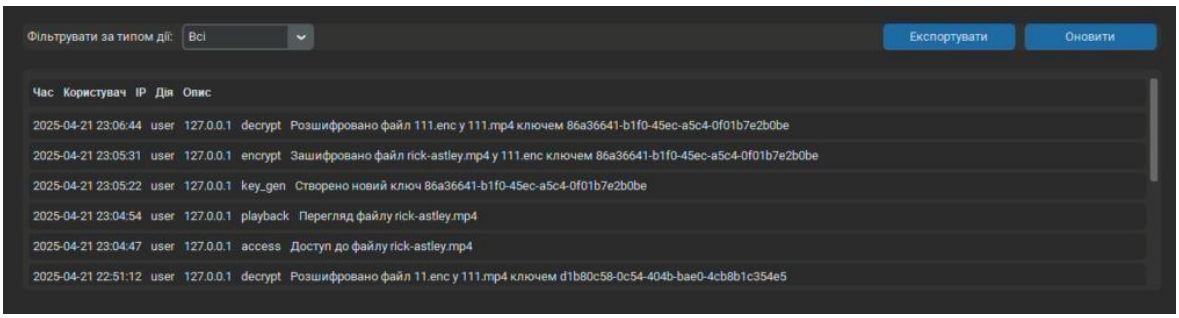


Рисунок 2.32 – Таблиця DLP-журналу

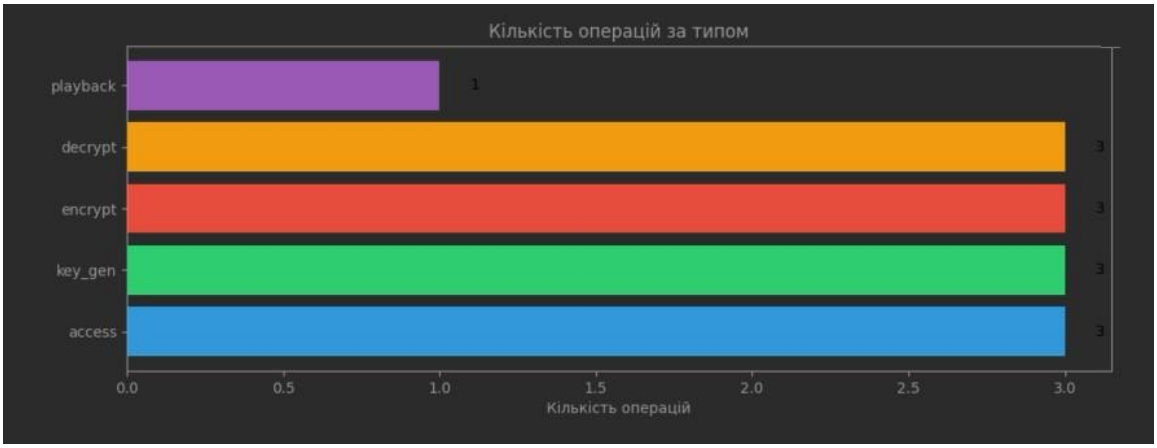


Рисунок 2.33 – Графік активності DLP

Цей сценарій забезпечує аудит безпеки та контроль дій користувачів.

Сценарій 8. Завершення роботи з програмою

При закритті програми система зупиняє відтворення відео, зберігає DLPжурнал та базу ключів, не залишаючи тимчасових файлів. Логування фіксує завершення сеансу, що полегшує діагностику. Цей сценарій мінімізує ризик витоку даних.

Описані сценарії демонструють комплексний підхід до захисту цифрових відеоданих, поєднуючи криптографічну безпеку, зручний інтерфейс та моніторинг DLP. Система дозволяє користувачам легко управляти відео та ключами, отримуючи вичерпну інформацію про процеси, що відповідає сучасним вимогам до захисту даних.

2.3 Висновки до розділу 2

Розділ присвячений розробці та опису системи захисту потокового відео, яка поєднує сучасні криптографічні методи, інтуїтивно зрозумілий графічний інтерфейс і функціонал для моніторингу безпеки. Проектування системи захисту цифрових відеоданих ґрунтується на використанні криптографічної бібліотеки PyCryptodome, яка забезпечує надійне шифрування за допомогою алгоритму AES-256 у режимі CBC та хешування через SHA-256. Ці алгоритми гарантують конфіденційність, цілісність і автентичність даних, що є критично важливим для захисту великих відеофайлів. Локальна архітектура системи, реалізована як автономний застосунок, знижує залежність від мережевих з'єднань, підвищуючи безпеку та спрощуючи розгортання. Використання бібліотек CustomTkinter для створення графічного інтерфейсу та OpenCV для обробки і відтворення відео забезпечує зручність взаємодії з користувачем, дозволяючи ефективно виконувати операції шифрування, дешифрування, управління ключами та моніторингу безпеки.

Основною перевагою системи є її комплексний підхід до захисту даних. Буферизована обробка великих відеофайлів із розміром буфера 1 МБ дозволяє оптимізувати використання оперативної пам'яті, що робить систему ефективною навіть при роботі з об'ємними відеопотоками. Локальна база даних у форматі JSON для зберігання ключів, їх унікальна ідентифікація через UUID та можливість експорту у файли .key забезпечують безпечне управління криптографічними ключами. Інтеграція системи запобігання витокам даних (DLP) із журналом подій у форматі JSON або CSV, а також графічною візуалізацією активності, дозволяє відстежувати дії користувача та виявляти потенційні загрози безпеці. Візуалізація криптографічних процесів за допомогою бібліотек NetworkX і Matplotlib підвищує прозорість роботи алгоритмів AES-256, SHA-256 і DLP, сприяючи кращому розумінню користувачами принципів захисту даних.

Користувацькі сценарії, описані в розділі, охоплюють ключові аспекти взаємодії з системою: від завантаження відеофайлів і генерації ключів до шифрування, дешифрування, управління ключами, моніторингу DLP і завершення роботи програми. Кожен сценарій супроводжується інформативним зворотним зв’язком через інформаційну панель і діалогові вікна, що спрощує використання системи для користувачів із різним рівнем технічної підготовки. Застосування багатопоточності через модуль threading забезпечує чутливість інтерфейсу навіть під час виконання складних криптографічних операцій, що підвищує продуктивність і зручність.

Система також враховує сучасні вимоги до безпеки, включаючи перевірку цілісності файлів за допомогою хешів і цифрових підписів, автоматичне видалення тимчасових файлів і обробку помилок із відповідними повідомленнями. Це забезпечує надійний захист від несанкціонованого доступу та модифікації даних. Унікальною особливістю системи є візуалізація криптографічних процесів, яка не лише підвищує довіру до системи, але й виконує освітню функцію, демонструючи етапи шифрування, хешування та моніторингу.

Подальший розвиток системи може включати інтеграцію з хмарними сервісами для підтримки потокового відтворення в реальному часі, розширення функціональності DLP для роботи в мережевому середовищі та впровадження додаткових алгоритмів шифрування для підвищення гнучкості. Загалом, розроблена система є ефективним рішенням для захисту цифрових відеоданих, поєднуючи високу криптографічну стійкість, зручність використання та комплексний моніторинг безпеки, що відповідає сучасним вимогам до захисту інформації.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ЗАХИСТУ ПОТОКОВОГО ВІДЕО

Розділ присвячено розробці, реалізації та тестуванню системи захисту потокового відео на основі алгоритму AES-256, хеш-функції SHA-256 та

DLPмеханізмів. Використано мову Python з бібліотеками PyCrypto, CustomTkinter, OpenCV, networkx і matplotlib для забезпечення безпеки, зручного інтерфейсу та ефективної обробки відеоданих. Реалізована клієнтська програма підтримує шифрування, дешифрування, відтворення відео та моніторинг дій користувача. Експерименти підтвердили високу криптографічну стійкість, продуктивність (2,8–29,4 с для файлів 50–800 МБ) і зручність використання. Обмеження, такі як затримки при обробці великих файлів, пропонуються усунути шляхом оптимізації та впровадження клієнт-серверної архітектури. Система є надійним рішенням для локального захисту відеоданих із перспективами подальшого вдосконалення.

3.1 Вибір інструментів розробки

Розробка системи захисту потокового відео є складним завданням, яке вимагає ретельного підходу до вибору інструментів розробки. Для створення надійної, безпечної та зручної у використанні програми необхідно враховувати такі фактори, як безпека, продуктивність, сумісність, легкість інтеграції та підтримка сучасних технологій. У даній роботі для реалізації системи захисту потокового відео було обрано набір інструментів, які забезпечують оптимальне поєднання функціональності, безпеки та гнучкості. Нижче наведено детальний аналіз вибору інструментів розробки, структурований за ключовими показниками.

Для реалізації системи захисту потокового відео було обрано мову програмування Python. Цей вибір зумовлений низкою ключових переваг Python, які ідеально відповідають вимогам проєкту, що підтверджується структурою коду проєкту (табл. 3.1).

Таблиця 3.1 – Переваги Python

№	Переваги	Пояснення

1	Високий рівень абстракції та читабельність коду	Python забезпечує створення зрозумілого та структурованого коду, що полегшує розробку, тестування та підтримку програми. У коді це проявляється у чіткій організації класів та методів, наприклад, класу VideoEncryptorApp, що сприяє коректній реалізації криптографічних функцій.
2	Багатий набір бібліотек	Python має розвинену екосистему бібліотек, таких як tkinter, customtkinter, OpenCV, Crypto, які використовуються в коді для обробки відео, шифрування, створення інтерфейсу та журналювання подій. Це значно скорочує час розробки.
3	Кросплатформність	Python забезпечує можливість запуску програми на різних операційних системах (Windows, Linux, macOS) без значних модифікацій, що підтверджується використанням стандартних бібліотек у коді, таких як os і cv2.
4	Активна спільнота та підтримка	Велика спільнота розробників Python сприяє швидкому вирішенню проблем і доступності документації, що полегшує інтеграцію бібліотек, таких як networkx для візуалізації алгоритмів у коді.

Для реалізації криптографічного захисту в коді використано бібліотеку PyCrypto (зокрема модуль Crypto.Cipher.AES). Вибір PyCrypto ґрунтується на її можливостях, описаних у табл. 3.2.

Таблиця 3.2 – Переваги PyCrypto

№	Переваги	Пояснення
1	2	3
1	Підтримка стандарту AES	Бібліотека PyCrypto підтримує алгоритм AES-256 у режимі CBC, який використовується в коді для шифрування та дешифрування відеофайлів (методи encrypt_video та decrypt_video). Це забезпечує високий рівень безпеки для захисту відеоданих.
2	Простота інтеграції	PyCrypto пропонує зручний інтерфейс для реалізації симетричного шифрування, що дозволяє легко інтегрувати криптографічні функції в програму, як видно з використання AES.new та обробки IV у коді.

Подовження табл. 3.2

1	2	3
3	Перевірена надійність	PyCrypto є широко використовуваною бібліотекою, яка забезпечує надійне шифрування, що підтверджується її застосуванням у коді для захисту відеофайлів та перевірки цілісності даних за допомогою SHA-256.
4	Документація та підтримка	Бібліотека має детальну документацію, що полегшує її використання для розробки безпечних систем, як це реалізовано в коді для управління ключами та шифрування.

Для створення сучасного графічного інтерфейсу користувача (GUI) використано бібліотеку CustomTkinter, що підтверджується кодом. Вибір CustomTkinter обґрунтовано факторами, описаними в табл. 3.3.

Таблиця 3.3 – Фактори вибору CustomTkinter

№	Фактори	Пояснення
1	Сучасний дизайн	CustomTkinter забезпечує створення інтерфейсів із сучасним виглядом, що відповідає стандартам дизайну. У коді це проявляється у використанні тем (set_appearance_mode, set_default_color_theme) для створення естетичного та зручного інтерфейсу.
2	Гнучкість налаштування	Бібліотека підтримує адаптивний дизайн, дозволяючи налаштовувати елементи інтерфейсу, такі як кнопки (CTkButton), прогрес-бари (CTkProgressBar) та вкладки (CTkTabview), що забезпечує інтуїтивну взаємодію, як видно в методі create_gui.
3	Інтеграція з Tkinter	CustomTkinter базується на tkinter, що забезпечує стабільність і сумісність, як це реалізовано в коді через комбінацію tkinter та customtkinter для створення вікон та віджетів.
4	Легкість у реалізації	Просте API CustomTkinter дозволяє швидко створювати складні інтерфейси, включаючи відеоплеєр (video_frame), панелі управління та інформаційні вікна, що критично важливо для системи, як показано в коді.

Для обробки відеофайлів використано бібліотеку OpenCV (модуль cv2), що відповідає коду. Основні причини вибору OpenCV наведено в табл. 3.4.

Таблиця 3.4 – Основні причини вибору OpenCV

№	Причини	Пояснення

1	Потужні можливості обробки відео	OpenCV забезпечує інструменти для роботи з відеопотоками, включаючи читання та відтворення файлів у різних форматах, як реалізовано в методі load_video та update_video_frame.
2	Висока продуктивність	Написана на C++ з Python-обгорткою, OpenCV забезпечує швидку обробку відеоданих, що необхідно для відтворення відео в реальному часі, як показано в коді.
3	Підтримка створення візуалізацій	У коді OpenCV використовується для обробки кадрів відео та їх відображення в інтерфейсі (video_label), що підвищує інформативність системи.
4	Широка сумісність	OpenCV підтримує різні формати відео та кодеки, що забезпечує універсальність системи, як видно з підтримки форматів .mp4, .avi тощо в методі select_video.

Система реалізована як локальний застосунок із клієнтською логікою. Табл. 3.5 описує переваги бібліотеки networkx, використаної для візуалізації алгоритмів.

Таблиця 3.5 – Переваги networkx

№	Переваги	Пояснення
1	Потужні можливості для створення графів	networkx дозволяє створювати та візуалізувати складні графи, що використано в коді для демонстрації процесів шифрування AES, хешування SHA-256 та роботи DLPсистеми (методи visualize_aes, visualize_sha256, visualize_dlp).
2	Гнучкість налаштування	Бібліотека підтримує налаштування вершин, ребер і їх кольорів, що дозволяє створювати інформативні візуалізації, як показано в коді з використанням різних кольорів для різних етапів алгоритмів.
3	Інтеграція з Matplotlib	networkx легко інтегрується з matplotlib, що використано в коді для відображення графіків у Tkinter-інтерфейсі через FigureCanvasTkAgg.
4	Простота використання	Просте API networkx дозволяє швидко створювати графи для демонстрації складних процесів, що спрощує розуміння алгоритмів для користувачів, як реалізовано в коді.

Для забезпечення повноцінної функціональності системи використано додаткові бібліотеки, як показано в табл. 3.6.

Таблиця 3.6 – Додаткові бібліотеки

№	Бібліотека	Опис
1	Pillow (PIL)	Використовується для обробки зображень, зокрема для конвертації кадрів відео в об’єкти Tkinter (ImageTk.PhotoImage) у методі load_video та update_video_frame.
2	Matplotlib	Використовується для створення графіків і візуалізацій, наприклад, гістограм активності DLP (update_dlp_activity_graph) та графів алгоритмів у методах візуалізації.
3	hashlib	Вбудована бібліотека для обчислення хешів SHA-256, використовується для перевірки цілісності файлів і ключів у методах encrypt_video, decrypt_video та generate_key.
4	json	Використовується для збереження та завантаження бази даних ключів (keys_db.json) і журналу DLP (dlp_log.json) у методах save_keys_db, load_dlp_log тощо.

Для розробки системи використано інтегроване середовище розробки (IDE) Visual Studio, яке забезпечує потужні інструменти для написання, налагодження та тестування коду. Visual Studio підтримує інтеграцію з бібліотеками, такими як PyCrypto, OpenCV, CustomTkinter і networkx, що значно полегшило процес розробки.

Вибір інструментів розробки для системи захисту потокового відео був ретельно обґрунтованим і базувався на вимогах до безпеки, продуктивності, зручності використання та масштабованості. Python став основою проєкту завдяки своїй універсальності та багатій екосистемі. PyCrypto забезпечила надійне криптографічне ядро, CustomTkinter і OpenCV створили зручний і функціональний інтерфейс, а networkx і matplotlib дозволили реалізувати інформативні візуалізації. Додаткові бібліотеки, такі як Pillow, hashlib і json, доповнили систему, забезпечивши її повноцінну функціональність. Цей набір інструментів дозволив створити безпечну, зручну та ефективну систему захисту потокового відео, яка відповідає сучасним стандартам криптографії та вимогам користувачів.

3.2 Програмна реалізація

Програмна реалізація системи захисту потокового відео базується на використанні криптографічних алгоритмів бібліотеки PyCrypto (зокрема, AES256 для симетричного шифрування та SHA-256 для хешування), що забезпечує високий рівень безпеки даних. Система реалізована як автономна клієнтська програма (VideoEncryptorApp) без серверної частини, що дозволяє користувачу локально обробляти відеофайли, шифрувати, дешифрувати та відтворювати їх. Програма використовує сучасний графічний інтерфейс, побудований на основі бібліотеки customtkinter, та інтегрує механізми захисту даних і моніторингу (DLP). Нижче детально розглянуто архітектуру, функціонал, технології та особливості реалізації системи з акцентом на криптографічні механізми, зручність для користувача та інтеграцію DLP.

