Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

Магістерська кваліфікаційна робота на тему:
«Метод та засіб автентифікації застосунків в операційній системі Android»

Виконав: студент групи 1БС-23м
спеціальності 125 Кібербезпека
_____ Владислав ШВЕЦЬ

Керівник: к. т. н., доцент, доцент каф. ЗІ
_____ Володимир ГАРНАГА
«16» 12 2024 р.

Опонент: к. т. н., доцент, доцент каф. ПЗ
_____ Олександр ХОШАБА
«16» 12 2024 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., професор
_____ ЛУЖЕЦЬКИЙ В. А.
«16» 12 2024 р.

Вінниця ВНТУ – 2024 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти магістр
Галузь знань – «Інформаційні технології»
Спеціальність – «Кібербезпека та захист інформації»
Освітня програма – «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., професор
_____Володимир ЛУЖЕЦЬКИЙ
«18» 09 2024 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Швецю Владиславу Вікторовичу

1. Тема роботи: «Метод та засіб автентифікації застосунків в операційній системі Android»
   керівник роботи: Гарнага Володимир Анатолійович, к. т. н., доцент, доцент кафедри ЗІ,
   затверджено наказом ректора ВНТУ від 17 вересня 2024 року №310.

2. Строк подання студентом роботи 16 грудня 2024 р.

3. Вихідні дані до роботи:
   - види автентифікації – через пароль та цифровий підпис;
   - тип шифрування – симетричного шифрування AES;
   - генерація ключової пари – ECDSA на основі кривої SECP256k1;

4. Зміст текстової частини: Вступ. 1. Поняття про автентифікацію. 2. Класифікація і аналіз методів шифрування. 3.Розробка програмного забезпечення для покращення автентифікації. 4. Економічна частина. Висновки. Список використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу: аналіз засобів захисту автентифікації, методи захисту інформації за допомогою шифрування, узагальнений алгоритм роботи засобу, алгоритм захисту шифрування та передачі шифрованих повідомлень, алгоритм генерації ключової пари, порівняльний аналіз методів розробки, результати тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Гарнага В. А., к. т. н., доц. каф. ЗІ	16.09.24	22.09.24
2	Гарнага В. А., к. т. н., доц. каф. ЗІ	14.09.24	04.10.24
3	Гарнага В. А., к. т. н., доц. каф. ЗІ	08.10.24	30.10.24
4	Ратушняк О.Г., к. т. н., доц. каф. ЕПВМ	11.11.24	17.11.24

7. Дата видачі завдання 1 вересня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістрерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	4.09.24 – 15.09.24	
2	Аналіз джерел за напрямком бакалаврської дипломної роботи	16.09.24 – 22.09.24	
3	Розробка рішень, моделей, алгоритмів	14.09.24 – 4.10.24	
4	Практична реалізація, моделювання, експериментування, результати	08.10.24 – 30.10.24	
5	Аналіз виконання ТЗ, висновки	01.11.24 – 15.11.24	
6	Оформлення пояснювальної записки	16.11.24 – 22.11.24	
7	Попередній захист та доопрацювання МКР	29.11.24 – 11.12.24	
8	Представлення МКР на захист	12.12.24 – 16.12.24	
9	Захист МКР	18.12.24 – 20.12.24	

Студент _____ Владислав ШВЕЦЬ

Керівник роботи _____ Володимир ГАРНАГА

АНОТАЦІЯ

Магістерська дипломна робота складається з 80 сторінок формату А4, на яких є 32 рисунків, 9 таблиці, список використаних джерел містить 11 найменувань.

У даній роботі було проведено порівняльний аналіз методів автентифікації, аналіз методів шифрування, сформовано вимоги для розробки програмного засобу. Розроблено узагальнену архітектуру засобу. Розроблено алгоритм роботи засобу для двох видів автентифікації. Розроблено програмний застосунок в операційній системі Android з інтерфейсом, з функцією розшифрування повідомлення для різних довжин на базі Xamarin для клієнтької сторони, додатково був розроблений сервер для побудови API, який обробляє запити від клієнтського додатку, побудовано графіки для порівняння різниці автентифікації.

Ключові слова: шифрування, розшифрування, підпис повідомлення, кросплатформенність, сервер, передача підписаних повідомлень, логування.

ABSTRACT

The master's thesis consists of 80 A4 pages, containing 32 figures, 9 tables, and a reference list with 11 entries.

The study conducted a comparative analysis of authentication methods, encryption methods analysis, and formulated requirements for the development of a software tool. A generalized architecture of the tool was developed. An algorithm for the operation of the tool was created for two types of authentication. Additionally, an Android application was developed with an interface that includes a decryption function for various message lengths based on Xamarin for the client side. A server for building an API was also developed, which processes requests from the client app. Graphs for comparing authentication differences were constructed.

Keywords: encryption, decryption, message signing, cross-platform compatibility, server, transfer of signed messages, logging.

ЗМІСТ

ВСТУП 9

1 ПОНЯТТЯ ПРО АВТЕНТИФІКАЦІЮ 12

1.1 Огляд технологій автентифікації в інформаційних системах 12

1.2 Способи і методи автентифікації 13

1.4 Аналіз методів автентифікації в Android 19

1.5 Імплементація Android SDK 23

1.6 Висновки до розділу 31

2 КЛАСИФІКАЦІЯ І АНАЛІЗ МЕТОДІВ ШИФРУВАННЯ 33

2.1 Методологія захисту інформації за допомогою шифрування 33

2.2 Стандарт симетричного шифрування DES (Data Encryption Standard). 34

2.3 Стандарт симетричного шифрування 3DES (Triple DES). 36

2.4 Стандарт симетричного шифрування AES (Advanced Encryption Standard). 37

2.5 MD5 (Message-Digest Algorithm 5) хешування. 39

2.6 SHA (Secure Hash Algorithm) хешування. 41

2.7 Стандарт асиметричного шифрування RSA (Rivest-Shamir-Adleman). 42

2.7 Стандарт асиметричного шифрування ECDSA (Elliptic Curve Digital Signature Algorithm) 43

2.8 Висновки з розділу 45

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОКРАЩЕННЯ АВТЕНТИФІКАЦІЇ 47

3.1 Автентифікація за допомогою пароля 47

3.1.1 Зберігання даних користувача в базі даних на сервері 47

3.1.2 Процес автентифікації: передача зашифрованого повідомлення та його дешифрування на сервері 48

3.2 Автентифікація за допомогою цифрового підпису 49

3.2.1 Використання асиметричних ключів для підпису та перевірки підписаних повідомлень 49

3.2.2 Процес перевірки автентичності повідомлення без необхідності його дешифрування на сервері 49

3.3 Розробка додатка для тестування підходів 50

3.3.1 Вибір платформи Xamarin для кросплатформеної розробки Android додатка 51

3.3.2 Інтеграція серверної частини на Flask для обробки запитів від клієнтського додатка 51

3.3.3 Реалізація функціоналу двох видів автентифікації через пароль і цифровий підпис 52

3.3.4 Важливість кросплатформенності та гнучкості рішення 55

3.4 Експеримент 56

3.4.1 Опис проведення експерименту для вимірювання швидкодії обох підходів 56

3.4.2 Оцінка результату 57

3.5 Загальні висновки щодо доцільності застосування кожного з підходів для автентифікації Android-додатків у майбутніх проектах 63

ЕКОНОМІЧНА ЧАСТИНА 65

4.1 Технологічний аудит розробки 65

4.2 Розрахунок витрат на впровадження МКР 68

4.2.1 Витрати на оплату праці 68

4.2.2 Відрахування на соціальні заходи 70

4.2.3 Сировина та матеріали 71

4.2.4 Програмне забезпечення для наукових робіт 71

4.2.5 Амортизація обладнання, програмних засобів та приміщень . 72

4.2.6 Паливо та енергія для науково-виробних цілей 72

4.2.7 Службові відрядження 73

4.2.8 Витрати на роботи, які виконують сторонні підприємства, установи та організації 74

4.3.1 Накладні (загальновиробничі) витрати 74

4.3.2 Оцінювання важливості на наукової значимості науководослідної роботи фундаментального чи пошукового характеру 75

4.4 Висновки 76

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 80

ДОДАТКИ 81

Додаток А 82

Додаток Б ..... **Error! Bookmark not defined.**

Додаток В 91

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі, де інформаційні технології швидко розвиваються, відбувається постійна зміна у способах захисту та доступу до інформації. Важливим аспектом цих змін є системи автентифікації, що є ключовими для захисту конфіденційності даних та запобігання неавторизованому доступу.

Розвиток інформаційних технологій постійно відкриває нові можливості для підвищення рівня безпеки, втім, це також збільшує кількість потенційних загроз і ризиків. Захист інформації за допомогою сучасних алгоритмів шифрування та цифрових підписів є одним зі способів ефективного реагування на ці виклики .

Розробка нового додатку для Android, який інтегруватиме прогресивні методи автентифікації, виходить на перший план у світлі загальної популярності мобільних технологій та зростання кількості кібератак. Цей додаток буде направлено на вдосконалення процесів верифікації користувачів за допомогою передових технологій шифрування і цифрового підпису, що сприятиме забезпеченню більшої безпеки.

Зв'язок роботи з науковими програмами, планами, темами. Дослідження проводилося у контексті актуальних наукових напрямків кафедри кібербезпеки.

Мета та завдання дослідження. Метою даної роботи є створення застосунку для систем Android, який не тільки підвищить захищеність даних, але й оптимізує процедури ідентифікації користувачів. Головними завданнями є:

1. Розробка методик автентифікації, заснованих на шифруванні і цифрових підписах для додатку в ОС Android;
2. Розробка додатку який буде написаний за допомогою таких технологій, як Xamarin та інші кросплатформенні інструменти;
3. Розробка серверу для побудови API.
4. Проведення експерименту для вимірювання швидкодії методик;

5. Порівняльний аналіз двох методик.

Об’єкт дослідження - процес автентифікації в мобільних системах, які включають шифрування повідомлення та підпис ключем.

Предмет дослідження – методи шифрування та цифрового підпису при автентифікації користувачів на мобільних пристроях.

Методи дослідження. У процесі виконання роботи використовувалися наступні методи:

– Методи дослідження автентифікації;
– методи аналізу та синтезу сучасних алгоритмів шифрування;
– програмна імплементація алгоритмів;
– тестування і верифікація систем автентифікації;
– графічне моделювання порівнянь методів автентифікацій, що дозволяє аналізувати дані для прийняття рішень.

Наукова новизна отриманих результатів - введення комплексного підходу до використання шифрування та цифрових підписів у системах автентифікації на Android, що розширює можливості забезпечення безпеки інформації з використанням сучасних технологій шифрування.

Наукова новизна отриманих результатів:

1. Подальшого розвитку отримав метод автентифікації для мобільних пристроїв на основі розподіленого реєстру, що дозволяє підвищити безпеку процесу автентифікації.

Практична цінність отриманих результатів. Розроблена програмна модель може бути впроваджена в комерційних та приватних мобільних застосунках для забезпечення вищого рівня безпеки користувача. Результати можуть бути також корисні для підвищення обізнаності розробників програмного забезпечення про новітні методи захисту даних.

Особистий внесок здобувача. Усі наукові результати, викладені в магістерській кваліфікаційній роботі, отримані автором особисто. У рамках дослідження здобувачем були розроблені архітектуру системи автентифікації, проведенно програмну реалізацію та тестування розробленого рішення. Також

було виконано аналіз і порівняння з методами автентифікації, що дозволило визначити переваги впроваджених методик у контексті мобільної безпеки.

1 ПОНЯТТЯ ПРО АВТЕНТИФІКАЦІЮ

1.1 Огляд технологій автентифікації в інформаційних системах

Автентифікація є ключовим елементом систем безпеки, оскільки вона забезпечує обслуговування авторизованих користувачів. Вона створює початковий захисний бар'єр або "щит" для оборони інформаційного середовища організації.

Теоретичний розгляд автентифікації включає дослідження технічних пристроїв, методологій і стратегій для точного ідентифікування особистості суб'єктів, що відіграють важливу роль у криміналістичних розслідуваннях.

Особиста автентифікація не тільки включає визначення особи, але й пов'язує її з конкретними характеристиками або атрибутами. У контексті інформаційних технологій це означає ідентифікацію користувача таким чином, що система може налаштувати алгоритми для доступу до комп'ютерних ресурсів, конфіденційної інформації та інших завдань, призначених для користувача.

Завдяки автентифікації, система забезпечує переконання в тому, що особа дійсно є тією, ким представляється. Термін "автентифікація" іноді може бути використаний як синонім до "підтвердження автентичності".

На рис. 1.1. зображено схему автентифікації з використанням ідентифікатора [1]

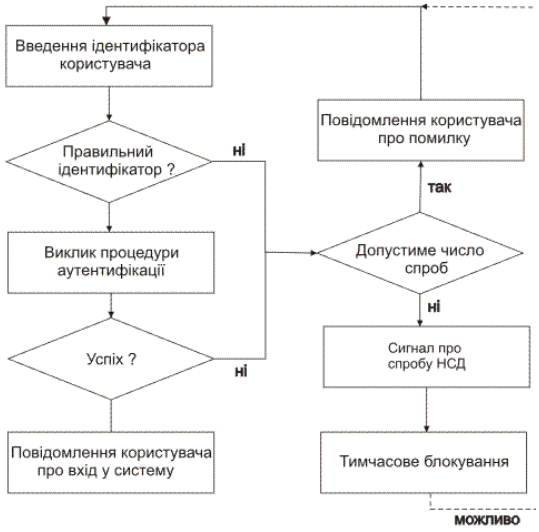


Рисунок 1.1 – Схема автентифікації з використанням ідентифікатора

В умовах відкритої мережі взаємодія учасників процесу ідентифікації та автентифікації часто відбувається без гарантованого безпечного шляху передачі даних. Це створює ризик, що інформація, надіслана однією стороною, може відрізнятися від тієї, що була фактично отримана та використана для підтвердження автентичності особи. Відповідно, надзвичайно важливим є впровадження заходів безпеки, що унеможливлюють нелегальне перехоплення, зміну або повторне використання даних.

1.2 Способи і методи автентифікації

Спосіб – це дія або система дій, що використовуються при виконанні будь-якої роботи.

Метод – це система правил і прийомів, що використовуються при вивченні явищ природи, суспільства, мислення.

Автентифікація відіграє вирішальну роль у захисті інформаційних систем, запропоновуючи різноманітні способи, за допомогою яких можливо підтвердити особу [2]. Ці способи можна класифікувати за трьома основними факторами: знає, має, є.

1) Що користувач знає (Knowledge Factors). Перше це паролі та ПІН-коди, вони вважаються найпоширеніший та базовий метод, що включає введення секретного слова чи комбінації чисел, які знає тільки користувач. В цей самий спосіб можна включити «секретні питання». Це коли користувач відповідає на обрані ним питання, відповіді на які він попередньо вказав при налаштуванні облікового запису.
2) Що користувач має (Possession Factors) включає в себе одноразові паролі (OTP). Це паролі, які дійсні тільки для один раз при введені. Також є токени - фізичні пристрої, які генерують безпековий код, який користувач вводить під час автентифікації. І мобільні пристрої, які використовуються в додатках на мобільних телефонах для генерації кодів автентифікації або як ключів доступу.

