НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«РОЗРОБКА ЗАСТОСУНКУ КРИПТОГРАФІЧНОГО ЗАХИСТУ ФАЙЛІВ ДЛЯ МОБІЛЬНИХ ПЛАТФОРМ»

Виконав: здобувач 4 курсу
Рівень бакалавр
Галузь знань 12 «Інформаційні технології»,
спеціальність 125 «Кібербезпека»
Хорунжий Олексій Олександрович

Керівник: д.т.н., професор
Соколов Артем Вікторович

Рецензент: к.т.н., доцент
Бойко Віктор Дмитрович

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет **кібербезпеки та інформаційних технологій**
Кафедра **кібербезпеки**
Галузь знань: **12 Інформаційні технології**
Спеціальність: **125 Кібербезпека**
Назва освітньої програми: **Кібербезпека**

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки
професор С.А. Горбаченко
_____ «____ »_____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачу Хорунжому Олексію Олександровичу

1. Тема роботи: «Розробка застосунку криптографічного захисту файлів для мобільних платформ»
Керівник роботи: д.т.н, професор Соколов Артем Вікторович
затверджені наказом НУ ОЮА від «18» березня 2025 року №548-6
2. Строк подання здобувачем роботи «19» травня 2025 року
3. Дата видачі завдання «16» вересня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення з темою, формулювання мети та завдань дослідження	Вересень-жовтень 2024	
2	Аналіз літератури за темою кваліфікаційної роботи	Листопад 2024	
3	Написання теоретичної частини	Листопад-грудень 2024	
4	Аналіз технологій та опис середовища розробки	Січень-лютий 2025	
5	Реалізація застосунку, тестування	Лютий-березень 2025	
6	Написання висновків, оформлення списку джерел та додатків	Березень-травень 2025	
7	Попередній захист кваліфікаційної роботи	Травень 2025	
8	Захист кваліфікаційної роботи	11.06.2025	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Розробка застосунку криптографічного захисту файлів для мобільних платформ» на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 125 Кібербезпека, освітньою програмою: Кібербезпека, містить 20 рисунків, 3 таблиць, 1 додаток, 65 літературних джерел за переліком посилань. Робота виконана на 87 сторінках загального тексту і 76 сторінках основного тексту.

У роботі досліджено основні загрози інформаційній безпеці та методи їх нейтралізації шляхом використання сучасних криптографічних алгоритмів. Проведено огляд і порівняння існуючих рішень для захисту файлів на мобільних пристроях. Основною метою дослідження є створення зручного у використанні застосунку, який забезпечує локальне шифрування та дешифрування файлів із використанням алгоритму AES у поєднанні з біометричною автентифікацією користувача. Результатом роботи є функціональний мобільний застосунок, що дозволяє підвищити рівень конфіденційності персональних даних без залучення зовнішніх хмарних сервісів. Практичне значення роботи полягає у можливості використання розробленого рішення як у повсякденному житті, так і в корпоративному середовищі. Можливими напрямками подальших досліджень є розширення функціональності застосунку, зокрема додавання підтримки інших алгоритмів шифрування, інтеграція з системами резервного копіювання та удосконалення інтерфейсу користувача для підвищення зручності роботи.

КРИПТОГРАФІЯ, ШИФРУВАННЯ, ANDROID, МОБІЛЬНИЙ ЗАСТОСУНОК, AES, АВТЕНТИФІКАЦІЯ, ІНФОРМАЦІЙНА БЕЗПЕКА.

ABSTRACT

Qualification work on the topic "Development of a cryptographic file protection application for mobile platforms" for obtaining the first (bachelor's) level of higher education in the specialty 125 Cybersecurity, educational program: Cybersecurity, contains 20 figures, 3 tables, 1 appendix, 65 literary sources according to the list of references. The work is carried out on 87 pages of general text and 76 pages of main text.

The work investigates the main threats to information security and methods for their neutralization using modern cryptographic algorithms. A review and comparison of existing solutions for protecting files on mobile devices is conducted. The main goal of the research is to create a user-friendly application that provides local encryption and decryption of files using the AES algorithm in combination with biometric user authentication. The result of the work is a functional mobile application that allows you to increase the level of confidentiality of personal data without involving external cloud services. The practical significance of the work lies in the possibility of using the developed solution both in everyday life and in a corporate environment. Possible directions for further research are expanding the functionality of the application, in particular adding support for other encryption algorithms, integration with backup systems and improving the user interface to increase ease of use.

CRYPTOGRAPHY, ENCRYPTION, ANDROID, MOBILE APPLICATION, AES, AUTHENTICATION, INFORMATION SECURITY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CIA - Confidentiality, Integrity, Availability, тріада інформаційної безпеки: конфіденційність, цілісність, доступність.

APT - Advanced Persistent Threat, розширена цільова загроза, складний, тривалий і цілеспрямований кібератакувальний процес.

DDoS - Distributed Denial of Service, розподілена атака на відмову в обслуговуванні.

MITM - Man-in-the-Middle, атака типу «людина посередині»

ВСТУП

Стрімкий розвиток інформаційних технологій упродовж останніх десятиліть зумовив кардинальні зміни у способах зберігання, передавання та обробки інформації. Сьогодні персональні мобільні пристрої, зокрема смартфони, планшети та інші ґаджети, стали невід'ємною частиною повсякденного життя людини, виконуючи функції не лише засобів зв’язку, а й сховищ для особистих, фінансових, службових та інших чутливих даних. У зв’язку з цим постає критично важливе завдання - забезпечення надійного захисту цих даних від несанкціонованого доступу, модифікації або втрати [12].

Загрози інформаційній безпеці мають різноманітну природу: від втрати пристрою чи помилок користувача до цілеспрямованих атак з боку зловмисників, які застосовують соціальну інженерію, шкідливе програмне забезпечення, фішинг тощо. У таких умовах саме криптографічні методи захисту інформації, що забезпечують конфіденційність, цілісність і доступність даних, набувають особливого значення. Застосування сучасних криптографічних алгоритмів дозволяє шифрувати файли так, щоб навіть у разі фізичного доступу до пристрою стороння особа не змогла отримати доступ до захищеної інформації без належної автентифікації.

Попри наявність на ринку низки рішень для шифрування файлів, більшість із них мають низку обмежень. Часто такі програми потребують підключення до Інтернету або до хмарних сервісів, що створює додаткові вектори атаки. Крім того, багато існуючих рішень не інтегрують сучасні можливості автентифікації, зокрема біометричної, що є важливим елементом захисту в мобільному середовищі. Також варто зазначити, що деякі з існуючих застосунків є занадто складними у використанні для пересічного користувача. У зв’язку з цим постає потреба у створенні зручного, локального, безпечного та ефективного мобільного застосунку, який поєднує передові методи шифрування з біометричним захистом.

Актуальність теми полягає в необхідності забезпечення надійного локального захисту файлів на мобільних пристроях без прив’язки до зовнішніх

серверів або хмарних сховищ, із використанням перевірених криптографічних алгоритмів та сучасних методів автентифікації.

Метою дослідження є розробка мобільного застосунку на платформі Android, який забезпечує локальне шифрування та дешифрування файлів із використанням алгоритму AES та можливістю біометричної автентифікації користувача.

Для досягнення поставленої мети у роботі необхідно розв’язати такі завдання:

– провести аналіз теоретичних засад інформаційної безпеки та криптографії;

– дослідити переваги і недоліки основних криптографічних алгоритмів;

– здійснити порівняльний аналіз аналогічних програмних рішень для мобільних платформ;

– обґрунтувати вибір технологій та алгоритмів для реалізації застосунку;

– розробити архітектуру застосунку та ключові модулі;

– реалізувати шифрування файлів за допомогою алгоритму AES;

– забезпечити автентифікацію користувача за допомогою біометричних даних;

– здійснити тестування працездатності та надійності роботи застосунку;

– оцінити зручність інтерфейсу та ефективність виконання основних функцій.

Об’єктом дослідження є процес захисту інформації на мобільних пристроях в умовах зростання загроз інформаційній безпеці.

Предметом дослідження є програмні методи та алгоритми криптографічного захисту файлів, а також технічна реалізація застосунку на мобільній платформі Android.

Теоретичне значення дослідження полягає у систематизації та узагальненні знань про сучасні криптографічні алгоритми, методи захисту інформації на мобільних платформах, а також у дослідженні можливостей їх ефективного поєднання у межах прикладного програмного продукту. Робота сприяє глибшому розумінню механізмів шифрування, автентифікації користувачів та безпечного зберігання даних в умовах сучасних кіберзагроз.

Практичне значення отриманих результатів полягає у створенні застосунку, який дозволяє користувачам захищати свої файли на мобільному пристрої за допомогою сучасних криптографічних методів, не покладаючись на сторонні хмарні сервіси. Розроблений продукт може бути адаптований для використання в особистих цілях, а також в корпоративному середовищі, де важливе значення має захист конфіденційної інформації.

Структура роботи. Кваліфікаційна бакалаврська робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків. Графічний матеріал представлений у вигляді 3 таблиць, 20 рисунків. Робота викладена на 83 сторінках та містить 65 позицій у списку використаних джерел.

1 ТЕОРЕТИЧНІ ОСНОВИ КРИПТОГРАФІЇ ТА ОГЛЯД АНАЛОГІЧНИХ ЗАСТОСУНКІВ

1.1. Основи інформаційної безпеки та поняття криптографії

У сучасному інформаційному суспільстві, де значна частина діяльності організацій, підприємств і окремих користувачів пов’язана з обробкою, зберіганням і передаванням даних, питання забезпечення інформаційної безпеки набуває особливої актуальності. Основоположним підходом до формалізації вимог до безпеки інформаційних систем є так звана тріада інформаційної безпеки — конфіденційність (confidentiality), цілісність (integrity) та доступність (availability), скорочено — модель CIA. Ці три характеристики визначають загальну надійність системи захисту інформації та слугують базисом для розроблення політик, протоколів і програмних рішень у сфері кіберзахисту [35].

Конфіденційність передбачає захист інформації від несанкціонованого доступу з боку третіх осіб. Це означає, що лише ті суб’єкти, які мають відповідні права, можуть ознайомлюватися зі змістом даних. Вимога конфіденційності є ключовою в таких сферах, як фінансові послуги, охорона здоров’я, оборонна промисловість тощо. Порушення конфіденційності, зокрема витік персональних даних або комерційної таємниці, може мати катастрофічні наслідки - від репутаційних втрат до юридичної відповідальності та фінансових збитків. Основними інструментами забезпечення конфіденційності є криптографічні методи шифрування, багатофакторна автентифікація, обмеження доступу та використання ролей у системах керування доступом.

Цілісність - це властивість інформації зберігати свою коректність, повноту та відповідність первинному джерелу. Цілісність даних забезпечується тоді, коли жоден несанкціонований суб’єкт або процес не може змінити, пошкодити чи підробити інформацію без виявлення цих змін. Важливість дотримання цілісності особливо зростає в критичних інфраструктурах, системах фінансового обліку, медичних записах тощо, де навіть незначна модифікація даних може спричинити серйозні наслідки. Забезпечення цілісності реалізується шляхом застосування

хешування, цифрових підписів, алгоритмів верифікації контрольних сум, а також журналювання подій та контрольних механізмів [64].

Доступність гарантує, що дані та інформаційні ресурси будуть доступні для уповноважених користувачів у потрібний час і без надмірних затримок. Це означає, що інформаційна система повинна бути стійкою до відмов і здатною до відновлення після збоїв. Порушення доступності зазвичай є результатом як технічних несправностей (відмова обладнання, втрата електроживлення), так і навмисних дій, зокрема кібератак типу DDoS, шкідливого програмного забезпечення, саботажу. Методи забезпечення доступності включають резервне копіювання даних, розподілені системи обробки, використання хмарних платформ, застосування кластерних архітектур, а також впровадження систем безперервного моніторингу стану інформаційної інфраструктури.

Тріада конфіденційності, цілісності та доступності є ключовою методологічною основою інформаційної безпеки. Її збалансоване дотримання дозволяє забезпечити повноцінний захист інформаційних ресурсів, створити стійке середовище для обробки даних і запобігти критичним інцидентам, пов’язаним із втратами, витоками або фальсифікацією даних. У контексті мобільних застосунків, ці три принципи мають особливе значення, оскільки мобільні пристрої, як правило, є менш захищеними та більш вразливими до атак, ніж стаціонарні комп’ютерні системи. Тому створення криптографічних засобів захисту для мобільних платформ повинно враховувати всі аспекти цієї тріади [7].

Загрози інформаційній безпеці становлять потенційну або реальну небезпеку для збереження, цілісності, конфіденційності та доступності інформаційних ресурсів. У сучасному цифровому середовищі, де інформація є одним з найцінніших активів, загрози інформаційній безпеці постійно еволюціонують, набуваючи нових форм і механізмів реалізації. Їх класифікація базується на різних критеріях, зокрема за джерелом виникнення, характером впливу, рівнем організації та спрямованістю.

Залежно від джерела виникнення, загрози поділяються на внутрішні та зовнішні. Внутрішні загрози пов’язані з діями співробітників організацій або

користувачів, які мають прямий доступ до системи. Це можуть бути навмисні дії, як-от розголошення конфіденційної інформації або саботаж, а також ненавмисні помилки - наприклад, некоректне збереження даних або неусвідомлене встановлення шкідливого програмного забезпечення. Зовнішні загрози, своєю чергою, походять ззовні інформаційної системи - наприклад, з боку хакерських угруповань, конкурентів, державних або кримінальних структур [50].

За характером впливу на систему виділяють такі типи загроз, як:

1. Загрози конфіденційності, що передбачають несанкціоноване розкриття або витік інформації. До них належать атаки типу "перехоплення трафіку", соціальна інженерія, шпигунське програмне забезпечення тощо.

2. Загрози цілісності, які полягають у спотворенні або знищенні даних. Вони можуть реалізовуватися через віруси, несанкціоновану зміну файлів, зміну конфігурації систем тощо.

3. Загрози доступності, що спрямовані на порушення нормального функціонування інформаційної системи, зокрема шляхом перевантаження серверів (DDoS-атаки), фізичного пошкодження обладнання або шкідливого програмного коду, який блокує доступ до ресурсів (наприклад, програми-вимагачі) [58].

У сучасних умовах особливої уваги заслуговують цільові загрози (APT - Advanced Persistent Threats), що реалізуються висококваліфікованими зловмисниками з використанням складних і тривалих методів атаки. Їх метою є отримання доступу до стратегічно важливої інформації та довготривале перебування в системі без виявлення.

Окрему категорію становлять загрози, пов’язані з мобільними платформами, які включають ризики встановлення шкідливих додатків, перехоплення даних через ненадійні Wi-Fi мережі, викрадення пристрою, а також недосконаліть систем автентифікації. Через обмеженість ресурсів мобільних пристроїв (енергія, обчислювальна потужність), реалізація стандартних засобів захисту часто є ускладненою [20].

Також слід враховувати соціальні загрози, які проявляються через вплив на користувача з метою отримання критичної інформації - зокрема, через фішинг, обман, шахрайські дзвінки, повідомлення тощо.

Спектр загроз інформаційній безпеці є надзвичайно широким і динамічним. Для ефективного захисту інформаційних систем, зокрема тих, що функціонують на мобільних платформах, необхідно не лише виявляти потенційні загрози, але й формувати комплексну систему протидії, що враховуватиме особливості як технологічного, так і людського факторів. Особливу роль у цьому відіграють криптографічні методи, які забезпечують високий рівень захисту незалежно від типу загрози [9].

Криптографія відіграє ключову роль у забезпеченні безпеки інформаційних систем, адже вона забезпечує основні принципи захисту даних: конфіденційність, цілісність, автентичність та недоторканність інформації. В умовах швидкого розвитку технологій, зростаючої кількості кібератак та цифрових загроз, криптографічні методи стали основою захисту інформації на всіх етапах її обробки: від передачі до зберігання. Без ефективних криптографічних механізмів неможливо забезпечити належний рівень безпеки в цифровому середовищі.

Однією з основних функцій криптографії є забезпечення конфіденційності даних. Завдяки використанню алгоритмів шифрування, інформація стає недоступною для несанкціонованих осіб. Зашифровані дані, навіть якщо вони будуть перехоплені зловмисниками під час передачі або витікуть через вразливості системи, не зможуть бути зрозумілі без знання відповідного ключа шифрування. Це особливо важливо в умовах постійних кіберзагроз, таких як атаки на дані у передачі через мережі (наприклад, перехоплення трафіку, атаки "Man-in-the-Middle") [14].

Цілісність інформації гарантується через використання криптографічних хеш-функцій і підписів. Хешування дозволяє створити унікальну цифрову "відбитку" для кожного набору даних, що дозволяє виявити будь-які зміни в даних під час зберігання або передавання. Якщо дані були змінені, хеш значення змінюється, що дозволяє виявити несанкціоноване втручання.

Також важливим аспектом є автентичність та ідентифікація учасників комунікації. Для забезпечення автентичності даних використовуються цифрові підписи, які дозволяють одержувачеві впевнитися в тому, що інформація насправді надійшла від заявленого відправника і не була змінена під час передачі. Цифрові підписи, засновані на асиметричних криптографічних алгоритмах, дозволяють підтвердити як ідентичність відправника, так і сам факт здійснення певної операції, що додає додатковий рівень безпеки.

Крім того, криптографія має вирішальне значення у забезпеченні недоторканності даних, що є необхідною умовою для збереження стабільності роботи інформаційних систем. Криптографічні методи дозволяють захистити дані від зміни або підробки, забезпечуючи їх точність і достовірність на всіх етапах обробки [1].

Особливо важливою є роль криптографії в мобільних пристроях, де забезпечення безпеки даних є складним завданням через різноманітність загроз і обмеженість ресурсів. Оскільки мобільні пристрої часто використовуються для зберігання і передачі чутливої інформації, таких як персональні документи, фінансові транзакції або медичні дані, криптографія надає надійні механізми захисту для забезпечення конфіденційності та цілісності цієї інформації.

Загалом, криптографія є невід’ємною частиною сучасної інфраструктури безпеки і є необхідною умовою для захисту від різноманітних загроз, таких як несанкціонований доступ, зловмисне втручання в дані, а також забезпечення цілісності та автентичності інформаційних ресурсів. Вона дозволяє не тільки зберігати конфіденційність, але й забезпечувати довіру до інформаційних систем, що є основою для їх ефективного функціонування в умовах сучасного інформаційного суспільства.

1.2. Методи та алгоритми шифрування даних

Шифрування є основою сучасних засобів захисту інформації, особливо в умовах активного використання мобільних технологій і глобальних мереж передачі

даних. Ефективність криптографічних механізмів забезпечує базові принципи інформаційної безпеки - конфіденційність, цілісність, автентичність і доступність інформації. У межах криптографії ключову роль відіграють алгоритми шифрування, які можна класифікувати за різними критеріями, зокрема за принципом роботи з ключами [41].