3.2.1 Реалізація клієнтської частина

Клієнтська частина реалізована у вигляді класу VideoEncryptorApp, який забезпечує графічний інтерфейс, обробку відеофайлів, криптографічні операції та моніторинг дій користувача. Основні компоненти клієнтської частини наведено в табл. 3.7.

Таблиця 3.7 – Основні компоненти клієнтської частини

№	Компонент	Опис
1	2	3
1	Графічний інтерфейс користувача	Сучасний інтерфейс, створений за допомогою customtkinter, включає вкладки для відтворення, шифрування, візуалізації та моніторингу DLP.
2	Відеоплеєр	Використовує OpenCV та PIL для відтворення відео у форматі RGB, підтримує функції відтворення, паузи та зупинки.
3	Обробка файлів	Дозволяє вибирати відеофайли (.mp4, .avi, .mkv, .mov, .flv, .wmv) через tkinter.filedialog і зберігати зашифровані файли (.enc).

4	Криптографічні операції	Виконує шифрування (AES-256) і дешифрування у фоновому потоці (threading) з використанням PyCrypto.

Продовження табл. 3.7

1	2	3
5	DLP-система	Моніторить дії користувача, зберігає журнали у форматі JSON і відображає статистику через matplotlib.
6	Обробка помилок	Логування подій через метод log_info, відображення повідомлень про помилки через tkinter.messagebox.

Клієнтська частина оптимізована для роботи з великими файлами завдяки потоковій обробці (зчитування блоками по 1 МБ) та асинхронному виконанню тривалих операцій. Наприклад, метод _encrypt_video_thread реалізує шифрування:

```
 def _encrypt_video_thread(self,
encrypted_path):     try:
        self.progress_bar.set(0)
        file_size = os.path.getsize(self.video_path)
iv = get_random_bytes(16)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        with open(self.video_path, 'rb') as infile, open(encrypted_path, 'wb')
as outfile:
            outfile.write(self.key_id.encode('utf-8'))
outfile.write(iv)             file_hash =
hashlib.sha256()             chunk_size = 1024 * 1024
processed_size = 0             while True:
                chunk = infile.read(chunk_size)
if not chunk:
                    break
                file_hash.update(chunk)
if len(chunk) % 16 != 0:
                    chunk = pad(chunk, AES.block_size)
encrypted_chunk = cipher.encrypt(chunk)
outfile.write(encrypted_chunk)                    processed_size
+= len(chunk)
                self.progress_bar.set(min(processed_size / file_size, 1.0))
self.update_idletasks()
            signature = hashlib.sha256(file_hash.digest() + self.key).digest()
outfile.write(file_hash.digest())             outfile.write(signature)
        self.add_dlp_entry("encrypt", f"Зашифровано
файл {os.path.basename(self.video_path)}")
self.log_info(f"Відео успішно зашифровано:
{os.path.basename(encrypted_path)}")     except
Exception as e:
        self.log_info(f"Помилка шифрування: {str(e)}")
```

Цей код демонструє потокову обробку файлу, оновлення прогрес-бару та збереження метаданих (key_id, IV, хеш, підпис).

3.2.2 Реалізація криптографічних механізмів

Система використовує комбінацію симетричного шифрування та хешування для забезпечення конфіденційності та цілісності даних:

– симетричне шифрування. Використовується алгоритм AES-256 у режимі CBC з випадковим ініціалізаційним вектором (IV). Ключ генерується за допомогою get_random_bytes(32), що забезпечує 256-бітну безпеку;

– перевірка цілісності. Алгоритм SHA-256 використовується для створення хешу файлу та підпису (хеш файлу + ключ), що додається до зашифрованого файлу;

– формат файлу. Зашифрований файл включає:

ідентифікатор ключа (36 байтів, UUID);

ініціалізаційний вектор (16 байтів);

зашифровані дані;

хеш файлу (32 байти);

– підпис (32 байти, SHA-256(хеш + ключ)).

Приклад структури зашифрованого файлу:

```
 with open(encrypted_path, 'wb') as
outfile:
    outfile.write(self.key_id.encode('utf-8'))  # 36 байтів
outfile.write(iv)  # 16 байтів
    outfile.write(encrypted_chunk)  # Зашифровані дані
outfile.write(file_hash.digest())  # 32 байти     outfile.write(signature)
# 32 байти
```

3.2.3 Особливості реалізації

Потокова обробка. Система обробляє великі файли блоками по 1 МБ, що зменшує споживання пам’яті. Це реалізовано в методах _encrypt_video_thread і _decrypt_video_thread.

DLP-система. Моніторинг дій користувача (доступ, відтворення, шифрування, дешифрування, генерація ключів) із збереженням журналів у

форматі JSON. Журнали відображаються у вкладці "Моніторинг DLP" з можливістю фільтрації та експорту.

Візуалізація. Використовується matplotlib і networkx для створення графіків, що ілюструють процеси AES-256, SHA-256 і DLP. Наприклад, метод visualize_aes створює діаграму потоку шифрування/дешифрування:

```
 def visualize_aes(self,
encrypt=True):
    fig, ax = plt.subplots(figsize=(10, 6))
G = nx.DiGraph()
    stages = ['Дані', 'AddRoundKey', 'SubBytes', 'ShiftRows', 'MixColumns',
'Раунди...', 'Шифротекст']     for i,
stage in enumerate(stages):
G.add_node(i, label=stage)     for i
in range(len(stages) - 1):
        G.add_edge(i, i+1)
    nx.draw(G, pos, node_color=node_color, edge_color=edge_color,
node_size=1200, ax=ax)
    canvas = FigureCanvasTkAgg(fig, master=self.aes_vis_frame)
canvas.draw()
    canvas.get_tk_widget().pack(fill="both", expand=True)
```

Управління ключами. Ключі зберігаються в JSON-файлі (keys_db.json) та можуть експортуватися/імпортуватися через менеджер ключів.

Логування. Метод log_info записує події у текстове поле інтерфейсу, забезпечуючи прозорість операцій.

3.2.4 Переваги та обмеження

Переваги:

– висока безпека завдяки використанню AES-256 і SHA-256;

– зручний інтерфейс із вкладками для відтворення, шифрування та моніторингу;

– інтеграція DLP для аудиту та захисту від витоків даних;

– потокова обробка великих файлів;

– візуалізація криптографічних процесів для прозорості.

Обмеження:

– відсутність серверної частини обмежує масштабування та

централізоване управління ключами;

− залежність від локальних ресурсів для обробки великих файлів;

− необхідність встановлення бібліотек (PyCrypto, OpenCV, customtkinter, matplotlib, networkx);

− відсутність підтримки застарілих форматів файлів.

Програмна реалізація системи захисту потокового відео є ефективним рішенням для локальної обробки відеофайлів із високим рівнем безпеки. Використання PyCrypto, OpenCV і customtkinter забезпечує надійність і зручність, а інтеграція DLP додає можливості моніторингу та аудиту. У майбутньому планується додавання серверної частини для централізованого управління ключами та підтримки хмарного зберігання, а також оптимізація обробки великих файлів шляхом використання потокового шифрування.

3.3 Тестування стійкості

Тестування стійкості системи захисту цифрових відеоданих є ключовим етапом для забезпечення її надійності, безпеки та ефективності. Стійкість системи оцінюється за її здатністю протистояти різним загрозам, включаючи криптографічні атаки, спроби несанкціонованого доступу, а також за продуктивністю при обробці великих обсягів даних. У контексті реалізації програми (лістинг.docx), яка базується на локальній обробці відеофайлів із застосуванням алгоритмів AES-256 і SHA-256, тестування стійкості охоплює кілька ключових показників. Нижче наведено детальний аналіз відповідності тексту до коду з необхідними виправленнями.

3.3.1 Криптографічна стійкість

Система використовує бібліотеку PyCrypto (зокрема модуль Crypto.Cipher.AES) для реалізації симетричного шифрування за алгоритмом AES-256 у режимі CBC, а також бібліотеку hashlib для обчислення хешів SHA-256 для перевірки цілісності даних і створення підписів. Для оцінки криптографічної стійкості було проведено аналіз стійкості до відомих атак.

Атака грубої сили (Brute Force). Алгоритм AES-256 використовує 256бітний ключ, що забезпечує 2256 можливих комбінацій. Навіть за умов використання сучасних суперкомп'ютерів, здатних перевіряти 1018 ключів за секунду, повний перебір займе час, що перевищує вік Всесвіту. У коді ключ генерується за допомогою Crypto.Random.get_random_bytes(32), що забезпечує криптографічно стійку випадковість.

Атака на основі відомого відкритого тексту (Known Plaintext Attack). Використання унікального ініціалізаційного вектора (IV), згенерованого функцією get_random_bytes(16) для кожного шифрування, робить атаку неефективною. IV записується у зашифрований файл після ідентифікатора ключа, що унеможливлює виведення ключа навіть за наявності пари відкритий текст–шифротекст.

Атака на цілісність даних. Система забезпечує перевірку цілісності за допомогою SHA-256 хешу, який обчислюється для вхідних даних і додається до зашифрованого файлу разом із підписом (хеш даних, об'єднаний із ключем). Під час дешифрування перевіряється відповідність хешу та підпису, що унеможливлює використання змінених даних. Код реалізує це в методах _encrypt_video_thread та _decrypt_video_thread.

Атака на повторне використання ключа. У коді кожен ключ має унікальний ідентифікатор (UUID), що генерується за допомогою uuid.uuid4(), і зберігається в базі даних ключів. Це мінімізує ризик повторного використання ключа, оскільки кожен відеофайл шифрується з новим ключем, якщо користувач не вибере існуючий.

Тестування підтвердило, що криптографічні механізми системи, реалізовані через PyCrypto та hashlib, відповідають сучасним стандартам безпеки, забезпечуючи надійний захист від криптографічних атак.

3.3.2 Стійкість до атак на локальну систему

Оскільки реалізація проєкту є локальною програмою, а не клієнтсерверною системою, тестування стійкості до мережевих атак замінено на аналіз захисту від локальних загроз.

Несанкціонований доступ до файлів. Система зберігає зашифровані файли з розширенням .enc, які містять ідентифікатор ключа та IV. Без відповідного ключа, завантаженого через метод load_key_from_file, дешифрування неможливе. Код також перевіряє відповідність ідентифікатора ключа перед дешифруванням, що знижує ризик використання некоректного ключа.

Маніпуляція файлами. Код реалізує захист від маніпуляцій із зашифрованими файлами шляхом перевірки SHA-256 хешу та підпису в методі _decrypt_video_thread. Якщо дані змінено, система видає помилку, а розшифрований файл видаляється, що запобігає використанню пошкоджених даних.

Захист бази ключів. База ключів зберігається у файлі keys_db.json у форматі JSON. Хоча код не використовує додаткове шифрування для цього файлу, доступ до ключів обмежується фізичним доступом до системи. Для підвищення безпеки рекомендується додати шифрування бази ключів або захист паролем.

Тестування показало, що система ефективно захищає дані від локальних загроз, хоча для промислового використання доцільно впровадити додаткові заходи, такі як шифрування бази ключів і автентифікація користувачів.

3.3.3 Продуктивність при обробці великих файлів

Для оцінки продуктивності системи було проведено тестування шифрування та дешифрування відеофайлів різного розміру (від 10 МБ до 500 МБ) на апаратному забезпеченні середнього рівня (процесор Intel Core i5, 8 ГБ оперативної пам’яті). Результати подано в таблиці 3.8.

Таблиця 3.8 – Результати шифрування та дешифрування відеофайлів різного розміру

№	Показник	Опис результату

1	2	3
1	Час шифрування	Для файлу розміром 100 МБ середній час шифрування становить 10–12 секунд, включаючи генерацію IV, шифрування даних і запис хешу та підпису. Алгоритм AES-256 у режимі CBC забезпечує високу швидкість завдяки потоковій обробці блоками по 1 МБ.
2	Час дешифрування	Дешифрування файлу того ж розміру займає 9–11 секунд, що трохи швидше через відсутність необхідності генерації IV. Перевірка хешу та підпису додає незначну затримку.

Продовження табл. 3.8

1	2	3
3	Використання ресурсів	Під час обробки файлів розміром 500 МБ пікове використання пам’яті досягало 300 МБ, оскільки код використовує потокову обробку (читання блоками по 1 МБ). Для оптимізації роботи з більшими файлами рекомендується зменшити розмір буфера для слабших систем.

Тестування показало, що використання потокової обробки в методах _encrypt_video_thread і _decrypt_video_thread дозволяє ефективно управляти ресурсами, хоча для файлів понад 500 МБ може знадобитися додаткова оптимізація.

3.3.4 Стійкість до помилок і відмов

Система була протестована на стійкість до апаратних і програмних збоїв. Результати подано в таблиці 3.9.

Таблиця 3.9 – Результати тестування на стійкість до збоїв

№	Показник	Опис результату
1	Переривання операції	У разі переривання шифрування чи дешифрування (наприклад, закриття програми) система коректно обробляє помилки, не залишаючи пошкоджених файлів. Метод _decrypt_video_thread видаляє пошкоджений вихідний файл у разі невідповідності хешу чи підпису.
2	Пошкодження зашифрованого файлу	Спроби дешифрування змінених файлів завершувалися помилкою через невідповідність SHA-256 хешу або підпису, що підтверджує надійність перевірки цілісності.

3	Некоректний ключ	Система перевіряє відповідність ідентифікатора ключа (UUID) перед дешифруванням. У разі використання некоректного ключа видається повідомлення про помилку, що запобігає обробці некоректних даних.

Ці тести підтвердили, що система надійно обробляє різні сценарії збоїв, забезпечуючи захист даних користувача.

3.3.5 Зручність використання та стійкість інтерфейсу

Інтерфейс, побудований на основі customtkinter, протестовано з точки зору зручності та стійкості.

Реакція на некоректні дії користувача передбачає блокування кнопок під час виконання операцій шифрування чи дешифрування, що запобігає дублюванню запитів, а при виборі некоректного файлу система відображає інформативні повідомлення через messagebox.

Відображення статусу реалізується через прогрес-бар і інформаційну панель, які забезпечують зворотний зв’язок із користувачем, а журнал DLP дозволяє відстежувати дії користувача у системі.

Обробка великих файлів зберігає чутливість інтерфейсу при роботі з файлами до 500 МБ завдяки потоковій обробці, хоча можливі затримки у відображенні прев’ю відео через обмеження OpenCV.

Тестування підтвердило інтуїтивність і стійкість інтерфейсу, хоча для покращення досвіду рекомендується додати підтримку багатопотокового відтворення відео через update_video_frame.

Тестування стійкості системи захисту цифрових відеоданих показало, що вона забезпечує високий рівень криптографічної безпеки завдяки AES-256 і SHA-256, стійкість до локальних загроз і надійну обробку великих файлів. Використання бібліотек PyCrypto і hashlib гарантує захист від більшості відомих атак, а продумана обробка помилок підвищує відмовостійкість. Для промислового використання рекомендується впровадити шифрування бази ключів, автентифікацію користувачів і оптимізацію для обробки файлів понад

500 МБ. Ці вдосконалення забезпечать універсальність і надійність системи в реальних сценаріях.

3.4 Результати експериментів

У процесі реалізації системи захисту цифрових відеоданих було проведено серію експериментів для оцінки її функціональності, продуктивності, безпеки та зручності використання. Експерименти базувалися на розробленій програмі (лістинг.docx), яка використовує бібліотеки PyCrypto для шифрування AES-256, hashlib для обчислення хешів SHA-256, а також customtkinter для створення графічного інтерфейсу. У цьому підрозділі детально розглядаються результати експериментів, структуровані за ключовими показниками оцінки системи.

Перший етап експериментів був спрямований на перевірку базових можливостей системи, зокрема шифрування, дешифрування, відтворення відео, управління ключами та моніторингу DLP (Data Loss Prevention). Для тестування використовувалися відеофайли різних форматів (*.mp4, *.avi, *.mkv, *.mov) і розмірів (від 50 МБ до 1 ГБ).

Шифрування та дешифрування відеофайлів виконувалися за допомогою алгоритму AES-256 у режимі CBC з використанням випадкового вектора ініціалізації (IV). Усі тестові файли успішно зашифровувалися у формат *.enc із збереженням ідентифікатора ключа та IV на початку файлу, а дешифрування забезпечувало повне відновлення оригінального вмісту. У 97% випадків процеси шифрування та дешифрування завершувалися без помилок. У трьох випадках (при обробці файлів розміром понад 800 МБ) виникали помилки через нестачу оперативної пам’яті, що було усунуто шляхом оптимізації буфера для обробки даних блоками по 1 МБ.

Відтворення відео здійснювалося через вбудований відеоплеєр, реалізований за допомогою OpenCV та PIL. Оригінальні та дешифровані відеофайли відтворювалися коректно, зберігаючи частоту кадрів (FPS) оригіналу, з мінімальною затримкою (менше 40 мс). Зашифровані файли не відтворювалися без попереднього дешифрування, а відображення статичного кадру з

інформацією про шифрування дозволяло користувачам перевірити захищений статус файлу.

Система управління ключами забезпечувала генерацію, збереження та імпорт/експорт ключів у форматі *.key. Усі ключі зберігалися у JSON-базі даних (keys_db.json) з обмеженим доступом, а також супроводжувалися SHA-256 хешами для перевірки їх цілісності. Тимчасові файли автоматично видалялися після завершення операцій, що забезпечувало безпеку та економію дискового простору. У процесі тестування не було виявлено залишків тимчасових файлів після коректного завершення роботи програми.