3) Ким користувач є (Inherence Factors) це біометричні методи. Вони ділються на: сканування відбитків пальців, розпізнавання обличчя, ірисочна сканування і розпізнавання голосу:

У сфері інформаційних технологій застосовуються наступні підходи до автентифікації:

Одностороння автентифікація: цей метод передбачає, що лише одна сторона, тобто клієнт або користувач системи, має довести свою автентичність для отримання доступу до інформації.

Двобічна автентифікація: в цьому випадку не тільки клієнт, але і сама система, наприклад банк, повинні взаємно підтвердити свою автентичність.

Тристороння автентифікація: тут використовується додаткова незалежна сторона, зазвичай нотаріальна служба, яка забезпечує підтвердження автентичності обох учасників обміну інформацією.

Кожен з цих методів забезпечує певний рівень безпеки і вибір конкретного методу залежить від потреб організації та типу даних, до яких здійснюється доступ.

На рис. 1.2. відображено методи автентифікації.

Рисунок. 1.2 – Методи ідентифікації

Методи автентифікації можна класифікувати на однофакторні та двофакторні.

Однофакторні методи включають:

– Логічні методи. Ці методи вимагають від користувача введення паролів або ключових фраз через комп'ютерну клавіатуру або спеціалізований пристрій.
– Ідентифікаційні методи. Ключова інформація переноситься фізичними об'єктами, такими як дискети, магнітні картки, смарт-карти, штрих-кодові картки та інші. Головні недоліки цього методу включають потребу в спеціальному рідері для читання інформації та ризик втрати чи крадіжки носія.
– Біометричні методи. Базуються на аналізі унікальних фізичних характеристик людини, таких як відбитки пальців, райдужка ока, голос, або обличчя. Недоліки включають високу ціну, складність у використанні та обслуговуванні, чутливість до змін параметрів носія, та обмежену надійність.

Окремої уваги заслуговують біометричні технології:

– Автентифікація за відбитками пальців. Вважається одним із надійних та простих у використанні методів з високою вірогідністю точності ідентифікації.
– Використання геометрії руки. Популярний у великих організаціях метод, котрий хоч і займає більше місця, має переваги за надійністю порівняно з відбитками пальців.
– Автентифікація за райдужною оболонкою ока. Має переваги ефективності завдяки можливості сканування на відстані, і підходить для людей із порушеннями зору, але із збереженою райдужкою.
– Сканування сітківки ока. Використовується в особливо суворих системах безпеки через високу точність і надзвичайно низький ризик несанкціонованого доступу.
– Розпізнавання обличчя. Ця технологія розвивається дуже швидко і

обіцяє забезпечити високий рівень безпеки, особливо в громадських місцях, таких як аеропорти для запобігання терористичним актам.

Кожен з цих методів вибирається на основі потреби в безпеці та ресурсів, доступних для впровадження [3].

Двофакторна автентифікація (2FA) представляє собою захисний механізм, що використовує два окремі типи даних для верифікації особи користувача при спробі доступу до системи чи облікового запису [4]. Ці дані зазвичай включають комбінацію чогось, що користувач знає (наприклад, пароль або ПІН-код) і чогось, що користувач має (наприклад, спеціальний жетон автентифікації або мобільний телефон).

Завдяки використанню двофакторної автентифікації, безпека облікових записів значно підвищується, оскільки зловмисникам стає важче здійснити несанкціонований доступ. Навіть якщо хтось отримає ваш пароль, це не буде достатньо, оскільки для повноцінного доступу потрібний додатковий фактор автентифікації, який значно ускладнює можливості для шахрайства та крадіжки даних.

Основні типи факторів, що застосовуються в двофакторній автентифікації, включають (рис. 1.3):

– Щось, що користувач знає: цей вид включає інформацію, яка має бути відома тільки користувачу, наприклад, паролі, ПІН-коди. Типовим сценарієм використання є необхідність введення пароля після логіну.
– Щось, що користувач має: категорія, що включає предмети або пристрої, які фізично належать користувачу. Зазвичай це мобільний пристрій, який використовується для отримання одноразових підтверджувальних кодів, що вводяться під час процесу автентифікації разом з основним паролем.
– Щось, що характеризує користувача як особу: включає унікальні біометричні особливості, такі як відбитки пальців, розпізнавання обличчя або сітківки ока, які слугують для додаткового підтвердження ідентичності користувача.

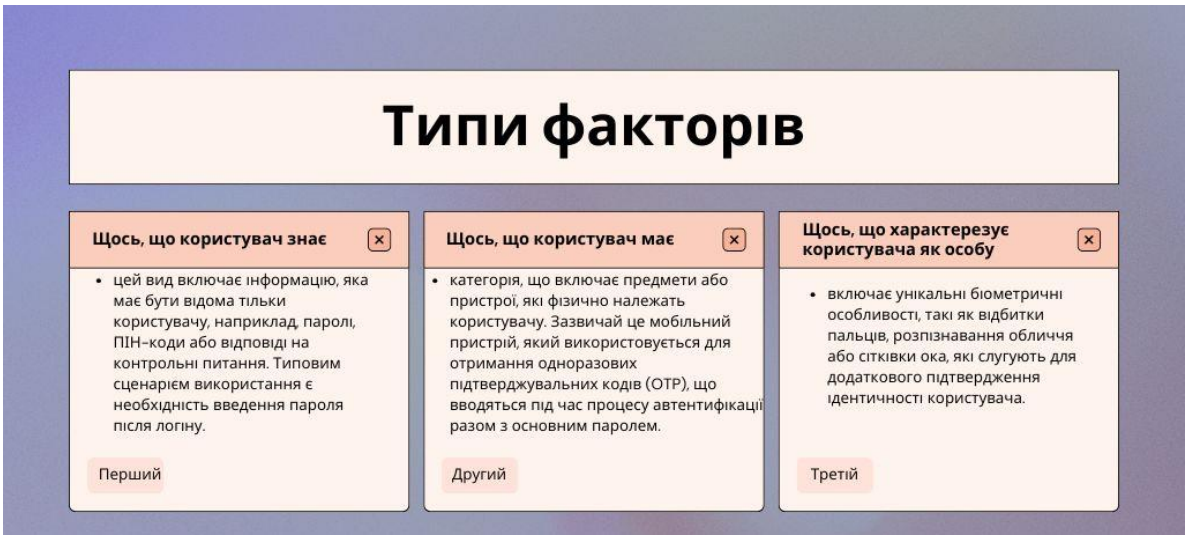


Рисунок. 1.3 – Типи факторів двофакторної автентифікації

Двофакторна автентифікація (2FA) часто використовується для зміцнення безпеки шляхом вимоги двох різних форм доказу ідентичності. Зазвичай це комбінація знань користувача, таких як пароль, та фізичного елементу, який перебуває в його володінні, такого як мобільний телефон, з якого надходить одноразовий код доступу. Ці два фактори разом створюють більш міцний шар захисту.

Впровадження 2FA рекомендується для сервісів, де висока безпека є необхідною, наприклад, в онлайн-банкінгу, електронній пошті чи соціальних мережах. Використання двофакторної автентифікації значно ускладнює потенційний несанкціонований доступ до важливих або конфіденційних даних, забезпечуючи додаткову безпеку користувачам.

Проте, важливо враховувати, що незважаючи на збільшену безпеку, викликану застосуванням двофакторної автентифікації, жодна система не є непорушною. Більшість методів автентифікації ідентифікують не саму особу, а лише дані для входу, що відповідають певному ідентифікатору. Це означає, що системи можуть бути вразливими у випадку компрометації цих даних для входу.

Багатофакторна автентифікація (MFA) представляє собою розширену технологію забезпечення безпеки, яка використовує декілька різних методів для підтвердження особистості користувачів у інформаційних системах [5]. Вона відрізняється від двофакторної автентифікації тим, що включає більше ніж два фактори, надаючи тим самим вищий рівень безпеки.

Основний принцип багатофакторної автентифікації полягає у використанні кількох відмінних факторів, кожен з яких базується на унікальних даних(рис 1.4) :

– Щось, що ви знаєте. Цей фактор включає конфіденційні дані як пароль, ПІН-код або відповіді на секретні запитання, які відомі лише користувачу.
– Щось, що ви маєте. В цю категорію входять пристрої, які фізично мають користувачі, такі як мобільні телефони, смарт-карти, USB-ключі або автентифікаційні токени, що використовуються для підтвердження особи.
– Щось, що ви є. Охоплює біометричні характеристики користувача, наприклад, відбитки пальців, розпізнавання обличчя або сканування сітківки ока, які служать для вірогідної ідентифікації.
– Щось, що ви робите. Включає в себе додаткові дії, які користувач повинен виконати для автентифікації, такі як введення одноразового коду, підтвердження дії через мобільний пристрій або використання другого пристрою для запуску підтвердження.

Рисунок 1.4 – Комбінація факторів багатофакторної автентифікації

Використання декількох факторів у багатофакторній автентифікації створює більш складну систему для верифікації особи, значно ускладнюючи можливості для несанкціонованого доступу до систем, навіть у випадку компрометації одного з факторів. Цей метод є особливо популярним у критично важливих секторах, таких як фінанси, електронна комерція, медицина і корпоративні мережі, де вимоги до безпеки даних є особливо високими.

1.4 Аналіз методів автентифікації в Android

Андроїд (або Android) - це операційна система для мобільних пристроїв, яка розроблена компанією Google. Вона використовується на різних пристроях, таких як смартфони, планшети, телевізори і навіть ноутбуки[6].

Історія Android бере початок у 2003 році, коли Енді Рубін, Річ Майнер, Нік Сірс і Кріс Вайт спільно заснували Android Inc. у Пало-Альто, Каліфорнія. Спочатку Android була розроблена не як операційна система для мобільних телефонів, а як платформа для цифрових камер. Розробники згодом зрозуміли, що ринок цифрових камер не мав достатнього потенціалу для їхньої продукції, тому зосередили свої зусилля на мобільних телефонах.

У 2005 році Google купив компанію Android Inc., вбачаючи значний потенціал у фірмі та її розробках. Цей крок відкрив новий розділ для Android, коли Google вирішив розробляти її як відкриту та безкоштовну операційну систему для виробників, що спростило процес її інтеграції в пристрої від різних компаній.

Перший смартфон на базі Android, T-Mobile G1 (також відомий як HTC Dream), був випущений в 2008 році. Операційна система мала інтеграцію з послугами Google, зручний інтерфейс, а також можливість завантажувати додатки з Android Market (який згодом перейменовано в Google Play) [7].

З цього часу Android стала провідною операційною системою на глобальному ринку мобільних пристроїв, регулярно оновлюючись з новими можливостями та вдосконаленнями. Її відкрита натура дозволяє багатьом

виробникам та розробникам адаптувати систему під свої специфічні потреби, що сприяє її популярності та широкому розповсюдженню.

Незважаючи, що перший смартфон на базі Android, був випущений в 2008 році, але важливі запровадження та змінні в механізмах автентифікації було змінено з версії Android 5.0 Lollipop в 2014 році. Декілька значущих версії Android, де були зроблені зміни автентифікації наведено нижче(рис. 1.5) [8].

Android 5.0 Lollipop (2014 рік) ввів заходи безпеки, такі як система захисту Factory Reset Protection (FRP). FRP зажадало від користувачів вводити дані свого облікового запису Google після скидання настроювань, що збільшило захист пристрою від несанкціонованого доступу.

З настанням Android 6.0 Marshmallow (2015 рік) значно покращилася біометрична автентифікація з появою API для сканерів відбитків пальців. Таке нововведення дозволило розробникам легше втілювати функціонал відбитків пальців для розблокування пристроїв та верифікації користувачів при виконанні транзакцій чи доступі до додатків [9].

Android 7.0 Nougat (2016 рік) приніс покращення в роботі API для відбитків пальців та ввів функцію Direct Boot. Direct Boot поліпшив функціональність пристроїв після перезавантажень, забезпечуючи обмежений доступ до додатків до того, як пристрій буде повністю розблокований [10].

У версії Android 9.0 Pie (2018 рік) була представлена система BiometricPrompt API, яка замінила стару систему сканування відбитків пальців. BiometricPrompt API створив єдиний універсальний діалог для всіх видів біометричної автентифікації, підтримуючи як відбитки пальців, так і інші методи, такі як розпізнавання обличчя.

Android 10 (2019 рік) приніс розширену підтримку біометричних технологій, зокрема автентифікацію за допомогою розпізнавання обличчя [11]. В цій версії також було зміцнено захист приватності, додаючи можливість додаткам запитувати дозвіл на доступ до місцезнаходження в процесі їх використання.

У 2020 році, з випуском Android 11, було введено нову функціональність, що дозволяє користувачам надавати додаткам дозволи на використання датчиків, таких як камера, мікрофон і GPS, на поодинокій основі. Ця функція надає користувачам більше контролю над їх приватністю. Додатково, Android 11 запровадив автоматичне скидання дозволів для додатків, які не були активними протягом певного часу, що змусить користувачів заново надати дозволи при наступному використанні додатків.

З появою Android 12 у 2021 році, система управління дозволами стала ще більш досконалою. Оновлення включає точніший контроль за параметрами розташування і новий дашборд приватності, який надає детальну інформацію про використання дозволів додатками. Крім того, Android 12 покращив інтеграцію біометричної автентифікації, розширюючи API для підтримки більш широкого спектру біометричних технологій.

З появою Android 13 у 2022 році відбулися значні вдосконалення в системах автентифікації. Була введена уніфікована платформа для біометричних технологій, що включала рівні надійності, як Strong - висока надійність (наприклад, сканери відбитків пальців, які базуються на ультразвукових технологіях), Weak - середня надійність (методи, що використовують просте розпізнавання обличчя за допомогою фронтальної камери) і Credentialed - поєднання методів (наприклад, комбінація PIN-коду з біометрією для виконання критично важливих операцій). Це значно спростило для розробників процес інтеграції біометричних систем в додатки. Крім того, Android 13 почав підтримувати Passkeys — новий стандарт безпечної автентифікації, що заміняє традиційні паролі і зберігає ключі доступу у захищеному сховищі. Ключі доступу зберігаються локально в апаратному сховищі пристрою. Синхронізація через Google Password Manager дозволяє використовувати Passkeys на різних пристроях. Це значно підвищило захищеність, адже Passkeys неможливо зламати методами підбору чи фішингу. Для розробників з’явився новий Credential Manager API, який об'єднує різні методи автентифікації — біометрію, паролі та Passkeys — в одному зручному інтерфейсі.

У 2023 році з випуском Android 14 точність і безпека біометричної автентифікації зазнали подальшого підвищення, зокрема у плані захисту від підроблених даних. Вдосконалення API BiometricPrompt і Credential Manager дозволили стороннім додаткам використовувати передові методи входу, включаючи автентифікацію на основі контексту. Розширена інтеграція Passkeys з вебсайтами і покращена синхронізація через Google Password Manager зробили використання цих технологій ще зручнішим.

Android 14 також удосконалив технології NFC та Bluetooth для безконтактної автентифікації, розширивши їх використання для смарт-пристроїв та забезпечивши кращий захист сигналів від перехоплення. Також змінили шифрування облікових даних, а саме Всі біометричні дані та Passkeys шифруються в межах Trusted Execution Environment (TEE), а їх видалення відбувається автоматично після тривалого невикористання.