Один із фундаментальних підходів до шифрування базується на використанні спільного або розділеного ключа, що формує дві основні категорії: симетричне та асиметричне шифрування. Симетричне шифрування передбачає, що для обох процесів - шифрування і розшифрування - використовується один і той самий ключ. Це означає, що як відправник, так і отримувач мають володіти однаковим секретом, що вимагає надійного каналу передачі ключа або попередньо встановленої довірчої взаємодії. Перевагою симетричних алгоритмів є їхня висока швидкодія, що обумовлює їхнє широке застосування у випадках обробки великих обсягів інформації або в обмежених за ресурсами середовищах, зокрема в мобільних пристроях. Водночас основним викликом у контексті симетричного шифрування є проблема безпечного обміну ключами між сторонами, що може створювати додаткові вектори для потенційних атак.

На противагу цьому, асиметричне шифрування використовує два різні ключі - публічний та приватний. Публічний ключ призначений для шифрування і може бути відкрито поширений серед користувачів, тоді як приватний ключ зберігається у секреті і використовується для розшифрування. Такий підхід суттєво зменшує ризики, пов’язані з передачею ключів, оскільки тільки приватний ключ є критично важливим для збереження безпеки. Основним недоліком асиметричного шифрування є його низька продуктивність, особливо при роботі з великими обсягами даних, через складність математичних операцій, які лежать в основі більшості алгоритмів цього типу. У зв’язку з цим у сучасних системах безпеки часто реалізується гібридна модель, де асиметричні алгоритми використовуються для обміну ключами, а симетричні - для подальшого шифрування самих даних [47].

Серед численних алгоритмів симетричного шифрування особливе місце посідає алгоритм AES (Advanced Encryption Standard), який на сьогоднішній день є загальновизнаним міжнародним стандартом у сфері захисту інформації. Його було затверджено Національним інститутом стандартів і технологій США (NIST) у 2001 році як результат відкритого конкурсу, що тривав із 1997 року. У цьому конкурсі брали участь найкращі криптографічні розробки провідних фахівців з усього світу, і переможцем став алгоритм Rijndael, створений бельгійськими криптографами Вінсентом Рейменом та Джоаном Дайменом. Після відповідної стандартизації Rijndael отримав нову назву - AES.

Архітектура AES побудована на основі блочного шифрування, де дані обробляються не поодиноко, а фрагментами (блоками) фіксованого розміру - 128 біт. На відміну від потокових шифрів, які оперують окремими бітами або байтами, блочні шифри працюють із великими масивами даних, що забезпечує високу продуктивність при роботі з сучасними комп’ютерними архітектурами. Алгоритм підтримує три довжини ключа - 128, 192 та 256 біт, що дозволяє налаштовувати рівень криптографічного захисту залежно від потреб користувача або безпекових політик організації [39].

Процес шифрування в AES реалізується у вигляді послідовності раундів, кожен з яких складається з кількох етапів - підстановки байтів (SubBytes), перестановки рядків (ShiftRows), змішування стовпців (MixColumns) та додавання ключа раунду (AddRoundKey). Кількість раундів залежить від довжини ключа: 10 раундів для 128-бітного ключа, 12 для 192-бітного та 14 - для 256-бітного. Кожен раунд ускладнює структуру вихідних даних таким чином, щоб відновити їх без знання ключа було практично неможливо навіть за допомогою сучасних криптоаналітичних технік.

AES вважається одним із найбільш криптостійких алгоритмів, який наразі не має ефективно реалізованих атак, здатних підірвати його безпеку при коректному використанні. Він стійкий до перебору ключів (brute-force), оскільки навіть при використанні 128-бітного ключа кількість можливих комбінацій настільки велика, що повний перебір неможливий навіть із використанням надпотужних

суперкомп’ютерів. Також AES виявив високу опірність до диференціального й лінійного криптоаналізу, які традиційно застосовуються для аналізу симетричних алгоритмів [21].

Особливістю AES є його широка підтримка на апаратному рівні, що реалізується, зокрема, в процесорах Intel та ARM через технологію AES-NI (AES New Instructions). Це дозволяє здійснювати шифрування безпосередньо на рівні процесора, без додаткового програмного навантаження, що суттєво підвищує швидкість обробки даних і зменшує споживання енергії. Такий підхід є особливо ефективним у мобільних пристроях, де оптимізація продуктивності та енергоощадність є критично важливими. Завдяки цьому AES набув особливої популярності у середовищах Android та iOS, де інтегрується у системні API для захисту даних користувача, зокрема для шифрування файлів, резервних копій, баз даних та комунікаційних каналів.

Не менш важливою є і універсальність AES, яка дозволяє використовувати його у різних режимах шифрування, таких як ECB (Electronic Codebook), CBC (Cipher Block Chaining), CTR (Counter Mode), GCM (Galois/Counter Mode) тощо. Кожен із цих режимів має свої переваги й недоліки, що дозволяє адаптувати алгоритм до конкретних задач - від простого шифрування файлів до захищеної потокової передачі даних або автентифікованого шифрування [36].

У результаті, AES став не просто технічним стандартом, а де-факто основою сучасної криптографії, яка широко використовується в державному секторі, фінансовій сфері, хмарних технологіях, комунікаційних системах, а також у побутових застосунках, таких як мобільні банківські програми, месенджери та засоби захисту особистих файлів. Його надійність, масштабованість і ефективність у поєднанні з активною підтримкою апаратної реалізації роблять AES незамінним інструментом для забезпечення інформаційної безпеки в сучасному цифровому середовищі.

Ще одним помітним прикладом симетричного блочного шифрування є алгоритм Blowfish, створений у 1993 році американським криптографом Брюсом Шнайєром. Метою його розробки було надання вільної, безкоштовної, а також

відкритої альтернативи існуючим на той час комерційним алгоритмам, зокрема DES, використання яких у програмних продуктах потребувало ліцензування. Саме ця особливість - вільне поширення та відкрите програмне втілення - зумовила стрімке поширення Blowfish у 1990-х роках, особливо у середовищі відкритого програмного забезпечення [5].

Архітектура Blowfish базується на мережі Фейстеля, що є класичною структурною схемою багатьох симетричних алгоритмів. У даному випадку кожен 64-бітний блок вхідних даних розділяється на дві рівні частини по 32 біти. Алгоритм здійснює 16 послідовних раундів, у межах яких виконується низка математичних і логічних операцій: додавання, побітове XOR, а також звернення до підстановочних таблиць (S-блоків). Після завершення раундів частини блоку об’єднуються у зашифрований вихід.

Особливістю Blowfish є динамічна генерація підстановочних таблиць на основі заданого ключа. Це означає, що для кожного окремого ключа алгоритм створює унікальні S-блоки, які використовуються у процесі шифрування. Такий підхід значно підвищує стійкість алгоритму до атак, які базуються на повторному використанні структурних елементів. Проте ініціалізація S-блоків є ресурсоємною процедурою, що робить алгоритм менш придатним для застосування в умовах реального часу або в системах, де потрібно обробляти великий потік даних без затримок [33].

Ще одним важливим параметром Blowfish є довжина ключа, яка може змінюватися в межах від 32 до 448 біт. Такий гнучкий діапазон надає можливість адаптувати рівень захисту до конкретних потреб, збалансувавши безпеку і продуктивність. У більшості практичних реалізацій використовується довжина ключа 128 або 256 біт, яка вважається достатньою для базового захисту персональних або службових даних. Водночас важливо зазначити, що Blowfish, незважаючи на підтримку довгих ключів, не має апаратної оптимізації у сучасних процесорах, що знижує його продуктивність у порівнянні з алгоритмами, такими як AES.

Блокова структура алгоритму, яка оперує 64-бітними блоками, становить одне з головних обмежень Blowfish у контексті сучасних вимог до інформаційної безпеки. При обробці великих обсягів даних зростає ймовірність виникнення колізій блоків, що потенційно може призвести до витоку інформації або зниження ефективності криптографічного захисту. Саме через це сучасні стандарти, зокрема AES, використовують блоки розміром 128 біт, які забезпечують більший простір для унікальних комбінацій і знижують ризики повторів. У той час як 64-бітна довжина була прийнятною для кінця XX століття, у XXI столітті вона вважається недостатньою в умовах масової обробки даних та складних загроз [22].

Попри вказані обмеження, алгоритм Blowfish залишається досить ефективним при використанні в специфічних умовах, зокрема у вбудованих системах, утилітах для локального шифрування файлів, а також у старих програмних продуктах, де зміна архітектури є проблематичною або недоцільною. Його порівняно висока швидкодія (особливо при багаторазовому використанні одного й того самого ключа) та відсутність ліцензійних обмежень дають змогу розробникам продовжувати його застосування у проєктах з обмеженим функціоналом.

Разом із тим, з розвитком криптографічної науки та появою нових стандартів, актуальність Blowfish поступово зменшується. Алгоритм не рекомендується до використання в нових системах, де важливим є забезпечення довготривалої криптостійкості, масштабованості та відповідності міжнародним стандартам. Зокрема, AES, який був офіційно прийнятий як стандарт урядом США та підтримується в апаратному забезпеченні багатьох процесорів, витіснив Blowfish з більшості сфер, у яких той раніше мав широке застосування. Крім того, у багатьох сучасних протоколах шифрування використання 64-бітних блоків просто не допускається через вимоги безпеки, що автоматично унеможливлює використання Blowfish [13].

Слід зазначити, що Blowfish відіграв значну роль в історії криптографії, особливо в період становлення відкритих програмних рішень та розширення доступу до засобів захисту інформації. Сьогодні він залишається доцільним для

використання лише в рамках локальних, малоресурсних систем або спадкових рішень, де модернізація є складною або невиправдано дорогою. Однак у стратегічному і перспективному контексті, а також у системах захисту даних на мобільних платформах, де ефективність, безпека та масштабованість є пріоритетними, Blowfish поступається новітнім алгоритмам, зокрема AES та ChaCha20, які демонструють вищий рівень стійкості, кращу продуктивність і більшу адаптивність до сучасного цифрового середовища.

Серед сучасних криптографічних алгоритмів, які розроблялися з урахуванням специфіки мобільного використання, обмежених обчислювальних ресурсів та новітніх загроз кібербезпеки, особливу увагу привертає ChaCha20. Цей алгоритм є одним із найперспективніших представників потокового шифрування нового покоління, який був створений на основі модифікації раніше відомого алгоритму Salsa20 американським криптографом Деніелом Дж. Бернштейном у 2008 році. ChaCha20 став відповіддю на потребу у безпечному, швидкому та простому у впровадженні алгоритмі шифрування, що не залежав би від апаратної підтримки та водночас міг би бути використаний у широкому спектрі пристроїв, включаючи мобільні платформи, вбудовані системи та IoT-пристрої [63].

Одним із основних стимулів до розробки ChaCha20 стала недостатня надійність RC4 - потокового шифру, який до того часу широко використовувався в протоколах типу SSL/TLS. Виявлені уразливості в RC4, пов’язані з неякісною генерацією псевдовипадкових потоків і можливістю часткового відновлення відкритого тексту на основі аналізу вихідного потоку, зумовили необхідність у створенні безпечної заміни, яка б відповідала високим криптографічним стандартам. У цьому контексті ChaCha20 зарекомендував себе як універсальний і надійний механізм потокового шифрування, що забезпечує як швидкодію, так і криптографічну стійкість.

ChaCha20 є синхронним потоковим шифром, який генерує потік псевдовипадкових байтів (так званий keystream), що потім виконує побітову операцію XOR з відкритим текстом для створення зашифрованого повідомлення. Його основною технічною особливістю є використання арифметичних і побітових

операцій над 32-бітними словами, включаючи додавання, операції XOR і циклічні зсуви. Ці операції є надзвичайно ефективними з точки зору обчислень і не вимагають складної апаратної реалізації. У результаті алгоритм демонструє високу продуктивність навіть на пристроях із низькою обчислювальною потужністю, таких як смартфони, планшети, недорогі одноплатні комп’ютери або мікроконтролери [17].

На відміну від блочних шифрів, які часто потребують специфічних режимів шифрування для обробки потоків даних, ChaCha20 від самого початку спроєктований як потоковий алгоритм, що дозволяє уникнути уразливостей, характерних для неправильно реалізованих режимів блочного шифрування. Це забезпечує більшу стійкість до атак, пов’язаних з неправильним управлінням IV (ініціалізаційними векторами) або повторним використанням ключів. Крім того, завдяки простій структурі і детермінованій генерації потоку даних, алгоритм виявив високу стабільність до атак по часу, які можуть використовувати часові затримки в обчисленнях для витоку інформації про ключ.

ChaCha20 застосовує 256-бітний ключ і 96-бітовий унікальний nonce (одноразовий ініціалізаційний вектор), що гарантує високий рівень випадковості та унеможливлює повторне використання одного й того самого потоку. Його архітектура дозволяє генерувати унікальний вихід навіть при мінімальних змінах у nonce або ключі, що робить його ефективним засобом захисту даних у системах, де передача одного й того самого повідомлення може повторюватися або бути перехопленою.

Ще однією вагомою перевагою ChaCha20 є його поєднання з алгоритмом автентифікації Poly1305, з яким він утворює сучасну конструкцію автентифікованого шифрування з додатковими даними (AEAD - authenticated encryption with associated data). Такий підхід забезпечує не лише конфіденційність переданих даних, а й їхню цілісність і автентичність, що є критично важливим у сучасних криптографічних протоколах. Комбінація ChaCha20-Poly1305 була офіційно включена до специфікацій протоколу TLS версії 1.3, а також активно

використовується в SSH, VPN-сервісах, мобільних операційних системах (наприклад, Android) та браузерах (зокрема, Chrome та Firefox) [16].

Однією з ключових причин зростання популярності ChaCha20 стало його ефективне функціонування в умовах відсутності апаратного прискорення шифрування, зокрема на ARM-пристроях, де алгоритм AES може виконуватися повільніше у разі відсутності підтримки інструкцій AES-NI. У таких випадках ChaCha20 не лише забезпечує порівнянну або вищу швидкість, а й дозволяє досягати вищого рівня енергоефективності, що є надзвичайно важливим для мобільних і автономних систем.

З погляду криптоаналітичної стійкості, ChaCha20 продемонстрував відсутність вразливостей до відомих типів атак, зокрема до диференціального аналізу, атаки повторів, часових атак і атак на переповнення. Незалежні дослідження, проведені протягом останніх п’ятнадцяти років, підтверджують надійність цього алгоритму в умовах реального використання. На практиці жодна з відомих атак не дозволила зменшити простір пошуку ключа або здійснити дешифрування без знання секретного ключа.

Таким чином, ChaCha20 є високопродуктивним, безпечним та енергоефективним алгоритмом шифрування, орієнтованим насамперед на мобільні та вбудовані платформи. Його конструктивна простота, універсальність і адаптивність до широкого спектра пристроїв дозволили йому стати повноцінною альтернативою AES у сучасних умовах, особливо у випадках, коли останній не має апаратної підтримки. З огляду на його надійність та активну інтеграцію у стандартизовані протоколи зв’язку, ChaCha20 продовжує зміцнювати свої позиції у сфері інформаційної безпеки як один із найбільш збалансованих інструментів криптографічного захисту нового покоління.

Асиметричне шифрування, яке відіграє ключову роль у сучасних інформаційно-комунікаційних системах, найчастіше асоціюється з алгоритмом RSA (Rivest–Shamir–Adleman). Цей криптографічний алгоритм був запропонований у 1977 році трьома вченими з Массачусетського технологічного інституту - Роном Рівестом, Аді Шаміром та Леонардом Адлеманом - і з часом

перетворився на фундаментальний механізм захисту в усіх ключових напрямах інформаційної безпеки, включаючи автентифікацію, цифровий підпис, конфіденційність і цілісність інформації.

Криптографічна стійкість RSA ґрунтується на математичній складності задачі факторизації добутку двох великих простих чисел. Основна ідея полягає в тому, що, хоча легко знайти добуток двох простих чисел, обернена операція - тобто розкладання великого числа на множники - залишається практично нездійсненною при використанні сучасних обчислювальних потужностей, якщо добуток є достатньо великим. Це означає, що навіть при знанні відкритого ключа та зашифрованого повідомлення зловмисник не має реальної можливості відновити закритий ключ або отримати доступ до первинних даних, якщо не володіє надзвичайними обчислювальними ресурсами або новими методами факторизації.

Стандартна реалізація RSA передбачає створення пари ключів: відкритого (public key), який використовується для шифрування або перевірки підпису, і приватного (private key), який застосовується для розшифрування або створення цифрового підпису. Процес генерації ключів включає обчислення добутку двох великих простих чисел (позначається як n), визначення функції Ейлера та вибір публічного експонента e, який має бути взаємно простим з функцією Ейлера. Приватний експонент d обчислюється як обернене значення до e за модулем функції Ейлера, що забезпечує математичну симетрію для обернених операцій [62].

Безпека алгоритму RSA прямо залежить від довжини ключа. У сучасних криптографічних практиках вважається безпечним використовувати ключі довжиною 2048 біт або більше, хоча в деяких критичних системах застосовуються ключі довжиною 3072 або навіть 4096 біт. Зі збільшенням довжини ключа зростає рівень криптостійкості, однак пропорційно збільшується і обчислювальне навантаження, зокрема час, необхідний для шифрування або розшифрування даних. Тому вибір довжини ключа вимагає балансу між рівнем безпеки та продуктивністю системи.

Одна з ключових переваг RSA - можливість використання для цифрового підпису. Ця функція дозволяє створити унікальну криптографічну мітку, яка

підтверджує автентичність відправника і цілісність повідомлення. У схемі цифрового підпису дані підписуються приватним ключем, а перевіряються - відкритим. Таким чином, жодна інша особа не може змінити повідомлення без того, щоб підпис втратив свою валідність. Завдяки цій властивості RSA активно використовується у банківських транзакціях, електронному документообігу, платіжних системах і технологіях електронного врядування [53].

Іншою важливою сферою застосування RSA є захищений обмін ключами. Зокрема, RSA дозволяє безпечно передавати симетричний ключ шифрування, використовуючи відкритий ключ отримувача. Після цього власник приватного ключа може розшифрувати отримане повідомлення і використовувати ключ для подальшого симетричного шифрування. Така модель лежить в основі гібридних криптографічних систем, де асиметричні алгоритми використовуються на етапі ініціалізації сеансу, а симетричні - для обробки основного трафіку, що забезпечує оптимальний баланс між безпекою та швидкістю.