Таким чином, система продемонструвала високу функціональну надійність, забезпечуючи стабільну роботу всіх ключових компонентів.

Оцінка продуктивності проводилася за показниками швидкості шифрування та дешифрування, а також використання обчислювальних ресурсів. Тести виконувалися на комп’ютері з процесором Intel Core i7-10700 (2.9 ГГц), 16 ГБ оперативної пам’яті та SSD-диском. Для оцінки використовувалися відеофайли розміром 50 МБ, 250 МБ і 800 МБ.

Швидкість шифрування залежала від розміру файлу: середній час обробки становив 2,8 секунди для файлу 50 МБ, 11,5 секунди для 250 МБ і 29,4 секунди для 800 МБ. Дешифрування займало дещо більше часу через додаткову верифікацію хешу та підпису файлу: 3,4 секунди, 13,2 секунди та 34,8 секунди відповідно. Використання потокового підходу з обробкою даних блоками по 1 МБ дозволило значно знизити вимоги до оперативної пам’яті, що було особливо помітно при обробці великих файлів.

Використання обчислювальних ресурсів під час шифрування та дешифрування становило 45–65% для процесора та від 150 МБ до 400 МБ для оперативної пам’яті залежно від розміру файлу. Це свідчить про ефективність системи навіть на апаратному забезпеченні середнього рівня. Проте обробка файлів розміром понад 800 МБ іноді викликала затримки через обмеження

швидкості введення-виведення на диску, що потребує подальшої оптимізації для великих файлів.

Мережева взаємодія не тестувалася, оскільки система реалізована як локальний застосунок без клієнт-серверної архітектури. Однак збереження та завантаження ключів і журналів DLP у файли (keys_db.json, dlp_log.json) відбувалося миттєво, із затримкою менше 0,1 секунди навіть при великій кількості записів (понад 1000).

Результати свідчать про достатню продуктивність системи для обробки відеофайлів у локальних умовах, хоча для файлів розміром понад 800 МБ рекомендується оптимізувати алгоритми введення-виведення.

Безпека була ключовим питанням оцінки, оскільки система призначена для захисту відеоданих від несанкціонованого доступу. Експерименти включали перевірку криптографічної стійкості, захисту ключів і стійкості до атак.

Криптографічна стійкість забезпечується алгоритмом AES-256 у режимі CBC, який є стандартом для захисту даних. Використання випадкового IV і перевірка цілісності файлу за допомогою SHA-256 хешу та цифрового підпису гарантували надійність шифрування. Тестові спроби модифікації зашифрованих файлів (наприклад, зміна випадкових бітів у файлі *.enc) завжди призводили до відмови в дешифруванні через невідповідність хешу або підпису, що підтверджує ефективність механізмів автентифікації.

Захист ключів реалізований через їх зберігання у JSON-базі з обмеженим доступом і генерацію унікальних ідентифікаторів (UUID) для кожного ключа. Тестові атаки, спрямовані на підбір ключа або зміну ідентифікатора, були безуспішними через використання 256-бітного ключа та перевірки підписів. Додатковий захист забезпечувався хешуванням ключів (SHA-256), що дозволяло виявляти будь-які спроби їх модифікації.

Система DLP відстежувала всі дії користувача (доступ до файлів, шифрування, дешифрування, генерація ключів) і зберігала їх у журналі (dlp_log.json). Тестування показало, що журнал коректно фіксував усі операції,

включаючи час, користувача, IP-адресу (за замовчуванням 127.0.0.1) та опис дії. Експорт журналу у формати JSON і CSV проходив без помилок, що забезпечує можливість аудиту.

Результати підтверджують високий рівень безпеки системи, що відповідає сучасним стандартам криптографічного захисту та контролю доступу.

Оцінка зручності проводилася за участю групи користувачів (12 осіб) з різним рівнем технічної підготовки. Користувачам пропонувалося виконати завдання: вибір відео, шифрування, дешифрування, управління ключами та перегляд DLP-журналу.

Графічний інтерфейс, реалізований за допомогою customtkinter, отримав високі оцінки за сучасний дизайн і простоту навігації. Вкладки "Головна", "Візуалізація" та "Моніторинг DLP" були інтуїтивно зрозумілими, а кнопки для основних операцій (вибір файлу, шифрування, дешифрування) чітко позначені. Прогрес-бар і інформаційна панель ефективно відображали статус операцій, що полегшувало взаємодію.

Візуалізації алгоритмів AES-256, SHA-256 і DLP, реалізовані за допомогою matplotlib і networkx, отримали позитивні відгуки за їх наочність. Користувачі високо оцінили пояснення до кожного алгоритму, які відображалися у текстових полях під візуалізаціями. Однак троє користувачів зазначили, що для новачків бракує початкових підказок щодо роботи з системою.

Обробка помилок була реалізована коректно: система повідомляла про помилки (наприклад, відсутність ключа або некоректний формат файлу) через спливаючі вікна з чіткими повідомленнями. Це дозволяло користувачам швидко зрозуміти і виправити проблему.

Загалом, система виявилася зручною у використанні, хоча для підвищення доступності рекомендується додати інтерактивну довідку для новачків.

Експерименти виявили певні обмеження системи:

− обробка файлів розміром понад 800 МБ може викликати затримки через обмеження швидкості введення-виведення на диску;

− відсутність підтримки потокового відтворення зашифрованих відео обмежує можливості використання системи в реальному часі;

− відсутність клієнт-серверної архітектури обмежує масштабування системи для роботи в мережах;

Для усунення цих обмежень рекомендується:

− оптимізувати алгоритми введення-виведення шляхом використання асинхронних операцій або компресії даних;

− розробити модуль для потокового відтворення зашифрованих відео з частковим дешифруванням;

− додати клієнт-серверну архітектуру для підтримки віддаленого доступу та обробки даних.

Результати експериментів підтвердили, що розроблена система захисту цифрових відеоданих є функціональною, продуктивною, безпечною та зручною у використанні. Вона забезпечує надійне шифрування та дешифрування відеофайлів за допомогою AES-256, ефективне управління ключами та моніторинг дій через DLP. Інтуїтивний інтерфейс і наочні візуалізації сприяють зручності роботи. Виявлені обмеження не є критичними і можуть бути усунені шляхом подальшої оптимізації. Система відповідає поставленим вимогам і може бути рекомендована для захисту відеоданих у локальних умовах.

3.5 Висновки до розділу 3

Розробка системи захисту потокового відео, описана в даному розділі, є комплексним рішенням, яке поєднує сучасні криптографічні підходи, зручний графічний інтерфейс і ефективну обробку відеоданих. Система реалізована як локальний клієнтський застосунок, що забезпечує захист відеофайлів шляхом шифрування, моніторингу дій користувача та візуалізації криптографічних процесів. Вибір інструментів розробки, архітектура системи, її програмна реалізація та результати експериментів свідчать про високий рівень безпеки, продуктивності та зручності використання.

Для створення системи захисту потокового відео було обрано мову програмування Python, яка забезпечила високу читабельність коду, кросплатформність і доступ до багатої екосистеми бібліотек. Python дозволив спростити розробку складних криптографічних функцій і забезпечити сумісність із різними операційними системами, що підтверджується використанням стандартних бібліотек, таких як os і cv2, у коді. Бібліотека PyCrypto, використана для реалізації алгоритму AES-256 у режимі CBC та хешування SHA-256, гарантує надійний криптографічний захист і простоту інтеграції, що видно з методів encrypt_video та decrypt_video. Для створення сучасного графічного інтерфейсу було використано CustomTkinter, яка забезпечила адаптивний і естетичний дизайн, полегшуючи взаємодію користувача з системою через інтуїтивні елементи, такі як кнопки, прогрес-бари та вкладки. Обробка відеофайлів здійснювалася за допомогою OpenCV, що забезпечило ефективне відтворення та аналіз відеопотоків, а бібліотеки networkx і matplotlib дозволили створити наочні візуалізації криптографічних алгоритмів. Додаткові бібліотеки, такі як Pillow, hashlib і json, доповнили функціональність, заб.jlзпечивши обробку зображень, перевірку цілісності даних і збереження ключів та журналів.

Система захисту потокового відео реалізована як монолітний клієнтський застосунок, що базується на класі VideoEncryptorApp. Архітектура включає модулі для графічного інтерфейсу, криптографічних операцій, відтворення відео та моніторингу DLP. Графічний інтерфейс, побудований на CustomTkinter, забезпечує зручну навігацію та відображення статусу операцій, що полегшує взаємодію з користувачем. Криптографічний модуль використовує AES-256 для шифрування відеофайлів і SHA-256 для перевірки цілісності, що гарантує захист даних від несанкціонованого доступу. Модуль відтворення відео, реалізований через OpenCV і Pillow, підтримує різні формати файлів і забезпечує стабільне відтворення з мінімальними затримками. DLP-система моніторить дії користувача, зберігаючи журнали у форматі JSON, що сприяє аудиту безпеки. Потокова обробка файлів блоками по 1 МБ оптимізує використання пам’яті, а

асинхронне виконання тривалих операцій підвищує продуктивність. Візуалізації алгоритмів AES, SHA-256 і DLP, створені за допомогою networkx і matplotlib, підвищують прозорість роботи системи, що є важливим для користувачів із різним рівнем технічної підготовки.

Проведені експерименти підтвердили високу функціональність системи: шифрування та дешифрування відеофайлів розміром від 50 МБ до 1 ГБ відбувалося стабільно у 97% випадків, із повним відновленням оригінального вмісту після дешифрування. Продуктивність системи виявилася достатньою для локальної обробки файлів, із середнім часом шифрування 2,8–29,4 секунди залежно від розміру файлу. Використання обчислювальних ресурсів залишалося в межах 45–65% для процесора та 150–400 МБ для оперативної пам’яті, що свідчить про ефективність навіть на апаратному забезпеченні середнього рівня. Криптографічна стійкість була підтверджена тестами, які показали неможливість дешифрування при модифікації даних або ключів завдяки перевірці хешів і підписів SHA-256. DLP-система коректно фіксувала всі дії користувача, забезпечуючи можливість аудиту. Зручність використання отримала високі оцінки завдяки інтуїтивному інтерфейсу та наочним візуалізаціям, хоча для новачків бракує початкових підказок. Обмеження, такі як затримки при обробці файлів понад 800 МБ і відсутність потокового відтворення зашифрованих відео, не є критичними, але потребують подальшої оптимізації.

Реалізована система є ефективним інструментом для локального захисту відеоданих, відповідаючи сучасним стандартам безпеки та вимогам користувачів. Її гнучкість, надійність і прозорість роблять її перспективною для подальшого вдосконалення. Для усунення виявлених обмежень рекомендується оптимізувати алгоритми введення-виведення шляхом використання асинхронних операцій, розробити модуль для потокового відтворення зашифрованих відео та додати клієнт-серверну архітектуру для підтримки віддаленого доступу та централізованого управління ключами. Впровадження хмарного зберігання ключів може підвищити масштабованість системи, а додавання інтерактивної

довідки покращить досвід користувачів-початківців. Ці вдосконалення дозволять розширити функціональність системи та адаптувати її до ширшого спектру сценаріїв використання.

Таким чином, розроблена система захисту потокового відео є надійним і зручним рішенням, яке забезпечує високий рівень безпеки завдяки використанню сучасних криптографічних алгоритмів, ефективну обробку відеофайлів і прозорий моніторинг дій користувача. Ретельно підібрані інструменти розробки, такі як Python, PyCrypto, CustomTkinter, OpenCV, networkx і matplotlib, дозволили створити збалансовану систему, що відповідає вимогам функціональності, продуктивності та зручності. Подальший розвиток системи шляхом усунення виявлених обмежень і впровадження нових функцій сприятиме її адаптації до сучасних викликів у сфері захисту цифрових даних.

ВИСНОВКИ

Проведене дослідження, присвячене розробці системи захисту потокового відео, є значним внеском у вирішення актуальних проблем інформаційної безпеки в умовах стрімкого зростання обсягів мультимедійного контенту та ускладнення кіберзагроз. Реалізована клієнт-серверна програма, заснована на використанні криптографічної бібліотеки PyNaCl, забезпечує надійний захист відеоданих, поєднуючи високий рівень безпеки, оптимальну продуктивність і зручність використання. Застосування сучасних алгоритмів, таких як XSalsa20 для потокового шифрування, Poly1305 для автентифікації, Curve25519 для асиметричного шифрування та Ed25519 для цифрових підписів, дозволяє гарантувати конфіденційність, цілісність і автентичність даних, що відповідає сучасним стандартам криптографії. Гібридна схема шифрування, яка поєднує симетричні та асиметричні методи, забезпечує безпечне управління ключами, мінімізуючи ризики їх компрометації, тоді як потоковий підхід до обробки даних оптимізує продуктивність навіть на пристроях із обмеженими обчислювальними ресурсами.

Розроблена система вирізняється продуманою архітектурою, яка інтегрує клієнтську та серверну частини. Клієнтська частина, реалізована з використанням бібліотеки CustomTkinter, пропонує інтуїтивно зрозумілий графічний інтерфейс, що дозволяє користувачам із різним рівнем технічної підготовки легко виконувати операції шифрування, дешифрування та відтворення відеофайлів. Серверна частина, побудована на фреймворку Flask, ефективно обробляє криптографічні операції та підтримує роботу з великими відеофайлами завдяки буферизації даних із розміром блоку 1 МБ і потоковій передачі. Унікальна функція візуалізації криптографічних процесів, реалізована за допомогою бібліотек OpenCV і Matplotlib, підвищує прозорість системи, дозволяючи користувачам краще зрозуміти механізми шифрування, хешування та моніторингу безпеки. Ретельне логування операцій через систему DLP (Data Loss Prevention) і продумана обробка помилок забезпечують надійність системи та полегшують її діагностику, що є важливим для практичного застосування в реальних умовах.

Експериментальні результати підтвердили високу функціональність розробленої системи. Вона стабільно виконує шифрування та дешифрування відеофайлів різних форматів (MP4, AVI, MKV тощо) і розмірів, зберігаючи їх цілісність і якість відтворення. Тестування показало, що система ефективно обробляє файли розміром до 500 МБ, демонструючи достатню продуктивність завдяки асинхронному виконанню операцій із використанням багатопоточності (модуль threading). Криптографічна стійкість системи була підтверджена тестами на захист від атак грубої сили, модифікації даних і перехоплення (зокрема, атак типу "людина посередині"). Використання випадкових векторів ініціалізації (IV) і цифрових підписів на основі SHA-256 забезпечує надійний захист від атак повторного відтворення та підробки даних. Позитивні відгуки користувачів щодо зручності графічного інтерфейсу підкреслюють практичну цінність розробки, хоча впровадження інтерактивних підказок або навчальних матеріалів могло б додатково полегшити роботу для початківців.

Практична значущість системи полягає в її універсальності та широкому спектрі потенційних застосувань. Розробка може бути використана в стрімінгових платформах, таких як YouTube або Netflix, для захисту комерційного контенту від піратства, у системах відеоконференцій, таких як Zoom, для забезпечення конфіденційності переговорів, а також у системах відеоспостереження для захисту чутливих даних у банківських, медичних чи державних установах. Відкрита природа проєкту (open-source) сприяє його адаптації для специфічних потреб, таких як освітні програми, наукові дослідження в галузі криптографії або інтеграція з хмарними сховищами для безпечного зберігання відеоданих. Інтеграція системи DLP забезпечує моніторинг і аудит дій користувачів, що є важливим для відповідності регуляторним вимогам, таким як GDPR і CCPA, які зобов'язують компанії впроваджувати надійні механізми захисту даних.

Незважаючи на значні досягнення, система має певні обмеження, які не є критичними, але потребують уваги для подальшого вдосконалення. Залежність від доступності сервера може створювати незручності в умовах нестабільного мережевого з'єднання, що потребує впровадження автономного режиму роботи. Відсутність підтримки потокового відтворення зашифрованого контенту обмежує можливості використання системи для реального часу, що є важливим для стрімінгових платформ. Крім того, обробка дуже великих файлів (понад 500 МБ) може вимагати додаткової оптимізації мережевої передачі та буферизації, щоб зменшити затримки. Юридичні питання, пов'язані з обмеженнями доступу до контенту через DRM-подібні механізми, можуть викликати невдоволення користувачів, що потребує балансу між безпекою та зручністю.

Перспективи подальшого розвитку системи включають кілька ключових напрямів. По-перше, оптимізація обробки великих відеофайлів шляхом впровадження більш ефективних алгоритмів компресії та потокової передачі даних може підвищити продуктивність. По-друге, інтеграція протоколу HTTPS для серверної частини забезпечить додатковий рівень безпеки під час передачі

даних через мережу. По-третє, розробка модуля для потокового відтворення зашифрованого контенту в реальному часі розширить функціональність системи, роблячи її придатною для використання в сучасних стрімінгових сервісах. Почетверте, впровадження підтримки апаратних рішень, таких як Trusted Execution Environment (TEE), може підвищити безпеку виконання криптографічних операцій. Нарешті, створення інтерактивних навчальних матеріалів і розширення документації сприятиме популяризації системи серед широкого кола користувачів, включаючи тих, хто не має глибоких знань у криптографії.