Версія Android	Рік випуску	Основні нововведення в автентифікації
Android 5.0 Lollipop	2014	Factory Reset Protection (FRP)
Android 6.0 Marshmallow	2015	API для сканерів відбитків пальців
Android 7.0 Nougat	2016	Покращення API відбитків пальців, функція Direct Boot
Android 9.0 Pie	2018	Впровадження API BiometricPrompt
Android 10	2019	Розширена підтримка біометричних технологій, включаючи розпізнавання обличчя
Android 11	2020	Дозволи на використання датчиків на разовій основі, автоматичне скидання дозволів
Android 12	2021	Удосконалена система управління дозволами, покращена біометрична автентифікація
Android 13	2022	Уніфікована платформа для біометричних технологій, підтримка Passkeys
Android 14	2023	Підвищена точність і безпека біометричної автентифікації, удосконалення NFC і Bluetooth

Рисунок 1.5 – Нововведення в автентифікації в різних версіях Android

Ці оновлення в операційній системі Android сприяли посиленню безпеки та надали користувачам ширший вибір методів автентифікації, підвищуючи тим самим загальний рівень захисту даних і конфіденційності на пристроях.

1.5 Імплементація Android SDK

Android SDK (Software Development Kit) є фундаментальним набором інструментів для розробників, які бажають створювати додатки для операційної системи Android. Включаючи компілятори, API, різноманітні інструменти, а також бібліотеки, спеціально призначені для Android, SDK допомагає в процесі розробки, тестуванні і аналізі додатків.

Основні компоненти Android SDK включають:

1. Android Studio: Це офіційне інтегроване середовище розробки (IDE), яке забезпечує полегшене написання коду, дебагінг, тестування та профілювання додатків.
2. SDK Tools: Цей комплекс інструментів включає в себе налагоджувачі, бібліотеки, емулятори, документацію, приклади коду та інше.
3. Platform-tools: Інструменти, специфічні для розробки на платформах Android, які включають утиліти для взаємодії з пристроями Android.
4. Build-tools: Інструментарій, необхідний для створення додатків на Android, наприклад, компілятори коду.
5. APIs: Інтерфейси програмування додатків, що дозволяють розробникам використовувати можливості пристроїв і сервісів Android.

Процес імплементації Android SDK включає наступні кроки:

1. Встановлення Android Studio: Почніть з інсталяції Android Studio, що включає у себе всі необхідні компоненти SDK.
2. Конфігурація нового проекту: Встановіть параметри нового проекту, визначте необхідні версії SDK та додайте потрібні бібліотеки.
3. Розробка додатку: За допомогою Java або Kotlin розробляйте ваш додаток, інтегруючи необхідний функціонал та інтерфейси.
4. Тестування та дебагінг: Використовуйте інтегровані інструменти Android Studio для аналізу та виправлення помилок у вашому додатку.
5. Деплоймент: Готові та протестовані додатки можна публікувати у Google Play Store або розповсюджувати іншими методами.

Додавання різних автентифікацій до програм з ОС Android 9 і вище:

1. Credential Manager;
2. Автентифікація користувачів за допомогою WebView.

Credential Manager - це API Jetpack, яке підтримує кілька методів входу, таких як ім'я користувача з паролем, passkeys та федеративні рішення входу (наприклад, вхід через Google) в єдиному API, спрощуючи інтеграцію для розробників.

Крім того, для користувачів Credential Manager уніфікує інтерфейс входу між методами автентифікації, роблячи процес входу в додатки зрозумілішим і зручнішим, незалежно від обраного методу. Passkeys– це більш безпечна та зручна заміна паролів. З допомогою passkeys користувачі можуть входити в додатки та на веб-сайти, використовуючи біометричний сенсор (такий як відбиток пальця або розпізнавання обличчя), PIN-код чи графічний ключ. Це забезпечує безшовний досвід входу, звільняючи користувачів від необхідності пам'ятати імена користувачів чи паролі.

Passkeys залежать від WebAuthn (Web Authentication), стандарту, розробленого спільно FIDO Alliance та World Wide Web Consortium (W3C). WebAuthn використовує криптографію з відкритим ключем для автентифікації користувача. Веб-сайт або додаток, в який користувач входить, може бачити та зберігати публічний ключ, але ніколи - приватний ключ. Приватний ключ залишається у таємниці та безпечності. Оскільки ключ є унікальним та пов'язаний з веб-сайтом або додатком, passkeys неможливо викрасти, що забезпечує додаткову безпеку.

Щоб увімкнути підтримку ключа доступу для програми Android, пов’яжіть свою програму з веб-сайтом, яким вона володіє. Ви можете оголосити цей зв’язок, виконавши такі дії:

1. Створити файл JSON Digital Asset Links. Наприклад, щоб оголосити, що веб-сайт https://signin.example.comі програма Android із такою назвою пакета com.example можуть спільно використовувати облікові дані для входу, створіть файл assetlinks.json із таким вмістом:

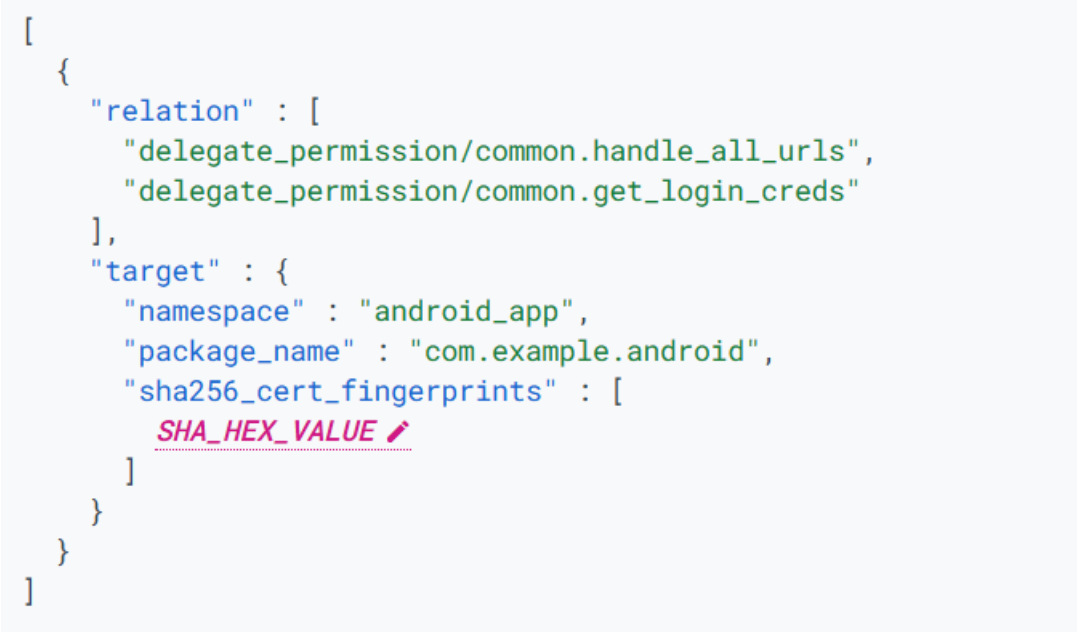


Рисунок 1.6 – файл JSON Digital Asset Links

Поле relation - це масив з одного або кількох рядків, які описують оголошений зв'язок. Щоб оголосити, що програми та сайти мають спільні облікові дані для входу, вказати зв'язки як delegate_permission/handle_all_urls і delegate_permission/common.get_login_creds.

Поле target - це об'єкт, який визначає актив, якого стосується декларація. Наступні поля ідентифікують веб-сайт: namespace, web, site. URL-адреса веб-сайту у форматі https://www.example.com.

2. Розмістити файл JSON Digital Assets Link у такому місці в домені входу: https://domain[:optional_port]/.well-known/assetlinks.json/
3. Переконатись, що хост дозволяє Google отримати файл Digital Asset Link. Якщо у вас є robots.txt файл, він повинен дозволити агенту Googlebot отримати /.well-known/assetlinks.json. Більшість сайтів можуть дозволити будь-якому автоматичному агенту отримувати файли на /.well-known/ шляху, щоб інші служби мали доступ до метаданих у цих файлах: User-agent: *; Allow: /.well-known/
4. Додати такий рядок до файлу маніфесту в розділі <application>:

<meta-data android:name="asset_statements" android:resource="@string/asset_statements" />

Тепер потрібно налаштувати диспетчер облікових даних.

Щоб налаштувати та ініціалізувати CredentialManager об’єкт, потрібно додати логіку, подібну до наступної (val credentialManager = CredentialManager.create(context))

Наступним кроком потрібно вказати поля облікових даних. У Android 14 та більш нових версіях атрибут isCredential можна застосовувати для позначення полів введення облікових даних, як-то імені користувача або поля введення пароля. Цей атрибут визначає, що дане поле є відповідальним за зберігання облікових даних, спрощуючи взаємодію з менеджером облікових даних та сторонніми провайдерами. Це також сприяє покращенню пропозицій автоввід (автозаповнення) від різних сервісів.

Коли додаток інтегрований з API менеджера облікових даних, відображається модальне вікно на зразок менеджера облікових даних з доступними опціями облікових записів. В такому випадку, не потрібно відображати окремі діалогові вікна для автозаповнення або збереження імен користувачів і паролів, оскільки додаток безпосередньо спілкується з Credential Manager API для зберігання чи вибору облікових даних.

Щоб користуватися атрибутом isCredential, його необхідно додати до відповідних елементів Views у вашому інтерфейсі. Таким чином, атрибут стає містком, що з'єднує поля вводу в додатку з системою управління обліковими даними, оптимізуючи процес автентифікації та зберігання даних.

```
<TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:isCredential="true"
...
 />
```

Щоб отримати всі параметри ключа доступу та пароля, пов’язані з обліковим записом користувача, потрібно:

1. Ініціалізувати параметри автентифікації пароля та ключа доступу:

```
val getPasswordOption = GetPasswordOption()
val getPublicKeyCredentialOption = GetPublicKeyCredentialOption(
    requestJson = requestJson
)
```

2. Використати параметри, отримані з попереднього кроку, щоб створити запит на вхід.

```
val getCredRequest = GetCredentialRequest(
    listOf(getPasswordOption, getPublicKeyCredentialOption)
)
```

3. Потрібно запустити процес входу.

У деяких випадках користувач може не мати доступних облікових даних або користувач може не надати згоду на використання доступних облікових даних. Якщо getCredential() викликано і жодних облікових даних не знайдено, NoCredentialException повертається. Якщо це станеться, код має обробляти NoCredentialException екземпляри.

```
try {
  val credential = credentialManager.getCredential(credentialRequest)
} catch (e: NoCredentialException) {
  Log.e("CredentialManager", "No credential available", e)
}
```

На Android 14 або новішої версії ви можете зменшити затримку під час показу селектора облікових записів за допомогою prepareGetCredential() методу перед викликом getCredential().

```
val response = credentialManager.prepareGetCredential(
  GetCredentialRequest(
    listOf(
      <getPublicKeyCredentialOption>,
      <getPasswordOption>
    )
  )}
```

Метод prepareGetCredential() не викликає елементів інтерфейсу користувача. Це лише допомагає виконати підготовчу роботу, щоб пізніше можна було запустити решту операції отримання облікових даних (яка включає інтерфейси користувача) через getCredential() API.

Кешовані дані повертаються в PrepareGetCredentialResponse об'єкті. Якщо наявні облікові дані, результати буде кешовано, і пізніше можливо запустити решту getCredential() API, щоб відкрити селектор облікових записів із кешованими даними.

Також можна встановити параметр authenticatorAttachment, але лише під час створення облікових даних. Можливо вказати platform, cross-platform, або не мати значення. У більшості випадків значення не рекомендовано.

- platform: щоб зареєструвати поточний пристрій користувача або запропонувати користувачеві ввести пароль для оновлення до ключів доступу після входу, установіть authenticatorAttachment значення platform.
- cross-platform: це значення зазвичай використовується під час реєстрації багатофакторних облікових даних і не використовується в контексті ключа доступу.
- Немає значення: щоб надати користувачам можливість створювати ключі доступу на своїх вибраних пристроях (наприклад, у налаштуваннях облікового запису), authenticatorAttachment параметр не слід вказувати, коли користувач вирішує додати ключ доступу. У більшості випадків найкращим варіантом буде залишити параметр невизначеним.

Для того щоб запобігти створенню нового ключа доступу, якщо він уже існує з тим самим постачальником ключа доступу, потрібно перелічити ідентифікатори облікових даних у додатковому масиві.

Якщо користувач надає ім'я користувача та пароль для процесу автентифікації у програмі, потрібно зареєструвати облікові дані користувача, які

можна використовувати для автентифікації користувача. Для цього потрібно створити CreatePasswordRequest об’єкт, за допомогою функції:

```
fun registerPassword(username: String, password: String) {
    val createPasswordRequest = CreatePasswordRequest(id = username, password = password)
val result = credentialManager.createCredential(activityContext,
    createPasswordRequest)
handleRegisterPasswordResult(result)}
```

Якщо користувач більше не має доступу до пристрою, на якому він зберігав свої облікові дані, йому може знадобитися відновити дані за допомогою безпечної онлайн-резервної копії.

Одна з проблем, з якою стикаються користувачі під час керування кількома ключами доступу, пов’язаними з даною програмою, полягає в тому, щоб визначити правильний ключ доступу для редагування або видалення. Щоб вирішити цю проблему, рекомендовано, щоб додатки та веб-сайти включали додаткову інформацію, як-от постачальника, який створив облікові дані, дату створення та дату останнього використання, у списку ключів доступу на екрані налаштувань програми. Інформацію про постачальника можна отримати шляхом вивчення AAGUID, пов’язаний із відповідним паролем. AAGUID можна знайти як частину даних автентифікатора ключа доступу.

Наприклад, якщо користувач створює ключ доступу на пристрої на базі Android за допомогою Менеджера паролів Google, RP потім отримує AAGUID, який виглядає приблизно так: "ea9b8d66-4d01-1d21-3ce4-b6b48cb575d4". Провіряюча сторона може примітити ключ доступу в списку ключів доступу, щоб вказати, що його було створено за допомогою Менеджера паролів Google.

Щоб зіставити AAGUID із постачальником ключа доступу, RP можуть використовувати репозиторій AAGUID, створений спільнотою..

Впровадження Android SDK вимагає певного рівня знань з програмування і розуміння мобільних платформ, але завдяки ресурсам та підтримці від Google, процес є доступним і зрозумілим для більшості розробників.

Автентифікація користувачів за допомогою WebView у Android дозволяє вбудовувати веб-контент безпосередньо в ваш додаток. WebView може відображати веб-сторінки, як і справжній браузер, але не має всіх можливостей нативного браузера. Тому, при використанні WebView для автентифікації користувачів, виникають певні питання безпеки та обмеження щодо функціональності, які потрібно враховувати. Створення реєстрації:

1. Сервер готує JSON, який включає всю необхідну інформацію для реєстрації, та надсилає його на веб-сторінку, яка відображена в WebView.
2. Веб-сторінка виступає як інтерфейс і використовує navigator.credentials.create() для ініціювання створення нових облікових даних. За допомогою ін'єкцій JavaScript можна перевизначити цей метод, щоб ініціювати запит до Android додатку.
3. Додаток на Android, використовуючи API диспетчера облікових даних, ініціює запит для створення облікових даних і подальше їх використання за допомогою createCredential.
4. API диспетчера облікових даних передає додатку облікові дані відкритого ключа.
5. Додаток надсилає облікові дані відкритого ключа назад на веб-сторінку для обробки за допомогою вбудованого JavaScript.
6. Веб-сторінка завершує процес, надсилаючи відкритий ключ на серверну частину для подальшої перевірки та збереження.