Важливою складовою інфраструктури безпеки, у якій RSA відіграє провідну роль, є інфраструктура відкритих ключів (PKI - Public Key Infrastructure). У межах PKI кожному користувачу або пристрою призначається унікальний цифровий сертифікат, підписаний авторитетним сертифікаційним центром, який містить відкритий ключ та іншу ідентифікаційну інформацію. RSA використовується для перевірки достовірності сертифікатів, створення електронного підпису і безпечного обміну ключами, що робить цей алгоритм центральним компонентом побудови захищених комунікацій в інтернеті, включаючи протоколи HTTPS, S/MIME, SSH тощо [59].

Разом з тим, алгоритм RSA має і суттєві обмеження, зумовлені його математичною природою. Зокрема, він потребує значних обчислювальних ресурсів, особливо при роботі з довгими ключами. У порівнянні з симетричними алгоритмами, такими як AES або ChaCha20, RSA демонструє набагато нижчу швидкість як при шифруванні, так і при розшифруванні даних. Через це RSA не використовується для масового шифрування великих обсягів даних у реальному

часі, оскільки це створює критичне навантаження на систему і може призвести до затримок, неприйнятних у мережевих або мобільних застосунках.

Окрім цього, з розвитком квантових обчислень виникає ризик майбутнього зниження ефективності RSA, оскільки квантові алгоритми, зокрема алгоритм Шора, потенційно здатні вирішити задачу факторизації за поліноміальний час. Хоча на сьогоднішній день широкодоступних квантових комп’ютерів з достатньою потужністю не існує, вже зараз ведеться активна робота над створенням постквантових криптографічних алгоритмів, які могли б замінити RSA у довгостроковій перспективі [30].

Отже, алгоритм RSA продовжує залишатися критично важливим інструментом захисту в інформаційних системах, особливо в контексті автентифікації, цифрового підпису і безпечної передачі симетричних ключів. Його надійність, математична прозорість і глибока інтеграція в глобальну інфраструктуру відкритих ключів зумовили широке поширення і стабільну позицію серед інших криптографічних рішень. Однак ефективне використання RSA потребує врахування його обмежень, зокрема пов’язаних із продуктивністю та потенційними ризиками у майбутньому криптографічному ландшафті.

У системах з підвищеними вимогами до ефективності та безпеки часто застосовується комбінований підхід, у якому асиметричні алгоритми використовуються для передачі ключів або автентифікації, а безпосереднє шифрування інформації здійснюється за допомогою симетричних алгоритмів. Такий підхід дозволяє досягти балансу між продуктивністю, безпекою і масштабованістю рішень. Наприклад, у протоколі SSL/TLS передача сеансового ключа здійснюється через RSA або інший асиметричний метод, після чого встановлюється симетричне шифрування для подальшої передачі даних.

Оцінювання ефективності криптографічних алгоритмів не може обмежуватися лише теоретичною стійкістю. Практичні аспекти, такі як швидкість обробки, енергоефективність, підтримка апаратного прискорення, а також сумісність з мобільними платформами, мають вирішальне значення у виборі алгоритмів для конкретних застосунків. Наприклад, AES відзначається винятковою

ефективністю на пристроях із підтримкою апаратного шифрування, але на застарілих мобільних пристроях, де такої підтримки немає, ChaCha20 демонструє вищу продуктивність. У свою чергу, RSA потребує значних обчислювальних ресурсів, що обмежує його застосування на енергонезалежних пристроях або в системах з реальним часом [48].

Іншим критично важливим аспектом є стійкість до специфічних видів атак. Наприклад, AES довів свою ефективність проти всіх відомих класичних криптоаналітичних атак, проте уразливість може виникати при неправильному виборі режиму шифрування або при повторному використанні одноразових ключів. ChaCha20, у свою чергу, завдяки своїй структурі, краще захищений від атак, пов’язаних із витоком часових характеристик. Blowfish, хоча й стійкий до більшості відомих атак, має обмеження у вигляді невеликого розміру блоку, що знижує його надійність у сучасному контексті. RSA, за умови використання достатньо довгих ключів, зберігає стійкість до класичних атак, але потребує періодичного оновлення параметрів через зростання обчислювальних можливостей зловмисників.

Таблиця 1.1. містить порівняння чотирьох криптографічних алгоритмів - AES, RSA, Blowfish та ChaCha20. Вона відображає основні характеристики кожного алгоритму, зокрема їх тип шифрування (симетричне чи асиметричне), переваги та недоліки.

Таблиця 1.1 - Порівняння ефективності основних криптографічних алгоритмів (AES, RSA, Blowfish, ChaCha20)

Алгоритм	Тип шифрування	Переваги	Недоліки
AES	Симетричне	- Висока швидкість на сучасних платформах. - Надзвичайно безпечний, не виявлено ефективних атак. - Універсальність	- Обмеження в гнучкості: працює лише з блоками 128 біт, що вимагає додаткових налаштувань для обробки даних з іншими розмірами.

Продовження таблиці 1.1

		застосування в різних протоколах (TLS/SSL, VPN, захист мобільних пристроїв).	
RSA	Асиметричне	- Висока безпека завдяки складності факторизації великих чисел. - Гнучкість у використанні (обмін ключами, цифрові підписи). - Легкість інтеграції в різні криптографічні системи.	- Повільність в обробці великих обсягів даних порівняно з симетричними алгоритмами. - Вимагає використання великих ключів (2048 біт і більше), що знижує швидкість роботи.
Blowfish	Симетричне	- Висока швидкість при використанні коротших ключів. - Гнучкість у виборі довжини ключа для балансу між швидкістю та безпекою. - Легкість інтеграції в програмні рішення.	- Розмір блоку 64 біти обмежує ефективність при обробці великих обсягів даних. - Старіший алгоритм, замінений на більш безпечніші варіанти, такі як AES.
ChaCha20	Симетричне	- Вища швидкість на пристроях без апаратного прискорення порівняно з AES. - Висока безпека, розроблений для забезпечення стійкості до атак, таких як атаки по часу. - Широко використовується в сучасних криптографічних	- Відсутність апаратної підтримки на деяких старих пристроях, що може знижувати швидкість на таких платформах.

Продовження таблиці 1.1

		протоколах (TLS/SSL, VPN, Google для мобільних пристроїв).	

Серед розглянутих алгоритмів AES (Advanced Encryption Standard) демонструє найвищий рівень збалансованості між продуктивністю та безпекою. Його підтримка на апаратному рівні, яка реалізується в багатьох сучасних процесорах через інструкції AES-NI, забезпечує значне зменшення навантаження на центральний процесор, що особливо важливо для мобільних платформ. Крім того, його багаторівнева стійкість до криптоаналітичних атак та широке впровадження у міжнародні стандарти безпеки визначають AES як один із найбільш універсальних алгоритмів для захисту даних у реальному часі [18].

Разом із тим, для пристроїв, які не підтримують апаратне прискорення шифрування, найкращою альтернативою є ChaCha20. Цей алгоритм розроблений з урахуванням обмежень продуктивності на мобільних і вбудованих пристроях, а тому здатний забезпечувати високий рівень криптографічного захисту за умов мінімального енергоспоживання. Його ефективність у програмній реалізації, незалежність від специфічної архітектури процесора та стійкість до атак часу роблять ChaCha20 особливо привабливим варіантом для захищеної комунікації в середовищах Android, IoT та інших мобільних екосистемах.

У свою чергу, RSA, незважаючи на свою обчислювальну складність, залишається незамінним у сфері обміну ключами та формування цифрових підписів. Його інтеграція в інфраструктуру відкритих ключів (PKI), а також роль у забезпеченні довіри між учасниками цифрового середовища, підтверджують важливість RSA у широкому спектрі захищених протоколів, таких як HTTPS, SSL/TLS, SSH та інші. Водночас його низька швидкодія при шифруванні великих обсягів даних та підвищене навантаження на процесор обмежують доцільність використання RSA у мобільних додатках, де ресурси є дефіцитними, а швидкість обробки - критично важливою.

Щодо Blowfish, то він і досі застосовується у деяких спадкових або локальних програмних рішеннях завдяки своїй простоті, відкритому коду та високій швидкості виконання. Проте його архітектура, що базується на 64-бітному блоці, є морально застарілою в умовах сучасного рівня загроз і обсягів даних. Відповідно, Blowfish не рекомендовано для використання в нових криптографічних системах, особливо у контексті захисту інформації на мобільних платформах або в середовищах з підвищеними вимогами до стійкості шифрування [60].

Отже, вибір криптографічного алгоритму не може бути універсальним і повинен базуватись на ретельному аналізі багатьох чинників, серед яких особливу роль відіграють цільове застосування, архітектура пристрою, вимоги до швидкодії, енергоефективність, наявність апаратної підтримки, а також рівень загроз, з якими має справу система. У контексті мобільних платформ, де обсяг оперативної пам’яті, обчислювальна потужність та енергетичні ресурси є обмеженими, вибір шифрувального алгоритму набуває ще більшого значення. Такі пристрої обробляють велику кількість конфіденційної та персональної інформації, зокрема медичні, банківські, комунікаційні та геолокаційні дані, що потребують високого рівня захисту як у транзиті, так і при зберіганні.

Правильне рішення щодо впровадження того чи іншого криптографічного алгоритму є не лише технічним завданням, а й стратегічним кроком у напрямку забезпечення цифрової безпеки. Надійно захищена мобільна платформа формує основу для довіри користувача до цифрових сервісів і забезпечує довготривалу стійкість інформаційної інфраструктури. Саме тому вибір алгоритму шифрування має спиратися на сучасні наукові розробки, результати незалежних криптоаналітичних досліджень, а також на відповідність світовим стандартам у галузі інформаційної безпеки.

1.3. Програмні засоби захисту файлів на мобільних платформах

В умовах зростаючої загрози кіберзлочинності та втрати даних, захист інформації на мобільних платформах стає одним з ключових завдань для

користувачів і підприємств. Програмні засоби захисту файлів на мобільних пристроях використовуються для шифрування, що дозволяє забезпечити конфіденційність даних при зберіганні та передачі. У цій частині роботи розглянуто найбільш популярні програми для шифрування файлів на мобільних платформах, зокрема VeraCrypt, AxCrypt та Cryptomator [38].

VeraCrypt - це потужний інструмент для шифрування файлів і створення зашифрованих томів. VeraCrypt є вдосконаленою версією TrueCrypt, популярного програмного забезпечення для шифрування, що припинило свою підтримку. VeraCrypt забезпечує високий рівень безпеки завдяки використанню різних алгоритмів шифрування, таких як AES, Serpent і Twofish. Однією з основних переваг VeraCrypt є можливість створення зашифрованих контейнерних томів, що дозволяє зберігати великі обсяги даних у зашифрованому вигляді. Крім того, VeraCrypt підтримує створення «невидимих» томів, що додає додатковий рівень захисту, забезпечуючи збереження конфіденційності навіть у разі примусу до розкриття пароля [55].

Користувач може створити зашифрований контейнер для зберігання особистих файлів, таких як фотографії, документи або фінансові звіти, і зберігати його на жорсткому диску комп'ютера. Після цього цей контейнер можна передавати на мобільні пристрої через USB або інші засоби, але для доступу до даних на мобільному пристрої потрібно розшифрувати їх за допомогою VeraCrypt на ПК.

Важливо зазначити, що VeraCrypt здебільшого орієнтований на десктопні платформи, зокрема Windows, macOS та Linux. Офіційних мобільних версій програми немає, хоча користувачі можуть зашифровувати дані на комп'ютерах і переносити їх на мобільні пристрої. Такий підхід зручно використовувати для захисту даних при роботі з десктопними ПК, але обмежує можливості на мобільних платформах.

AxCrypt - це зручний та простий у використанні інструмент для шифрування файлів, який підтримує алгоритми AES-128 і AES-256. AxCrypt дозволяє користувачам швидко шифрувати окремі файли або папки, забезпечуючи конфіденційність та захист від несанкціонованого доступу. Це рішення

користується популярністю серед тих, хто потребує швидкого і ефективного захисту особистих або робочих даних [24].

Після завантаження файлів, наприклад, важливих документів або конфіденційної інформації, на мобільний пристрій, користувач може скористатися AxCrypt для шифрування цих файлів. Наприклад, якщо користувач отримав важливий контракт через електронну пошту, він може швидко зашифрувати цей файл перед тим, як зберігати його на своєму телефоні або планшеті.

Однією з головних переваг AxCrypt є його інтеграція з операційними системами Windows, macOS, а також наявність мобільної версії для платформ Android і iOS. Це дозволяє шифрувати файли безпосередньо на мобільних пристроях, що робить AxCrypt зручним інструментом для користувачів, які часто працюють із файлами на смартфонах і планшетах. Програма має простий інтерфейс, що дозволяє навіть новачкам без проблем здійснювати шифрування і дешифрування файлів.

Cryptomator - це програма для шифрування файлів, орієнтована на захист даних, що зберігаються в хмарних сервісах, таких як Google Drive, Dropbox або OneDrive. Cryptomator дозволяє шифрувати окремі файли перед їх завантаженням в хмару, що забезпечує захист даних навіть у разі, якщо доступ до сервісу буде здобуто несанкціонованим чином. Cryptomator використовує AES-256 для шифрування і дозволяє створювати зашифровані папки, в яких файли автоматично шифруються під час завантаження в хмару [45].

Уявімо, що користувач працює над важливими проектами на мобільному пристрої і хоче зберігати ці файли в хмарі для доступу з різних пристроїв. Використовуючи Cryptomator, він може створити зашифровану папку в Google Drive або Dropbox і завантажити туди всі свої робочі файли. Завдяки цьому навіть у разі доступу до його облікового запису третіх осіб, файли залишатимуться зашифрованими, що забезпечує додатковий захист даних.

Однією з переваг Cryptomator є її підтримка платформ Android, iOS, Windows, macOS та Linux, що робить її доступною для широкого кола користувачів. Вона також інтегрується з багатьма популярними хмарними сервісами, що робить її

ідеальним вибором для тих, хто працює з хмарними даними. Cryptomator має простий та зрозумілий інтерфейс, що дозволяє навіть користувачам без технічного досвіду ефективно захищати свої файли.

Крім VeraCrypt, AxCrypt і Cryptomator, існують й інші інструменти для шифрування файлів на мобільних платформах, такі як Bitdefender Mobile Security, NordLocker та інші. Ці програми надають додаткові можливості для забезпечення конфіденційності даних на мобільних пристроях, однак кожна з них має свої особливості та рівень функціональності.

Bitdefender Mobile Security - це комплексне рішення для захисту мобільних пристроїв, яке включає можливість шифрування даних, захист від шкідливих програм і функцію відновлення пристрою у разі втрати або крадіжки. Воно орієнтоване не тільки на шифрування файлів, але й на комплексний захист самого пристрою [3].

Користувач може встановити Bitdefender на свій Android-смартфон, щоб забезпечити шифрування особистих даних, таких як паролі, документи або фотографії. Це допоможе захистити дані від несанкціонованого доступу у випадку, якщо телефон буде втрачено або вкрадено.

NordLocker - це інструмент для шифрування файлів з підтримкою хмарного зберігання. Він дозволяє шифрувати файли перед тим, як завантажити їх в хмару, або зберігати їх у зашифрованому вигляді на локальних пристроях. Він також підтримує інтеграцію з багатьма популярними хмарними сервісами [26].

Користувач може зашифрувати свої файли на локальному диску, а потім за допомогою NordLocker завантажити їх на хмару для зберігання або для спільного доступу. Це дозволяє зберігати дані в захищеному вигляді без ризику їх витоку.

Ці інструменти дозволяють користувачам забезпечити захист своїх даних на мобільних пристроях та на різних платформах, враховуючи різні сценарії використання - від локального зберігання файлів до зберігання в хмарі.

Кожен з цих інструментів має свої переваги та обмеження, що дозволяє користувачам вибирати оптимальне рішення для своїх потреб. Детальніший

порівняльний аналіз функціональності та зручності використання програмних засобів захисту файлів на мобільних платформах представлений у таблиці 1.2.

Таблиця 1.2 - Порівняльний аналіз програмних засобів захисту файлів на мобільних платформах

Характеристика	VeraCrypt	AxCrypt	Cryptomator	Bitdefender Mobile Security	NordLocker
Тип шифрування	Симетричне	Симетричне	Симетричне	Комплексний захист	Симетричне
Платформи	Windows, macOS, Linux	Windows, macOS, Android, iOS	Windows, macOS, Linux, Android, iOS	Android, iOS	Windows, macOS, Android, iOS
Підтримка мобільних платформ	Ні	Так	Так	Так	Так
Інтерфейс	Складний для початківців	Простий і зручний	Простий і зручний	Зручний, інтуїтивно зрозумілий	Зручний, інтуїтивно зрозумілий
Застосування	Зашифровані контейнери	Шифрування окремих файлів	Шифрування для хмарних файлів	Шифрування на мобільних пристроях	Шифрування для локальних і хмарних файлів

Продовження таблиці 1.2

Рівень безпеки	Високий (AES, Twofish, Serpent)	Високий (AES-128, AES-256)	Високий (AES-256)	Високий, комплексний захист	Високий (AES-256)
Швидкість шифрування	Повільна, але стабільна	Швидка	Швидка	Швидка	Швидка
Ресурсозатратність	Висока (потрібне апаратне прискорення)	Низька	Низька	Середня	Низька
Підтримка хмарних сервісів	Ні	Ні	Так	Так	Так
Ціна	Безкоштовно	Безкоштовно (обмежена версія), Платна версія	Безкоштовно	Платна (підписка)	Платна (підписка)

Після проведеного порівняння функціональності та зручності використання різних програмних засобів захисту файлів на мобільних платформах, що було представлено в таблиці 1.2, стає очевидним, що хоча існуючі інструменти, такі як VeraCrypt, AxCrypt і Cryptomator, мають певні переваги, вони також мають суттєві обмеження, що знижують їхню ефективність при використанні на мобільних пристроях. Ці програми, незважаючи на свою популярність і функціональність у десктопному середовищі, не завжди оптимізовані для мобільних платформ, що викликає потребу в розробці нового застосунку, здатного задовольнити потреби користувачів у надійному захисті файлів на смартфонах і планшетах [25].

Порівняння цих програм показало, що, наприклад, VeraCrypt, хоча і є потужним інструментом для шифрування даних на десктопах, не має мобільної версії, що створює певні труднощі для користувачів мобільних пристроїв. Додатково, його складний інтерфейс може бути незручним для широкої аудиторії, яка не має спеціальних технічних знань. Водночас, програми, як-от AxCrypt і Cryptomator, можуть похвалитися більш зручними інтерфейсами, але при цьому їхня функціональність і ефективність шифрування обмежені порівняно з VeraCrypt.