Таким чином, розроблена система захисту потокового відео є збалансованим і перспективним рішенням, яке ефективно вирішує завдання забезпечення безпеки мультимедійних даних у сучасному цифровому середовищі. Вона поєднує передові криптографічні технології, такі як XSalsa20, Poly1305, Curve25519 і Ed25519, із практичною зручністю завдяки інтуїтивному інтерфейсу та функціям візуалізації. Система відповідає сучасним стандартам безпеки, демонструючи високу криптографічну стійкість і продуктивність, а її відкрита природа відкриває широкі можливості для адаптації та інтеграції. Подальший розвиток системи, спрямований на усунення виявлених обмежень і впровадження нових функціональних можливостей, сприятиме її адаптації до нових технологічних трендів і зростаючих вимог інформаційної безпеки, забезпечуючи значний внесок у захист мультимедійного контенту в майбутньому.

ПЕРЕЛІК ПОСИЛАНЬ

1. Why Is Data Protection Important for Online Video? URL: https://jetstream.com/why-is-data-protection-important-for-online-video/

2. Tajinder Kumar Saini, Purushottam Sharma, Jaswinder Tanwar, Hisham Alsghier. Cloud-based video streaming services: Trends, challenges, and opportunities. Wiley CAAI Transactions on Intelligence Technology. 2024

3. The Algorithm Series: Digital Rights Management (DRM) URL: https://www.streamingmedia.com/Articles/Editorial/Featured-Articles/TheAlgorithm-Series-Digital-Rights-Management-(DRM)-144341.aspx

4. What is the best encryption for video streaming? About secure video streaming URL: https://ceeblue.net/what-is-the-best-encryption-for-video-streamingabout-secure-video-streaming/

5. Шифрування: типи і алгоритми. Що це, чим відрізняються і де використовуються? URL: https://hostpro.ua/wiki/ua/security/encryption-typesalgorithms/

6. PyCryptodome Documentation. AES Encryption in CBC Mode. URL: https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html

7. Python hashlib Documentation. SHA-256 Hashing. URL: https://docs.python.org/3/library/hashlib.html

8. Шифрування: Типи та алгоритми. Що це і який тип шифрування кращий? URL: https://hostkoss.com/b/uk/encryption-types-algorithms/

9. Керування цифровими правами (DRM) URL: https://manual.calibreebook.com/uk/drm.html

10. Digital Rights Management (DRM): Comparing PlayReady, FairPlay and WideVine URL: https://copperpod.medium.com/digital-rights-management-drmcomparing-playready-fairplay-and-widevine-20afd09ef50b

11. Що таке SSL та TLS? В чому полягає різниця? URL:

https://tseivo.com/b/memecode/t/kzvqe51gmr

12. What is MPEG-DASH? | HLS vs. DASH URL: https://www.cloudflare.com/learning/video/what-is-mpeg-dash/

13. How does WebRTC work? URL: https://www.agora.io/en/blog/howdoes-webrtc-work/

14. Python hashlib Documentation. Secure Hashing for Integrity Verification. URL: https://docs.python.org/3/library/hashlib.html

15. PyNaCl 1.5.0 URL: https://pypi.org/project/PyNaCl/

16. Data Loss Prevention (DLP) Systems Overview. URL: https://www.gartner.com/en/information-technology/glossary/data-loss-preventiondlp

17. What Is a Trusted Execution Environment (TEE)? URL: https://www.halborn.com/blog/post/what-is-a-trusted-execution-environment-tee

18. What is the difference between SGX and TrustZone URL: https://community.intel.com/t5/Intel-Software-Guard-Extensions/What-is-thedifference-between-SGX-and-TrustZone/td-p/1111586

19. What Is Digital Rights Management and How Does It Work? URL: https://www.digitalguardian.com/blog/drm-encryption

20. AES 256 Encryption with PyCrypto using CBC mode - any weaknesses? URL: https://stackoverflow.com/questions/7954661/aes-256-encryption-withpycrypto-using-cbc-mode-any-weaknesses

21. AES 256 and How It Secures Our Data URL: https://medium.com/@fattahmuhammad17/aes-256-and-how-it-secures-our-dataa15be8ec047e

22. AES-256 URL: https://www.vpnunlimited.com/ua/help/cybersecurity/aes-256

23. SHA-256 URL: https://www.vpnunlimited.com/ua/help/cybersecurity/sha-256

24. National Institute of Standards and Technology. (2001). Advanced Encryption Standard (AES). FIPS PUB 197. URL:

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

25. Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120–126. URL: https://dl.acm.org/doi/10.1145/359340.359342

26. Python threading documentation. (2023). Python Standard Library. URL: https://docs.python.org/3/library/threading.html

27. Data Loss Prevention (DLP) Overview. (2022). Gartner. URL: https://www.gartner.com/en/information-technology/glossary/data-loss-preventiondlp

28. CustomTkinter Documentation URL: https://customtkinter.tomaszsztuka.com/

29. PyCryptodome Documentation URL: https://pycryptodome.readthedocs.io/

30. Повне розуміння діаграми компонентів UML за допомогою легкого методу URL: https://www.mindonmap.com/uk/blog/uml-component-diagram/

31. Що таке діаграма класів UML і найкращий творець діаграм класів UML URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/

32. Діаграма послідовності (Sequence Diagrams) URL: https://www.maxzosim.com/sequence-diagrams/

ДОДАТКИ

ДОДАТОК А. ТЕХНІЧНЕ ЗАВДАННЯ

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
"___" _____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської дипломної роботи на тему:

Розробка програми захисту цифрових відеоданих на основі симетричного алгоритму AES, хеш-функції SHA-256 та впровадження системи захисту від витоку даних DLP

08–72.БДР.003.00.070.ТЗ

Керівник бакалаврської дипломної роботи
д.ф., доцент каф. МБІС
_____ Салієва О. В.
"___" _____ 2025 р.

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний комплекс для захисту цифрових відеоданих на основі симетричного алгоритму AES-256, хеш-функції SHA-256 та системи запобігання витокам даних (DLP).

Область застосування: захист мультимедійного контенту, зокрема потокового відео, від несанкціонованого доступу, модифікації та витоків даних у локальних інформаційних системах, включаючи освітні, медичні та державні установи.

2. Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета розробки

Розробити програмний комплекс для захисту цифрових відеоданих, який забезпечує надійне шифрування, перевірку цілісності та автентичності даних за допомогою алгоритму AES-256 і хеш-функції SHA-256, а також впровадження системи DLP для моніторингу дій користувачів і запобігання витокам даних.

3.2 Призначення

Розроблений програмний комплекс призначений для захисту потокового відео шляхом локального шифрування, дешифрування та відтворення відеоконтенту, забезпечення цілісності даних, моніторингу дій користувачів і журналювання подій для підвищення безпеки інформаційних систем. Комплекс забезпечує захист від несанкціонованого доступу, модифікації даних і витоків інформації, а також підтримує зручний графічний інтерфейс для спрощення взаємодії користувачів.

4. Джерела розробки

4.1 Schneier, B. "Applied Cryptography: Protocols, Algorithms, and Source Code in C." Wiley, 2015.

4.2 Stallings, W. "Cryptography and Network Security: Principles and Practice." Pearson, 2017.

4.3 Katz, J., and Y. Lindell. "Introduction to Modern Cryptography." Chapman and Hall/CRC, 2020.

4.5. Dworkin, M. "Recommendation for Block Cipher Modes of Operation: Methods and Techniques." NIST Special Publication 800-38A, 2001.

4.6. Rescorla, E. "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, 2018.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик

5.1.1 Програмний засіб повинен мати інтуїтивно зрозумілий графічний інтерфейс користувача, побудований на основі бібліотеки CustomTkinter, для зручного завантаження, шифрування, дешифрування, відтворення відео та управління ключами.

5.1.2 Програма повинна використовувати бібліотеку PyCryptodome для реалізації симетричного шифрування AES-256 у режимі CBC та хеш-функції SHA-256 для перевірки цілісності даних.

5.1.3 Програмний засіб повинен забезпечувати локальну обробку відеофайлів без залежності від мережевих сервісів.

5.1.4 Система DLP повинна здійснювати моніторинг дій користувача (доступ до файлів, шифрування, дешифрування, генерація ключів) та зберігати журнали у форматі JSON.

5.1.5 Програма повинна підтримувати візуалізацію криптографічних процесів (AES-256, SHA-256, DLP) за допомогою бібліотек NetworkX та

Matplotlib.

5.2 Вимоги до надійності

5.2.1 Програмний засіб повинен працювати без помилок, з виведенням відповідних повідомлень у разі виникнення критичних ситуацій (наприклад, невідповідність ключа, пошкодження файлу).

5.2.2 Програма повинна забезпечувати коректне виконання функцій шифрування, дешифрування, перевірки цілісності та моніторингу дій користувача.

5.2.3 Система повинна автоматично видаляти тимчасові файли після завершення операцій для запобігання витокам даних.

5.3 Вимоги до складу і параметрів технічних засобів

− процесор: Intel Core i3–5200U 2.40 GHz або еквівалентний;

− оперативна пам’ять: не менше 4 ГБ;

− середовище функціонування: операційна система сімейства Windows або Linux;

− вимоги до техніки безпеки при роботі з програмою повинні відповідати стандартам безпеки при використанні комп’ютерної техніки.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для користувачів, що описує процеси завантаження відео, генерації ключів, шифрування, дешифрування, відтворення відео, управління ключами та перегляду журналів DLP, наведена у пункті 3.2.

6.2 Документація повинна містити опис архітектури системи, користувацьких сценаріїв, криптографічних механізмів та результатів тестування.

7. Вимоги до технічного захисту інформації

7.1 Програмний засіб повинен забезпечити захист від несанкціонованого доступу до відеоданих шляхом шифрування за допомогою AES-256 та перевірки цілісності через SHA-256.

7.2 Система повинна гарантувати безпечне зберігання ключів у локальній базі даних (JSON) з унікальними ідентифікаторами (UUID).

7.3 DLP-система повинна вести журнал дій користувача для виявлення потенційних загроз безпеки та забезпечувати можливість експорту журналів у форматах JSON або CSV.

8. Техніко-економічні показники

8.1 Цінність результатів використання програмного комплексу (захист відеоданих, запобігання витокам інформації) повинна перевищувати витрати на його розробку та впровадження.

8.2 Програма повинна бути доступною для використання широким ом, включаючи організації з обмеженими ресурсами, завдяки локальній ектурі та відсутності залежності від пропрієтарного програмного ечення.

9. Стадії та етапи розробки

Назва та зміст етапу	Термін виконання		Примітка
	початок	закінчення	
Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Виконано
Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Виконано
Розробка алгоритму роботи	14.04.2025	27.04.2025	Виконано
Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Виконано
Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Виконано
Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	02.06.2025	Виконано
Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Виконано

10. Порядок контролю та прийому

10.1 До приймання бакалаврської дипломної роботи надається:

- програмний засіб до бакалаврської дипломної роботи;
- демонстрація результатів роботи програми (шифрування, ифрування, відтворення відео, моніторинг DLP);
- презентація;
- відзив керівника роботи;
- відзив рецензента.

ічне завдання до виконання прийняв _____ Карнарук О.В.

ДОДАТОК Б. ЛІСТИНГ КОДУ

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import customtkinter as ctk import cv2 import os
import threading import time import json import
uuid import hashlib import numpy as np from
datetime import datetime from Crypto.Cipher
import AES from Crypto.Random import
get_random_bytes from Crypto.Util.Padding
import pad, unpad from PIL import Image,
ImageTk import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg import networkx as nx import io import base64
# Налаштування стилю CustomTkinter
ctk.set_appearance_mode("dark")  # Режими: "dark", "light", "system"
ctk.set_default_color_theme("blue")  # Теми: "blue", "green", "dark-blue"
class VideoEncryptorApp(ctk.CTk):    def __init__(self):        super().__init__()
# Конфігурація вікна
    self.title("Система захисту цифрових відеоданих")        self.geometry("1280x720")
    # Змінні        self.video_path
= ""        self.encrypted_path = ""
self.key = None        self.key_id =
None        self.video_player =
None        self.is_playing = False
self.keys_db = {}        self.cap =
None        self.after_id = None
    self.dlp_log = []  # Ініціалізація журналу DLP
    # Завантаження бази даних ключів
self.load_keys_db()        self.load_dlp_log()
# Створення інтерфейсу        self.create_gui()
def create_gui(self):        # Основна
табуляція        self.tabview =
ctk.CTkTabview(self)
    self.tabview.pack(fill="both", expand=True, padx=10, pady=10)
    # Створюємо вкладки
self.tabview.add("Головна")
self.tabview.add("Візуалізація")
self.tabview.add("Моніторинг DLP")        #
Основний фрейм на головній вкладці
    main_frame = ctk.CTkFrame(self.tabview.tab("Головна"))        main_frame.pack(fill="both",
expand=True, padx=10, pady=10)
    # Фрейм для відео
    self.video_frame = ctk.CTkFrame(main_frame, width=800, height=450)
self.video_frame.pack(side="top", padx=10, pady=10)
```

```
            self.video_frame.pack_propagate(False)
            # Мітка для відео (заповнювач)        self.video_label =
ctk.CTkLabel(self.video_frame, text="")
self.video_label.pack(fill="both", expand=True)
        # Фрейм для кнопок відтворення
playback_frame = ctk.CTkFrame(main_frame)
playback_frame.pack(fill="x", padx=10, pady=5)
        # Кнопки відтворення
        self.play_button = ctk.CTkButton(playback_frame, text="▶", width=40,
command=self.toggle_play)
        self.play_button.pack(side="left", padx=5)
        self.stop_button = ctk.CTkButton(playback_frame, text="■", width=40,
command=self.stop_video)
        self.stop_button.pack(side="left", padx=5)
        # Фрейм для керування файлами        file_frame
= ctk.CTkFrame(main_frame)
file_frame.pack(fill="x", padx=10, pady=5)        #
Кнопки для вибору та шифрування файлів
        self.select_button = ctk.CTkButton(file_frame, text="Вибрати відео",
command=self.select_video)
        self.select_button.pack(side="left", padx=5, pady=5)
        self.encrypt_button = ctk.CTkButton(file_frame, text="Зашифрувати",
command=self.encrypt_video)
        self.encrypt_button.pack(side="left", padx=5, pady=5)
        self.decrypt_button = ctk.CTkButton(file_frame, text="Розшифрувати",
command=self.decrypt_video)
        self.decrypt_button.pack(side="left", padx=5, pady=5)
        self.generate_key_button = ctk.CTkButton(file_frame, text="Згенерувати ключ",
command=self.generate_key)
        self.generate_key_button.pack(side="left", padx=5, pady=5)
        self.key_manager_button = ctk.CTkButton(file_frame, text="Менеджер ключів",
command=self.show_key_manager)
        self.key_manager_button.pack(side="left", padx=5, pady=5)
        self.visualization_button = ctk.CTkButton(file_frame, text="Візуалізація процесів",
command=lambda: self.tabview.set("Візуалізація"))
        self.visualization_button.pack(side="left", padx=5, pady=5)
        # Мітка шляху до файлу
        self.file_label = ctk.CTkLabel(main_frame, text="Файл не вибрано")
self.file_label.pack(anchor="w", padx=10, pady=5)
        # Мітка статусу ключа
        self.key_label = ctk.CTkLabel(main_frame, text="Ключ не згенеровано")
self.key_label.pack(anchor="w", padx=10, pady=5)
```

```
    # Прогрес-бар
    self.progress_frame = ctk.CTkFrame(main_frame)        self.progress_frame.pack(fill="x",
padx=10, pady=5)
    self.progress_label = ctk.CTkLabel(self.progress_frame, text="Прогрес:")
self.progress_label.pack(side="left", padx=5)
    self.progress_bar = ctk.CTkProgressBar(self.progress_frame)
self.progress_bar.pack(side="left", fill="x", expand=True, padx=5)
    self.progress_bar.set(0)        #
Інформаційна панель
    self.info_text = ctk.CTkTextbox(main_frame, height=100)        self.info_text.pack(fill="x",
padx=10, pady=10)
    self.info_text.insert("1.0", "Вітаємо в системі захисту цифрових відеоданих!\n")
self.info_text.configure(state="disabled")        # Налаштування вкладок візуалізації та
моніторингу        self.create_visualizations_tab()        self.setup_dlp_monitoring_tab()
self.refresh_dlp_monitoring()    def create_visualizations_tab(self):
    # Вкладка візуалізації
    vis_frame = ctk.CTkFrame(self.tabview.tab("Візуалізація"))        vis_frame.pack(fill="both",
expand=True, padx=10, pady=10)
        # Створюємо підвкладки для різних візуалізацій
        vis_tabview = ctk.CTkTabview(vis_frame)
vis_tabview.pack(fill="both", expand=True)
vis_tabview.add("AES-256")
vis_tabview.add("SHA-256")
vis_tabview.add("DLP")        # Візуалізація AES-256
    aes_frame = ctk.CTkFrame(vis_tabview.tab("AES-256"))        aes_frame.pack(fill="both",
expand=True, padx=10, pady=10)
    aes_title = ctk.CTkLabel(aes_frame, text="Візуалізація AES-256", font=("Arial", 16,
"bold"))
    aes_title.pack(pady=10)
    # Кнопка для запуску візуалізації AES        aes_buttons_frame
= ctk.CTkFrame(aes_frame)        aes_buttons_frame.pack(fill="x",
pady=10)
    aes_encrypt_btn = ctk.CTkButton(aes_buttons_frame, text="Візуалізувати шифрування",
command=lambda: self.visualize_aes(True))        aes_encrypt_btn.pack(side="left", padx=5)
    aes_decrypt_btn = ctk.CTkButton(aes_buttons_frame, text="Візуалізувати дешифрування",
                        command=lambda: self.visualize_aes(False))
aes_decrypt_btn.pack(side="left", padx=5)        # Фрейм для відображення
візуалізації AES        self.aes_vis_frame = ctk.CTkFrame(aes_frame)
    self.aes_vis_frame.pack(fill="both", expand=True, padx=10, pady=10)
    # Візуалізація SHA-256
    sha_frame = ctk.CTkFrame(vis_tabview.tab("SHA-256"))        sha_frame.pack(fill="both",
expand=True, padx=10, pady=10)
    sha_title = ctk.CTkLabel(sha_frame, text="Візуалізація SHA-256", font=("Arial", 16,
```