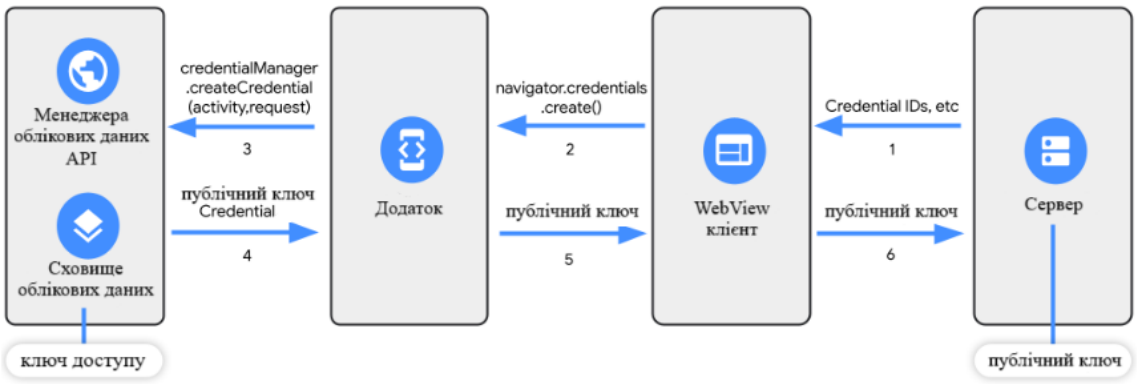


Рисунок 1.7 – Потік реєстрації ключа доступу

Після створення реєстрації, потрібно створити автентифікацію для отримання ключу доступу:

1. Сервер створює JSON для автентифікації з метою отримання облікових даних та відправляє його на веб-сторінку, яка зображується у клієнті WebView.
2. На веб-сторінці використовується navigator.credentials.get(). Через ін’єкційний JavaScript цей метод замінюється таким чином, аби перенаправити запит до Android-додатку.
3. Android-додаток викликає getCredential через API диспетчера облікових даних, які, в свою чергу, передають додатку потрібні облікові дані.
4. За допомогою облікових даних, додаток генерує цифровий підпис з використанням приватного ключа та передає його назад на веб-сторінку для додаткової обробки ін'єктованим JavaScript.
5. Веб-сторінка, отримавши цифровий підпис, направляє його на сервер. Сервер використовує відповідний відкритий ключ, щоб перевірити цифровий підпис. Якщо перевірка успішна, це підтверджує справжність облікових даних користувача.

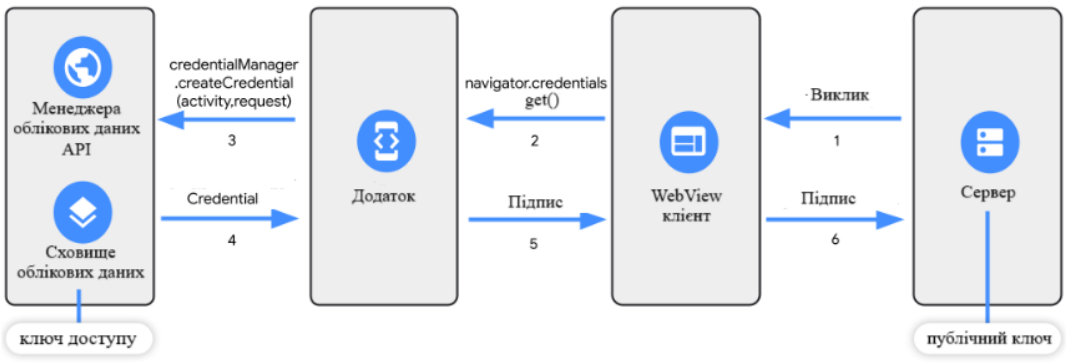


Рисунок 1.8 – Потік автентифікації за допомогою ключа доступу

Той самий потік можна використовувати для паролів або систем об’єднаної ідентифікації.

1.6 Висновки до розділу

У цьому розділі визначено основні поняття про автентифікацію в цілому та Android, шо дає змогу зрозуміти цю тему більш детальніше. Окрім цього, було розглянуто:

- технології автентифікації в інформаційних системах, зокрема, як саме працює за допомогою схеми.
- Способи та методи автентифікації, а саме: що користувач знає, що користувач має, ким користувач є та методи такі як однофакторні, двофакторні і багатофакторні.

Історія Android, зокрема, розглянуто версії в яких були значущі зміни в механізмах автентифікації Android 5.0 Lollipop, Android 6.0 Marshmallow, Android 7.0 Nougat, Android 9.0 Pie, Android 10, Android 11, Android 12, Android 13 та імплементацію Android SDK, а саме Credential Manager та автентифікацію користувачів за допомогою WebView.

2 КЛАСИФІКАЦІЯ І АНАЛІЗ МЕТОДІВ ШИФРУВАННЯ

2.1 Методологія захисту інформації за допомогою шифрування

Шифрування вважається одним із стовпів кібербезпеки та приватності. Воно дозволяє захистити цілісність, конфіденційність та доступність інформації. Основні типи шифрування поділяються на дві групи: симетричне шифрування та асиметричне шифрування. Також, існують також інші методи та підходи до шифрування, які використовуються для специфічних цілей. Детальніше про кожний тип проаналізовано нижче:

1. Симетричне шифрування.

Симетричне шифрування включає використання однин і той ж ключ для шифрування та дешифрування інформації, що робить більшу швидкість та ефективність при роботі з великими даними.

- DES (Data Encryption Standard). Цей ранній стандарт наразі застарілий та недостатньо безпечний для сучасних потреб.
- AES (Advanced Encryption Standard). Є сучасним стандартом, який на даний момент вважається еталоном безпеки і рекомендується для використання урядовими організаціями США.
- 3DES (Triple DES). Це покращена версія DES, яка, використовуючи три ключі замість одного, значно підвищує рівень безпеки.

2. Асиметричне шифрування.

Асиметричне шифрування використовує пару ключів — публічний для шифрування та приватний для дешифрування, забезпечуючи вищу безпеку, але з більшою вимогою до обчислювальних ресурсів.

- RSA (Rivest–Shamir–Adleman). Це відомий метод, який широко використовується для шифрування даних та цифрових підписів.
- ECDSA (Elliptic Curve Digital Signature Algorithm). Використовує принципи еліптичних кривих для зменшення розміру ключів при збереженні того ж рівня безпеки, що й RSA.

– Diffie-Hellman. Цей метод, хоч і використовується переважно для безпечного обміну ключами, є фундаментальним для асиметричного шифрування.

3. Хешування.

Хоч технічно хешування не є шифруванням, хеш-функції часто поєднуються з шифруванням для забезпечення цілісності даних.

– SHA (Secure Hash Algorithm). Широко використовується для створення унікальних хешів, які гарантують, що дані не були змінені.
– MD5 (Message-Digest Algorithm 5). На сьогодні вважається ненадійним для багатьох застосувань через виявлені вразливості.

В цьому розділі проаналізовано основні методи шифрування, їхню важливість та використання з метою забезпечення конфіденційності, цілісності та доступності інформації, що є життєво необхідним в умовах сучасного цифрового світу.

2.2 Стандарт симетричного шифрування DES (Data Encryption Standard).

Стандарт DES був розроблений у 1970-х роках співпрацею між компанією IBM та Агентством національної безпеки США (NSA). Завданням стандарту було створити надійний та стандартизований метод шифрування, придатний для урядових і комерційних потреб у безпеці даних. DES став одним з перших широко розповсюджених методів симетричного шифрування, що використовувалися протягом десятиліть, доки не був замінений на більш сучасний і безпечний AES (Advanced Encryption Standard).

Технічні характеристики DES:

– Тип шифру: Блочний шифр.
– Розмір блоку: 64 біти.
– Довжина ключа: Ефективна довжина ключа 56 біт із загальною довжиною 64 біти (8 біт використовуються для контролю парності).
– Кількість раундів шифрування: 16.

Якщо розглянути алгоритм процесу шифрування DES, то він може бути розділений на декілька етапів:

1. Початкова перестановка (IP). Перед тим, як розпочнуться раунди шифрування, вхідний блок даних проходить через початковий етап перестановки, розподіляючи біти згідно визначеному порядку.
2. Фаза основних раундів. Розширення - права частина блоку даних розширюється з 32 до 48 біт за допомогою таблиці розширення. Застосування підключа - 48-бітний результат розширення комбінується з фазовим ключем через операцію XOR. S-блоки - сформований 48-бітний блок проходить через 8 S-блоків (substitution boxes), кожен з яких перетворює 6 вхідних бітів у 4 вихідних біта, використовуючи представлені таблиці підстановок. P-перестановка - 32-бітний результат, отриманий після S-блоків, піддається додатковій перестановці згідно із P-таблицею.
3. Фінальне злиття. Після завершення 16 раундів, дві частини блоку обмінюються місцями та проходять останню перестановку, яка нівелює ефект початкової перестановки.

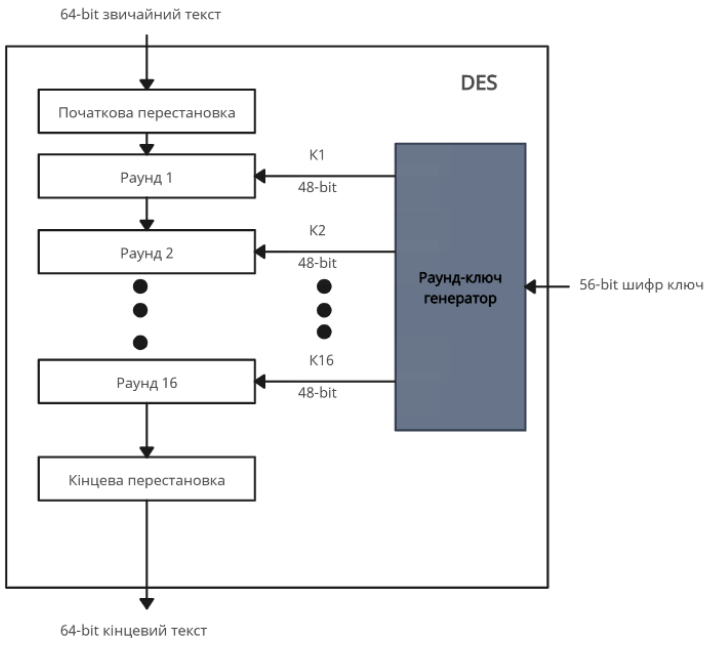


Рисунок 2.1 – Алгоритм процесу шифрування DES

Завершальний етап шифрування включає застосування перестановки оберненої до початкової ($IP\hat{}(-1)$), яка формує остаточний зашифрований блок. Незважаючи на свою важливість та широке використання в минулому, DES був поступово вилучений через свою вразливість до таких атак як перебір ключів і диференційний криптоаналіз, що зробило його ненадійним для сучасних вимог безпеки.

2.3 Стандарт симетричного шифрування 3DES (Triple DES).

Triple DES було створене як удосконалення базового стандарту DES, що виявився недостатньо захищеним через відносно коротку довжину ключа в 56 бітів, яка дозволяла атаки методом "грубої сили". З появою у 1990-х роках, 3DES отримало широке поширення, особливо у фінансовому секторі та урядових структурах, завдяки значному підвищенню рівня захисту даних.

Технічні характеристики 3DES:

- Тип шифру: Блочний шифр.
- Розмір даних для шифрування: як і в DES, 64 біти.
- Довжина ключа: Використовується три ключі по 56 біт, загальна довжина 168 біт.

3DES підвищує безпеку шляхом трьох послідовних процесів шифрування DES:

1. Перший етап з ключем K1. Шифрування даних здійснюється використовуючи перший 56-бітний ключ.
2. Другий етап з ключем K2. На цій стадії виконується дешифрування результату попереднього шифрування за другим ключем, що ускладнює отримання початкових даних без знання K1.
3. Третій етап з ключем K3. Фінальне шифрування результату виконується третім ключем, що максимально забезпечує цілісність даних.

Також, 3DES може працювати з різними режимами блочного шифрування такі як:

– ECB (Electronic Codebook) — базовий, але менш безпечний для даних з повторюваними шаблонами.
– CBC (Cipher Block Chaining) — залучає попередній блок даних до шифрування поточного.
– CFB (Cipher Feedback) та OFB (Output Feedback) — обидва режими дозволяють шифрування у режимі потоку.

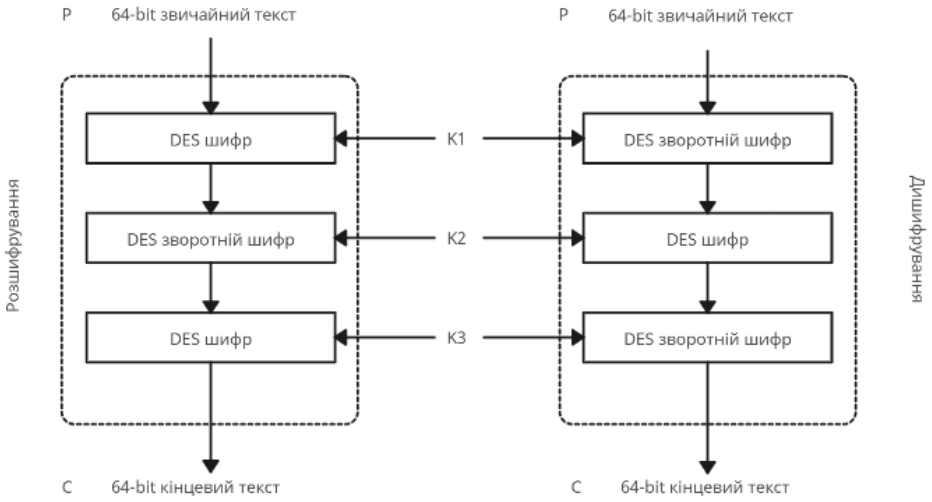


Рисунок 2.2 – Алгоритм процесу шифрування 3DES

Незважаючи на сумісність з DES-обладнанням та високий рівень захисту, 3DES зараз поступово застаріває та замінюється на більш сучасний AES, який володіє кращою продуктивністю та стійкістю до численних сучасних загроз в області криптографії. Існуючим користувачам 3DES рекомендується поступово переходити на стійкіші та ефективніші стандарти шифрування.

2.4 Стандарт симетричного шифрування AES (Advanced Encryption Standard).

Стандарт Advanced Encryption Standard (AES) було розроблено і впроваджено Національним інститутом стандартів і технологій США (NIST) в 2001 році як вдосконалення попередніх стандартів DES та 3DES, які вже не відповідали потребам сучасної кібербезпеки. AES було розроблено на базі алгоритму "Rijndael", який був розроблений бельгійськими криптографами Вінсентом Рейменом та Йоаном Даменом. Завдяки своїм безпековим параметрам та високій продуктивності, AES став одним з найбільш поширених алгоритмів шифрування в усьому світі.