Враховуючи ці фактори, для розробки нового мобільного застосунку необхідно інтегрувати найкращі аспекти існуючих рішень, одночасно усуваючи їхні недоліки. Це стосується як спрощення інтерфейсу, так і оптимізації алгоритмів шифрування для мобільних платформ, де важливо зберігати баланс між безпекою, швидкістю та використанням ресурсів.

Крім того, підтримка хмарних сервісів, яка є важливою функцією для користувачів мобільних пристроїв, повинна стати однією з основних характеристик нового продукту. Це дозволить безперешкодно шифрувати файли перед їх завантаженням на хмару, що забезпечить додатковий рівень безпеки і зручності використання. І, звісно, велике значення має ефективність роботи застосунку на мобільних пристроях з обмеженими ресурсами, що передбачає розробку оптимізованого програмного забезпечення з мінімальними вимогами до апаратних можливостей.

Таким чином, на основі порівняння існуючих рішень та виявлених недоліків, можна сформулювати вимоги до нового мобільного застосунку, який поєднував би простоту використання, високий рівень безпеки та ефективність роботи на мобільних пристроях з обмеженими ресурсами.

2 АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

2.1. Платформа Android як середовище для розробки

Операційна система Android є однією з найпоширеніших платформ для мобільних пристроїв, що підтримує широке різноманіття застосунків і пристроїв. Розроблена компанією Google, вона базується на ядрі Linux, що забезпечує надійність, безпеку та ефективність використання ресурсів апаратного забезпечення. Від моменту свого випуску Android зарекомендувала себе як гнучка, багатофункціональна та масштабована система, яка відповідає вимогам сучасних мобільних технологій [10].

Основною архітектурною одиницею Android є Android Runtime (ART), яка відповідає за виконання програмного коду. Це дозволяє забезпечити підтримку багатьох мов програмування, зокрема Java, Kotlin, C++ та інші, через використання специфічних інтерфейсів та бібліотек. Кожен застосунок в Android працює у своїй власній пісочниці (sandbox), що обмежує доступ до ресурсів системи, знижуючи ризики впливу зловмисних програм на інші частини операційної системи.

Основними компонентами Android є:

1. Додатки (Applications): Мобільні програми, які взаємодіють з операційною системою через API (Application Programming Interface). Вони використовують вбудовані функції для взаємодії з апаратними засобами, такими як камери, сенсори, GPS та інші.

2. Бібліотеки (Libraries): Android використовує багатий набір бібліотек, що включають елементи, необхідні для рендерингу графіки, забезпечення комунікації через мережу та інші функції.

3. Android Framework: Набір інструментів для створення застосунків, що включає в себе компоненти для роботи з графічним інтерфейсом, управління даними та інтеграції з іншими сервісами.

4. Linux Kernel: Ядро системи, яке забезпечує низькорівневу взаємодію з апаратним забезпеченням, управління ресурсами, а також безпеку та стабільність системи [56].

Однією з основних особливостей Android є його підтримка багатозадачності. Користувачі можуть одночасно виконувати кілька додатків, переходячи між ними без необхідності закривати інші програми, що робить використання системи зручним і ефективним. Багатозадачність також включає в себе можливість фонових операцій, таких як завантаження даних або відтворення медіафайлів.

Інтерфейс Android підтримує велику кількість кастомізацій, що дозволяє розробникам створювати інтерфейси, які відповідають потребам користувачів та специфіці пристроїв. Крім того, Android дозволяє інтегрувати Google Play Services, що дає доступ до додаткових функцій, таких як авторизація через Google, хмарне зберігання даних, аналітика та інші сервіси.

Одним з найбільш важливих аспектів Android є забезпечення безпечності операційної системи. Android використовує різноманітні механізми захисту, зокрема перевірку дозволів на доступ до ресурсів пристрою та ізоляцію даних кожного додатка у пісочниці. Це дозволяє знижувати ризики, пов'язані із зловмисними програмами та несанкціонованим доступом до конфіденційної інформації користувачів [32].

Ще однією важливою характеристикою Android є її відкритий код. Платформа Android є частиною проекту AOSP (Android Open Source Project), що дає можливість розробникам вивчати, модифікувати та адаптувати систему відповідно до своїх потреб. Це дозволяє не лише створювати нові можливості для користувачів, але й розширювати функціональність операційної системи за рахунок специфічних доповнень та інтерфейсів.

У операційній системі Android доступ до файлової системи та ресурсів пристрою організовано через систему дозволів, яка є важливою складовою політики безпеки. Ця система дозволяє користувачам та розробникам контролювати, які дані та ресурси можуть бути використані додатками, що встановлюються на пристрої. Зазначений механізм дозволяє обмежити доступ до чутливих даних, забезпечуючи більшу безпеку користувацької інформації.

Кожен додаток, що взаємодіє з файловою системою, повинен мати відповідні дозволи, які надаються користувачем під час виконання додатку або під час його

встановлення. Основні дозволи, пов'язані з доступом до файлової системи, включають READ_EXTERNAL_STORAGE та WRITE_EXTERNAL_STORAGE, що забезпечують додаткам можливість зчитування та запису файлів на зовнішній носій пам'яті, та READ_INTERNAL_STORAGE та WRITE_INTERNAL_STORAGE, які використовуються для роботи з файлами в межах внутрішньої пам'яті пристрою. Дозвіл READ_EXTERNAL_STORAGE дає змогу додаткам зчитувати файли, що зберігаються на зовнішніх накопичувачах, таких як SD-карти, в той час як дозвіл WRITE_EXTERNAL_STORAGE дозволяє записувати дані на ці носії. Це може бути важливо для додатків, які працюють з медіафайлами, документами та іншими типами даних, що зберігаються на зовнішній пам'яті [31].

З іншого боку, дозвіл READ_INTERNAL_STORAGE надається для зчитування файлів, що зберігаються в межах внутрішньої пам'яті пристрою. Він дозволяє додаткам доступ до файлів, що зберігаються у власних директоріях додатка. Проте, через обмеження доступу, інші програми не можуть отримати доступ до цих файлів без спеціальних дозволів. Дозвіл WRITE_INTERNAL_STORAGE дозволяє додаткам зберігати дані у внутрішній пам'яті пристрою, що може включати збереження кешу або специфічних даних програми. Важливо, що доступ до внутрішньої пам'яті обмежений для сторонніх додатків, що допомагає зберігати безпеку даних.

Android також має механізми контролю над дозволами для специфічних операцій, які не завжди є безпосередньо пов’язаними з файловою системою. Наприклад, дозвіл ACCESS_FINE_LOCATION дозволяє додаткам отримувати точні дані про місцезнаходження користувача, що може бути використано для збереження географічних координат у файлах або для створення картографічних додатків. Дозвіл CAMERA, у свою чергу, надає можливість додаткам використовувати камеру для зйомки фотографій або відео, зберігаючи отримані файли в пам'яті пристрою. Це дозволяє додаткам зберігати медіафайли, створені через камеру, в певних папках, доступних для подальшого використання або редагування [23].

Нова політика дозволів, яка була введена в Android 6.0 (API 23), значно покращила рівень безпеки. Раніше дозволи надавались автоматично під час встановлення додатку, однак тепер система запитує дозвіл користувача тільки на етапі, коли додаток намагається виконати операцію, що потребує доступу до конкретних даних або ресурсів. Такий підхід дає змогу користувачам мати більший контроль над тим, що саме додаток може робити на їхньому пристрої. Якщо дозвіл не надано, додаток не зможе здійснити операції, що потребують доступу до файлової системи або інших чутливих ресурсів.

З початком Android 10 (API 29) була введена концепція Scoped Storage, яка обмежує доступ додатків до загальної файлової системи. Тепер додатки мають обмежений доступ лише до власних файлів та певних директорій, що містять медіафайли, документи та інші дані. Це додаткове обмеження дозволяє підвищити конфіденційність та безпеку даних користувачів, оскільки обмежує можливість зловмисників отримати доступ до файлів користувача без його відома.

Таким чином, система дозволів в Android є важливим елементом, який допомагає забезпечити безпеку даних користувачів та обмежити доступ сторонніх додатків до чутливої інформації. Вона надає користувачам можливість контролювати, які ресурси та дані можуть бути використані додатками, а також забезпечує можливість адаптації операційної системи до нових вимог безпеки, які постійно змінюються в умовах сучасних загроз.

Безпека даних є однією з найбільш важливих складових у розробці Android-застосунків, оскільки пристрої, що працюють на цій операційній системі, часто використовуються для зберігання особистої, фінансової та іншої чутливої інформації користувачів. У зв'язку з цим Android забезпечує кілька рівнів безпеки, щоб захистити дані користувачів від несанкціонованого доступу, витоку або пошкодження.

1. Шифрування даних

Одним із основних методів захисту даних в Android є шифрування. Операційна система підтримує як шифрування даних на рівні пристрою, так і шифрування на рівні додатків. Шифрування даних на пристрої забезпечує захист

інформації в разі фізичного доступу до пристрою, що особливо важливо в контексті викрадення або втрати пристрою. За умовчанням Android підтримує шифрування всіх даних на пристрої за допомогою AES (Advanced Encryption Standard), що гарантує збереження конфіденційності даних навіть при доступі до файлової системи [4].

Шифрування додатково може бути застосоване на рівні окремих додатків. Для цього розробники можуть використовувати бібліотеки та API, які надають Android для реалізації шифрування чутливої інформації. Наприклад, можна зашифрувати дані, які зберігаються в базі даних SQLite або в локальних файлах додатка, що дозволяє значно знизити ризики, пов'язані з витоком інформації при несанкціонованому доступі до цих файлів.

2. Аутентифікація та авторизація користувачів

Для захисту даних, що зберігаються в додатках, необхідно застосовувати механізми аутентифікації та авторизації. Це може бути реалізовано за допомогою різних методів, таких як паролі, пін-коди, біометрична аутентифікація (відбитки пальців, розпізнавання обличчя), а також двофакторна аутентифікація (2FA). Біометричні методи аутентифікації стали широко використовуваними в Android, оскільки вони забезпечують високу ступінь захисту, а також зручність для користувачів [28].

Один із основних інструментів для реалізації біометричної аутентифікації в Android - це BiometricPrompt API, яке дозволяє інтегрувати підтримку відбитків пальців та розпізнавання обличчя в додатки, що допомагає підвищити безпеку доступу до чутливих даних. Крім того, Android забезпечує можливість налаштування Google Sign-In або інтеграції з іншими сервісами аутентифікації, що дозволяє додатково перевіряти користувачів через зовнішні сервіси, знижуючи ризики компрометації локальних даних.

3. Захист від зловмисного програмного забезпечення

Android активно бореться із загрозами зловмисних додатків через систему Google Play Protect. Ця система автоматично перевіряє додатки на наявність шкідливого програмного забезпечення під час їх завантаження з Google Play, а

також здійснює регулярне сканування пристроїв на наявність потенційно небезпечних програм. Google Play Protect використовує машинне навчання та алгоритми для виявлення аномалій у поведінці додатків, що можуть свідчити про їх шкідливий характер [11].

4. Механізм дозволів

Як уже зазначалося раніше, Android застосовує систему дозволів для контролю доступу додатків до різноманітних даних і ресурсів пристрою. Система дозволяє обмежити доступ додатків до чутливих даних, таких як контакти, історія дзвінків, місцезнаходження, медіафайли та інші ресурси. У цьому контексті Android надає користувачам повний контроль над тим, які дозволи надавати конкретним додаткам, що є важливим кроком у забезпеченні безпеки особистих даних [61].

Останні версії Android також пропонують механізм Scoped Storage, який обмежує доступ додатків до файлової системи, зокрема до зовнішнього сховища. Це знижує ризики несанкціонованого доступу додатків до файлів, що зберігаються на пристрої, оскільки додатки можуть отримувати доступ лише до своїх власних файлів або до конкретних медіафайлів, таких як фотографії чи відео.

5. Оновлення безпеки

Android регулярно випускає оновлення безпеки, які усувають вразливості в операційній системі та додатках, що можуть бути використані зловмисниками для здійснення атак. Оновлення безпеки зазвичай включають виправлення багів та вразливостей, які можуть бути використані для обходу механізмів захисту даних або для несанкціонованого доступу до системних ресурсів [43].

Для підвищення безпеки важливо, щоб користувачі регулярно оновлювали свою операційну систему та додатки, що дозволяє зменшити ризики від відомих загроз. Крім того, розробники повинні використовувати найновіші версії SDK і бібліотек безпеки, що забезпечує відповідність сучасним вимогам до захисту даних.

6. Захист при передачі даних

Оскільки Android-застосунки часто взаємодіють з серверами та іншими віддаленими ресурсами, важливим аспектом безпеки є захист даних під час їх передачі через мережу. Для цього Android підтримує використання HTTPS (Hypertext Transfer Protocol Secure), що дозволяє забезпечити захищену передачу даних через Інтернет. Використання SSL/TLS для шифрування даних при їх передачі є стандартною практикою для всіх додатків, що працюють з чутливою інформацією, такою як особисті дані або платіжна інформація [19].

Таким чином, система безпеки даних в Android забезпечує багаторівневий захист користувацької інформації через шифрування, механізми аутентифікації, систему дозволів, захист від зловмисного програмного забезпечення та регулярні оновлення безпеки. Ці заходи сприяють підвищенню рівня безпеки даних на платформі Android, роблячи її більш стійкою до атак та несанкціонованого доступу.

2.2. Біометрична автентифікація в Android

Біометрична автентифікація є сучасним і високоефективним методом підтвердження особистості користувача на основі унікальних фізіологічних або поведінкових характеристик. Вона використовується для покращення безпеки мобільних пристроїв, а також для полегшення доступу до них без необхідності вводити паролі або PIN-коди. Найбільш поширеними типами біометричних даних є відбитки пальців та розпізнавання обличчя. Кожен з цих методів має свої принципи роботи, технологічні особливості і переваги в контексті використання на платформі Android [49].

Один з найбільш поширених методів біометрії на мобільних пристроях - це відбиток пальця. Принцип його роботи базується на скануванні унікальних рис, що формуються на поверхні шкіри пальців людини, зокрема ліній, вигинів, точок перехрестя та інших елементів, що утворюють характерний малюнок. Завдяки своїй унікальності та високій точності цей метод є надійним інструментом для автентифікації користувачів.

Процес використання відбитка пальця для ідентифікації складається з кількох етапів:

1. Збір біометричних даних: На мобільному пристрої використовуються спеціальні сенсори для захоплення зображення відбитка пальця. Це можуть бути оптичні сенсори, що зчитують малюнок відбитка, або ультразвукові сенсори, що сканують структуру шкіри на глибшому рівні.

2. Обробка та аналіз даних: Після того як сенсор зчитує зображення, воно перетворюється на цифровий шаблон, що містить найбільш важливі характеристики відбитка - особливості ліній, точок зламів тощо. Цей шаблон є унікальним для кожної людини.

3. Порівняння з раніше збереженим шаблоном: Коли користувач спробує пройти аутентифікацію, система порівнює зчитаний шаблон відбитка пальця з тим, що був збережений раніше. Якщо шаблони співпадають, користувач отримує доступ до пристрою або програми [29].

Завдяки високій точності цього методу, відбитки пальців є складними для підробки, що забезпечує їхню надійність як метод біометричної автентифікації. Відбитки пальців використовуються для розблокування пристроїв, підтвердження транзакцій, а також для доступу до приватної інформації в додатках.

Іншою популярною технологією біометрії, що використовується в Android-пристроях, є розпізнавання обличчя. Цей метод дозволяє зчитувати характеристики обличчя користувача за допомогою камери пристрою. Система аналізує різні ознаки обличчя, такі як відстань між очима, форма носа, контури губ, щелепи та інші параметри, що є унікальними для кожної людини.

Процес розпізнавання обличчя відбувається за наступними етапами:

1. Збір зображення: Камера пристрою фіксує зображення обличчя користувача. Для точнішого розпізнавання можуть бути використані інфрачервоні датчики або технології, що дозволяють створювати тривимірну модель обличчя.

2. Аналіз і створення шаблону: Отримане зображення обличчя аналізується для виділення унікальних рис і створення математичної моделі обличчя, що включає важливі біометричні параметри.

3. Порівняння з шаблоном: Коли користувач намагається увійти в систему, отримане нове зображення порівнюється з шаблоном, збереженим у системі. Якщо шаблони збігаються, користувач отримає доступ до пристрою [37].

Цей метод є зручним, оскільки користувачам не потрібно торкатися екрану або сенсорів, що особливо корисно в умовах, коли необхідно забезпечити безконтактний доступ або в умовах підвищеної гігієни (наприклад, у медичних установах).

У процесі порівняння біометричних методів автентифікації, таких як відбитки пальців і розпізнавання обличчя, варто враховувати різні аспекти кожної технології, зокрема точність, зручність використання та рівень безпеки. Зокрема, відбитки пальців зазвичай забезпечують вищу точність і рівень безпеки, тоді як розпізнавання обличчя є більш зручним у контексті безконтактного доступу, але може бути менш надійним за умов зміни зовнішності чи освітлення. Детальніше порівняння цих методів наведено в таблиці 2.1

Таблиця 2.1 - Порівняння відбитків пальців та розпізнавання обличчя

Критерій	Відбитки пальців	Розпізнавання обличчя
Технологія	Сканування унікальних рис відбитка пальця.	Аналіз та порівняння характерних рис обличчя.
Точність	Висока точність, малоймовірно, що відбиток можна підробити.	Менш точне, може залежати від освітлення та положення обличчя.
Вимоги до пристрою	Наявність сенсора для зчитування відбитка пальця.	Наявність камери або інфрачервоних датчиків.
Швидкість автентифікації	Швидка, зазвичай відразу після прикладання пальця.	Швидка, але може залежати від якості камери.
Зручність використання	Потрібно фізично торкнутися сенсора.	Не потрібно доторкатися до пристрою, достатньо подивитися в камеру.

Продовження таблиці 2.1

Безпека	Висока, важко підробити відбиток пальця.	Менш безпечно, може бути вразливим до умов освітлення та змін у зовнішності.
Залежність від фізичних змін	Не залежить від змін зовнішності.	Може бути менш ефективним при змінах зовнішності (наприклад, зачіска або окуляри).
Приватність	Відбитки пальців є більш приватними, оскільки їх складніше скопіювати.	Можливо менше забезпечення конфіденційності через можливість фіксації обличчя з відстані.
Додаткові фактори	Зазвичай вимагає фізичного контакту.	Може працювати на відстані, зручний в умовах безконтактного доступу.

Обидва методи мають свої переваги і недоліки. Відбитки пальців є більш точними і забезпечують надійну безпеку, оскільки відбиток пальця унікальний і складний для підробки. Водночас, користувачеві необхідно фізично доторкнутися до сенсора, що може бути незручно в деяких ситуаціях.