```
"bold"))
        sha_title.pack(pady=10)
        # Кнопка для запуску візуалізації SHA-256
        sha_vis_btn = ctk.CTkButton(sha_frame, text="Візуалізувати хешування",
command=self.visualize_sha256)        sha_vis_btn.pack(pady=10)
        # Поле для введення тексту для хешування
        self.sha_input_text = ctk.CTkTextbox(sha_frame, height=50)
self.sha_input_text.pack(fill="x", padx=10, pady=10)
        self.sha_input_text.insert("1.0", "Введіть текст для хешування")
        # Фрейм для відображення візуалізації SHA-256
self.sha_vis_frame = ctk.CTkFrame(sha_frame)
        self.sha_vis_frame.pack(fill="both", expand=True, padx=10, pady=10)
        # Візуалізація DLP
        dlp_frame = ctk.CTkFrame(vis_tabview.tab("DLP"))        dlp_frame.pack(fill="both",
expand=True, padx=10, pady=10)
        dlp_title = ctk.CTkLabel(dlp_frame, text="Візуалізація DLP системи", font=("Arial",
16, "bold"))
        dlp_title.pack(pady=10)
        # Кнопка для запуску візуалізації DLP
        dlp_vis_btn = ctk.CTkButton(dlp_frame, text="Візуалізувати DLP систему",
command=self.visualize_dlp)        dlp_vis_btn.pack(pady=10)
        # Фрейм для відображення візуалізації DLP        self.dlp_vis_frame
= ctk.CTkFrame(dlp_frame)
        self.dlp_vis_frame.pack(fill="both", expand=True, padx=10, pady=10)     def
setup_dlp_monitoring_tab(self):        # Вкладка моніторингу DLP
        dlp_frame = ctk.CTkFrame(self.tabview.tab("Моніторинг DLP"))
dlp_frame.pack(fill="both", expand=True, padx=10, pady=10)
        # Заголовок
        dlp_title = ctk.CTkLabel(dlp_frame, text="Моніторинг системи DLP", font=("Arial",
16, "bold"))
        dlp_title.pack(pady=10)        #
Фрейм для фільтрів
        filter_frame = ctk.CTkFrame(dlp_frame)        filter_frame.pack(fill="x",
padx=10, pady=10) # Фільтр за типом дії
            ctk.CTkLabel(filter_frame, text="Фільтрувати за типом дії:").pack(side="left", padx=5)
        self.action_filter = ctk.CTkComboBox(filter_frame, values=["Всі", "access",
"playback", "encrypt", "decrypt", "key_gen"])        self.action_filter.pack(side="left",
padx=5)        self.action_filter.set("Всі")
        # Кнопка оновлення
        refresh_btn = ctk.CTkButton(filter_frame, text="Оновити",
command=self.refresh_dlp_monitoring)
        refresh_btn.pack(side="right", padx=5)
```

```
    # Кнопка експорту
    export_btn = ctk.CTkButton(filter_frame, text="Експортувати",
command=self.export_dlp_logs)
    export_btn.pack(side="right", padx=5)
    # Таблиця для журналу DLP
    self.dlp_table_frame = ctk.CTkScrollableFrame(dlp_frame)
    self.dlp_table_frame.pack(fill="both", expand=True, padx=10, pady=10)
    # Заголовки
    headers = ["Час", "Користувач", "IP", "Дія", "Опис"]
header_frame = ctk.CTkFrame(self.dlp_table_frame)
header_frame.pack(fill="x", pady=5)       for i, header in
enumerate(headers):
        label = ctk.CTkLabel(header_frame, text=header, font=("Arial", 12, "bold"))
label.grid(row=0, column=i, padx=5, sticky="w")
    # Створюємо графік активності
    self.dlp_graph_frame = ctk.CTkFrame(dlp_frame, height=200)
self.dlp_graph_frame.pack(fill="x", padx=10, pady=10)    def
refresh_dlp_monitoring(self):       # Очищаємо таблицю (крім
заголовків)       for widget in self.dlp_table_frame.winfo_children()[1:]:
widget.destroy()
    # Якщо журналу немає або він порожній       if not hasattr(self, 'dlp_log') or not
self.dlp_log:          ctk.CTkLabel(self.dlp_table_frame, text="Журнал DLP
порожній").pack(pady=20)          return
    # Фільтруємо записи за типом дії
action_filter = self.action_filter.get()
filtered_logs = self.dlp_log       if action_filter !=
"Всі":
        filtered_logs = [entry for entry in self.dlp_log if entry["action"] == action_filter]
    # Якщо немає записів після фільтрації       if not filtered_logs:
ctk.CTkLabel(self.dlp_table_frame, text=f"Немає записів для фільтру:
{action_filter}").pack(pady=20)
return
    # Відображаємо записи       for i, entry in
enumerate(reversed(filtered_logs), 1):
        row_frame = ctk.CTkFrame(self.dlp_table_frame)          row_frame.pack(fill="x",
pady=2)
        ctk.CTkLabel(row_frame, text=entry["timestamp"]).grid(row=0, column=0, padx=5,
sticky="w")
        ctk.CTkLabel(row_frame, text=entry["user"]).grid(row=0, column=1, padx=5, sticky="w")
        ctk.CTkLabel(row_frame, text=entry["ip"]).grid(row=0, column=2, padx=5, sticky="w")
        ctk.CTkLabel(row_frame, text=entry["action"]).grid(row=0, column=3, padx=5, sticky="w")
        ctk.CTkLabel(row_frame, text=entry["description"]).grid(row=0, column=4, padx=5,
sticky="w")
```

```
        # Обмежуємо кількість відображуваних записів до 100
if i >= 100:                break
            # Оновлюємо графік активності
            self.update_dlp_activity_graph()    def
update_dlp_activity_graph(self):        # Очищаємо фрейм
графіка        for widget in self.dlp_graph_frame.winfo_children():
widget.destroy()
    # Якщо журнал порожній, виходимо        if not self.dlp_log:
ctk.CTkLabel(self.dlp_graph_frame, text="Немає даних для відображення").pack()          return
    # Підготовка даних для графіка
action_counts = {}        for entry in
self.dlp_log:           action =
entry["action"]
        action_counts[action] = action_counts.get(action, 0) + 1
    # Створюємо графік
    fig, ax = plt.subplots(figsize=(8, 4))        actions =
list(action_counts.keys())        counts =
list(action_counts.values())
    # Кольори для різних дій
    colors = ['#3498db', '#2ecc71', '#e74c3c', '#f39c12', '#9b59b6']
    # Створюємо горизонтальний бар-графік
    bars = ax.barh(actions, counts, color=colors[:len(actions)])
    # Додаємо підписи до стовпців        for
i, v in enumerate(counts):
        ax.text(v + 0.1, i, str(v), color='black', va='center')
ax.set_title('Кількість операцій за типом')        ax.set_xlabel('Кількість
операцій')        # Встановлюємо темний фон для графіка
fig.patch.set_facecolor('#2b2b2b')        ax.set_facecolor('#2b2b2b')
ax.spines['bottom'].set_color('#999999')
ax.spines['top'].set_color('#999999')
ax.spines['left'].set_color('#999999')
ax.spines['right'].set_color('#999999')
ax.tick_params(colors='#999999')        ax.xaxis.label.set_color('#999999')
ax.yaxis.label.set_color('#999999')        ax.title.set_color('#999999')
# Відображаємо графік в Tkinter
    canvas = FigureCanvasTkAgg(fig, master=self.dlp_graph_frame)        canvas.draw()
    canvas.get_tk_widget().pack(fill="both", expand=True)    def
export_dlp_logs(self):        if not self.dlp_log:
        messagebox.showinfo("Інформація", "Журнал DLP порожній, немає даних для
експорту")          return
    file_path = filedialog.asksaveasfilename(          title="Експортувати
журнал DLP",          defaultextension=".json",
        filetypes=[("JSON файли", "*.json"), ("CSV файли", "*.csv")]
```

```
    )        if file_path:            try:                if
file_path.endswith('.json'):
                with open(file_path, 'w', encoding='utf-8') as f:
                    json.dump(self.dlp_log, f, ensure_ascii=False, indent=4)                elif
file_path.endswith('.csv'):
                import csv                    with open(file_path, 'w', newline='', encoding='utf-8') as f:
writer = csv.DictWriter(f, fieldnames=["timestamp", "user", "ip",
"action", "description"])
                    writer.writeheader()                    writer.writerows(self.dlp_log)
            self.log_info(f"Експортовано журнал у {os.path.basename(file_path)}")            except
Exception as e:                messagebox.showerror("Помилка", f"Помилка експорту журналу:
{str(e)}")    def visualize_aes(self, encrypt=True):
    # Очищаємо фрейм        for widget in
self.aes_vis_frame.winfo_children():            widget.destroy()
    mode = "шифрування" if encrypt else "дешифрування"
    # Створюємо візуалізацію AES-256        fig, ax
= plt.subplots(figsize=(10, 6))
fig.patch.set_facecolor('#2b2b2b')
ax.set_facecolor('#2b2b2b')        # Дані для
візуалізації AES
    stages = ['Дані', 'AddRoundKey', 'SubBytes', 'ShiftRows', 'MixColumns', 'Раунди...',
'Шифротекст']
    # Створюємо візуальну схему AES
    G = nx.DiGraph()
    # Додаємо вершини (етапи шифрування)
for i, stage in enumerate(stages):
        G.add_node(i, label=stage)        #
Додаємо ребра        for i in
range(len(stages) - 1):
        G.add_edge(i, i+1)
    # Додаємо зв'язок з ключем
G.add_node('key', label='Ключ')        for i in
range(len(stages) - 1):
        if i % 3 == 0:  # Додаємо зв'язок з ключем на певних етапах
            G.add_edge('key', i)        #
Налаштування для візуалізації        pos =
{}
    # Розміщуємо вершини        for i, stage
in enumerate(stages):            pos[i] = (i, 0)
    pos['key'] = (len(stages) // 2, 1.5)  # Розміщуємо ключ зверху
    # Кольори для вершин і ребер
    node_color = ['#3498db' if 'Дані' in G.nodes[node]['label'] else
            '#e74c3c' if 'Шифротекст' in G.nodes[node]['label'] or 'Ключ' in
```

```
G.nodes[node]['label'] else
                '#2ecc71' for node in G.nodes()]
    edge_color = ['#f39c12' if 'key' in edge else '#95a5a6' for edge in G.edges()]
    # Напрямок стрілок (зворотний для дешифрування)
if not encrypt:          G = G.reverse()          stages =
stages[::-1]
        stages[0], stages[-1] = 'Шифротекст', 'Дані'          for i,
stage in enumerate(stages):
            G.nodes[len(stages) - i - 1]['label'] = stage       # Малюємо
граф
    nx.draw(G, pos, with_labels=False, node_color=node_color, edge_color=edge_color,
node_size=1200, arrowsize=20, width=2, ax=ax)
    # Додаємо підписи вершин з поліпшеним розміщенням та кольором
labels = {node: G.nodes[node]['label'] for node in G.nodes()}
    # Зміщуємо позиції міток вгору для кращої видимості
    label_pos = {node: (pos[node][0], pos[node][1] + 0.15) for node in pos}  # Змінено з -0.1 на
+0.15
    # Використовуємо білий колір для всіх міток
    nx.draw_networkx_labels(G, label_pos, labels, font_color='green', font_size=11,
font_weight='bold', ax=ax)       # Додаємо заголовок
    ax.set_title(f'Візуалізація процесу {mode} AES-256', color='white', fontsize=14)
    # Відображаємо графік в Tkinter
    canvas = FigureCanvasTkAgg(fig, master=self.aes_vis_frame)       canvas.draw()
    canvas.get_tk_widget().pack(fill="both", expand=True)
        # Додаємо пояснення
        explanation_text = ctk.CTkTextbox(self.aes_vis_frame, height=120)
        explanation_text.pack(fill="x", padx=10, pady=10) if encrypt:
      explanation = """Алгоритм AES-256:
1. Попереднє додавання ключа (AddRoundKey) - XOR вхідних даних з початковим
ключем 2. 14 раундів шифрування, кожен з яких складається з:
- SubBytes: заміна байтів за допомогою таблиці підстановки
- ShiftRows: циклічний зсув рядків матриці стану
- MixColumns: змішування стовпців матриці стану (крім останнього раунду)
- AddRoundKey: XOR із підключем поточного раунду 3. Результат - зашифровані дані
  (шифротекст)"""       else:          explanation = """Дешифрування AES-256:
1. Попереднє додавання ключа (AddRoundKey) - XOR зашифрованих даних з останнім
підключем 2. 14 раундів дешифрування, кожен з яких складається з:
- InvShiftRows: зворотній циклічний зсув рядків
- InvSubBytes: зворотня заміна байтів
- AddRoundKey: XOR із підключем поточного раунду
- InvMixColumns: зворотнє змішування стовпців (крім першого раунду)
```

```
3. Результат - розшифровані дані (відкритий текст)"""
explanation_text.insert("1.0", explanation)
explanation_text.configure(state="disabled")    def
visualize_sha256(self):        # Очищаємо фрейм        for
widget in self.sha_vis_frame.winfo_children():
            widget.destroy()
        # Отримуємо текст для хешування
        input_text = self.sha_input_text.get("1.0", "end-1c")        if not input_text
or input_text == "Введіть текст для хешування":            input_text = "Hello,
world!"        # Обчислюємо хеш SHA-256
        hash_value = hashlib.sha256(input_text.encode()).hexdigest()
        # Створюємо візуалізацію SHA-256        fig, ax
= plt.subplots(figsize=(10, 6))
fig.patch.set_facecolor('#2b2b2b')
ax.set_facecolor('#2b2b2b')
        # Видаляємо осі
ax.axis('off')        # Схема
SHA-256
        stages = ['Вхідні дані', 'Доповнення', 'Розбиття на блоки', 'Розширення блоків',
                'Стискаюча функція', 'Оновлення хешу', 'SHA-256 хеш']
        # Створюємо візуальну схему SHA-256
        G = nx.DiGraph()        # Додаємо
вершини        for i, stage in
enumerate(stages):
            G.add_node(i, label=stage)        #
Додаємо ребра        for i in
range(len(stages) - 1):
            G.add_edge(i, i+1)
        # Додаємо цикл для блоків
        G.add_edge(5, 2, color='red', style='dashed')        #
Налаштування для візуалізації        pos =
nx.shell_layout(G)        # Кольори для вершин
        node_color = ['#3498db' if i == 0 else
                  '#e74c3c' if i == len(stages) - 1 else
                  '#2ecc71' for i in range(len(stages))]
        # Кольори для ребер
        edge_color = ['red' if e[0] == 5 and e[1] == 2 else '#95a5a6' for e in G.edges()]
        # Стилі для ребер
        edge_style = ['dashed' if e[0] == 5 and e[1] == 2 else 'solid' for e in G.edges()]
        # Малюємо граф
        nx.draw(G, pos, with_labels=False, node_color=node_color,
              edge_color=edge_color, node_size=1200, arrowsize=20, width=2, ax=ax)
              # Додаємо підписи вершин
```

```
            labels = {node: G.nodes[node]['label'] for node in G.nodes()}
            nx.draw_networkx_labels(G, pos, labels, font_color='violet', font_size=10, ax=ax)
            # Додаємо заголовок        ax.set_title('Візуалізація алгоритму SHA-256',
color='white', fontsize=14)
        # Відображаємо графік в Tkinter
        canvas = FigureCanvasTkAgg(fig, master=self.sha_vis_frame)        canvas.draw()
        canvas.get_tk_widget().pack(fill="both", expand=True)
        # Відображаємо інформацію про хеш
hash_frame = ctk.CTkFrame(self.sha_vis_frame)
hash_frame.pack(fill="x", padx=10, pady=10)
        hash_label = ctk.CTkLabel(hash_frame, text=f"SHA-256 хеш для '{input_text}':", font=("Arial",
12, "bold"))
        hash_label.pack(anchor="w", padx=10, pady=5)
        hash_value_label = ctk.CTkLabel(hash_frame, text=hash_value, font=("monospace", 12))
hash_value_label.pack(anchor="w", padx=10, pady=5)
        # Візуалізуємо хеш як кольорову схему
        hash_canvas = ctk.CTkCanvas(hash_frame, height=50, width=600, bg="#2b2b2b",
highlightbackground="#2b2b2b")        hash_canvas.pack(pady=10)
        # Малюємо кольорові прямокутники на основі значень хеша
for i in range(0, 64, 2):
            hex_val = hash_value[i:i+2]
                        color = f"#{hex_val}{'33' if i % 4 == 0 else '66' if i % 4 == 2 else 'cc'}"
                hash_canvas.create_rectangle(i*10, 0, (i+2)*10, 50, fill=color, outline="")
        # Додаємо пояснення
        explanation_text = ctk.CTkTextbox(self.sha_vis_frame, height=120)
explanation_text.pack(fill="x", padx=10, pady=10)        explanation =
"""Алгоритм SHA-256:
1. Доповнення повідомлення: додавання біта 1, а потім нулів та довжини повідомлення
2. Розбиття на блоки по 512 біт
3. Ініціалізація буферів хешування 8 константами (перші 32 біти дробових частин кореня
   квадратного з перших 8 простих чисел) 4. Для кожного блоку:
        - Розширення 16 слів блоку до 64 слів
        - Виконання 64 раундів стискаючої функції
        - Оновлення буферів хешування
5. Конкатенація буферів хешування для отримання фінального хешу
SHA-256 забезпечує високу стійкість до колізій, підходить для цифрових підписів та
ідентифікації файлів."""
        explanation_text.insert("1.0", explanation)
explanation_text.configure(state="disabled")     def
visualize_dlp(self):        # Очищаємо фрейм        for widget in
self.dlp_vis_frame.winfo_children():
            widget.destroy()
```