Технічні характеристики AES :

- Тип шифру: Блочний шифр.
- Розмір блоку: 128 біт.
- Довжини ключів: Можливі варіанти 128, 192 або 256 біт.

Процес алгоритму шифрування AES складається з кількох раундів обробки даних, кількість яких залежить від довжини ключа:

- 128-бітний ключ: 10 раундів.
- 192-бітний ключ: 12 раундів.
- 256-бітний ключ: 14 раундів.

Кожен раунд складається із чотирьох фаз:

1. SubBytes: кожен байт замінюється згідно з попередньо визначеною таблицею (S-box).
2. ShiftRows: перестановка рядків у матриці, зміщення варіюється залежно від рядка.
3. MixColumns: лінійне перетворення, яке змішує дані всередині кожного стовпця.
4. AddRoundKey: в кожному раунді до блоку даних додається раундовий ключ через XOR.

Також AES можна застосовувати у декількох різних режимах блочного шифрування:

- Тип шифру: Блочний шифр.

– ECB (Electronic Codebook)
– CBC (Cipher Block Chaining)
– CFB (Cipher Feedback)
– OFB (Output Feedback)
– CTR (Counter)

Ці режими дозволяють налаштовувати процес шифрування під конкретні потреби обробки даних.

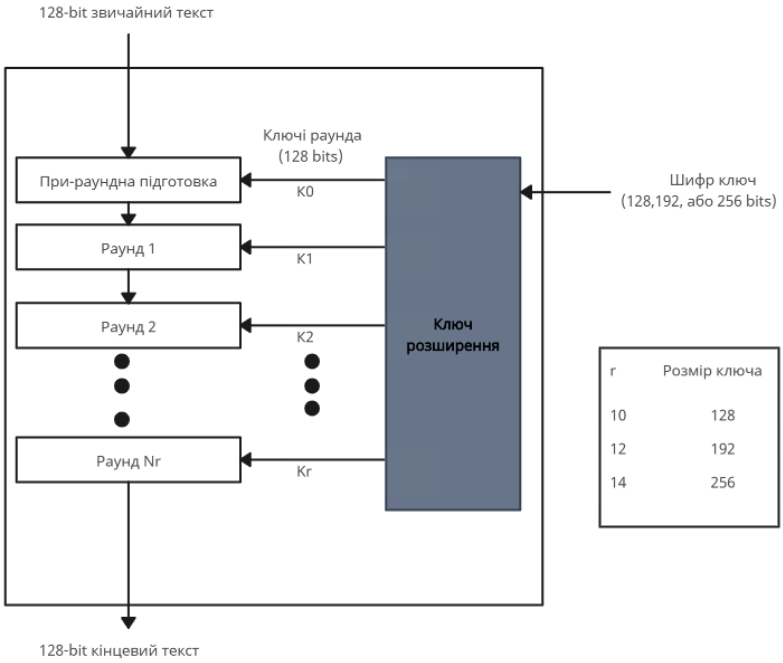


Рисунок 2.3 – Алгоритм процесу шифрування AES

AES є ключовим інструментом у захисті державних, корпоративних та персональних даних, широко використовується у забезпеченні безпеки мобільних додатків та захищених інтернет-з'єднань, включаючи HTTPS. Рекомендований за його надійність та стійкість до атак, AES має продовжувати залишатися в авангарді засобів криптографічного захисту інформації, з огляду на постійне зростання вимог до безпеки в цифровому світі. Саме тому я використовую саме це шифрування в своїй роботі.

2.5 MD5 (Message-Digest Algorithm 5) хешування.

Алгоритм хешування MD5 був створений Рональдом Рівестом у 1991 році як поліпшення попередньої версії MD4. Цей алгоритм був розроблений з метою підвищення стійкості до криптографічних атак і уповільнення процесу генерації хешу порівняно з MD4. MD5 швидко набув популярності і використовувався в широкому спектрі застосувань для перевірки цілісності даних і автентифікації повідомлень.

MD5 призначений для створення 128-бітного хешу з будь-яких вхідних даних незалежно від їхнього розміру. Ця хеш-функція перетворює великі обсяги інформації в короткий, фіксований розмір дайджесту (хеш), що дозволяє ефективно перевіряти унікальність і цілісність даних.

Процес обробки даних в MD5 ділиться на такі кроки:

1. Додавання доповнення до повідомлення. Дані доповнюються так, щоб їхня довжина стала кратною 512 бітам за винятком 64 бітів.
2. Ініціалізація буферу. Використовується чотири місця буфера (A, B, C, D), це є ключовими елементами в алгоритмі.
3. Обробка блоків. Кожний оброблений блок по 512 біт впливає на буфер за допомогою спеціально розроблених операцій.
4. Формування кінцевого хешу. По завершенні обробки всіх блоків стан буфера конвертується у 128-бітний хеш.

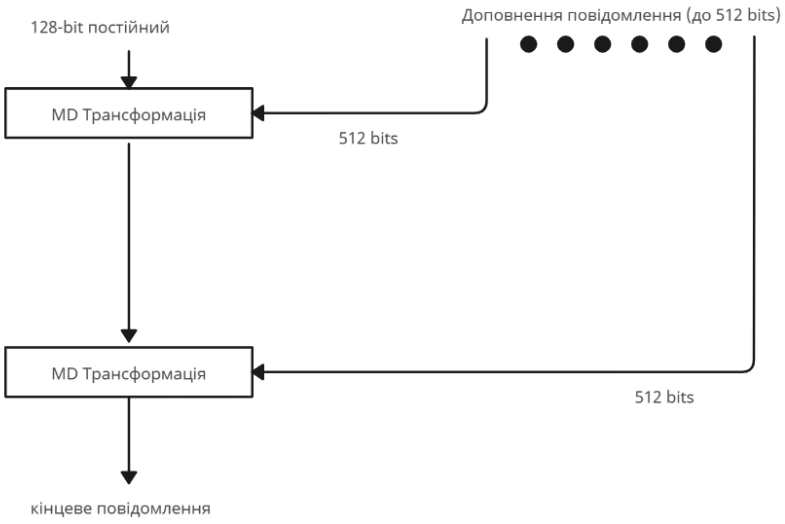


Рисунок 2.4 – Алгоритм хешування MD5

MD5 з часом продемонстрував значні вразливості до криптографічних атак, особливо до тих, які використовують колізії, коли дві різні вхідні послідовності дають однаковий хеш. Це серйозно підриває безпеку MD5 у прикладах, де важлива цілісність даних. Сьогодні фахівці в області IT безпеки не рекомендують використовувати MD5 для криптографічних цілей, особливо у сценаріях, де необхідний високий рівень стійкості до атак. Замість MD5 рекомендуються більш сучасні і безпечні алгоритми, такі як SHA-256 і SHA-3.

2.6 SHA (Secure Hash Algorithm) хешування.

Алгоритми SHA, знані як Secure Hash Algorithms, були розроблені спільними зусиллями Національного інституту стандартів і технологій (NIST) США та Агентства національної безпеки (NSA). Алгоритм SHA-0 вперше з'явився у 1993 році, але був швидко замінений на SHA-1 у 1995 році через виявлені слабості. У відповідь на зростаючі вимоги до безпеки та потужності обчислювальних систем, були розроблені вдосконалені версії SHA-2 та SHA-3.

Алгоритми SHA:

- SHA-1. Створює 160-бітний хеш. Раніше він був основним варіантом для систем безпеки, сьогодні він вважається застарілим та вразливим до колізій.
- SHA-2. Включає декілька версій з різним розміром хешів: SHA-224, SHA-256, SHA-384, та SHA-512. Зараз використання цих алгоритмів є стандартом у забезпеченні вищого рівня безпеки.
- SHA-3. Розроблений на основі алгоритму Keccak, SHA-3 був переможцем конкурсу NIST в 2007 році для створення нового стандарту хешування. SHA-3 доповнює SHA-2, запропонуючи збереження рівня безпеки навіть у випадку можливих слабкостей SHA-2.

Процес хешування з алгоритмами SHA має наступні основні кроки:

1. Підготовка повідомлення. Розбиття вхідного повідомлення на блоки певного розміру з додаванням наприкінці спеціальної доповнювальної послідовності.

2. Ініціалізація. Застосування початкові значення хешу, для початку процесу хешування.
3. Компресійна функція. Обробка кожного блоку повідомлення з оновленням проміжного хеш-стану.
4. Фінал. Завершальний результат процесу хешування отримується як кінцевий стан хеш-функції.

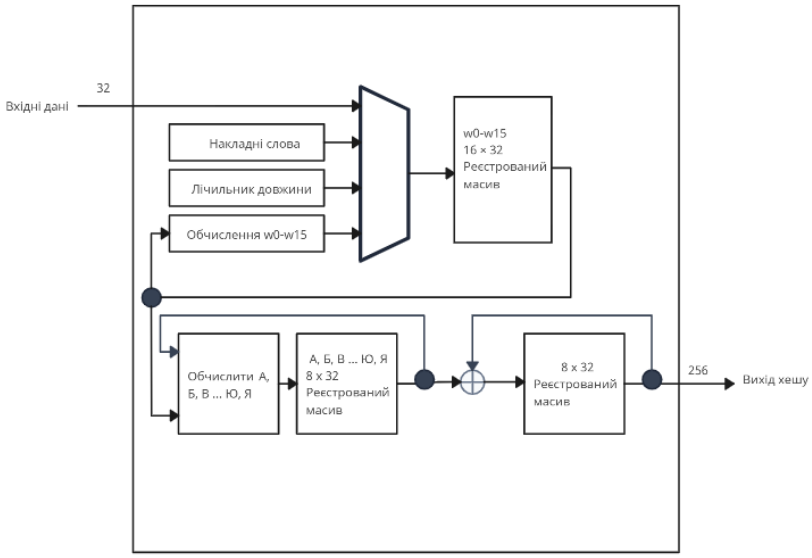


Рисунок 2.5 – Алгоритм хешування SHA-256

Алгоритми SHA знаходять застосування в різноманітних областях: від перевірки цілісності даних до застосувань у сфері криптовалют та блокчейн технологій. Функції хешування SHA є фундаментом для цифрових підписів, систем шифрування та автентифікації, тому саме її для шифрування повідомлень я буду використовувати.

2.7 Стандарт асиметричного шифрування RSA (Rivest-Shamir-Adleman).

Алгоритм RSA, що отримав назву від прізвищ своїх винахідників — Рональда Ривеста, Аді Шаміра та Леонарда Адлемана, був сформульований і представлений у 1977 році. Як один із перших алгоритмів, що дозволяє як

шифрування, так і цифровий підпис, RSA став важливим кроком в застосуванні теорії чисел і складність факторизації чисел як основу для забезпечення безпеки.

RSA використовує систему з двох ключів — публічного та приватного для шифрування та розшифрування даних. Ось як відбувається їх генерація:

1. Вибір простих чисел. Вибираються два великі прості числа (p) і (q).
2. Модуль (n) - обчислюється їх добуток ($n = p \times q$), який використовується як модуль для ключів.
3. Функція Ейлера - визначається за формулою ($\phi(n) = (p-1) \times (q-1)$).
4. Відкрита експонента (e) - вибирається число (e), що задовольняє умовам ($1 < e < \phi(n)$) і ($\text{gcd}(e,\phi(n)) = 1$).
5. Закрита експонента (d) - обчислюється (d) так, що ($d = e^{-1} \mod \phi(n)$).

Процеси шифрування та розшифрування виконується наступним чином:

– Шифрування. Щоб зашифрувати повідомлення (M), перетворюють його у число, менше (n), та використовують формулу ($C = M^e \mod n$), де (C) - шифртекст.

– Розшифрування. Для відновлення первісного повідомлення (M) з шифртексту (C) використовують приватний ключ із формулою ($M = C^d \mod n$).

Безпека RSA ґрунтується на складності задачі факторизації великих чисел. Постійний розвиток обчислювальних технологій, зокрема створення квантових комп'ютерів, може у майбутньому вплинути на надійність RSA, тому рекомендується використання досить великих ключів, наприклад, 2048 бітів або більше.

2.7 Стандарт асиметричного шифрування ECDSA (Elliptic Curve Digital Signature Algorithm).

ECDSA (Elliptical Curve Digital Signature Algorithm) використовує принципи еліптичних кривих для створення більш компактних і безпечних ключів, порівняно з традиційними методами цифрових підписів як DSA. З'явившись у 1990-х роках, ECDSA впровадив революцію у сфері криптографічного захисту, ставши особливо популярним у технології блокчейн, включаючи Bitcoin.

Алгоритм базується на властивостях еліптичних кривих над конечним полем. Структура ECDSA включає наступні основні аспекти:

– Вибір кривої та параметрів: Залежно від вимог безпеки, вибирається відповідна еліптична крива і її параметри.
– Генерація ключів: Приватний ключ (d) вибирається випадково, а публічний ключ (Q) обчислюється як ($Q = dG$), де (G) - базова точка кривої.

Алгоритм створення підпису ECDSA описано в наступних кроках:

1. Вибрати випадкове (k) з діапазону $[1, (n-1)]$, де (n) - порядок точки (G).
2. Обчислити точку кривої ($(x\_1, y\_1) = kG$).
3. Вирішити ($r = x\_1 \backslash mod\, n$). Переконатись, що ($r \backslash neq\, 0$); інакше вибрати інше (k).
4. Обчислити ($s = k\^{}\{-1\}(H(m) + dr) \backslash mod\, n$), де (H(m)) - хеш повідомлення.
5. Відсилка ((r, s)) як підпис повідомлення.

Перевірка підпису:

1. Перевірити чи (r) і (s) у встановлених межах.
2. Обчислити ($w = s\^{}\{-1\} \backslash mod\, n$).
3. Обчислити ($u\_1 = H(m)w \backslash mod\, n$) і ($u\_2 = rw \backslash mod\, n$).
4. Визначити точку ($(x\_1, y\_1) = u\_1G + u\_2Q$).
5. Якщо ($x\_1 \backslash mod\, n = r$), підпис валідний.

Переваги ECDSA в тому, що в нього вищий рівень безпеки при порівняно меншому розмірі ключа, швидша обробка та менше споживання пам'яті та

можливість вибору різних еліптичних кривих для задоволення специфічних вимог.

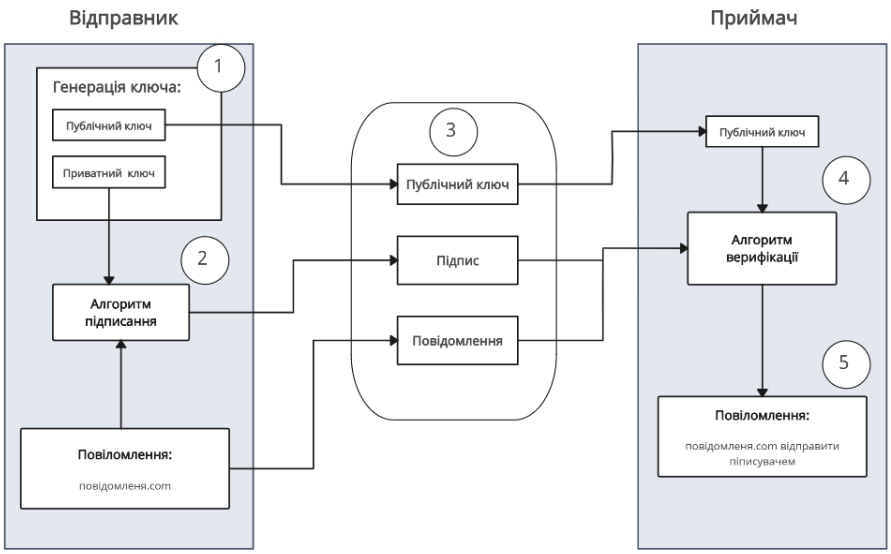


Рисунок 2.6 – Алгоритм роботи ECDSA

ECDSA надає сучасний, потужний та гнучкий інструментарій для гарантування цифрової безпеки. Його широке застосування в технологіях кібербезпеки і блокчейн показує значущість та ефективність алгоритму в різних сферах.