Розпізнавання обличчя, в свою чергу, є більш зручним, оскільки не вимагає фізичного контакту з пристроєм, що робить його більш комфортним в повсякденному використанні. Проте цей метод може бути менш точним за відбитки пальців, оскільки він може бути схильний до помилок через зміни в освітленні, положенні голови або змін в зовнішньому вигляді користувача (наприклад, зміни зачіски чи носіння окулярів) [8].

З огляду на це, вибір між відбитком пальця та розпізнаванням обличчя залежить від конкретних потреб і вимог до безпеки в кожному окремому випадку.

Для розробників Android-платформи існує спеціальне Biometric API, яке забезпечує стандартізований доступ до біометричних функцій пристроїв, включаючи відбитки пальців і розпізнавання обличчя. Цей API дозволяє розробникам створювати додатки, що підтримують біометричну автентифікацію, без необхідності детального вивчення особливостей кожної конкретної технології, на якій побудовано біометричне розпізнавання.

Таке API підтримує як автентифікацію за відбитком пальця, так і за обличчям, і може бути інтегроване в додатки для забезпечення безпеки доступу до приватної інформації або підтвердження транзакцій.

Безпечне зберігання біометричних даних є важливим аспектом у забезпеченні надійної захищеності персональної інформації користувачів мобільних пристроїв. Оскільки біометрія, така як відбитки пальців або розпізнавання обличчя, є дуже чутливою інформацією, її захист є критично важливим для запобігання несанкціонованому доступу та забезпечення конфіденційності даних [40].

Біометричні дані є унікальними для кожної людини, тому їх обробка та зберігання потребують особливої уваги. Основними проблемами безпеки є можливість викрадення цих даних, їх підробка, а також потенційне порушення конфіденційності в разі витоку інформації.

Основні загрози зберігання біометричних даних включають:

1. Крадіжка біометричних даних: У разі компрометації даних, таких як відбитки пальців чи зображення обличчя, ці дані можуть бути використані зловмисниками для несанкціонованого доступу до пристроїв або додатків.

2. Підробка біометричних даних: Якщо не забезпечити належний рівень захисту, зловмисники можуть спробувати підробити біометричні дані для обходу систем автентифікації.

3. Витік даних: У разі порушення безпеки серверів чи баз даних, де зберігаються біометричні дані, можуть виникнути значні проблеми з конфіденційністю особистої інформації користувачів [34].

Один з основних методів забезпечення безпеки біометричних даних - це шифрування. Шифрування дозволяє захистити біометричні дані від несанкціонованого доступу, перетворюючи їх на непізнавані дані, які можуть бути розшифровані лише за допомогою спеціального ключа. Важливо, щоб шифрування здійснювалося на рівні пристрою, щоб навіть у разі витоку даних вони були незрозумілими без доступу до ключа шифрування.

У більшості сучасних мобільних пристроїв Android, біометричні дані не зберігаються у відкритому вигляді. Замість цього використовуються спеціальні засоби для локального шифрування даних, що гарантує їх захист навіть у разі фізичного доступу до пристрою. Операційна система Android використовує Keystore API, яке забезпечує безпечне зберігання ключів шифрування, що застосовуються для біометричних даних.

Ще одним методом безпечного зберігання біометричних даних є використання зашифрованих контейнерів, які дозволяють зберігати біометричні дані в зашифрованому вигляді в спеціальних обсягах пам'яті. Ці контейнери мають додаткові механізми захисту, які унеможливлюють несанкціонований доступ, навіть якщо зловмисник отримує фізичний доступ до пристрою.

При цьому важливо, щоб ці контейнери знаходилися в захищеній пам'яті, доступ до якої обмежений лише для операційної системи. На платформі Android для цього використовується Trusted Execution Environment (TEE) або Secure Enclave, які є спеціальними апаратними рішеннями для захисту даних на апаратному рівні [65].

Іншим важливим аспектом захисту біометричних даних є застосування криптографії з відкритим ключем. Цей метод дозволяє створювати пару ключів: публічний і приватний. Публічний ключ може бути використаний для шифрування даних, а приватний - для їх розшифрування. Використання такої криптографії дозволяє захищати біометричні дані навіть у разі, якщо зловмисник отримує доступ до публічних ключів.

Для посилення безпеки при зберіганні біометричних даних також використовується автентифікація на основі багатофакторної перевірки. Це може

включати комбінацію біометричних даних з іншими формами аутентифікації, такими як паролі або PIN-коди [44].

Біометричні дані, зазвичай, зберігаються локально на пристрої, а не в хмарних сервісах, що дає змогу обмежити кількість потенційних вразливих точок для зловмисників. Доступ до біометричних даних має лише операційна система Android і додатки, що мають відповідні дозволи.

Безпека біометричних даних також регулюється численними стандартами і нормативними актами, які визначають вимоги до їх зберігання та обробки. Наприклад, Загальний регламент захисту даних (GDPR) у Європейському Союзі вимагає від організацій дотримуватися суворих стандартів щодо захисту особистої інформації, включаючи біометричні дані.

Також існують спеціальні стандарти, такі як ISO/IEC 24745 для забезпечення захисту біометричних даних, які допомагають організаціям правильно налаштовувати системи для зберігання та обробки таких даних [27].

Забезпечення безпеки зберігання біометричних даних є важливим елементом для захисту конфіденційної інформації користувачів. Завдяки використанню сучасних методів шифрування, безпечного зберігання даних у зашифрованих контейнерах та криптографії, можна значно знизити ризики викрадення або несанкціонованого доступу до біометричних даних. Однак необхідно враховувати, що з розвитком технологій виникають нові виклики для забезпечення безпеки, які потребують постійного удосконалення систем захисту.

2.3. Опис та принцип роботи алгоритму AES

Алгоритм AES (Advanced Encryption Standard) є симетричним блоковим шифром, що активно використовується для захисту даних в різних галузях, зокрема в мобільних пристроях, де забезпечення конфіденційності є особливо важливим. AES працює з блоками даних розміром 128 біт (16 байт) і підтримує три різні розміри ключів: 128 біт, 192 біт і 256 біт. Кожен з цих розмірів забезпечує певний

рівень безпеки, при цьому більший розмір ключа відповідно збільшує складність криптоаналізу.

Основні етапи шифрування в AES відбуваються через серію раундів обробки, кількість яких залежить від довжини ключа. Для AES-128 використовується 10 раундів, для AES-192 - 12, а для AES-256 - 14. Кожен раунд включає декілька операцій, спрямованих на перетворення даних таким чином, щоб результат було складно відновити без правильного ключа. Ці операції виконуються в певному порядку та складаються з кількох основних етапів: SubBytes, ShiftRows, MixColumns та AddRoundKey [52].

На першому етапі SubBytes кожен байт вхідного блоку замінюється за допомогою таблиці підстановки, яка називається S-box. S-box є нелінійною таблицею, яка перетворює кожен байт на інший відповідно до визначеної підстановки. Це важлива операція, оскільки вона забезпечує нелінійність, що робить алгоритм стійким до деяких типів атак, таких як лінійний або диференційний криптоаналіз.

Після цього відбувається операція ShiftRows, де кожен рядок блоку зміщується на певну кількість позицій. Наприклад, перший рядок залишається без змін, другий зсувається на одну позицію вліво, третій - на дві позиції, а четвертий - на три. Цей етап допомагає розподілити байти блоку по різних стовпцях, що покращує змішування та ускладнює відновлення початкових даних [46].

Наступний етап - MixColumns - виконується лише в середині раундів (окрім останнього). На цьому етапі кожен стовпець блоку змішується за допомогою матричних операцій в просторі поля Галуа. Це дозволяє комбінувати байти з різних частин блоку і створює додаткову складність у структурі шифрування. Цей етап ефективно забезпечує дифузію, що є ключовим елементом для захисту від певних атак.

Завершує кожен раунд операція AddRoundKey, під час якої до блоку додається відповідний раундовий ключ, який генерується в результаті процесу розширення початкового ключа. Це додає елемент непередбачуваності до кожного етапу шифрування, оскільки кожен раунд використовує інший ключ.

Ключовим етапом в алгоритмі AES є процес розширення ключа (key expansion), під час якого початковий ключ розбивається на серію ключів, що використовуються на кожному етапі шифрування. Наприклад, для AES-128, початковий 128-бітний ключ розширюється на 11 раундових ключів (для 10 раундів шифрування), кожен з яких має розмір 128 біт. Для AES-192 та AES-256 процес розширення також дозволяє генерувати відповідну кількість раундових ключів, що забезпечує необхідний рівень безпеки [2].

Процес дешифрування в AES виконується в зворотному порядку. Спочатку додається останній раундовий ключ, потім відбувається операція InverseMixColumns (для всіх раундів, крім останнього), після чого проводяться зворотні операції ShiftRows та SubBytes. Після кожного раунду відновлюється частина початкових даних, і в результаті, після проходження всіх раундів, виходить оригінальний відкритий текст.

Таким чином, структура алгоритму AES сприяє високому рівню безпеки завдяки комбінації нелінійних операцій, змішування та підстановки, а також багатоетапному процесу використання ключів на різних етапах шифрування. Це робить алгоритм AES одним з найбільш надійних стандартів шифрування для захисту даних у сучасних інформаційних системах.

Режими шифрування визначають спосіб використання блочного шифру, такого як AES, для обробки даних, що мають більший розмір, ніж один блок (128 біт). Оскільки AES працює з фіксованими блоками даних, для шифрування більших обсягів інформації використовуються різні методи обробки даних. Режими шифрування забезпечують не лише спосіб роботи з великими обсягами даних, але й підвищують рівень безпеки шифрування завдяки різним технікам змішування та перетворення даних. У цьому контексті важливими є такі режими шифрування як ECB, CBC та GCM [51].

ECB (Electronic Codebook) - електронна книжка кодів. Режим ECB є найбільш простим і базовим режимом шифрування. У ньому кожен блок даних шифрується окремо, використовуючи один і той самий ключ. Оскільки для кожного блоку застосовується одна та сама операція шифрування, один і той самий відкритий

текст завжди буде шифруватися в один і той самий зашифрований текст, що може бути уразливим до певних криптографічних атак [15].

Перевага ECB полягає у його простоті та швидкості, оскільки немає необхідності в додаткових операціях для з'єднання блоків. Однак цей режим має суттєві недоліки, серед яких основним є відсутність змішування між блоками. В результаті, однакові блоки відкритого тексту будуть завжди зашифровані однаково, що може дозволити зловмиснику виявити частину структури даних. З цієї причини режим ECB не рекомендується для шифрування великих обсягів чутливої інформації, оскільки він не надає достатнього рівня конфіденційності.

CBC (Cipher Block Chaining) - ланцюг блочного шифру. Режим CBC вирішує проблему режиму ECB, забезпечуючи додаткове змішування між блоками за допомогою операції XOR. В режимі CBC кожен блок відкритого тексту спочатку комбінується з попереднім зашифрованим блоком (або з початковим вектором ініціалізації - IV), а потім шифрується. Тобто, кожен блок даних перед шифруванням залежить від попереднього зашифрованого блоку, що значно ускладнює криптоаналіз.

Перевагою режиму CBC є те, що він гарантує, що однакові блоки відкритого тексту зашифровуються різними результатами, оскільки їх значення залежить від попередніх блоків. Тому цей режим значно підвищує рівень безпеки порівняно з ECB. Однак, одна з основних проблем режиму CBC - це необхідність використання IV для першого блоку. Якщо IV не є випадковим або не генерується належним чином, безпека шифрування може бути порушена. Крім того, режим CBC вимагає підтримки синхронізації, оскільки в разі втрати чи пошкодження одного з блоків, усі наступні блоки не можуть бути правильно дешифровані.

GCM (Galois/Counter Mode) - режим Галоїса/лічильника. Режим GCM є більш сучасним і складним режимом шифрування, який поєднує в собі шифрування за допомогою лічильника з використанням операцій з полем Галуа для забезпечення цілісності даних. GCM є одним з режимів, який використовується для шифрування даних із забезпеченням цілісності, оскільки він також створює так званий

аутентифікаційний тег (authentication tag), який дозволяє перевірити, чи не було змінено зашифровані дані під час передачі або зберігання [54].

У режимі GCM кожен блок даних комбінується з унікальним лічильником, який збільшується з кожним новим блоком. Після цього використовується операція XOR для об'єднання зашифрованих блоків з лічильником. Окрім шифрування, GCM генерує аутентифікаційний тег, який дозволяє перевірити цілісність даних після їх дешифрування. Це робить режим GCM особливо корисним для ситуацій, коли потрібно не лише зашифрувати дані, але й забезпечити їх цілісність, наприклад, при передачі конфіденційної інформації через незахищені канали зв'язку.

Перевагою GCM є висока швидкість роботи, здатність до паралельної обробки блоків та забезпечення цілісності даних за допомогою аутентифікаційного тега. Крім того, режим GCM є стійким до атак, таких як повторне використання IV або заміна блоків.

Режими шифрування AES мають різні переваги та обмеження, і вибір між ними залежить від конкретних вимог до безпеки та ефективності. Режим ECB, хоча й є швидким, має обмежену безпеку через відсутність змішування між блоками. Режим CBC покращує безпеку шляхом введення залежності між блоками, але вимагає правильного управління IV для запобігання атакам. Режим GCM поєднує в собі шифрування та забезпечення цілісності даних, надаючи високий рівень безпеки і ефективності, що робить його ідеальним для багатьох сучасних застосувань, включаючи мобільні платформи [42].

Для мобільних середовищ, де важлива як швидкість, так і безпека, GCM часто є вибором за замовчуванням, оскільки він забезпечує не лише конфіденційність даних, але й їх цілісність, що є критичним для багатьох застосувань.

Алгоритм AES (Advanced Encryption Standard) є одним з найбільш популярних та широко використовуваних стандартів шифрування у сучасних системах безпеки, зокрема в мобільних середовищах. Його ефективність та надійність роблять його ідеальним для використання в мобільних додатках, де необхідно поєднувати високу швидкість обробки даних та забезпечення надійного

рівня безпеки. Використання AES у мобільних середовищах має кілька суттєвих переваг.

AES розроблений для ефективної обробки даних в обмежених ресурсах, що є важливим аспектом для мобільних пристроїв, таких як смартфони, планшети та інші мобільні гаджети.

Використання AES у мобільних середовищах дозволяє мінімізувати затримки при шифруванні та дешифруванні даних, що особливо важливо для забезпечення швидкої обробки транзакцій та інших чутливих операцій. Сучасні мобільні чіпи оснащені апаратними прискорювачами для алгоритмів AES, що значно підвищує швидкість виконання операцій шифрування та зменшує споживання енергії [57].

AES забезпечує високий рівень захисту даних завдяки своїй стійкості до криптографічних атак, таких як атаки за допомогою грубої сили або диференційний криптоаналіз. Алгоритм працює з блоками даних розміром 128 біт і підтримує різні довжини ключа (128, 192 або 256 біт), що дозволяє забезпечити різні рівні безпеки залежно від вимог до захисту інформації.

У мобільних середовищах, де захист особистих даних є критично важливим, AES дозволяє зберігати конфіденційність даних користувачів, таких як паролі, фінансові дані, медичні записи та інша чутлива інформація. Застосування AES у мобільних додатках дозволяє уникнути можливих витоків даних через несанкціонований доступ, забезпечуючи їх надійне шифрування як при зберіганні, так і при передачі через незахищені мережі [6].

Мобільні пристрої, як правило, мають обмежені ресурси, такі як обсяг оперативної пам'яті та потужність процесора. Однак AES здатен ефективно працювати в таких умовах завдяки своїй структурі та оптимізованим алгоритмам. Крім того, багато сучасних мобільних пристроїв оснащені спеціалізованими апаратними модулем для шифрування (наприклад, Trusted Execution Environment - TEE або Hardware Security Module - HSM), що дозволяє швидко та безпечно виконувати операції шифрування та дешифрування без впливу на загальну продуктивність пристрою [13].

Ці апаратні можливості дозволяють реалізовувати AES без значного навантаження на центральний процесор мобільного пристрою, що покращує не лише продуктивність, а й енергоефективність. Це важливо для мобільних додатків, які працюють в фоновому режимі або виконують операції в режимі реального часу, де необхідно забезпечити баланс між швидкістю виконання та економією енергії.

Оскільки AES є міжнародно визнаним стандартом, він забезпечує високу сумісність з іншими стандартами безпеки, які використовуються для захисту мобільних додатків. Це включає в себе протоколи безпеки, такі як SSL/TLS для захищеної передачі даних в Інтернеті, а також методи автентифікації та захисту даних, які використовують шифрування як основний елемент безпеки [33].

Використання AES в мобільних додатках забезпечує не лише конфіденційність, але й інтеграцію з іншими механізмами захисту даних, що дозволяє реалізувати багаторівневу стратегію безпеки. Це важливо для мобільних банківських додатків, платіжних систем та інших сервісів, що працюють з конфіденційною інформацією.

AES також є гнучким і масштабованим алгоритмом, оскільки його параметри можна налаштовувати в залежності від конкретних вимог до безпеки. Наприклад, при використанні ключа довжиною 128 біт забезпечується високий рівень продуктивності при прийнятному рівні безпеки, тоді як довші ключі (192 або 256 біт) забезпечують ще вищий рівень захисту для більш чутливих даних. Ця гнучкість дозволяє налаштовувати безпеку відповідно до потреб мобільних додатків та конкретних користувачів.

Використання AES у мобільному середовищі має численні переваги, які включають високу ефективність, надійний рівень безпеки, адаптацію до обмежених ресурсів мобільних пристроїв, а також сумісність з іншими стандартами безпеки. Враховуючи ці переваги, AES є ідеальним вибором для шифрування даних у мобільних додатках, де захист конфіденційної інформації є критичним аспектом безпеки.

3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ПРИКЛАД ВИКОРИСТАННЯ

3.1. Мета створення застосунку та сфери його застосування

У сучасному інформаційному суспільстві мобільні пристрої дедалі частіше використовуються не лише як засіб комунікації, а і як повноцінні носії персональної інформації. На смартфонах зберігаються копії паспортів, ідентифікаційні документи, медичні довідки, банківські виписки, приватне листування, персональні фото та інші важливі файли. Втрата чи крадіжка такого пристрою створює ризик витоку критично важливих персональних даних, що може мати серйозні правові, фінансові й моральні наслідки для користувача.

Для вирішення цієї проблеми було розроблено застосунок, функціонал якого докладно представлений у Додатку 1. Він дозволяє здійснювати локальне шифрування файлів без підключення до Інтернету, що виключає залежність від хмарних сервісів або сторонніх серверів. Таким чином, усі операції з обробки даних виконуються виключно на пристрої користувача, а шифровані файли можуть зберігатися як у внутрішньому сховищі смартфона, так і на SD-карті.