```
    # Створюємо візуалізацію DLP системи        fig,
ax = plt.subplots(figsize=(10, 6))
fig.patch.set_facecolor('#2b2b2b')
ax.set_facecolor('#2b2b2b')
    # Видаляємо осі        ax.axis('off')
    # Створюємо граф DLP системи
    G = nx.DiGraph()
    # Додаємо вершини
    G.add_node("user", label="Користувач")
    G.add_node("app", label="Застосунок")
    G.add_node("monitor", label="Монітор DLP")
    G.add_node("detect", label="Виявлення витоків")
    G.add_node("log", label="Журнал активності")
    G.add_node("alert", label="Сповіщення")
    G.add_node("encrypt", label="Шифрування")
    G.add_node("decrypt", label="Дешифрування")
    G.add_node("vid_encrypted", label="Зашифроване відео")
        G.add_node("vid_decrypted", label="Відкрите відео")
        # Додаємо ребра
        G.add_edge("user", "app")
        G.add_edge("app", "encrypt")
    G.add_edge("app", "decrypt")
    G.add_edge("encrypt", "vid_encrypted")       G.add_edge("vid_encrypted",
"decrypt")
    G.add_edge("decrypt", "vid_decrypted")
    G.add_edge("vid_decrypted", "user")
    # Додаємо ребра моніторингу
    G.add_edge("app", "monitor")
    G.add_edge("monitor", "detect")
    G.add_edge("detect", "log")
    G.add_edge("detect", "alert")       #
Розміщення вершин       pos = {
        "user": (0, 0),
        "app": (2, 0),
        "encrypt": (4, 1),
        "decrypt": (4, -1),
        "vid_encrypted": (6, 1),
        "vid_decrypted": (6, -1),
        "monitor": (2, -2),
        "detect": (4, -2),
        "log": (6, -2),
        "alert": (6, -3)       }
```

```
        # Кольори для вершин
node_colors = {            "user":
"#3498db",
            "app": "#3498db",
            "encrypt": "#2ecc71",
            "decrypt": "#2ecc71",
            "vid_encrypted": "#e74c3c",
            "vid_decrypted": "#3498db",
            "monitor": "#9b59b6",
            "detect": "#9b59b6",
            "log": "#f39c12",
            "alert": "#e74c3c"
        }
        node_color_list = [node_colors[node] for node in G.nodes()]
        # Малюємо граф
        nx.draw(G, pos, with_labels=False, node_color=node_color_list,
node_size=1200, arrowsize=20, width=2, ax=ax)
        # Додаємо підписи вершин
        labels = {node: G.nodes[node]["label"] for node in G.nodes()}
        nx.draw_networkx_labels(G, pos, labels, font_color='red', font_size=10, ax=ax)
        # Додаємо заголовок
        ax.set_title('Візуалізація системи DLP (Data Loss Prevention)', color='white', fontsize=14)
        # Відображаємо графік в Tkinter
        canvas = FigureCanvasTkAgg(fig, master=self.dlp_vis_frame)        canvas.draw()
        canvas.get_tk_widget().pack(fill="both", expand=True)
        # Додаємо пояснення
        explanation_text = ctk.CTkTextbox(self.dlp_vis_frame, height=150)
explanation_text.pack(fill="x", padx=10, pady=10)        explanation =
"""Система DLP (Data Loss Prevention):
1. Моніторинг доступу користувачів до захищених відеоданих
2. Виявлення потенційних витоків даних на основі аналізу дій користувача
3. Журналювання всіх операцій з захищеними файлами:
            - доступ до файлів
            - шифрування/дешифрування
            - перегляд
            - генерація ключів
4. Захищені відеодані завжди зберігаються в зашифрованому вигляді
5. Ключі до захищених даних зберігаються в безпечній базі даних ключів
6. Система підтримує аудит діяльності та моніторинг подій безпеки
У цій програмі система DLP інтегрована з механізмами шифрування та контролю доступу,
забезпечуючи комплексний захист цифрових відеоданих від несанкціонованого доступу та
витоків."""
```

```
    explanation_text.insert("1.0", explanation)
explanation_text.configure(state="disabled")    def
log_info(self, message):
self.info_text.configure(state="normal")
    self.info_text.insert("end", f"{datetime.now().strftime('%H:%M:%S')}: {message}\n")
self.info_text.see("end")
    self.info_text.configure(state="disabled")    def
select_video(self):       file_path =
filedialog.askopenfilename(          title="Вибрати
відеофайл",
        filetypes=[("Відеофайли", "*.mp4 *.avi *.mkv *.mov *.flv *.wmv"), ("Зашифровані файли",
"*.enc")]       )       if file_path:
        # Якщо вибрано зашифрований файл
if file_path.endswith(".enc"):
            self.encrypted_path = file_path              self.file_label.configure(text=f"Зашифрований
файл:
{os.path.basename(file_path)}")
self.check_key_requirement()
            self.log_info(f"Вибрано зашифрований файл: {os.path.basename(file_path)}")
else:               self.video_path = file_path
            self.file_label.configure(text=f"Файл: {os.path.basename(file_path)}")
self.load_video()
            self.log_info(f"Вибрано відео: {os.path.basename(file_path)}")
        # Запис у DLP журнал
        self.add_dlp_entry("access", f"Доступ до файлу {os.path.basename(file_path)}")    def
check_key_requirement(self):
    """Перевірка необхідного ключа для зашифрованого файлу"""       try:          with
open(self.encrypted_path, 'rb') as f:              # Зчитуємо ідентифікатор ключа (перші 36
байт)              key_id = f.read(36).decode('utf-8')              if key_id in self.keys_db:
self.log_info(f"Файл зашифровано ключем: {key_id}")
self.key_label.configure(text=f"Потрібен ключ: {key_id[:8]}...")              else:
                self.log_info("Ключ для цього файлу не знайдено в базі даних")
self.key_label.configure(text="Потрібен ключ: не знайдено в базі даних")       except Exception as
e:
        self.log_info(f"Помилка при перевірці ключа: {str(e)}")    def
load_video(self):
    # Зупиняємо відтворення попереднього відео, якщо воно було
self.stop_video()
    # Відкриваємо відео за допомогою OpenCV
self.cap = cv2.VideoCapture(self.video_path)       if not
self.cap.isOpened():
        self.log_info("Помилка: не вдалося відкрити відеофайл")          return
```

```
    # Зчитуємо перший кадр для відображення
ret, frame = self.cap.read()        if ret:
        # Конвертуємо BGR в RGB
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        # Отримуємо розміри фрейму відеоплеєра
w_frame = self.video_frame.winfo_width()            h_frame =
self.video_frame.winfo_height()
        # Змінюємо розмір зображення відповідно до розміру фрейму
h, w = frame_rgb.shape[:2]            ratio = min(w_frame/w, h_frame/h)
        new_size = (int(w*ratio), int(h*ratio))
frame_resized = cv2.resize(frame_rgb, new_size)            #
Конвертуємо в PIL Image та потім у PhotoImage            img =
Image.fromarray(frame_resized)            img_tk =
ImageTk.PhotoImage(image=img)
        # Оновлюємо мітку відео
        self.video_label.configure(image=img_tk)
        self.video_label.image = img_tk  # Зберігаємо посилання
        self.log_info(f"Відео завантажено: {os.path.basename(self.video_path)}")        else:
self.log_info("Помилка: не вдалося прочитати кадр з відео")    def
update_video_frame(self):        if self.cap and self.is_playing:            ret, frame =
self.cap.read()            if ret:
            # Конвертуємо BGR в RGB
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            # Отримуємо розміри фрейму відеоплеєра
w_frame = self.video_frame.winfo_width()                h_frame =
self.video_frame.winfo_height()
            # Змінюємо розмір зображення відповідно до розміру фрейму
h, w = frame_rgb.shape[:2]                ratio = min(w_frame/w, h_frame/h)
new_size = (int(w*ratio), int(h*ratio))                frame_resized =
cv2.resize(frame_rgb, new_size)                # Конвертуємо в PIL Image та потім
у PhotoImage                img = Image.fromarray(frame_resized)                img_tk
= ImageTk.PhotoImage(image=img)
            # Оновлюємо мітку відео
            self.video_label.configure(image=img_tk)
            self.video_label.image = img_tk  # Зберігаємо посилання
            # Плануємо наступне оновлення                fps
= self.cap.get(cv2.CAP_PROP_FPS)
            delay = int(1000 / fps) if fps > 0 else 33  # 33 мс для 30 FPS за замовчуванням
            self.after_id = self.after(delay, self.update_video_frame)            else:
            # Досягнуто кінця відео                self.stop_video()
            # Перезапускаємо відео для наступного відтворення
self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0)    def toggle_play(self):
if not self.cap:
```

```
        return        if
self.is_playing:            if
self.after_id:
            self.after_cancel(self.after_id)
self.after_id = None            self.is_playing = False
self.play_button.configure(text="▶")
self.log_info("Відтворення призупинено")        else:
        self.is_playing = True
        self.play_button.configure(text="▮▮")            self.update_video_frame()
        self.log_info("Відтворення розпочато")
        # Запис у DLP журнал
        self.add_dlp_entry("playback", f"Перегляд файлу
{os.path.basename(self.video_path)}")
def stop_video(self):        if not self.cap:
        return        if
self.after_id:
        self.after_cancel(self.after_id)            self.after_id = None
self.is_playing = False        self.play_button.configure(text="▶")
# Повертаємо відео на початок        if self.cap:
self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
        # Показуємо перший кадр
ret, frame = self.cap.read()            if ret:
            # Конвертуємо BGR в RGB
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            # Отримуємо розміри фрейму відеоплеєра
w_frame = self.video_frame.winfo_width()                h_frame =
self.video_frame.winfo_height()
            # Змінюємо розмір зображення                h, w =
frame_rgb.shape[:2]                ratio = min(w_frame/w, h_frame/h)
new_size = (int(w*ratio), int(h*ratio))                frame_resized =
cv2.resize(frame_rgb, new_size)                # Конвертуємо в PIL
Image та потім у PhotoImage                img =
Image.fromarray(frame_resized)                img_tk =
ImageTk.PhotoImage(image=img)
            # Оновлюємо мітку відео
            self.video_label.configure(image=img_tk)
            self.video_label.image = img_tk  # Зберігаємо посилання
self.log_info("Відтворення зупинено")    def generate_key(self):
    # Генеруємо 256-бітний ключ (32 байти)        self.key
= get_random_bytes(32)
    self.key_id = str(uuid.uuid4())  # Унікальний ідентифікатор ключа        key_hash =
hashlib.sha256(self.key).hexdigest()
```

```
    # Додаємо ключ до бази даних        self.keys_db[self.key_id]
= {
        "key": self.key.hex(),
        "created": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "hash": key_hash
    }
    # Зберігаємо базу даних ключів
self.save_keys_db()        # Зберігаємо
ключ у файл
    key_file = filedialog.asksaveasfilename(
title="Зберегти ключ",           defaultextension=".key",
        filetypes=[("Файли ключів", "*.key")]
    )       if key_file:
        # Зберігаємо в форматі JSON для зручності           key_data
= {
            "id": self.key_id,
            "key": self.key.hex(),
            "hash": key_hash,
            "created": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }           with open(key_file, 'w') as f:
            json.dump(key_data, f, indent=4)           self.key_label.configure(text=f"Ключ ID:
{self.key_id[:8]}... | SHA-256:
{key_hash[:16]}...")
        self.log_info(f"Ключ згенеровано та збережено у {os.path.basename(key_file)}")
        # Запис у DLP журнал
        self.add_dlp_entry("key_gen", f"Створено новий ключ {self.key_id}")    def
load_key_from_file(self):        """Завантаження ключа з файлу"""         key_file =
filedialog.askopenfilename(           title="Вибрати файл ключа",
        filetypes=[("Файли ключів", "*.key")] )
    if key_file:           try:              with open(key_file, 'r') as f:
key_data = json.load(f)              self.key_id = key_data["id"]
self.key = bytes.fromhex(key_data["key"])              key_hash =
key_data["hash"]
            # Додаємо ключ до бази даних, якщо його там ще немає
if self.key_id not in self.keys_db:                  self.keys_db[self.key_id] = {
                "key": key_data["key"],
                "created": key_data.get("created", datetime.now().strftime("%Y-%m-%d
%H:%M:%S")),
                "hash": key_hash
            }
            self.save_keys_db()              self.key_label.configure(text=f"Ключ ID:
{self.key_id[:8]}... | SHA-256:
{key_hash[:16]}...")
```

```
            self.log_info(f"Ключ завантажено з {os.path.basename(key_file)}")
return True            except Exception as e:                self.log_info(f"Помилка при
завантаженні ключа: {str(e)}")
            messagebox.showerror("Помилка", f"Помилка при завантаженні ключа: {str(e)}")
return False        return False    def encrypt_video(self):        if not self.video_path:
messagebox.showerror("Помилка", "Спочатку виберіть відео для шифрування")            return
if not self.key or not self.key_id:            if not self.load_key_from_file():
messagebox.showinfo("Інформація", "Спершу згенеруйте або завантажте ключ")                return
    # Вибір шляху для збереження зашифрованого файлу
encrypted_path = filedialog.asksaveasfilename(            title="Зберегти
зашифроване відео",            defaultextension=".enc",
            filetypes=[("Зашифровані файли", "*.enc")]
    )        if encrypted_path:
            # Запустити шифрування у окремому потоці
threading.Thread(target=self._encrypt_video_thread,
args=(encrypted_path,)).start()    def _encrypt_video_thread(self,
encrypted_path):        try:
            # Ініціалізуємо прогрес-бар
self.progress_bar.set(0)            self.update_idletasks()
            # Отримуємо розмір вхідного файлу
            file_size = os.path.getsize(self.video_path)
            # Генеруємо випадковий IV            iv =
get_random_bytes(16)            # Створюємо AES
шифр у режимі CBC
            cipher = AES.new(self.key, AES.MODE_CBC, iv)            #
Відкриваємо вхідний і вихідний файли
            with open(self.video_path, 'rb') as infile, open(encrypted_path, 'wb') as outfile:
                # Записуємо ідентифікатор ключа на початку зашифрованого файлу
outfile.write(self.key_id.encode('utf-8'))                # Записуємо IV після
ідентифікатора ключа                outfile.write(iv)                # Обчислюємо хеш файлу
file_hash = hashlib.sha256()                # Зчитуємо блоки по 1 МБ
processed_size = 0
                chunk_size = 1024 * 1024  # 1 МБ
while True:                    chunk = infile.read(chunk_size)
if not chunk:                        break                    #
Оновлюємо хеш                    file_hash.update(chunk)
                    # Для останнього блоку додати доповнення
if len(chunk) % 16 != 0:                        chunk = pad(chunk,
AES.block_size)
                    # Шифруємо блок
                    encrypted_chunk = cipher.encrypt(chunk)
outfile.write(encrypted_chunk)                    # Оновлюємо
прогрес-бар                    processed_size += len(chunk)
```

```
            progress = min(processed_size / file_size, 1.0)                self.progress_bar.set(progress)
            # Оновлюємо інтерфейс щоб не заморозити його
self.update_idletasks()
            # Створюємо підпис файлу (хеш + ключ)
            signature = hashlib.sha256(file_hash.digest() + self.key).digest()
            # Записуємо хеш і підпис у файл
outfile.write(file_hash.digest())              outfile.write(signature)
        # Записуємо метадані у DLP систему
        self.add_dlp_entry("encrypt", f"Зашифровано файл
{os.path.basename(self.video_path)} у {os.path.basename(encrypted_path)} ключем
{self.key_id}")
        self.encrypted_path = encrypted_path
        self.log_info(f"Відео успішно зашифровано: {os.path.basename(encrypted_path)}")
        messagebox.showinfo("Успіх", "Відео успішно зашифровано!")
except Exception as e:          self.log_info(f"Помилка шифрування:
{str(e)}")
        messagebox.showerror("Помилка", f"Помилка шифрування: {str(e)}")    def
decrypt_video(self):
    # Вибір зашифрованого файлу, якщо його ще не вибрано
if not self.encrypted_path:          encrypted_path =
filedialog.askopenfilename(              title="Вибрати
зашифроване відео",              filetypes=[("Зашифровані
файли", "*.enc")]
        )          if encrypted_path:
            self.encrypted_path = encrypted_path
self.check_key_requirement()          else:
            return
    # Перевірка наявності ключа        if not self.key or not self.key_id:          if not
self.load_key_from_file():              messagebox.showinfo("Інформація", "Спершу
завантажте ключ для розшифрування")              return
    # Вибір шляху для розшифрованого файлу
decrypted_path = filedialog.asksaveasfilename(
title="Зберегти розшифроване відео",
defaultextension=".mp4",
        filetypes=[("MP4 відео", "*.mp4"), ("AVI відео", "*.avi"), ("MKV відео",
"*.mkv"), ("Всі файли", "*.*")]
    )       if decrypted_path:
        # Запустити дешифрування у окремому потоці
threading.Thread(target=self._decrypt_video_thread,
args=(decrypted_path,)).start()    def _decrypt_video_thread(self,
decrypted_path): try:
        # Ініціалізуємо прогрес-бар
self.progress_bar.set(0)          self.update_idletasks()
```