2.8 Висновки з розділу

У цьому розділі було аналізовано методи шифрування: від симетричних до асиметричних технологій:

- DES (Data Encryption Standard), зокрема: його історію, технічні характеристики та алгоритм роботи.
- 3DES (Triple DES), а саме: чим відрізняється від попереднього DES.
- AES (Advanced Encryption Standard), зокрема: історії, характеристики алгоритми роботи, склавши блок схему.
- MD5 (Message-Digest Algorithm 5), а саме: яка роль у сучасному інформаційному просторі.

– SHA (Secure Hash Algorithm), зокрема: аналіз алгоритму побудова, процес хешування.
– RSA (Rivest–Shamir–Adleman), а саме: як відбувається генерація ключів, процеси шифрування та розшифрування.
– ECDSA (Elliptic Curve Digital Signature Algorithm), зокрема: структура, алгоритми створювання підпису, перевірка підпису та переваги.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОКРАЩЕННЯ АВТЕНТИФІКАЦІЇ

3.1 Автентифікація за допомогою пароля

Автентифікація за допомогою пароля є одним із найбільш розповсюджених методів ідентифікації користувачів у веб- та мобільних додатках. Цей підхід забезпечує базовий рівень безпеки, оскільки доступ користувача до системи обмежується знанням пароля, який є секретним елементом інформації. Зберігання хешованих паролів у базі даних дозволяє знизити ризик викрадення паролів, навіть якщо дані будуть скомпрометовані. Такий метод широко використовується у різних сферах, включаючи електронну комерцію, соціальні мережі та банківські додатки.

Попри те, що метод надійний, він має свої обмеження. Наприклад, існує ризик, що користувач може вибрати слабкий пароль, який легко піддається атакам типу "грубої сили" (brute force). Тому методи шифрування та зберігання паролів повинні бути досить стійкими для протидії можливим загрозам. У даному додатку для зменшення ризиків використовується зберігання хешованих значень паролів та додаткові криптографічні алгоритми для шифрування повідомлень під час їх передачі.

3.1.1 Зберігання даних користувача в базі даних на сервері

Для організації безпечного доступу користувачів до сервера за допомогою пароля потрібно зберігати дані користувача в базі даних. Основна інформація, необхідна для автентифікації, включає:

- Ім'я користувача (унікальний ідентифікатор клієнта).
- Зашифрований пароль або ключ дешифрування для перевірки користувача.

Ці дані зберігаються у захищеному вигляді на сервері та використовуються під час процесу автентифікації для порівняння переданого повідомлення та даних в базі даних.

3.1.2 Процес автентифікації: передача зашифрованого повідомлення та його дешифрування на сервері

Процес автентифікації за допомогою пароля включає такі етапи:

1. Користувач вводить свої дані (ім'я користувача та пароль) у додатку на Android.
2. Додаток шифрує повідомлення, яке містить ці дані, використовуючи стандартні криптографічні методи.
3. Шифроване повідомлення передається на сервер для обробки.
4. Сервер отримує повідомлення та розшифровує його, використовуючи збережений ключ дешифрування.

Сервер порівнює розшифровані дані з інформацією в базі даних: Якщо дані збігаються, автентифікація успішна, і сервер надає доступ. Якщо дані не збігаються, сервер повертає помилку автентифікації.

На рис. 3.1. Зображено алгоритм автентифікації.

Рисунок 3.1 – Алгоритм автентифікації паролем

3.2 Автентифікація за допомогою цифрового підпису

Автентифікація за допомогою цифрового підпису є сучасним методом, який забезпечує високий рівень безпеки без необхідності дешифрування повідомлень на сервері. Цей метод базується на використанні асиметричних криптографічних ключів: приватного ключа, який зберігається на стороні клієнта і використовується для підпису повідомлення, та публічного ключа, який зберігається на сервері і використовується для перевірки підпису. Підписані повідомлення забезпечують цілісність даних та автентичність відправника, оскільки будь-яка зміна повідомлення порушить його підпис, що зробить його недійсним.

Цифрові підписи широко використовуються у сфері електронного документообігу, банківських послуг та інших високозахищених систем, оскільки вони дозволяють підтверджувати особу відправника без потреби передавати або зберігати паролі. Використання асиметричних ключів також знижує ризик несанкціонованого доступу до конфіденційних даних, оскільки для створення підпису потрібно мати доступ до приватного ключа, який зберігається лише на пристрої клієнта.

3.2.1 Використання асиметричних ключів для підпису та перевірки підписаних повідомлень

Для створення цифрового підпису клієнт використовує свій приватний ключ для підпису повідомлення. Після підпису додаток надсилає підписане повідомлення на сервер. Сервер отримує повідомлення та використовує публічний ключ клієнта для перевірки підпису.

3.2.2 Процес перевірки автентичності повідомлення без необхідності його дешифрування на сервері

Процес перевірки автентичності повідомлення за допомогою цифрового підпису складається з таких етапів:

1. Клієнт підписує повідомлення приватним ключем.
2. Підписане повідомлення передається на сервер.
3. Сервер отримує повідомлення та підпис і використовує публічний ключ для перевірки підпису.
4. Якщо підпис збігається, сервер підтверджує автентичність повідомлення.
5. У разі невідповідності підпису сервер повертає помилку автентифікації.

Цей підхід дозволяє уникнути необхідності дешифрування повідомлення на сервері, що зменшує навантаження на сервер та підвищує швидкість обробки запитів.

На рис. 3.2. Зображено алгоритм автентифікації для підписаного ключа.

Рисунок 3.2 – Алгоритм перевірки автентичності повідомлення за допомогою цифрового підпису

3.3 Розробка додатка для тестування підходів

Для тестування автентифікації було створено додаток на базі Xamarin для клієнтської сторони, що забезпечує кросплатформенну розробку, та сервер на

Flask для обробки запитів. Використання цих технологій дозволяє реалізувати обидва підходи: автентифікацію за допомогою пароля (через шифрування повідомлень) та автентифікацію за допомогою цифрового підпису. Завдяки цим методам клієнт може надсилати дані для перевірки на сервері, а сервер обробляє та логує результати для подальшого аналізу швидкодії та надійності.

3.3.1 Вибір платформи Xamarin для кросплатформеної розробки Android додатка

Платформа Xamarin була обрана для створення кросплатформеного Android-додатка, оскільки вона дозволяє використовувати єдину базу коду для розробки додатків для кількох операційних систем. Це значно знижує витрати на підтримку та розвиток додатка, забезпечуючи при цьому доступ до необхідних нативних функцій Android.

3.3.2 Інтеграція серверної частини на Flask для обробки запитів від клієнтського додатка

На серверній стороні використовується Python Flask для побудови API, який обробляє запити від клієнтського додатка. Сервер має функції для:

- прийому та обробки зашифрованих повідомлень;
- перевірки цифрових підписів;
- логування результатів обробки в CSV файл.

Код сервера включає функції для дешифрування повідомлень, обробки підписаних повідомлень, та логування часу обробки та результатів для подальшого аналізу.

На рис. 3.3. Зображено загальний алгоритм роботи

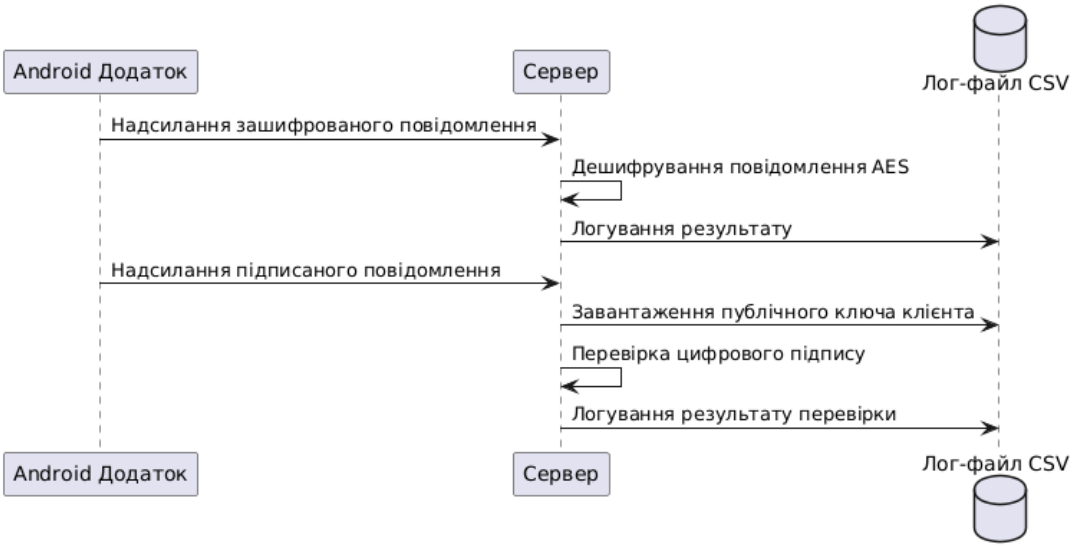


Рисунок 3.3 – Загальний алгоритм роботи

3.3.3 Реалізація функціоналу двох видів автентифікації через пароль і цифровий підпис

Для реалізації функціоналу двох видів автентифікації – через пароль і цифровий підпис – у мобільному застосунку на Android використано наступні технології та підходи:

1) Автентифікація через пароль проходить за наступними етапами:

– Шифрування повідомлень**:** Для захисту даних, що передаються, застосунок використовує алгоритм симетричного шифрування AES (Advanced Encryption Standard) із ключем, який генерується на основі пароля. Функція GenerateKey створює 256-бітний ключ, застосовуючи алгоритм хешування SHA-256 до пароля користувача. Цей ключ використовується для шифрування тексту повідомлення у функції EncryptMessage.

– Передача зашифрованих повідомлень**:** Шифроване повідомлення відправляється на сервер за допомогою HttpClient. Запит формується у вигляді JSON і надсилається на сервер для розшифрування і обробки. Час на відправку та отримання відповіді вимірюється для подальшого аналізу швидкодії.

2) Автентифікація через цифровий підпис:

– Генерація ключової пари ECDSA**:** Для реалізації автентифікації через цифровий підпис застосовується алгоритм ЕЦП (еліптичної криптографії) ECDSA на основі кривої SECP256k1. За допомогою бібліотеки BouncyCastle генерується ключова пара (відкритий і приватний ключі) за заданими параметрами кривої. Відкритий ключ завантажується на сервер для подальшої перевірки підписів.

Для контролю використовувалось логування в консоль. На рис. 3.4 зображено результат логування ключа в консолі.

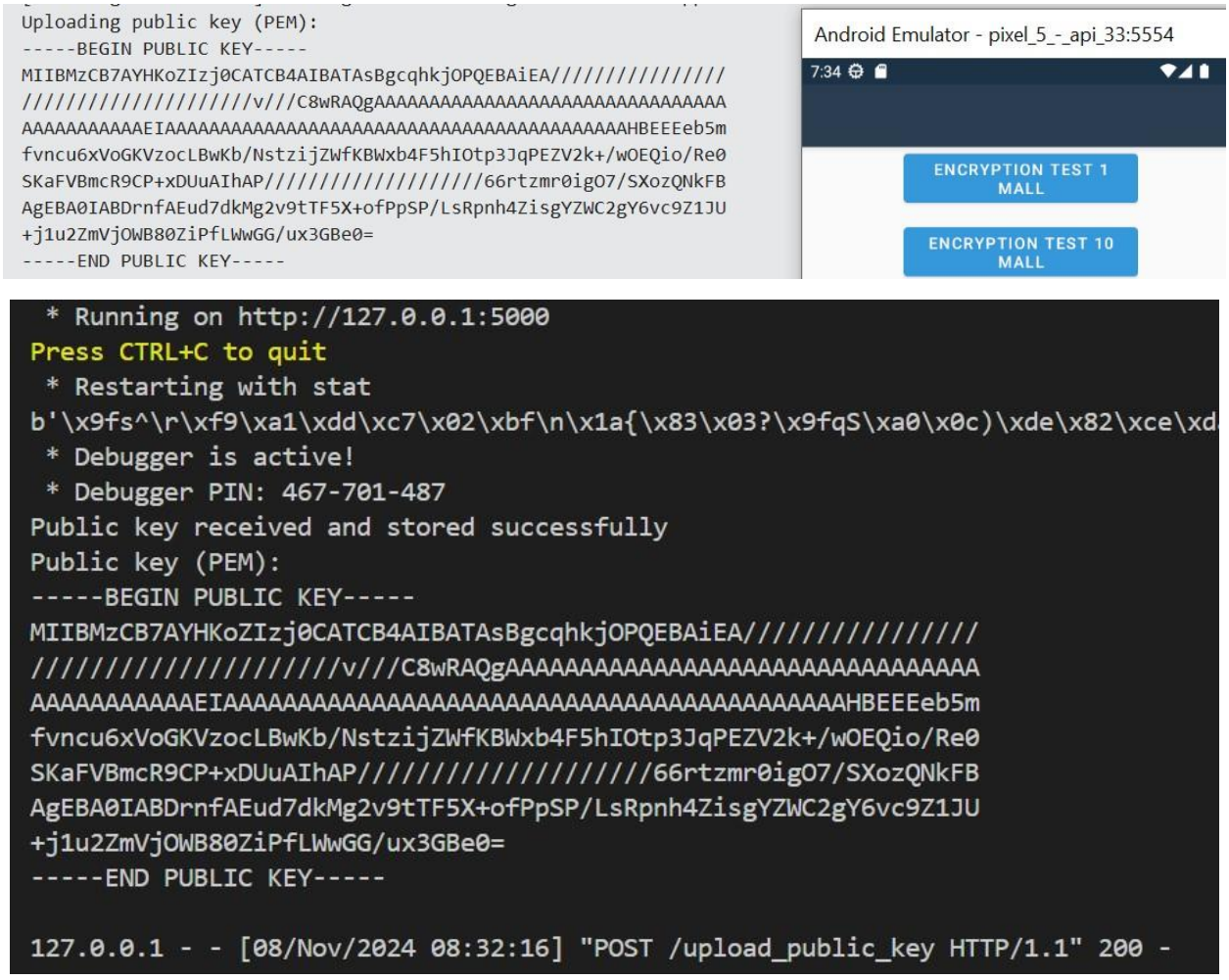


Рисунок 3.4 – Логування ключа в консолі

Після передачі ключа можна відправляти повідомлення:

– Підписування повідомлень: Повідомлення підписується за допомогою функції SignMessage, яка застосовує приватний ключ для створення

цифрового підпису повідомлення. Підпис кодується у форматі DER і конвертується у Base64, що полегшує його передачу у JSON-форматі.