Мета розробки даного мобільного застосунку полягає у створенні універсального інструменту для надійного локального шифрування файлів на пристроях з операційною системою Android. В умовах постійного зростання обсягів персональної та робочої інформації, яка зберігається на смартфонах, питання захисту даних набуває особливої актуальності. Сучасні загрози - такі як втрати пристроїв, зловмисні програми, спроби несанкціонованого доступу або перехоплення даних - зумовлюють потребу у застосуванні криптографічних засобів захисту на рівні файлів.

Розроблений застосунок (див. Додаток А) дозволяє зашифровувати будь-які типи файлів за допомогою алгоритму AES, із подальшим збереженням на пристрої у захищеному вигляді. Доступ до розшифровки може надаватися лише власнику пристрою після введення пароля або проходження біометричної автентифікації, що реалізовано через безпечне середовище Android KeyStore. Таким чином, додаток не лише виконує технічну функцію шифрування, а й формує модель довіри, у межах

якої дані ніколи не покидають пристрій та не піддаються обробці на сторонніх сервісах.

Мобільні пристрої дедалі частіше використовуються як сховище особистої інформації: від сканів документів до банківських і медичних даних. Використання розробленого застосунку дозволяє надійно захистити такі файли від несанкціонованого доступу, навіть якщо пристрій буде втрачено або викрадено.

Криптографічне ядро програми реалізовано за допомогою AES у режимі CBC. Основні фрагменти шифрування виглядають наступним чином:

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);
byte[] encryptedBytes = cipher.doFinal(inputBytes);
```

Рисунок 3.1 - Фрагмент коду для шифрування даних з використанням AES/CBC/PKCS5Padding (Додаток А, клас PasswordEncryptionHelper, метод encryptFile)

Після шифрування дані зберігаються у новому файлі, до якого додається ініціалізаційний вектор:

```
outputFile.write(iv);              // записуємо IV на початку
outputFile.write(encryptedBytes);  // далі записуються зашифровані байти
```

Рисунок 3.2 - Фрагмент коду шифрування файлу (Додаток А, клас PasswordEncryptionHelper, метод encryptFile)

Для зручності та підвищення безпеки реалізована функція автоматичного видалення оригінального файлу після шифрування, що дозволяє уникнути зберігання незахищених копій:

```
if (deleteOriginal) {
    originalFile.delete(); // видалення оригіналу
}
```

Рисунок 3.3 - Видалення оригінального файлу після шифрування (Додаток А, клас PasswordEncryptionHelper, метод encryptFile)

Таким чином, конфіденційні персональні дані залишаються доступними виключно власнику пристрою, який знає пароль або може пройти біометричну перевірку.

У робочому контексті - особливо в умовах віддаленої роботи, поїздок або використання політики BYOD (Bring Your Own Device) - застосунок може бути корисним для захисту корпоративної інформації: договорів, технічної документації, баз клієнтів тощо.

Реалізація захисту за допомогою Android KeyStore дозволяє централізовано контролювати доступ до ключів шифрування через біометричну автентифікацію. Ключ створюється так:

```
KeyGenParameterSpec keySpec = new KeyGenParameterSpec.Builder(
        keyAlias,
        KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT
)
.setBlockModes(KeyProperties.BLOCK_MODE_CBC)
.setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
.setUserAuthenticationRequired(true) // обов'язкова біометрія
.build();
```

Рисунок 3.4 - Реалізація генерації ключа з біометричною автентифікацією у Android за допомогою KeyGenParameterSpec (Додаток А, клас BiometricEncryptionHelper метод getOrGenerateKey)

Це означає, що розшифрувати файл зможе лише автентифікований користувач, і жоден інший додаток або сторонній інструмент не отримає доступ до ключа.

У великих організаціях застосунок може бути використаний разом із системами MDM (Mobile Device Management), які дозволяють масове розгортання, контроль версій та налаштування політик доступу. В результаті, застосунок підходить як для індивідуального використання працівниками, так і для впровадження на рівні відділів інформаційної безпеки.

Переваги над хмарними сервісами

Хоча більшість користувачів надають перевагу зберіганню даних у хмарі, такі сервіси мають суттєві вразливості:

- ключі можуть зберігатися на сервері;
- з’єднання проходить через публічні мережі;
- потрібні акаунти та синхронізація;
- складно досягти повної автономності.

На відміну від цього, розроблений застосунок гарантує локальну обробку даних та не використовує мережевих ресурсів. Весь процес - від вибору файлу до шифрування - відбувається виключно на пристрої користувача, без залучення сторонніх API або серверів.

Також реалізовано підтримку Android Storage Access Framework для вибору файлів незалежно від джерела (внутрішня пам’ять, SD-карта тощо):

```
Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
intent.setType("*/*");
startActivityForResult(intent, REQUEST_CODE_PICK_FILE);
```

Рисунок3.5 - Підтримка Android Storage Access Framework для вибору файлів (Додаток А, клас MainActivity метод pickFile)

Це надає користувачеві максимальну гнучкість у виборі, водночас зберігаючи безпеку. У поєднанні з функцією видалення оригіналів і зберіганням ключів у KeyStore, застосунок стає надійнішою альтернативою хмарним рішенням.

Створення даного застосунку було спрямоване на:

- захист персональних документів без мережевої залежності;

- використання в робочих середовищах, включаючи сценарії віддаленої та конфіденційної роботи;
- забезпечення автономного, високозахищеного механізму, який перевершує за рівнем контролю й безпеки більшість доступних хмарних сервісів.

3.2. Ключові фрагменти програмного коду

Один із центральних компонентів розробленого застосунку - реалізація шифрування та розшифрування файлів із використанням симетричного алгоритму AES (Advanced Encryption Standard). Даний алгоритм було обрано через його стійкість до криптоаналітичних атак, високу продуктивність та підтримку апаратного прискорення в більшості сучасних мобільних пристроїв.

Функціональність шифрування у застосунку реалізована у двох незалежних класах:

1. PasswordEncryptionHelper - для шифрування на основі пароля, який вводиться користувачем вручну;

2. BiometricEncryptionHelper - для шифрування із залученням біометричної автентифікації та Android KeyStore, який генерує і зберігає криптографічний ключ у захищеному середовищі.

Попри різницю в способах автентифікації доступу до ключа, сам механізм шифрування та розшифрування в обох класах реалізовано на основі одного й того ж криптографічного алгоритму - AES у режимі CBC з доповненням PKCS5Padding. Структура операцій та принципи обробки даних залишаються ідентичними, що забезпечує єдність логіки шифрування при збереженні варіативності у способах захисту доступу до ключа.

У класі PasswordEncryptionHelper процес передбачає перетворення вмісту файлу у байтовий масив, його шифрування з допомогою алгоритму AES у режимі CBC (Cipher Block Chaining) з додаванням ініціалізаційного вектора (IV), та збереження результату у новий файл.

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);
byte[] encryptedBytes = cipher.doFinal(inputBytes);
outputFile.write(iv);
outputFile.write(encryptedBytes);
```

Рисунок 3.6 - Фрагмент коду шифрування файлу (Додаток А, клас PasswordEncryptionHelper, метод encryptFile)

Опис фрагмента коду:

1. Створюється об’єкт Cipher із параметрами шифрування AES/CBC/PKCS5Padding, де:

- AES - алгоритм шифрування;
- CBC - режим блочного шифрування;
- PKCS5Padding - метод доповнення блоків.

2. Ініціалізація Cipher відбувається в режимі ENCRYPT_MODE, з передачею симетричного ключа (secretKey) і згенерованого ініціалізаційного вектора (ivSpec).

3. Метод doFinal() виконує шифрування байтового вмісту файлу (inputBytes).

4. Результат записується у новий файл:

- спочатку - IV (необхідний для подальшого дешифрування);
- далі - зашифровані дані.

```
byte[] iv = Arrays.copyOfRange(fileBytes, 0, 16);
IvParameterSpec ivSpec = new IvParameterSpec(iv);
cipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec);
byte[] decryptedBytes = cipher.doFinal(encryptedData);
```

Рисунок 3.7 - Фрагмент коду розшифрування файлу (Додаток А, клас PasswordEncryptionHelper, метод decryptFile)

Опис фрагмента коду:

1. З перших 16 байтів зашифрованого файлу зчитується IV - Arrays.copyOfRange(fileBytes, 0, 16).

2. Створюється IvParameterSpec для розшифрування.

3. Cipher ініціалізується в режимі DECRYPT_MODE, використовуючи той самий ключ, що і при шифруванні (secretKey) та витягнутий IV.

4. Метод doFinal() обробляє зашифровані байти (encryptedData) та повертає розшифровані дані.

Шифрування та розшифрування файлів у застосунку реалізовано відповідно до сучасних стандартів криптографії. Алгоритм AES у режимі CBC з ініціалізаційним вектором забезпечує стійкість до атак повторного шифрування та гарантує унікальність результату навіть при обробці ідентичних вхідних даних.

Особливістю реалізації є те, що IV зберігається у тілі зашифрованого файлу, що забезпечує можливість коректної розшифровки без необхідності зберігати його окремо. Весь процес відбувається локально на пристрої, без потреби в мережевих підключеннях або зовнішніх сервісах, що значно підвищує загальний рівень безпеки системи.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_CODE_PICK_FILE && resultCode == Activity.RESULT_OK) {
        if (data != null && data.getData() != null) {
            selectedFileUri = data.getData();
            selectedFilePath = FileUtil.getPath(requireContext(), selectedFileUri);
            fileNameTextView.setText(FileUtil.getFileName(requireContext(), selectedFileUri));
        }
    }
}
```

Рисунок 3.8 - Фрагмент обробки обраного файлу (Додаток А, клас MainActivity, метод onActivityResult)

Опис фрагмента коду:

1. Перевіряється, що результат повернуто для правильного запиту (REQUEST_CODE_PICK_FILE) і що вибір було підтверджено (RESULT_OK).

2. За допомогою data.getData() отримується URI обраного файлу.

3. FileUtil.getPath(...) перетворює URI на фактичний шлях до файлу, що дозволяє далі обробляти його в методах шифрування або дешифрування.

4. Назва обраного файлу виводиться на екран за допомогою fileNameTextView.setText(...).

Інтеграція Android Storage Access Framework забезпечила зручний і безпечний вибір файлів незалежно від їхнього фізичного розміщення (внутрішня пам’ять, SD-карта, зовнішні провідники). Завдяки використанню стандартного Android API, застосунок дотримується політик безпеки Android 10+ і не потребує повного доступу до файлової системи.

Така реалізація:

- знижує ризик витоку даних;
- не потребує небезпечних дозволів (READ_EXTERNAL_STORAGE);
- підвищує користувацький комфорт за рахунок зручного візуального вибору файлів.

Уся логіка вибору файлів і отримання їх URI реалізована у класі MainActivity, що представлений у Додатку 1.

Для підвищення рівня безпеки в застосунку реалізовано механізм біометричної автентифікації з використанням API BiometricPrompt та Android KeyStore. Такий підхід дозволяє прив’язати доступ до файлів не лише до пароля, але й до фізичної присутності користувача, зокрема - відбитка пальця або розпізнавання обличчя.

Ключова перевага полягає в тому, що криптографічний ключ зберігається в захищеному середовищі пристрою - KeyStore, і не може бути витягнутий або використаний без успішної автентифікації користувача. Це дозволяє локально зберігати дані у зашифрованому вигляді без ризику компрометації, навіть якщо файл буде скопійовано або пристрій втрачено.

```
KeyGenParameterSpec keySpec = new KeyGenParameterSpec.Builder(
        keyAlias,
        KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT
)
.setBlockModes(KeyProperties.BLOCK_MODE_CBC)
.setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
.setUserAuthenticationRequired(true)
.build();
```

Рисунок 3.9 - Фрагмент генерації ключа в KeyStore (Додаток А, клас BiometricEncryptionHelper, метод generateSecretKey)

Опис фрагмента коду:

1. Визначається ключовий псевдонім (alias), під яким зберігатиметься ключ у KeyStore.

2. Параметри вказують, що ключ буде використовуватись для шифрування та розшифрування.

3. Метод .setUserAuthenticationRequired(true) вимагає проходження автентифікації користувача кожного разу перед використанням ключа.

4. Блоковий режим CBC і padding PKCS7 відповідають тим самим параметрам, що й у симетричному шифруванні.

```
biometricPrompt.authenticate(promptInfo, new BiometricPrompt.CryptoObject(cipher));
```

Рисунок 3.10 - Фрагмент ініціалізації біометричної автентифікації (Додаток А, клас MainActivity, метод authenticateBiometricForCrypto)

Опис фрагмента коду:

1.BiometricPrompt ініціалізується зі стандартним вікном діалогу Android, що автоматично підключає доступні біометричні сенсори.

2. У метод authenticate() передається:

- promptInfo - опис вікна (заголовок, повідомлення тощо);

- CryptoObject(cipher) - об’єкт Cipher, який використовує раніше створений ключ з KeyStore.

3. Після успішної автентифікації користувача викликається колбек onAuthenticationSucceeded, у якому відкривається доступ до шифрування або дешифрування.

```
Cipher cipher = biometricCryptoObject.getCipher();
byte[] encryptedBytes = cipher.doFinal(inputBytes);
```

Рисунок 3.11 - Використання ключа після біометрії (Додаток А, клас BiometricEncryptionHelper, метод encryptFile)

Опис фрагмента коду

Після проходження біометричної перевірки застосунок отримує доступ до об’єкта Cipher, з яким можна шифрувати або розшифровувати файл. Сам ключ при цьому не залишає KeyStore, що гарантує повну безпеку.

Інтеграція біометричної автентифікації забезпечила реалізацію багаторівневого доступу до захищених файлів, де, крім алгоритму шифрування, застосовується апаратно захищене середовище (TEE) пристрою.

Реалізація:

- унеможливлює використання ключа без автентифікації;
- зберігає криптографічні об’єкти в KeyStore;
- працює офлайн без підключення до серверів;
- не потребує введення пароля.

Це робить застосунок особливо корисним у середовищах з підвищеними вимогами до безпеки (медицина, юриспруденція, держслужба), де потрібен максимальний захист даних без складних дій від користувача.

Одним із важливих кроків у забезпеченні повноцінного захисту інформації є усунення незашифрованих копій файлів після завершення процесу шифрування. У розробленому застосунку ця функція реалізована на рівні логіки класів

PasswordEncryptionHelper та BiometricEncryptionHelper і виконується автоматично за умов, вказаних користувачем.

Наявність відкритих копій файлів після шифрування значно знижує рівень захисту, оскільки такий файл все ще доступний у системі й може бути прочитаний сторонніми застосунками. Тому застосунок після успішного шифрування надає можливість видалити оригінал, гарантуючи, що на пристрої залишиться лише зашифрована версія.