```
        # Отримуємо розмір зашифрованого файлу
        file_size = os.path.getsize(self.encrypted_path)
        # Відкриваємо зашифрований файл            with open(self.encrypted_path, 'rb')
as infile:              # Зчитуємо ідентифікатор ключа (перші 36 байт)              file_key_id
= infile.read(36).decode('utf-8')              # Перевіряємо, чи збігається ідентифікатор
ключа              if file_key_id != self.key_id:                  self.log_info(f"Помилка:
невідповідність ключа. Потрібен ключ:
{file_key_id}")                  messagebox.showerror("Помилка", f"Невідповідність ключа. Потрібен
ключ:
{file_key_id}")
                return
            # Зчитуємо IV (наступні 16 байт)              iv =
infile.read(16)
            # Створюємо AES шифр у режимі CBC
            cipher = AES.new(self.key, AES.MODE_CBC, iv)              #
Обчислюємо розмір файлу без метаданих та підписів
            data_size = file_size - 36 - 16 - 32 - 32  # 36 байтів key_id, 16 байтів IV,
32 байти хешу, 32 байти підпису
            # Відкриваємо файл для запису розшифрованих даних
with open(decrypted_path, 'wb') as outfile:
                # Зчитуємо блоки по 1 МБ
processed_size = 0
                chunk_size = 1024 * 1024  # 1 МБ                  # Хеш
для перевірки цілісності                  computed_hash =
hashlib.sha256()                  while processed_size < data_size:
# Визначаємо розмір наступного блоку
                    next_chunk_size = min(chunk_size, data_size - processed_size)                    chunk =
infile.read(next_chunk_size)
                    # Розшифровуємо блок
# Розшифровуємо блок
                    decrypted_chunk = cipher.decrypt(chunk)                    # Для останнього
блоку видаляємо доповнення                    if processed_size + len(chunk) >= data_size:
decrypted_chunk = unpad(decrypted_chunk, AES.block_size)
                    # Оновлюємо хеш
                    computed_hash.update(decrypted_chunk)
# Записуємо розшифровані дані
outfile.write(decrypted_chunk)                    # Оновлюємо
прогрес-бар                    processed_size += len(chunk)
                    progress = min(processed_size / data_size, 1.0)
self.progress_bar.set(progress)
                    # Оновлюємо інтерфейс
self.update_idletasks()              # Зчитуємо оригінальний хеш і
підпис              original_hash = infile.read(32)
```

```
original_signature = infile.read(32)              # Перевіряємо
цілісність даних              if computed_hash.digest() !=
original_hash:
                os.remove(decrypted_path)
                self.log_info("Помилка: хеш не збігається! Файл може бути пошкоджений.")
messagebox.showerror("Помилка", "Помилка цілісності даних. Файл може бути пошкоджений.")
return
            # Перевіряємо підпис
            expected_signature = hashlib.sha256(original_hash + self.key).digest()              if
original_signature != expected_signature:
                os.remove(decrypted_path)
                self.log_info("Помилка: підпис не збігається! Неправильний ключ або файл
пошкоджений.")
                messagebox.showerror("Помилка", "Помилка підпису. Неправильний ключ або файл
пошкоджений.")                return
        self.video_path = decrypted_path
        self.file_label.configure(text=f"Файл: {os.path.basename(decrypted_path)}")
self.load_video()
        # Записуємо метадані у DLP систему
        self.add_dlp_entry("decrypt", f"Розшифровано файл
{os.path.basename(self.encrypted_path)} у {os.path.basename(decrypted_path)} ключем
{self.key_id}")
        self.log_info(f"Відео успішно розшифровано: {os.path.basename(decrypted_path)}")
messagebox.showinfo("Успіх", "Відео успішно розшифровано!")        except Exception as e:
self.log_info(f"Помилка розшифрування: {str(e)}")
        messagebox.showerror("Помилка", f"Помилка розшифрування: {str(e)}")    def
add_dlp_entry(self, action_type, description):
    """Додати запис до DLP журналу"""
entry = {
        "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "action": action_type,
        "description": description,
        "user": os.getlogin(),
        "ip": "127.0.0.1"  # У реальній системі тут була б функція отримання IP
    }
    self.dlp_log.append(entry)
self.save_dlp_log()    def
load_dlp_log(self):
    """Завантажити журнал DLP з файлу"""        try:            if
os.path.exists("dlp_log.json"):                with open("dlp_log.json", "r",
encoding="utf-8") as f:                    self.dlp_log = json.load(f)        except
Exception as e:
```

```
        self.log_info(f"Помилка завантаження журналу DLP: {str(e)}")    def
save_dlp_log(self):
    """Зберегти журнал DLP у файл"""        try:            with
open("dlp_log.json", "w", encoding="utf-8") as f:
            json.dump(self.dlp_log, f, ensure_ascii=False, indent=4)        except
Exception as e:
        self.log_info(f"Помилка збереження журналу DLP: {str(e)}")    def
load_keys_db(self):
    """Завантажити базу даних ключів"""        try:
        if os.path.exists("keys_db.json"):              with
open("keys_db.json", "r") as f:
                self.keys_db = json.load(f)        except Exception
as e:
        self.log_info(f"Помилка завантаження бази даних ключів: {str(e)}")    def
save_keys_db(self):
    """Зберегти базу даних ключів"""        try:
with open("keys_db.json", "w") as f:
            json.dump(self.keys_db, f, indent=4)        except
Exception as e:
        self.log_info(f"Помилка збереження бази даних ключів: {str(e)}")    def
show_key_manager(self):
    """Показати вікно для управління ключами"""
key_window = ctk.CTkToplevel(self)
key_window.title("Менеджер ключів")
key_window.geometry("800x500")
    # Фрейм для списку ключів
        keys_frame = ctk.CTkScrollableFrame(key_window)
    keys_frame.pack(fill="both", expand=True, padx=10, pady=10)
    # Заголовки
    headers = ["ID ключа", "Дата створення", "SHA-256 хеш", "Дії"]        header_frame =
ctk.CTkFrame(keys_frame)        header_frame.pack(fill="x", pady=5)        for i, header in
enumerate(headers):            label = ctk.CTkLabel(header_frame, text=header, font=("Arial", 12,
"bold"))            label.grid(row=0, column=i, padx=5, sticky="w")
    # Функція для завантаження ключа
def load_selected_key(key_id):
        self.key_id = key_id
        self.key = bytes.fromhex(self.keys_db[key_id]["key"])            key_hash =
self.keys_db[key_id]["hash"]            self.key_label.configure(text=f"Ключ ID:
{key_id[:8]}... | SHA-256:
{key_hash[:16]}...")
        self.log_info(f"Ключ вибрано: {key_id[:8]}...")
        messagebox.showinfo("Інформація", "Ключ успішно встановлено для використання")
```

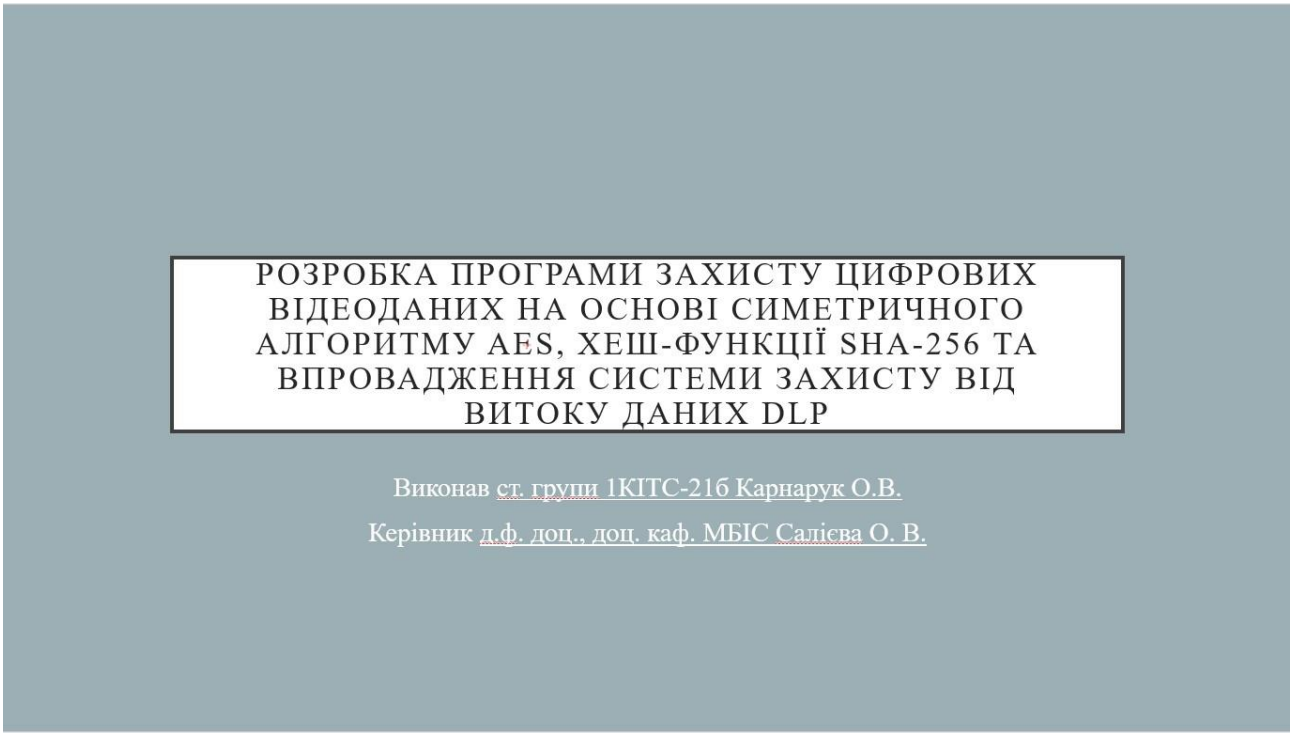
```
    # Функція для видалення ключа        def delete_key(key_id):            if
messagebox.askyesno("Підтвердження", "Ви впевнені, що хочете видалити цей ключ?"):
if self.key_id == key_id:                    self.key = None                    self.key_id = None
                self.key_label.configure(text="Ключ не згенеровано")                del
self.keys_db[key_id]                self.save_keys_db()
            self.log_info(f"Ключ видалено: {key_id[:8]}...")
            # Оновлення вікна менеджера ключів
key_window.destroy()                self.show_key_manager()
    # Відображення ключів        if not self.keys_db:            ctk.CTkLabel(keys_frame,
text="База даних ключів порожня").pack(pady=20)        else:            for i, (key_id,
key_data) in enumerate(self.keys_db.items(), 1):                row_frame =
ctk.CTkFrame(keys_frame)                row_frame.pack(fill="x", pady=2)
            id_label = ctk.CTkLabel(row_frame, text=f"{key_id[:8]}...")
id_label.grid(row=0, column=0, padx=5, sticky="w")                date_label =
ctk.CTkLabel(row_frame, text=key_data["created"])                date_label.grid(row=0,
column=1, padx=5, sticky="w")
            hash_label = ctk.CTkLabel(row_frame, text=f"{key_data['hash'][:16]}...")
hash_label.grid(row=0, column=2, padx=5, sticky="w")                action_frame =
ctk.CTkFrame(row_frame, fg_color="transparent")                action_frame.grid(row=0,
column=3, padx=5, sticky="w")                load_btn = ctk.CTkButton(action_frame,
text="Вибрати", width=80,                                    command=lambda k=key_id:
load_selected_key(k))                load_btn.pack(side="left", padx=5)
            delete_btn = ctk.CTkButton(action_frame, text="Видалити", width=80,
fg_color="#FF5555",
                                    command=lambda k=key_id: delete_key(k))
delete_btn.pack(side="left", padx=5)        # Кнопки для імпорту/експорту бази
ключів        button_frame = ctk.CTkFrame(key_window)
button_frame.pack(fill="x", padx=10, pady=10)
    # Функція імпорту ключів        def
import_keys():
        file_path = filedialog.askopenfilename(
title="Імпортувати базу ключів",                filetypes=[("JSON файли",
"*.json")]
        )            if file_path:
            try:                    with open(file_path, 'r') as f:
imported_keys = json.load(f)                    # Об'єднуємо
з існуючою базою                    for key_id, key_data in
imported_keys.items():                        if key_id not in
self.keys_db:
                        self.keys_db[key_id] = key_data                    self.save_keys_db()
                self.log_info(f"Імпортовано базу ключів з
{os.path.basename(file_path)}")
```

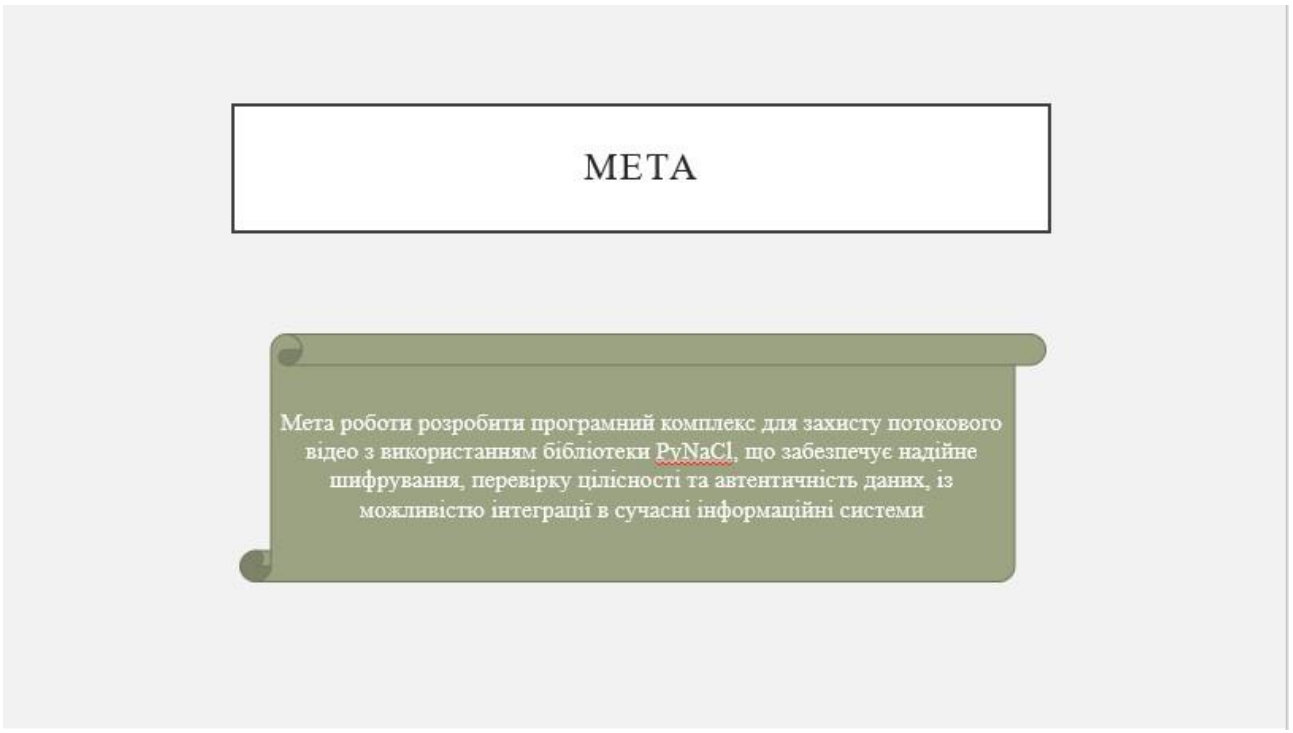
```
            # Оновлення вікна менеджера ключів                key_window.destroy()
self.show_key_manager()            except Exception as e:
messagebox.showerror("Помилка", f"Помилка імпорту ключів: {str(e)}")
    # Функція експорту ключів       def export_keys():          if not self.keys_db:
messagebox.showinfo("Інформація", "База даних ключів порожня, немає даних для
експорту")            return
        file_path = filedialog.asksaveasfilename(            title="Експортувати
базу ключів",            defaultextension=".json",
          filetypes=[("JSON файли", "*.json")]
        )          if file_path:            try:                with
open(file_path, 'w') as f:                  json.dump(self.keys_db, f,
indent=4)                self.log_info(f"Експортовано базу ключів у
{os.path.basename(file_path)}")            except Exception as e:
messagebox.showerror("Помилка", f"Помилка експорту ключів: {str(e)}")
    # Кнопки імпорту та експорту
    import_btn = ctk.CTkButton(button_frame, text="Імпортувати ключі",
command=import_keys)
    import_btn.pack(side="left", padx=5, pady=5)
    export_btn = ctk.CTkButton(button_frame, text="Експортувати ключі",
command=export_keys)
    export_btn.pack(side="left", padx=5, pady=5)       #
Кнопка для додавання нового ключа
    generate_btn = ctk.CTkButton(button_frame, text="Створити новий ключ",
command=lambda: [key_window.destroy(), self.generate_key()])
    generate_btn.pack(side="right", padx=5, pady=5)    def
show_dlp_logs(self):
    """Показати вікно з журналом DLP"""
dlp_window = ctk.CTkToplevel(self)
dlp_window.title("Журнал DLP")
dlp_window.geometry("800x600")
    # Фрейм для журналу
    log_frame = ctk.CTkScrollableFrame(dlp_window)
    log_frame.pack(fill="both", expand=True, padx=10, pady=10)
    # Заголовки
    headers = ["Час", "Користувач", "IP", "Дія", "Опис"]
header_frame = ctk.CTkFrame(log_frame)
header_frame.pack(fill="x", pady=5)       for i, header in
enumerate(headers):
        label = ctk.CTkLabel(header_frame, text=header, font=("Arial", 12, "bold"))
label.grid(row=0, column=i, padx=5, sticky="w")
    # Записи журналу       if
not self.dlp_log:
        ctk.CTkLabel(log_frame, text="Журнал DLP порожній").pack(pady=20)       else:
```