– Передача підписаних повідомлень: Підписане повідомлення, разом із підписом, надсилається на сервер. Там перевіряється справжність підпису з використанням попередньо завантаженого відкритого ключа. Це гарантує, що повідомлення дійсно надіслане авторизованим користувачем.

На рис. 3.5. Зображено загальний алгоритм двох видів автентифікації

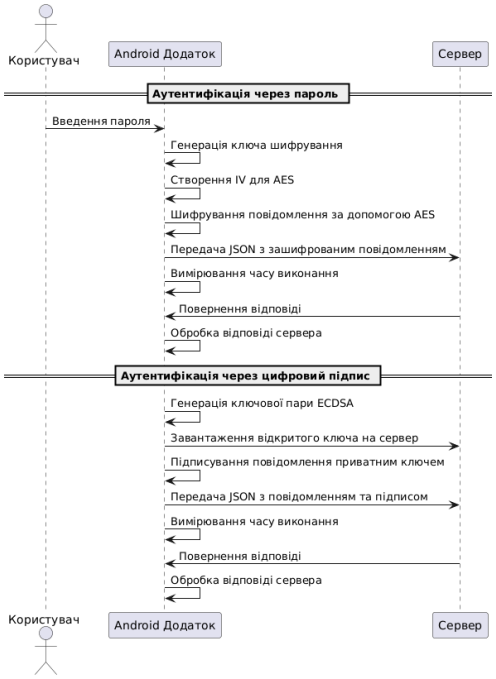


Рисунок 3.5 – Реалізація функціоналу двох видів автентифікації

На рис. 3.6. Зображено логування передачі та відповідь сервера

```
Sending message: In the heart of a bustling city
Message bytes: 73, 110, 32, 116, 104, 101, 32, 104, 101, 97, 114, 116, 32, 111, 102, 32, 97, 32, 98
Generated signature (base64): MEQCIDDQ3semc6Wt46f1oPCFrKrkL4V+xW9zhkMexiB3b6vIAiAh7/TnLWJG3Ai5f+vQa
Generated signature (bytes): MEQCIDDQ3semc6Wt46f1oPCFrKrkL4V+xW9zhkMexiB3b6vIAiAh7/TnLWJG3Ai5f+vQa4
Response: {
  "count_a": 2,
  "processing_time": 0.0
}
Time taken: 142.747 ms
```

```
127.0.0.1 - - [08/Nov/2024 08:32:16] "POST /upload_public_key HTTP/1.1" 200 -
Received message: In the heart of a bustling city
Message bytes: [73, 110, 32, 116, 104, 101, 32, 104, 101, 97, 114, 116, 32, 111, 102, 32, 97, 32, 98, 117, 115, 1
Received signature (base64): MEQCIDDQ3semc6Wt46f1oPCFrKrkL4V+xW9zhkMexiB3b6vIAiAh7/TnLWJG3Ai5f+vQa4AKPgokcAnBtGes
Signature verification: success
127.0.0.1 - - [08/Nov/2024 08:34:33] "POST /process_signed_message HTTP/1.1" 200 -
```

Рисунок 3.6 – Логування передачі та відповідь сервера

3.3.4 Важливість кросплатформенності та гнучкості рішення

Кросплатформенність дозволяє реалізувати додаток, який може працювати на різних операційних системах і пристроях (наприклад, Android, iOS, Windows). Це суттєво спрощує підтримку програми та знижує витрати на її розробку і тестування. Додаток написаний за допомогою таких технологій, як Xamarin та інші кросплатформенні інструменти, що гарантує роботу однаково на різних платформах, зменшуючи час на адаптацію коду.

Гнучкість рішення досягається завдяки використанню стандартних криптографічних алгоритмів (ECDSA, AES), які сумісні на багатьох платформах і бібліотеках. Це дозволяє легко замінити або модифікувати окремі компоненти без істотних змін в архітектурі додатку, що покращує підтримуваність і адаптивність рішення.

Забезпечення гнучкості та кросплатформенності дозволяє легко інтегрувати нові функціонали (наприклад, нові методи автентифікації) та адаптувати програму до змін вимог без значних витрат.

На рис. 3.7. Зображено загальний алгоритм проекту

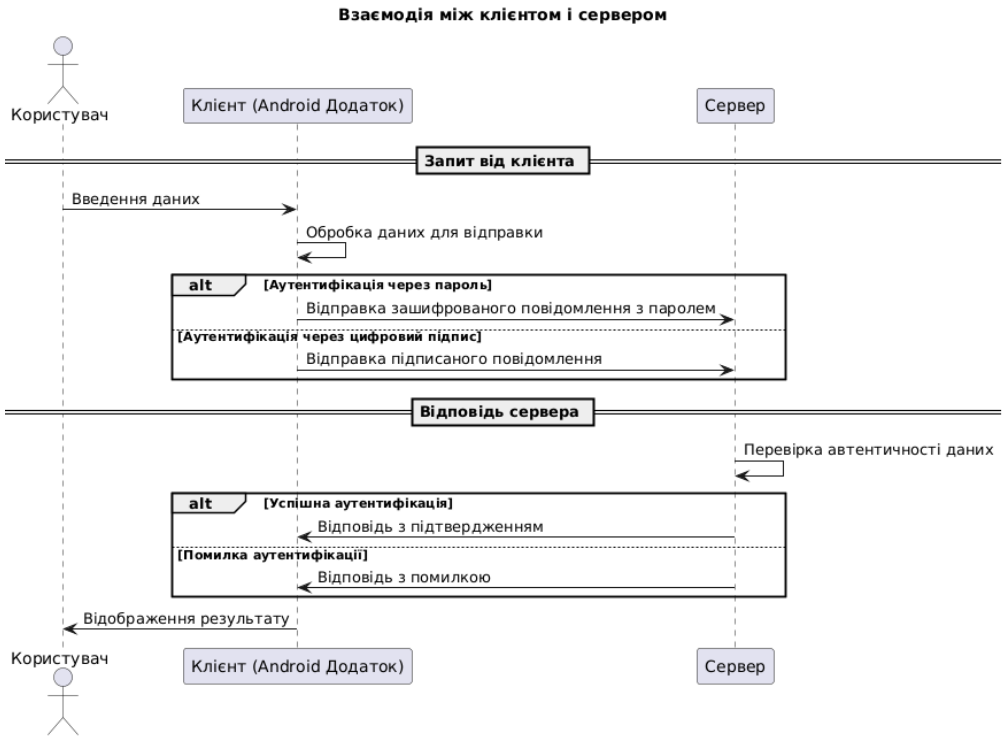


Рисунок 3.7 – Загальний алгоритм взаємодії між клієнтом і сервісом

3.4 Експеримент

3.4.1 Опис проведення експерименту для вимірювання швидкодії обох підходів

Для оцінки продуктивності методів автентифікації було проведено серію експериментів з різними довжинами повідомлень. Кожен експеримент складався з декількох етапів:

1. Для кожного повідомлення певної довжини виконувалось 10 окремих вимірювань швидкості передачі та обробки.
2. Після кожного набору з 10 вимірювань довжина повідомлення збільшувалася шляхом додавання повторюваного тексту, а експеримент повторювався.

```
// по 10 раз
Toast.MakeText(this, "Test started!", ToastLength.Short).Show();
string nMess = "";
string endpoint = "/process_signed_message_ohneLog";
string s;
var resultsTextView = FindViewById<TextView>(Resource.Id.resultsTextView);
resultsTextView.Text = "Тест з підписом \n";
for (int x = 0; x < 10; x++)
{
    for (int i = 0; i < 10000; i++)
    {
        nMess += message;
    }
    resultsTextView.Text += "Тест " + nMess.Length + " символів \n";
    for (int j = 0; j < 10; j++)
    {
        s = await SendSignedMessageWithTimeMeasurement(nMess, endpoint, false);
    }

}
resultsTextView.Text += "Тест завершено\n";
```

Рисунок 3.8 – Код для проведення експерименту

Результати кожного вимірювання фіксувалися для подальшого аналізу, щоб визначити середню та максимальну швидкість для кожного типу автентифікації (через пароль або цифровий підпис). Для реалізації експерименту було використано метод, який запускає серію тестів для цифрового підпису з поступовим збільшенням довжини повідомлення.

На рис. 3.9. Зображено алгоритм проведення експерименту

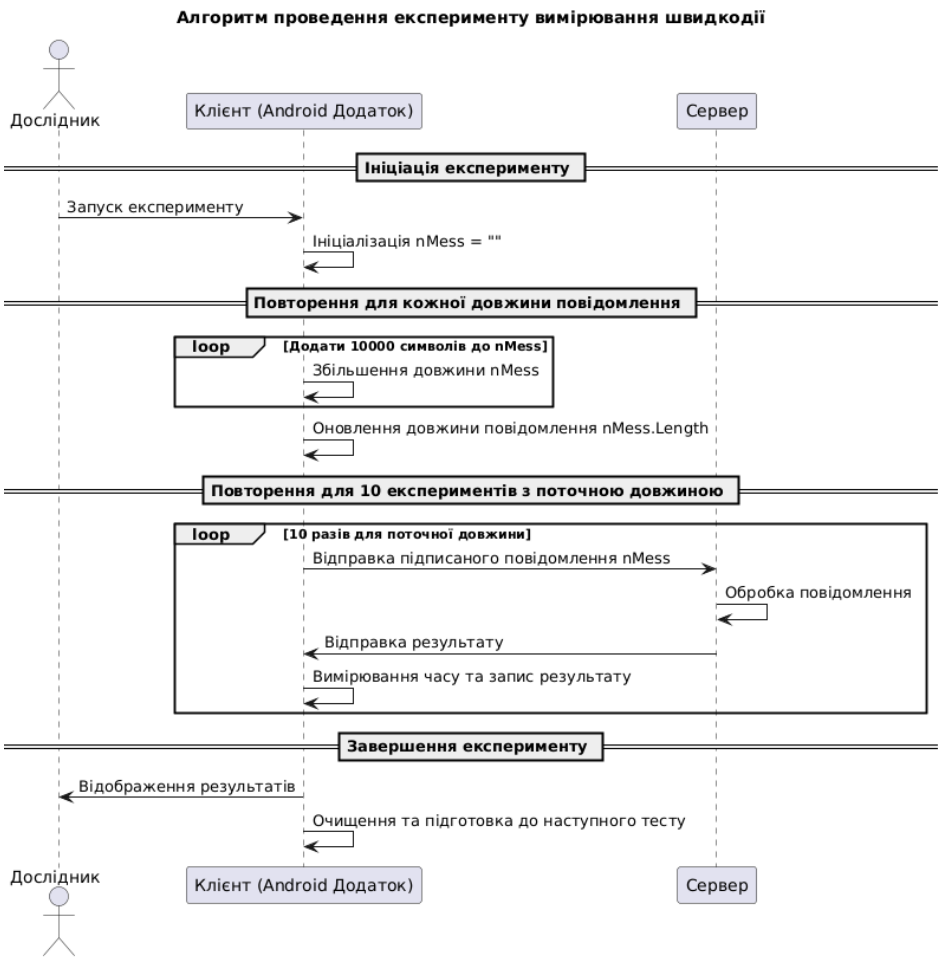


Рисунок 3.9 – Алгоритм проведення експерименту вимірювання швидкості

3.4.2 Оцінка результату

Після проведення експерименту були побудовані графічні залежності часу обробки від довжини повідомлення для кожного з підходів — автентифікації через пароль та автентифікації за допомогою цифрового підпису.

Для кожного набору тестів були зібрані значення часу обробки, які потім використовувалися для аналізу та побудови графіків. Це дозволило оцінити вплив довжини повідомлення на продуктивність обох методів. Завдяки графічним залежностям було виявлено тенденції зміни часу обробки разом.

На рис. 3.10. Зображено інтерфейс додатку при тестуванні

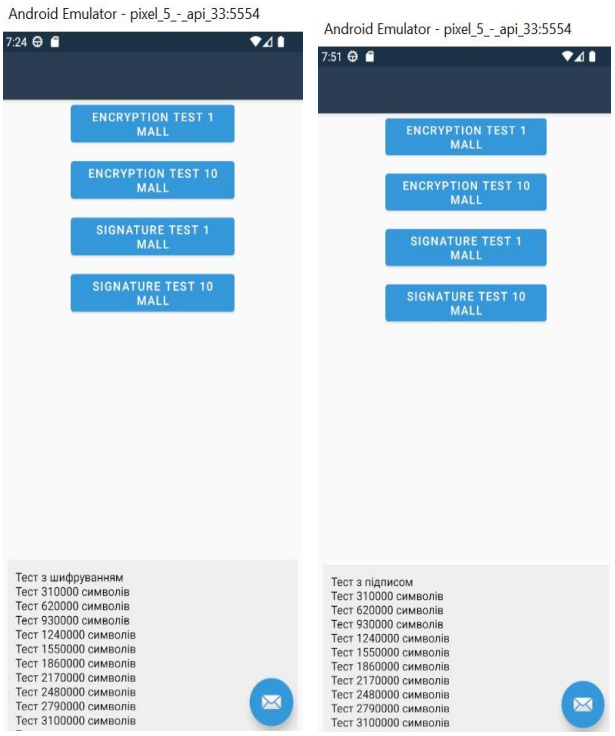


Рисунок 3.10 – Інтерфейс додатку при тестуванні

Слід додати, що вимірювання часу обробки проводилось саме по 10 разів для кожної наступної довжини повідомлення, виходячи з різної завантаженості процесора під час тестування. Ми звісно отримаємо дещо різні результати, по яким для даної ітерації можна обчислити середній час

Так для підписаних повідомлень отримані наступні результати при довжині повідомлення 310000 символів

Час запиту	Довжина повідомлення		Час обробки на сервері (с)	Результат обробки	
08.11.2024 7:47	310000	0,31	0,028	20000	signed success
08.11.2024 7:47	310000	0,31	0,030	20000	signed success
08.11.2024 7:47	310000	0,31	0,037	20000	signed success
08.11.2024 7:47	310000	0,31	0,030	20000	signed success
08.11.2024 7:47	310000	0,31	0,029	20000	signed success
08.11.2024 7:47	310000	0,31	0,039	20000	signed success
08.11.2024 7:47	310000	0,31	0,032	20000	signed success
08.11.2024 7:47	310000	0,31	0,024	20000	signed success
08.11.2024 7:47	310000	0,31	0,031	20000	signed success
08.11.2024 7:47	310000	0,31	0,026	20000	signed success

Рисунок 3.11 – Результат підписаних повідомлень

Побудована графічна залежність часу обробки від номера експерименту.

Рисунок 3.12 – Графічна залежність для підписаних повідомлень

Таким самим чином було оброблено повідомлення, але за допомоги шифрування.