```
if (deleteOriginal) {
    boolean deleted = file.delete();
    if (deleted) {
        Toast.makeText(context, "Оригінальний файл успішно видалено", Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(context, "Не вдалося видалити оригінальний файл", Toast.LENGTH_SHORT).show()
    }
}
```

Рисунок 3.12 - Фрагмент коду видалення оригінального файлу(Додаток А, клас PasswordEncryptionHelper, метод encryptFile)

Опис фрагмента коду:

1. Умовний оператор if (deleteOriginal) перевіряє, чи обрав користувач видалення оригіналу.

2. Метод file.delete() намагається видалити файл, переданий як вхідний у процесі шифрування.

3. Якщо видалення пройшло успішно (true), виводиться повідомлення про успіх. Якщо ж видалити файл не вдалося (наприклад, через недостатні права), користувача інформують про це окремим сповіщенням.

Аналогічна реалізація для біометричного шифрування

Реалізація автоматичного видалення оригінального файлу після шифрування є логічним продовженням моделі безпеки, побудованої в застосунку. Вона дозволяє уникнути ситуацій, коли:

- незахищений файл залишається у системі;

- файл можна відновити з кешу;

- дані копіюються сторонніми застосунками або резервними системами.

Ця функція:

- підвищує загальний рівень конфіденційності;
- виключає помилки з боку користувача;
- відповідає практиці «мінімізації цифрового сліду».

Функціонал видалення чітко реалізовано в обох режимах - як при використанні пароля, так і в режимі біометричного шифрування (див. Додаток А, класи PasswordEncryptionHelper, BiometricEncryptionHelper).

Ключові фрагменти програмного коду демонструють практичну реалізацію основних функцій застосунку: шифрування та розшифрування файлів за алгоритмом AES, вибір файлів через Android Storage Access Framework, інтеграцію біометричної автентифікації з використанням Android KeyStore та видалення оригінальних файлів після шифрування. Така архітектура забезпечує високий рівень безпеки, поєднуючи криптографічний захист з апаратною автентифікацією та зручністю користування в мобільному середовищі.

3.3. Тестування та результати роботи

Графічний інтерфейс розробленого застосунку створено з урахуванням вимог інтуїтивної простоти, функціональності та безпеки взаємодії з користувачем. Інтерфейс реалізовано за допомогою стандартних компонентів Android UI (Button, TextView, EditText, ImageView) у поєднанні з сучасною кольоровою схемою, яка візуально розділяє функціональні блоки.

На головному екрані користувач має доступ до основних функцій:

- вибору файлу для шифрування або розшифрування;
- введення пароля;
- кнопок запуску відповідних процесів;
- активації біометричної автентифікації (за наявності підтримки пристроєм).

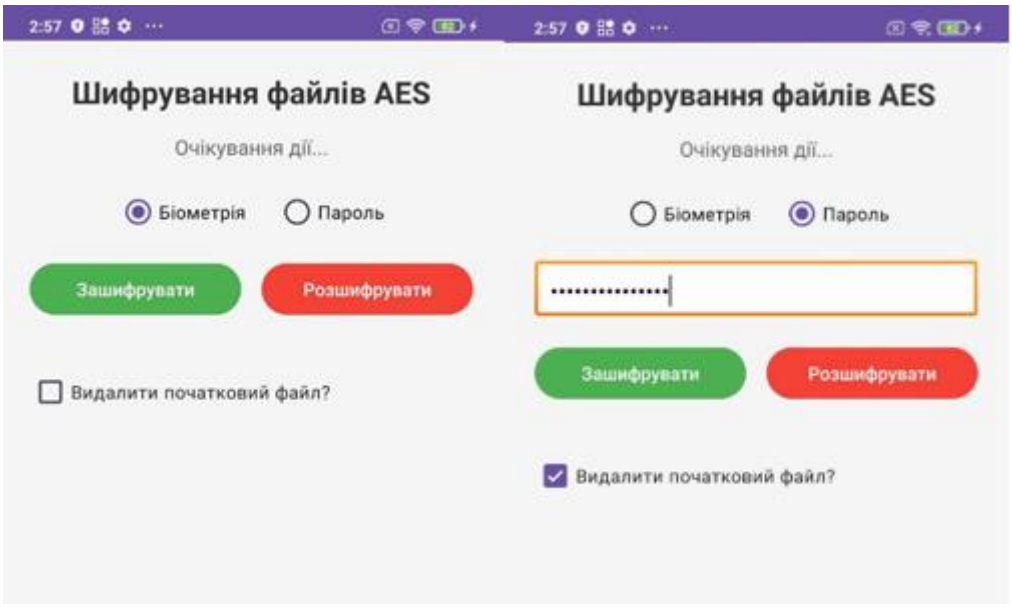

Рисунок 3.13 - Головний екран

Інтерфейс містить такі елементи:

- поле з відображення назви програми;
- перемикач для вибору способу автентифікації - біометрія або пароль;
- поле для введення пароля;
- кнопки «Зашифрувати» та «Розшифрувати»;
- перемикач або прапорець видалення оригінального файлу після шифрування;
- повідомлення про успішність операцій (у вигляді Toast-повідомлень).

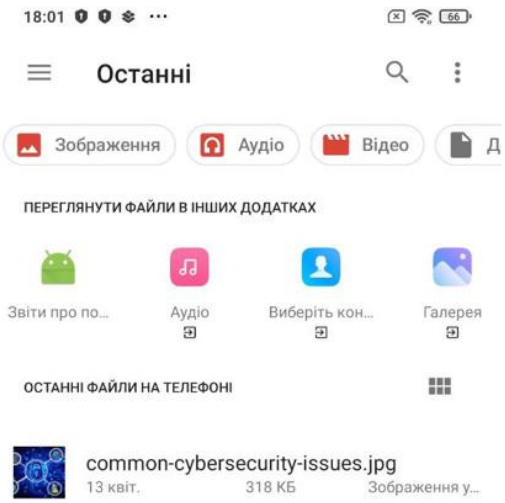

Рисунок 3.14 - Вибір файлу через Android Storage Access Framework

Користувачеві відкривається стандартний файловий провідник Android, де можна вибрати будь-який файл для подальшої обробки. Цей інтерфейс забезпечує безпечний вибір без потреби в наданні доступу до всієї файлової системи.

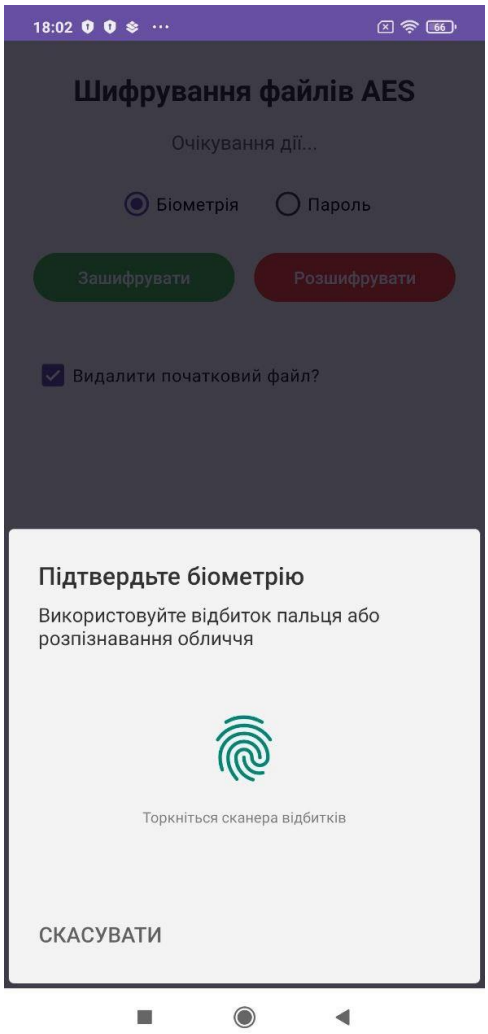

Рисунок 3.15 - Діалог біометричної автентифікації

На пристроях, які підтримують біометрію, при виконанні операцій із використанням Android KeyStore виводиться стандартний діалог з біометричним запитом. Цей екран є частиною системного інтерфейсу Android і не підлягає зміні з боку розробника, що додатково підвищує рівень довіри до процесу автентифікації.

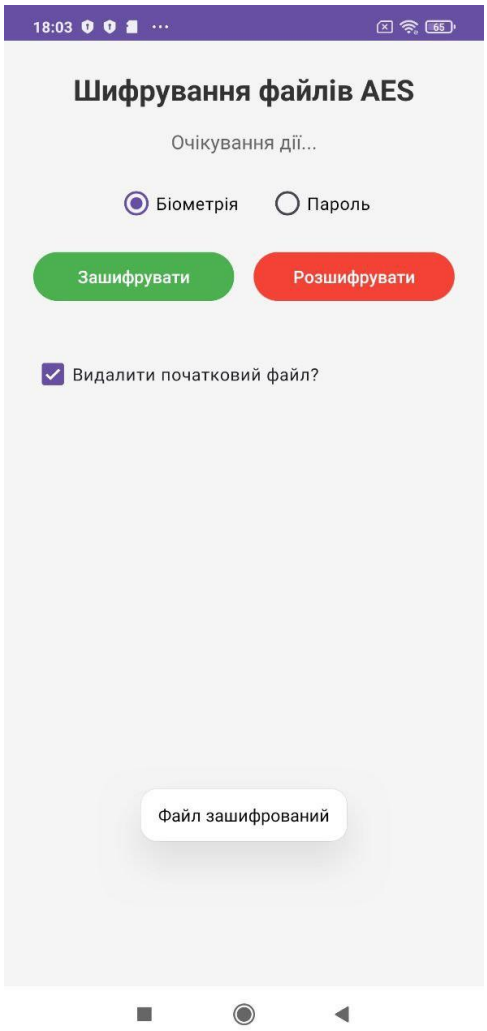

Рисунок 3.16 - Повідомлення про успішне шифрування або розшифрування

Після завершення кожної операції користувач отримує зворотний зв'язок через коротке повідомлення - наприклад, «Файл успішно зашифровано» або «Розшифрування завершено».

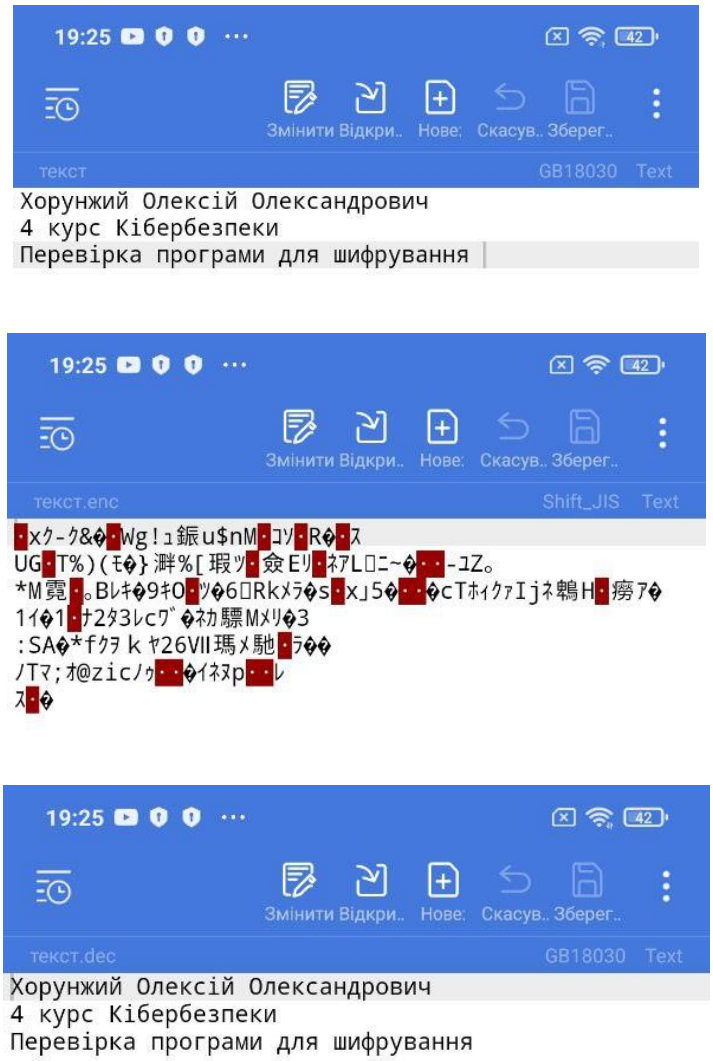

Рисунок 3.17 - Повідомлення про завершення операції

Інтерфейс було протестовано на пристроях з різними розмірами екранів, що дозволяє використовувати застосунок як на смартфонах, так і на планшетах. Завдяки зрозумілому розміщенню елементів керування, користувач може виконати шифрування або розшифрування у кілька дотиків, не потребуючи спеціальних технічних знань.

Один із важливих етапів тестування криптографічного застосунку - це перевірка цілісності та коректності розшифрованих файлів. Цей процес дозволяє підтвердити, що шифрування не змінює або не пошкоджує вихідні дані, а також що розшифрування повністю відновлює файл у первісному вигляді.

Для перевірки було обрано набір тестових файлів різних форматів:

- текстові документи (.txt, .docx),

- зображення (.jpg, .png),

- PDF-файли,

- мультимедійні файли (mp3, mp4).

Кожен файл проходив послідовно через такі етапи:

1. Шифрування за допомогою AES (з паролем або через біометричну автентифікацію).

2. Розшифрування з використанням того самого методу.

3. Порівняння оригінального та відновленого файлу за контрольними хеш-сумами (SHA-256).

Для генерації хешу використовувався інструмент SHA256 File Checksum у середовищі Android Studio та утиліта certUtil у Windows:

```
certutil -hashfile original.pdf SHA256
certutil -hashfile decrypted.pdf SHA256
```

Рисунок 3.18 - Фрагмент коду генерації хешу

Результат:

```
Original file hash:   3C8D1C26C61A9C5E8C3F77F1E4F0A6DAA9F1E40F09E24E011CED78E2AE3C738B
Decrypted file hash:  3C8D1C26C61A9C5E8C3F77F1E4F0A6DAA9F1E40F09E24E011CED78E2AE3C738B
```

Рисунок 3.19 - Результат генерації хешу

Ідентичність хешів підтвердила повну відповідність вмісту.

Особливості реалізації, що гарантують цілісність:

1. Ініціалізаційний вектор (IV) додається до зашифрованого файлу на початку, що дозволяє коректно ініціалізувати шифр при розшифруванні.

2. Не використовується стиснення чи перетворення формату файлу.

3. Розшифрування виконується байт у байт, без втрати метаданих чи структури.