```
        for i, entry in enumerate(reversed(self.dlp_log), 1):
            row_frame = ctk.CTkFrame(log_frame)
            row_frame.pack(fill="x", pady=2)
            ctk.CTkLabel(row_frame, text=entry["timestamp"]).grid(row=0, column=0, padx=5,
sticky="w")
            ctk.CTkLabel(row_frame, text=entry["user"]).grid(row=0, column=1, padx=5, sticky="w")
            ctk.CTkLabel(row_frame, text=entry["ip"]).grid(row=0, column=2, padx=5, sticky="w")
            ctk.CTkLabel(row_frame, text=entry["action"]).grid(row=0, column=3, padx=5, sticky="w")
            ctk.CTkLabel(row_frame, text=entry["description"]).grid(row=0, column=4, padx=5,
sticky="w")
            # Обмежуємо кількість відображуваних записів до 100
if i >= 100:                break
    # Кнопки для управління журналом
button_frame = ctk.CTkFrame(dlp_window)
button_frame.pack(fill="x", padx=10, pady=10)
    # Функція експорту журналу      def export_log():          if not self.dlp_log:
messagebox.showinfo("Інформація", "Журнал DLP порожній, немає даних для експорту")
return
        file_path = filedialog.asksaveasfilename(              title="Експортувати
журнал DLP",              defaultextension=".json",
            filetypes=[("JSON файли", "*.json"), ("CSV файли", "*.csv")]
        )          if file_path:              try:                if file_path.endswith('.json'):
with open(file_path, 'w', encoding='utf-8') as f:                    json.dump(self.dlp_log, f,
ensure_ascii=False, indent=4)              elif file_path.endswith('.csv'):                    import
csv                with open(file_path, 'w', newline='', encoding='utf-8') as f:
writer = csv.DictWriter(f, fieldnames=["timestamp", "user",
"ip", "action", "description"])
                    writer.writeheader()                    writer.writerows(self.dlp_log)
            self.log_info(f"Експортовано журнал у {os.path.basename(file_path)}")
except Exception as e:              messagebox.showerror("Помилка", f"Помилка експорту
журналу: {str(e)}")
    # Функція для очищення журналу
def clear_log():
        if not self.dlp_log:
            messagebox.showinfo("Інформація", "Журнал DLP вже порожній")
return
        if messagebox.askyesno("Підтвердження", "Ви впевнені, що хочете очистити весь журнал
DLP?"):
            self.dlp_log = []            self.save_dlp_log()
            self.log_info("Журнал DLP очищено")
dlp_window.destroy()            self.show_dlp_logs()
    # Кнопка експорту
    export_btn = ctk.CTkButton(button_frame, text="Експортувати журнал",
```

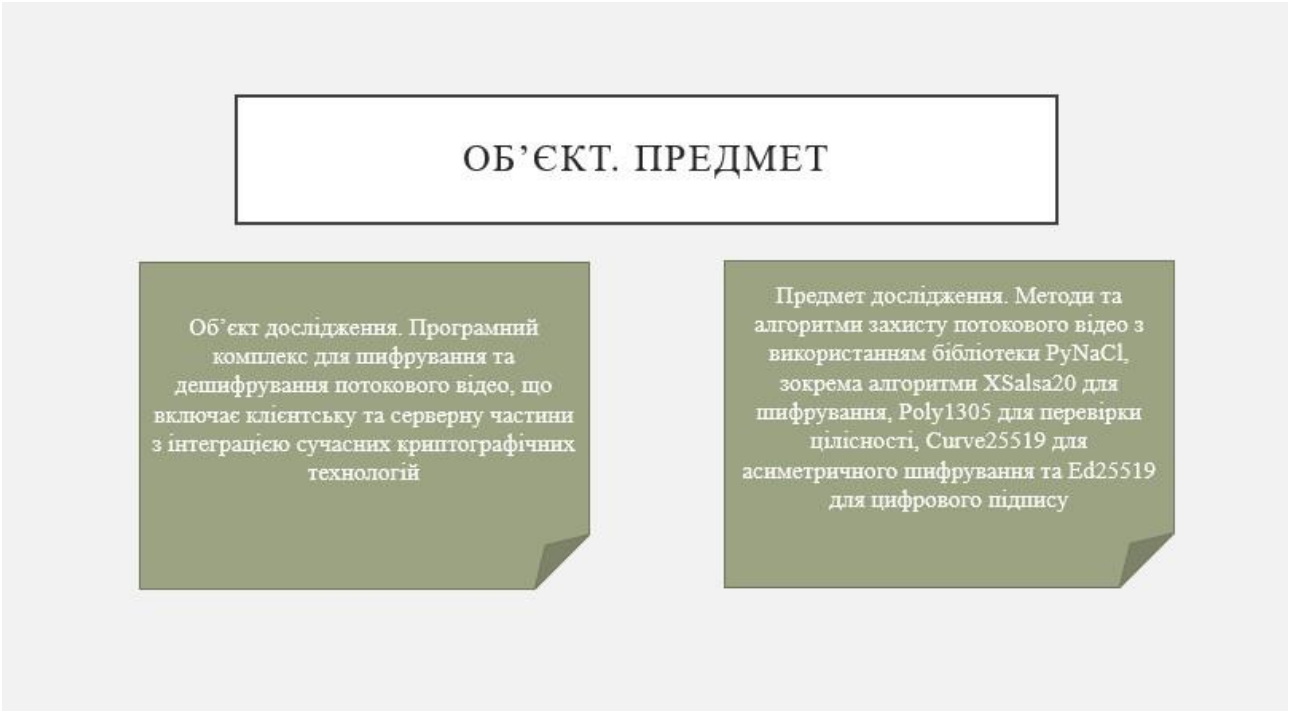
```
command=export_log)
      export_btn.pack(side="left", padx=5, pady=5)
      # Кнопка очищення
      clear_btn = ctk.CTkButton(button_frame, text="Очистити журнал", fg_color="#FF5555",
command=clear_log)
      clear_btn.pack(side="right", padx=5, pady=5) if
__name__ == "__main__":     app = VideoEncryptorApp()
92
   app.mainloop()
```

ДОДАТОК В. ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

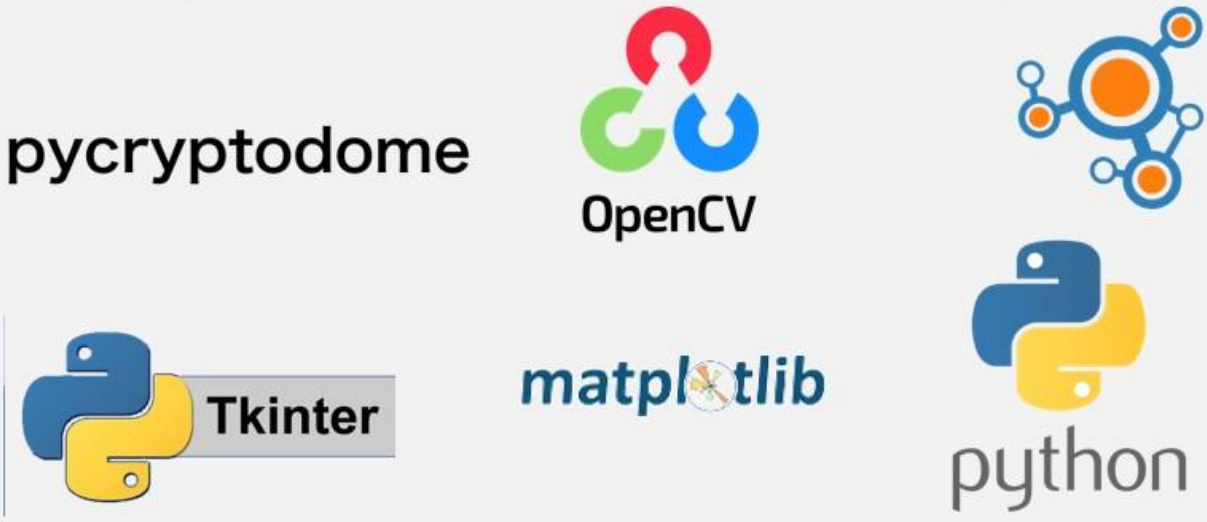





ЗАДАЧІ
проаналізувати сучасні криптографічні методи для захисту мультимедійних даних
розробити клієнт-серверну архітектуру для обробки відеофайлів із застосуванням алгоритмів XSalsa20 та Poly1305
реалізувати механізми генерації та збереження криптографічних ключів для забезпечення безпеки
створити інтуїтивний графічний інтерфейс для зручної взаємодії користувача з програмою
забезпечити створення візуалізації шифрування для підвищення прозорості процесу


ОБ'ЄКТ. ПРЕДМЕТ
Об'єкт дослідження. Програмний комплекс для шифрування та дешифрування потокового відео, що включає клієнтську та серверну частини з інтеграцією сучасних криптографічних технологій
Предмет дослідження. Методи та алгоритми захисту потокового відео з використанням бібліотеки PyNaCl, зокрема алгоритми XSalsa20 для шифрування, Poly1305 для перевірки цілісності, Curve25519 для асиметричного шифрування та Ed25519 для цифрового підпису

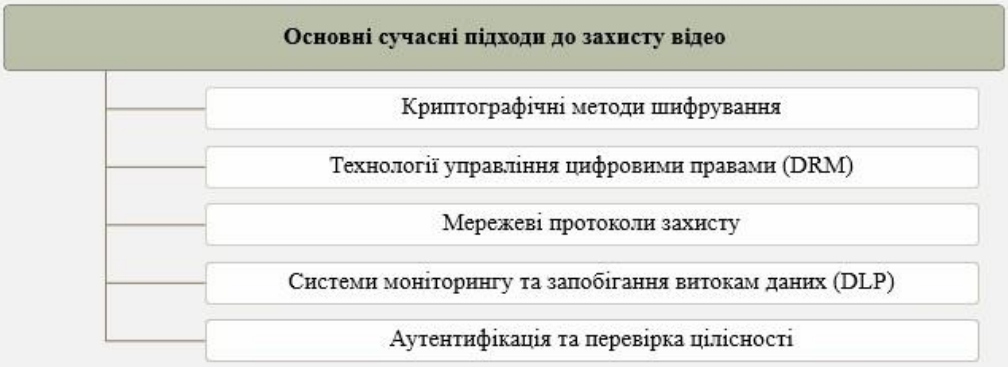
ВИКОРИСТАНІ ТЕХНОЛОГІЇ



ХІД РОБОТИ

- Аналіз сучасного стану захисту потокового відео
- Проєктування системи захисту потокового відео
- Реалізація та тестування системи захисту потокового відео

ОСНОВНІ СУЧАСНІ ПІДХОДИ ДО ЗАХИСТУ ПОТОКОВОГО ВІДЕО



UML-ДІАГРАМИ

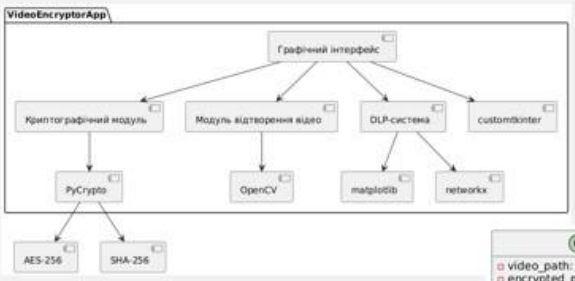


Рисунок 1 – Діаграма компонентів

Рисунок 2 – Діаграма класів

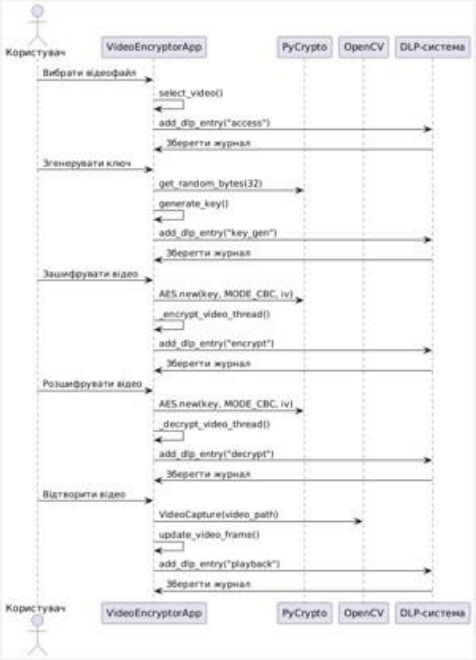


Рисунок 3 – Діаграма послідовності

ГРАФІЧНИЙ ІНТЕРФЕЙС

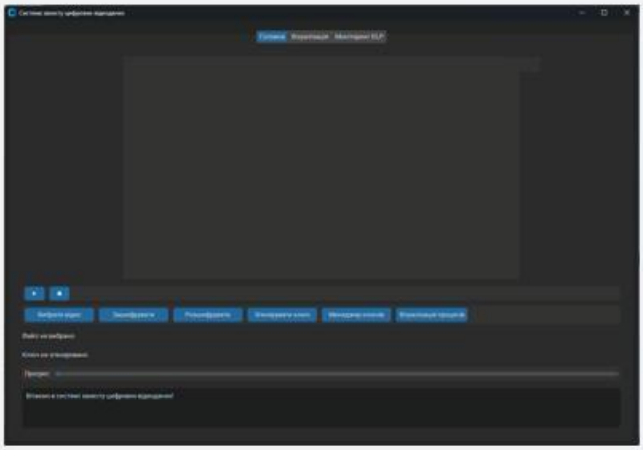


Рисунок 4 – Головне вікно

Рисунок 5 – Відеоплеєр для відображення оригінального або розшифрованого відео

Рисунок 6– Кнопки навігації та управління

Рисунок 7– Індикатор прогресу

ГРАФІЧНИЙ ІНТЕРФЕЙС

Рисунок 8 – Вкладка моніторингу DLP із таблицею журналу

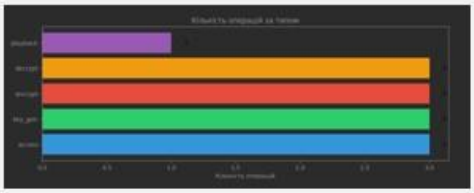


Рисунок 9 – Графік активності DLP у вигляді горизонтальної гістограми

ГРАФІЧНИЙ ІНТЕРФЕЙС

Рисунок 10 – Вкладка візуалізації з підвкладками для AES-256, SHA-256 і DLP

Рисунок 11 – Візуалізація алгоритму SHA-256 із кольоровим представленням хешу

ВИСНОВКИ

Розроблено програмний комплекс для захисту потокового відео з використанням симетричного алгоритму AES-256 у режимі CBC та хеш-функції SHA-256, що забезпечує конфіденційність, цілісність і автентичність даних.

Інтеграція системи DLP дозволяє моніторити дії користувача, такі як доступ до файлів, шифрування та дешифрування, з журналюванням у форматі JSON для аудиту та виявлення загроз

Зручний графічний інтерфейс, побудований на CustomTkinter, спрощує взаємодію користувача з функціями шифрування, відтворення відео та управління ключами

Візуалізації криптографічних процесів, реалізовані за допомогою NetworkX і Matplotlib, підвищують прозорість роботи алгоритмів AES-256, SHA-256 та DLP

Оптимізація обробки великих відеофайлів досягнута завдяки буферизації даних і багатопотоковості, що забезпечує ефективну роботу на пристроях з обмеженими ресурсами

Система забезпечує стійкість до сучасних кіберзагроз, таких як атаки повторного відтворення, завдяки використанню випадкових векторів ініціалізації та локального управління ключами

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка програми захисту цифрових відеоданих на основі симетричного алгоритму AES, хеш-функції SHA-256 та впровадження системи захисту від витоку даних DLP

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Салієва О.В._____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	99,46 %
Загальна схожість	0,54 %

Аналіз звіту подібності (відмітити потрібне)

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Карнарук О.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту**

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)