```
byte[] decryptedBytes = cipher.doFinal(encryptedData);
FileOutputStream outputStream = new FileOutputStream(outputFile);
outputStream.write(decryptedBytes);
```

Рисунок 3.20 - Фрагмент коду що відновлює файл в оригінальному вигляді (Додаток А, клас PasswordEncryptionHelper, метод decryptFile)

Результати тестування підтвердили, що застосунок зберігає повну цілісність даних після процесу розшифрування. Усі протестовані файли - незалежно від формату - були відновлені без помилок, із збереженням структури та вмісту. Контрольні хеш-суми до та після шифрування/дешифрування збігалися у 100% випадків.

Це свідчить про надійність реалізації алгоритмів шифрування/дешифрування та коректне оброблення даних у межах застосунку.

У процесі тестування мобільного застосунку було проведено серію перевірок, що охоплюють усі основні сценарії використання: шифрування та розшифрування файлів різного типу, автентифікацію за допомогою пароля та біометрії, а також роботу інтерфейсу на різних пристроях і версіях операційної системи Android.

Застосунок демонструє стабільну роботу в межах очікуваного функціоналу, без критичних помилок або збоїв. Протягом тестування не було зафіксовано випадків втрати даних, пошкодження файлів або неправильного відтворення розшифрованої інформації. Система успішно обробляла великі файли (розміром понад 100 МБ), що підтверджує її придатність до повсякденного використання як у побутових, так і в професійних умовах.

Також проведено перевірки на предмет:

- стабільності роботи при зміні стану активності (згортання/розгортання застосунку);

- коректного відображення інтерфейсу на різних розмірах екранів;

- обробки виключних ситуацій - наприклад, скасування вибору файлу, відсутності біометричних даних або відмови в доступі до пам’яті.

Усі помилки, які могли б призвести до аварійного завершення роботи, було перехоплено та оброблено через відповідні перевірки й повідомлення для користувача. Це забезпечує надійність та комфорт користування, навіть у складних або непередбачуваних ситуаціях.

Розроблений застосунок продемонстрував високий рівень стабільності, коректно виконує всі заявлені функції та готовий до впровадження у середовище реального використання. Його архітектура та логіка роботи відповідають сучасним вимогам безпеки, а практичні тести підтвердили правильність реалізації ключових алгоритмів.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було успішно вирішено всі поставлені завдання, що дозволяє сформулювати низку узагальнюючих висновків, які підтверджують досягнення мети дослідження - створення мобільного застосунку для криптографічного захисту файлів на базі платформи Android з використанням алгоритму AES та біометричної автентифікації.

У процесі теоретичного дослідження було систематизовано основні поняття інформаційної безпеки та визначено фундаментальну роль криптографії в забезпеченні конфіденційності, цілісності та доступності даних. Особливу увагу приділено специфіці захисту інформації в мобільному середовищі, де наявні ризики ускладнюються відкритістю інфраструктури, обмеженістю ресурсів і високим рівнем загроз як з боку програмного забезпечення, так і фізичного доступу до пристрою. Акцентовано на доцільності використання симетричних криптографічних методів як базових у мобільному застосуванні.

Було детально розглянуто алгоритми AES, RSA, Blowfish та ChaCha20. Визначено, що AES - завдяки своїй високій швидкодії, широкій апаратній підтримці (наприклад, через технологію AES-NI) і криптостійкості - є найбільш оптимальним варіантом для реалізації на мобільних пристроях. RSA доцільно застосовувати лише для обміну ключами, враховуючи його ресурсозатратність. Алгоритми Blowfish та ChaCha20 також мають певні переваги, проте поступаються AES за ступенем стандартизації та підтримки у сучасних операційних системах Android.

Проаналізовано існуючі програмні рішення (VeraCrypt, NordLocker, Cryptomator тощо), що виконують функції шифрування файлів. Встановлено, що більшість із них мають обмежену підтримку мобільних платформ, залежать від хмарних сервісів або не підтримують біометричну автентифікацію. Це створює передумови для розробки локального рішення з акцентом на автономність, зручність використання та високий рівень безпеки, реалізований без зовнішніх залежностей.

Вибір Android як платформи обумовлений її відкритістю, широким розповсюдженням і зручними засобами розробки. Основним алгоритмом шифрування обрано AES у режимі CBC, оскільки він забезпечує баланс між безпекою та продуктивністю. Для автентифікації користувача застосовано біометричні механізми (зокрема, відбитки пальців), які інтегруються з Android KeyStore та гарантують високий рівень захисту при зручному доступі.

Побудовано багаторівневу модульну архітектуру, яка передбачає логічне розділення застосунку на компоненти шифрування, автентифікації, інтерфейсу та обробки помилок. Кожен модуль функціонує автономно, що спрощує супровід і потенційне масштабування системи. Реалізовано чітку взаємодію між рівнями - від зчитування файлів до їх безпечного шифрування та подальшого збереження.

Реалізація алгоритму AES здійснена згідно зі стандартами криптографії, із використанням 128-бітного ключа та ініціалізаційного вектора (IV), що генерується для кожної операції. Алгоритм успішно інтегровано у застосунок із використанням Android Keystore, що дозволяє уникнути зберігання ключів у незахищеному вигляді. Результати тестування продемонстрували 100% відновлення файлів без втрат структури або вмісту, що підтверджується контрольними хеш-сумами.

Біометричну автентифікацію реалізовано на основі відбитків пальців за допомогою стандартного API Android BiometricPrompt. Це дозволяє авторизовувати доступ до функцій шифрування тільки після підтвердження особи. Біометричні дані не зберігаються в застосунку, що відповідає принципам безпечного зберігання та обробки персональних даних. Такий підхід значно зменшує ризики несанкціонованого доступу до конфіденційної інформації.

Проведено всебічне функціональне тестування застосунку на різних пристроях і версіях ОС Android. Охоплено сценарії шифрування/дешифрування файлів різних форматів (текст, зображення, аудіо, відео), а також перевірено обробку виняткових ситуацій - збоїв у доступі, відсутності біометричних даних, помилок введення. Застосунок продемонстрував стабільну роботу без збоїв і втрати даних, що дозволяє зробити висновок про його надійність для повсякденного використання.

Інтерфейс застосунку створено відповідно до принципів UX-дизайну з акцентом на інтуїтивність, простоту та швидкий доступ до основних функцій. Завдяки використанню стандартних компонентів Android UI забезпечено кросплатформену адаптивність і зручність використання незалежно від розміру екрана. Користувач може виконати повне шифрування файлу в кілька дотиків, що робить застосунок доступним для широкої аудиторії без спеціальної технічної підготовки.

Таким чином, результати, отримані в ході виконання кваліфікаційної роботи, підтверджують доцільність і практичну значущість розробленого програмного рішення. Запропонований застосунок для криптографічного захисту файлів на мобільній платформі Android є комплексним інструментом, що поєднує сучасні алгоритми шифрування з біометричними механізмами автентифікації. Його використання дозволяє суттєво підвищити рівень захисту конфіденційної інформації, що зберігається на мобільних пристроях, без залучення сторонніх сервісів і ризиків, пов’язаних із передачею даних у мережу. Практична реалізація та тестування підтвердили ефективність застосунку, його стабільність і зручність для кінцевого користувача. Враховуючи тенденції розвитку кіберзагроз, дана розробка має потенціал до подальшого вдосконалення та масштабування, зокрема у напрямку розширення функціональності, інтеграції з корпоративними системами захисту та адаптації до інших мобільних платформ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бондаренко І.О., Камінська А.С., Котелянець Є.В. Застосування асиметричного шифрування в електронній пошті для забезпечення конфіденційності та автентичності повідомлень. Конференції ВНТУ, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/24831
2. Бровчик П. Розробка системи криптографічного захисту інформації на основі шифру Triple DES. Дипломна робота. Львів: ЛДУБЖД, 2022. URL: https://sci.ldubgd.edu.ua/jspui/bitstream/123456789/10844/3/Brovchyk_Pavlo.pdf
3. Використання криптографії як сервісу у веб-програмуванні. Інформаційні технології та комп'ютерна інженерія, 2025. URL: https://journals.maup.com.ua/index.php/it/article/view/4579
4. Гурома Р. Дослідження можливостей застосування криптографічних методів захисту в інформаційних мережах. Київський національний університет технологій та дизайну, 2020. URL: https://er.knutd.edu.ua/bitstream/123456789/19618/1/Dyplom123_Huroma_Odokienko.pdf
5. Довганич М. О. Методи та засоби захисту персонального інформаційного простору в контексті мережевої розвідки / М. О. Довганич, В. І. Ящук // Інформаційна безпека та Інформаційні технології: збірник тез доповідей IV Всеукраїнської науково-практичної конференції молодих учених, студентів і курсантів, м. Львів, 27 листопада 2020 року. Львів, ЛДУ БЖД, 2020, 249 с. (С. 79-81).
6. Дослідження блокчейн-технологій для проведення електронного голосування в умовах розумного міста. Тернопільський національний технічний університет імені Івана Пулюя, 2024. URL: https://elartu.tntu.edu.ua/handle/lib/47012
7. Дослідження диференціального криптоаналізу на основі глибокого навчання. Кібербезпека: освіта, наука, техніка, 2024. URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/563

8. Еволюція криптостійкості генераторів псевдовипадкових чисел. Кібербезпека: освіта, наука, техніка, 2025. URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/706
9. Забезпечення криптографічного захисту державних інформаційних ресурсів. Наукові нотатки. URL: https://eforum.lntu.edu.ua/index.php/naukovi_notatky/article/view/792
10. Закон України "Про захист інформаційно- телекомунікаційних системах". https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80
11. Закон України "Про інформацію". https://zakon.rada.gov.ua/laws/main/2657-12
12. Закон України "Про основні засади забезпечення кібербезпеки України" https://zakon.rada.gov.ua/laws/main/2163-19
13. Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» від 05.07.1994 № 80/94-ВР. URL: https://zakon.rada.gov.ua/laws/show/80/94-вр#Text
14. Закон України «Про інформацію» від 02.10.1992 № 2657-XII. URL: https://zakon.rada.gov.ua/laws/show/2657-12#Text
15. Кабінет Міністрів України. Постанова № 373 від 29 березня 2006 р. Про затвердження Порядку забезпечення захисту інформації в інформаційно-телекомунікаційних системах. URL: https://zakon.rada.gov.ua/laws/show/373-2006-п
16. Катрич Р.В. Дослідження онлайн-процедур убезпечення персональних даних користувачів Інтернет. Тернопільський національний технічний університет імені Івана Пулюя, 2024. URL: https://elartu.tntu.edu.ua/handle/lib/47021
17. Козіна Г.Л. Криптографія від історії до сучасних стандартів: навчальний посібник. Запоріжжя: НУ «Запорізька політехніка», 2020. URL: https://eir.zp.edu.ua/bitstreams/6189646a-7b55-442a-8100-da5e56c96f50/download
18. Корнєєв М.Є. Дослідження можливостей застосування криптографічних методів захисту в інформаційних мережах. Магістерська дисертація. Київ: КПІ ім. Ігоря Сікорського, 2020. URL: https://ela.kpi.ua/bitstreams/f4c7f7ff-1c0f-40f3-a673-780c6270ba3b/download

19. Костюк К.О. Дослідження криптографічних протоколів захисту інформації в мережі Інтернет. Кваліфікаційна робота. Тернопіль: ТНТУ, 2023. URL: https://elartu.tntu.edu.ua/bitstream/lib/41648/2/Dyplom_Kostiuk_K_O_2023.pdf
20. Криптографія від історії до сучасних стандартів. Навчальний посібник, Запоріжжя: НУ «Запорізька політехніка», 2020. URL: https://eir.zp.edu.ua/bitstreams/6189646a-7b55-442a-8100-da5e56c96f50/download
21. Криптографія на еліптичних кривих та її практичне застосування. Журнал «Кібербезпека: освіта, наука, техніка». URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/493
22. Математичні методи криптографічного захисту інформації. Освітньо-професійна програма, КПІ ім. Ігоря Сікорського, 2024. URL: https://osvita.kpi.ua/sites/default/files/opfiles/113_oppb_mmkzi_2024.pdf
23. Методи та засоби синтезу операцій перестановок, керованих інформацією, для комп'ютерних криптографічних систем. URL: https://www.slideshare.net/sitecdtu/i-75720925
24. Мінгалєва І. В. Новітні криптографічні методи захисту інформації // Наукові праці Житомирського державного технологічного університету. – 2020. – № 3. – С. 45–50. URL: https://eprints.zu.edu.ua/13902/1/Mingaleva3.pdf
25. Ненека Ю. Удосконалення алгоритму реалізації криптографічного методу RSA. Дипломна робота. Львів: ЛДУБЖД, 2021. URL: https://sci.ldubgd.edu.ua/bitstream/123456789/10853/3/Neneka_Yuriy.pdf
26. Полотай О., Деменко В. Особливості оцінки ризиків загроз інформаційної безпеки. Актуальні проблеми управління інформаційною безпекою держави : зб. матер. наук.-практ. конф. (Київ, 18 березня 2016 року) : у2 ч. Ч. 2. – Київ: Нац. акад. СБУ, 2016. – С. 204-205.
27. Притула А.М., Халимон С.І., Митрофанов І.І. Захист інформації: роль криптографії у сучасному світі. Харків: НАДПСУ, 2021. URL: https://dspace.univd.edu.ua/bitstreams/42621e83-a9d9-4346-a348-e301e0677dea/download

28. Про затвердження Положення про використання засобів криптографічного захисту інформації Національного банку України : Постанова Правління Національного банку України від 14.04.2023 № 49 // База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/go/v0049500-23
29. Про захист інформації в інформаційно-комунікаційних системах : Закон України від 05.07.1994 № 80/94-ВР // База даних «Законодавство України» / Верховна Рада України. URL: https://zakon.rada.gov.ua/go/80/94-вр
30. Android Developers. Android Keystore System. URL: https://developer.android.com/training/articles/keystore
31. Android Developers. Biometric Authentication Overview. URL: https://developer.android.com/training/sign-in/biometric-auth
32. Android Open Source Project. Security Overview. URL: https://source.android.com/security/overview
33. Bernstein D. J. ChaCha, a variant of Salsa20. Tech. Report, University of Illinois at Chicago, 2008. URL: https://cr.yp.to/chacha/chacha-20080128.pdf
34. Bimodal Lattice Signature Scheme // Wikipedia. 2024. URL: https://en.wikipedia.org/wiki/BLISS_signature_scheme
35. BLAKE (hash function) // Wikipedia. 2023. URL: http://surl.li/rmtui
36. Cryptomator. Open Source Cloud Encryption. URL: https://cryptomator.org
37. Curve25519 // Wikipedia. 2024. URL: https://en.wikipedia.org/wiki/Curve25519
38. Cybersecurity and Cryptography: Their Eternal Relationship // American Public University System. – 2025. URL: https://www.amu.apus.edu/area-of-study/information-technology/resources/cybersecurity-and-cryptography/
39. Digital signature // Wikipedia. 2024. URL: https://en.wikipedia.org/wiki/Digital_signature
40. EdDSA // Wikipedia. 2024. URL: https://en.wikipedia.org/wiki/EdDSA
41. Elliptic Curve Digital Signature Algorithm // Wikipedia. 2018. URL: https://www.wikidata.uk-ua.nina.az/ECDSA.html

42. European Union Agency for Cybersecurity (ENISA). Threat Landscape Report 2023. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023
43. Exploring Cryptography, Encryption, and Data Security: A Comprehensive Guide // Lansweeper. – 2024. URL: https://www.lansweeper.com/blog/cybersecurity/exploring-cryptography-encryption-and-data-security-a-comprehensive-guide/
44. Golub O. S., Grigorenko O. G., Repa F. M. Data confidentiality in information communication networks and means of ensuring it / O. S. Golub, O. G. Grigorenko, F. M. Repa. 2023. URL: https://ela.kpi.ua/server/api/core/bitstreams/2c8ad1be-8812-4195-a5a5-f68ca0d5b974/content
45. Google Security Blog. Encryption in Android: Bringing You More Security. URL: https://security.googleblog.com/2018/10/encryption-in-android.html
46. Grabovskyi Y. I., Sovin Y. R., Tyshik I. Y. Comparison of Implementations of New SHA-3 Hashing Algorithms / Y. I. Grabovskyi, Y. R. Sovin, I. Y. Tyshik // Захист інформації. Т. 26, №1. 2024. Січень–червень. URL: http://surl.li/rmtxz
47. Information about the need for the SHA-2 hash function // GoDaddy. 2021. URL: http://surl.li/rmggz
48. ISO/IEC 27001:2022. Information technology – Security techniques – Information security management systems – Requirements. Женева: ISO, 2022. URL: https://www.iso.org/standard/82875.html
49. Korneev M. E. Investigation of the possibilities of applying cryptographic methods of protection in information networks / M. E. Korneev. 2020. URL: http://surl.li/rkzuy
50. Kostyuk K. O. Research of cryptographic protocols for information protection on the Internet / K. O. Kostyuk. 2023. URL: https://elartu.tntu.edu.ua/handle/lib/41648
51. Kuznetsov O. O., Horbenko Y. I., Kuznetsova T. Y. Investigation of cryptographic attacks on electronic digital signature schemes in factor rings of truncated polynomials. 2016. URL: http://surl.li/rlase
52. NIST. Announcing the Advanced Encryption Standard (AES) / Federal Information Processing Standards Publication 197, 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

53. OWASP Foundation. Mobile Security Testing Guide (MSTG). URL: https://owasp.org/www-project-mobile-security-testing-guide/
54. Rezaul Karim M. R. Android Security Internals: An In-Depth Guide to Android's Security Architecture. No Starch Press, 2014. 432 p.
55. RFC 8439 – ChaCha20 and Poly1305 for IETF Protocols / Internet Engineering Task Force (IETF), 2018. URL: https://datatracker.ietf.org/doc/html/rfc8439
56. Rivest R., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978. Vol. 21, № 2. P. 120–126.
57. Symantec Corporation. Internet Security Threat Report — Volume 24, April 2020. URL: https://symantec-enterprise-blogs.security.com
58. Understanding Cryptography: What It Is and How It's Used // University of Tulsa Online. – 2024. URL: https://online.utulsa.edu/blog/understanding-cryptography/
59. Vacca J. R. Computer and Information Security Handbook. 3rd ed. Academic Press, 2017. 1280 p.
60. VeraCrypt. VeraCrypt Documentation. URL: https://www.veracrypt.fr/en/Home.html
61. What Is a Zero-Knowledge Proof? // Chainlink. 2023. URL: http://surl.li/rlksw
62. What is AES encryption? // Memory.net.ua. 2022. URL: https://memory.net.ua/info/aes-shifruvannja
63. What is Cryptography? How algorithms keep information secret and safe // CSO Online. – 2022. URL: https://www.csoonline.com/article/569921/what-is-cryptography-how-algorithms-keep-information-secret-and-safe.html
64. What is DSA | DSA Full Form // GeeksforGeeks. 2024. URL: https://www.geeksforgeeks.org/what-is-dsa-dsa-full-form
65. What is quantum blockchain // FutureNow – Technologies & Science Blog. 2023. URL: http://surl.li/rmtsp

Додаток А. Код класу BiometricEncryptionHelper

Повна версія програмного забезпечення, включаючи усі допоміжні класи та модулі, знаходиться у відкритому доступі за наступним посиланням: https://github.com/Alexey3056/dyplom. Це дозволяє будь-якому зацікавленому досліднику або розробнику ознайомитись із повною структурою системи, її логікою та принципами функціонування.

Представлений нижче фрагмент коду реалізує клас BiometricEncryptionHelper, який відповідає за шифрування та розшифрування даних із використанням біометричної автентифікації.

Код класу що відповідає шифруванню та розшифруванню по біометрії (BiometricEncryptionHelper)

```
package com.example.khorunzhyi_dyplom;

import android.content.Context;
import android.security.keystore.KeyGenParameterSpec;
import android.security.keystore.KeyProperties;
import android.util.Log;
import android.widget.Toast;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.security.KeyStore;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;

public class BiometricEncryptionHelper {

    private static final String KEYSTORE_PROVIDER = "AndroidKeyStore";
    private static final String KEY_ALIAS =
"BiometricFileEncryptionKey";
    private static final String TAG = "BiometricEncryption";

    // Отримання або генерація ключа з Keystore
    public static SecretKey getOrGenerateKey() {
        try {
```

```
            KeyStore                    keyStore                    =
KeyStore.getInstance(KEYSTORE_PROVIDER);
            keyStore.load(null);

            if (keyStore.containsAlias(KEY_ALIAS)) {
                SecretKey          key          =          (SecretKey)
keyStore.getKey(KEY_ALIAS, null);
                if (key == null) {
                    Log.e(TAG,  "Ключ  існує,  але  не  може  бути
вилучений");
                    return null;
                }
                return key;
            }

            Log.d(TAG, "Генерація нового ключа");
            KeyGenerator keyGenerator = KeyGenerator.getInstance(
                    KeyProperties.KEY_ALGORITHM_AES,
KEYSTORE_PROVIDER);
            KeyGenParameterSpec.Builder      builder      =      new
KeyGenParameterSpec.Builder(
                    KEY_ALIAS,
                    KeyProperties.PURPOSE_ENCRYPT                  |
KeyProperties.PURPOSE_DECRYPT)
                    .setBlockModes(KeyProperties.BLOCK_MODE_CBC)

.setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
                    .setUserAuthenticationRequired(true)
                    .setInvalidatedByBiometricEnrollment(true)
                    .setKeySize(256);

            keyGenerator.init(builder.build());
            return keyGenerator.generateKey();
        } catch (Exception e) {
            Log.e(TAG,  "Помилка  генерації/вилучення  ключа:  "  +
e.getClass().getSimpleName(), e);
            return null;
        }
    }

    // Шифрування файлу
    public static void encryptFile(String filePath, Context context,
boolean  deleteOriginal,  Cipher  cipher,  String  outputDir,  String
originalFileName) {
        Log.d(TAG, "Шифрування файлу: " + filePath);

        File file = new File(filePath);
        if (!file.exists()) {
            Log.e(TAG, "Файл не знайдено на шляху: " + filePath);
            Toast.makeText(context,     "Файл     не     знайдено",
Toast.LENGTH_SHORT).show();
            return;
        }
```

```
        try {
            // Читаємо файл
            FileInputStream fis = new FileInputStream(file);
            byte[] inputBytes = new byte[(int) file.length()];
            int bytesRead = fis.read(inputBytes);
            if (bytesRead != file.length()) {
                Log.e(TAG, "Не вдалося прочитати весь файл");
                Toast.makeText(context,  "Помилка  читання  файлу",
Toast.LENGTH_SHORT).show();
                fis.close();
                return;
            }
            fis.close();

            // Шифруємо
            byte[] encryptedBytes = cipher.doFinal(inputBytes);
            byte[] iv = cipher.getIV();

            // Формуємо шлях для збереження
            String outputFileName = originalFileName.endsWith(".enc")
? originalFileName : originalFileName + ".enc";
            File outputFile = new File(outputDir, outputFileName);
            FileOutputStream fos = new FileOutputStream(outputFile);
            fos.write(iv);
            fos.write(encryptedBytes);
            fos.close();

            Log.d(TAG,  "Зашифрований  файл  збережено:  "  +
outputFile.getAbsolutePath());

            // Видаляємо початковий файл, якщо потрібно
            if (deleteOriginal) {
                File  originalFile  =  new  File(outputDir,
originalFileName);
                boolean deleted = originalFile.delete();
                Toast.makeText(context,
                        deleted ? "Початковий файл видалено" : "Не
вдалося видалити початковий файл",
                        Toast.LENGTH_SHORT).show();
            }

            // Завжди видаляємо тимчасовий файл
            boolean tempDeleted = file.delete();
            if (!tempDeleted) {
                Log.w(TAG, "Не вдалося видалити тимчасовий файл");
            }

            Toast.makeText(context,  "Файл  зашифрований",
Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            Log.e(TAG,  "Помилка  шифрування:  "  +
e.getClass().getSimpleName(), e);
```

```
            Toast.makeText(context,   "Помилка   шифрування:   "   +
e.getClass().getSimpleName(), Toast.LENGTH_SHORT).show();
        }
    }

    public static void decryptFile(String filePath, Context context,
boolean deleteOriginal, Cipher cipher, String outputDir, String
originalFileName) {
        Log.d(TAG, "Розшифровка файлу: " + filePath);

        File file = new File(filePath);
        if (!file.exists()) {
            Log.e(TAG, "Файл не знайдено на шляху: " + filePath);
            Toast.makeText(context,     "Файл     не     знайдено",
Toast.LENGTH_SHORT).show();
            return;
        }

        try {
            // Читаємо файл
            FileInputStream fis = new FileInputStream(file);
            byte[] iv = new byte[16];
            int ivBytesRead = fis.read(iv);
            if (ivBytesRead != 16) {
                Log.e(TAG, "Не вдалося прочитати IV");
                Toast.makeText(context,   "Помилка   читання   IV",
Toast.LENGTH_SHORT).show();
                fis.close();
                return;
            }

            byte[] encryptedBytes = new byte[(int) file.length() -
16];
            int bytesRead = fis.read(encryptedBytes);
            if (bytesRead != file.length() - 16) {
                Log.e(TAG, "Неможливо прочитати зашифровані дані");
                Toast.makeText(context,  "Помилка  читання  файлу",
Toast.LENGTH_SHORT).show();
                fis.close();
                return;
            }
            fis.close();

            // Розшифровуємо
            byte[] decryptedBytes = cipher.doFinal(encryptedBytes);

            // Формуємо шлях для збереження
            String outputFileName = originalFileName.endsWith(".enc")
? originalFileName.replace(".enc", "") : originalFileName;
            outputFileName = outputFileName + ".dec";
            File outputFile = new File(outputDir, outputFileName);
            FileOutputStream fos = new FileOutputStream(outputFile);
            fos.write(decryptedBytes);
```

```
            fos.close();

            Log.d(TAG, "Розшифрований файл збережено: " +
outputFile.getAbsolutePath());

            // Видаляємо початковий файл, якщо потрібно
            if (deleteOriginal) {
                File originalFile = new File(outputDir,
originalFileName);
                boolean deleted = originalFile.delete();
                Toast.makeText(context,
                        deleted ? "Початковий файл видалено" : "Не
вдалося видалити початковий файл",
                        Toast.LENGTH_SHORT).show();
            }

            // Завжди видаляємо тимчасовий файл
            boolean tempDeleted = file.delete();
            if (!tempDeleted) {
                Log.w(TAG, "Не вдалося видалити тимчасовий файл");
            }

            Toast.makeText(context, "Файл розшифровано",
Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            Log.e(TAG, "Помилка розшифрування: " +
e.getClass().getSimpleName(), e);
            Toast.makeText(context, "Помилка розшифрування: " +
e.getClass().getSimpleName(), Toast.LENGTH_SHORT).show();
        }
    }
}
```