Дата	Довжина повідомлення	Час обробки	Результат	
08.11.2024 7:40	310000	0,039	20000	encrypted success
08.11.2024 7:40	310000	0,040	20000	encrypted success
08.11.2024 7:40	310000	0,040	20000	encrypted success
08.11.2024 7:40	310000	0,044	20000	encrypted success
08.11.2024 7:40	310000	0,036	20000	encrypted success
08.11.2024 7:40	310000	0,040	20000	encrypted success
08.11.2024 7:40	310000	0,039	20000	encrypted success
08.11.2024 7:40	310000	0,033	20000	encrypted success
08.11.2024 7:40	310000	0,037	20000	encrypted success
08.11.2024 7:40	310000	0,029	20000	encrypted success

Рисунок 3.13 – Результат шифрованих повідомлень

Рисунок 3.14 – Графічна залежність для шифрованих повідомлень

Набагато інформативнішим виявились залежності при поступовій збільшенні довжини повідомлення. Так для підпису і шифрування побудовані наступні залежності. Це залежності часу обробки (в секундах) від довжини повідомлення (в мегасимволах).

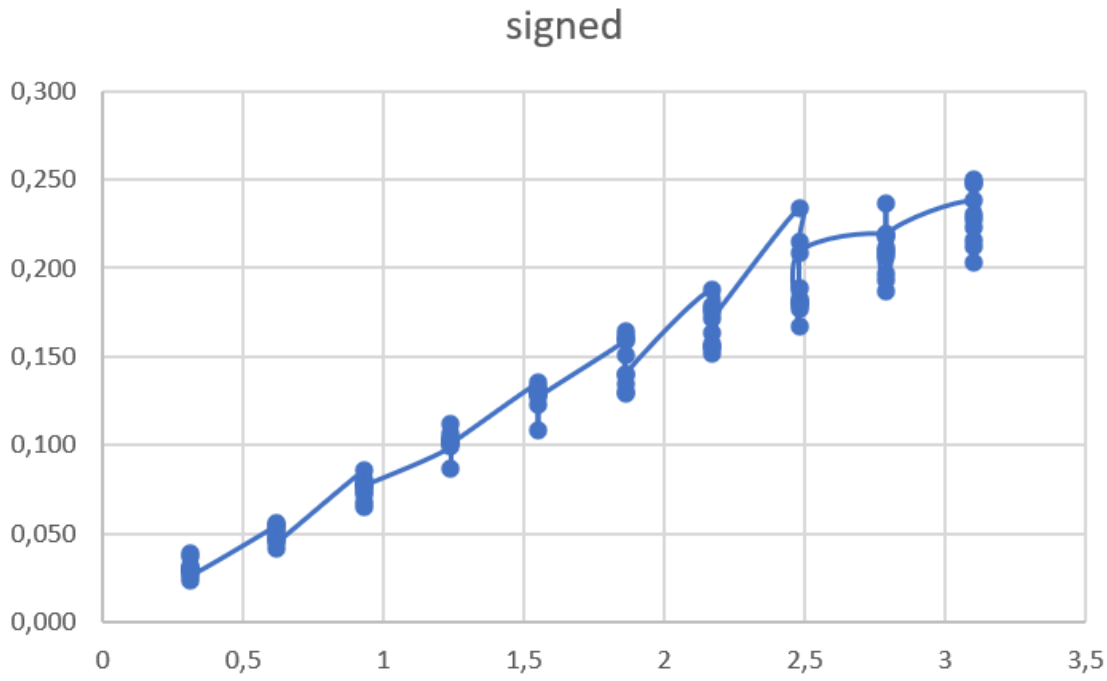


Рисунок 3.15 – Графічна залежність для підписаних повідомлень в секундах

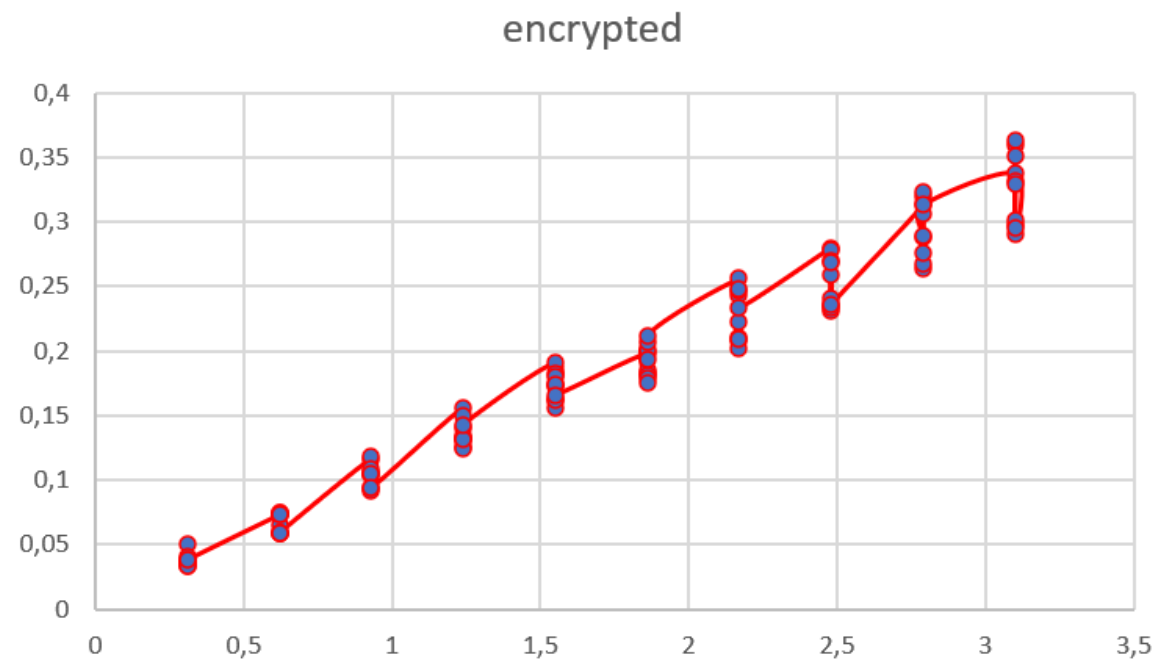


Рисунок 3.16 – Графічна залежність для шифрованих повідомлень в секундах

Фінальний результат можна побачити побудувавши залежності на одній координатній площині. А загальну тенденцію можна побачити апроксимувавши залежності наприклад лінійним трендом. Залежності представлені нижче.

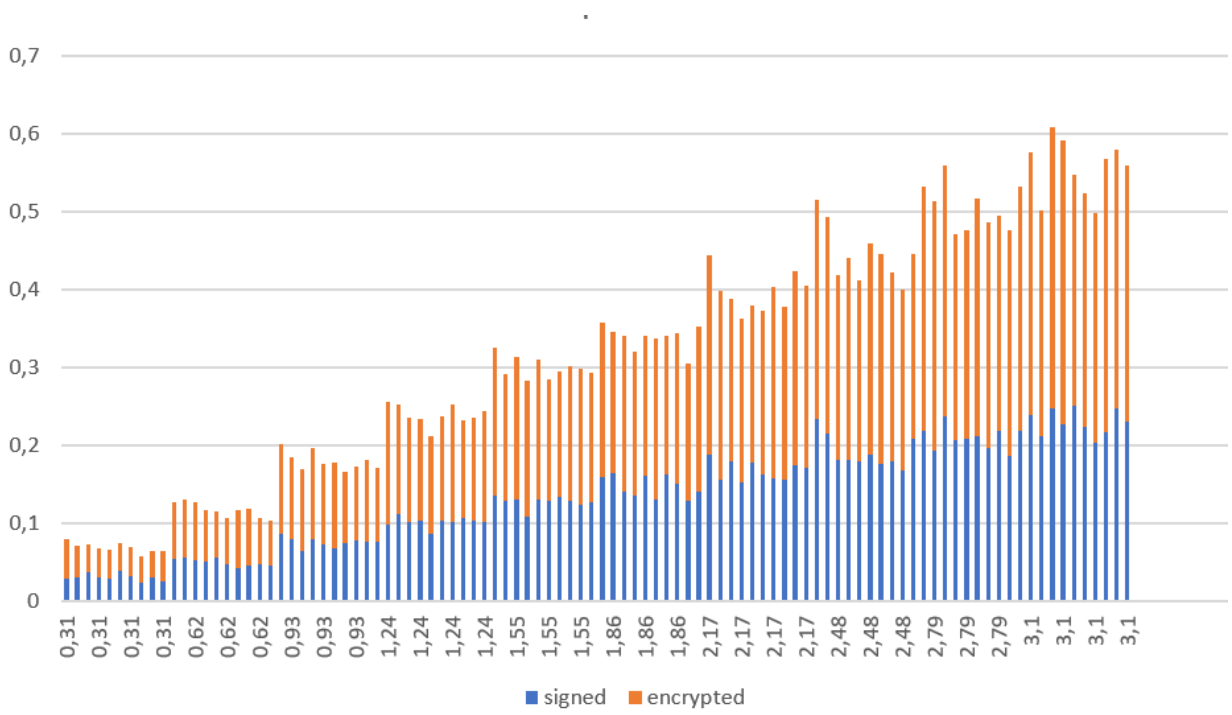


Рисунок 3.17 – Залежності на одній координатній площині

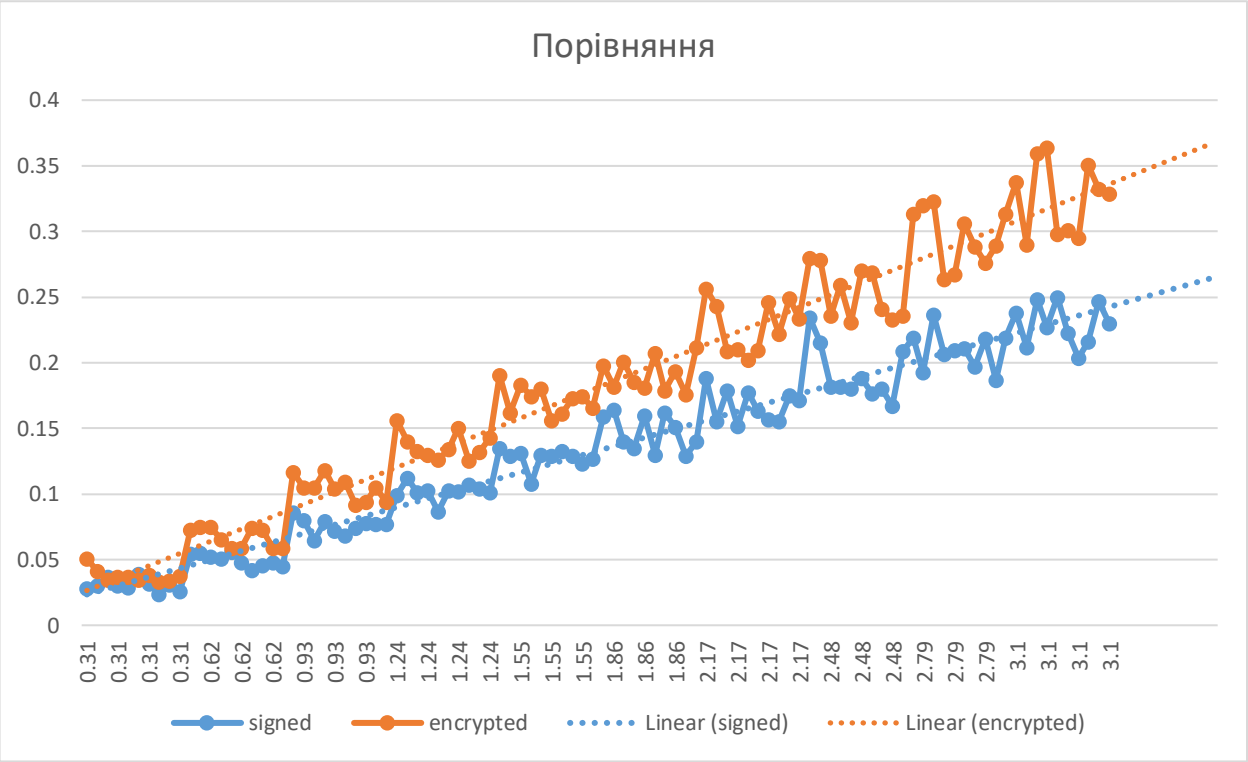


Рисунок 3.17 – Різниця шифрування та підпису

Незважаючи на то, що на сервері отримано значну перевагу в швидкості обробки підписаного повідомлення, було виміряно середній час від формування запиту на сервер для додатку.

Можна було бачити, що для тестової довжини повідомлення 310000 символів ми маємо значне з збільшення часу очікування відповіді. 592 мс для підпису проти 50 мс для шифрування. На коротких повідомленнях швидкість обробки на сервері майже однакова. Було висловлено припущення, що підписання повідомлення для цього алгоритму займає багато часу.

Цифровий підпис, особливо на основі алгоритму ECDSA (еліптичні криві), вимагає значних обчислювальних ресурсів, особливо при збільшенні розміру вхідного повідомлення. Хоча сама операція підпису не змінюється зі збільшенням довжини повідомлення, час обробки зростає через додаткову обробку, пов'язану з хешуванням великих обсягів даних. Для цифрового підпису система повинна спочатку обчислити хеш повідомлення, і це може значно збільшити час обробки для довгих повідомлень. Шифрування зазвичай реалізується з використанням симетричних алгоритмів (наприклад, AES), які

відомі своєю високою швидкістю та ефективністю. Для шифрування довгих повідомлень симетричні алгоритми обробляють інформацію блоками, що в результаті часто займає менше часу, ніж підписання.

Проведений експеримент показав, що час обробки повідомлень на сервері для обох підходів (цифровий підпис і шифрування) залежить від довжини повідомлення, хоча результати для довгих і коротких повідомлень мають деякі ключові відмінності.

При довжині повідомлення 3.1 мільйона символів час обробки на сервері склав:

- 25 мс для підписаного повідомлення,
- 35 мс для зашифрованого повідомлення.

При довжині повідомлення 310 тисяч символів обробка обох типів повідомлень тривала близько 3 мс. Однак затримка для отримання відповіді на клієнті зростала майже вдвічі для всіх випадків, при цьому підписане повідомлення потребувало більше часу, ніж зашифроване.

З огляду на отримані дані, було виявлено лінійну залежність часу обробки повідомлення від його довжини.

Так для шифрування це $y=10.75 \cdot x$, а для підпису це $y=7.17 \cdot x$, де (y) – час в мілісекундах, (x) – довжина в мегасимволах

3.5 Загальні висновки щодо доцільності застосування кожного з підходів для автентифікації Android-додатків у майбутніх проектах

На основі експериментальних результатів можна зробити такі висновки щодо застосування обох підходів для автентифікації Android-додатків:

1. Шифрування повідомлень — оптимальний варіант для коротких повідомлень, оскільки він демонструє стабільний час обробки на сервері і меншу затримку відповіді для клієнта. Це доцільно, якщо швидкість є критичною, а вимоги до підтвердження цілісності не є першочерговими.

2. Цифровий підпис — підходить для довгих повідомлень, оскільки забезпечує кращу продуктивність на сервері та підвищує рівень безпеки завдяки автентифікації відправника і перевірці цілісності повідомлення. Використання цифрового підпису також може знизити загальний обсяг обчислень для серверної частини при обробці великих повідомлень.

Рекомендації щодо майбутніх проектів в якості покращення: для ефективного поєднання безпеки і швидкодії можна розглянути комбінований підхід. Зокрема, використовувати шифрування для коротких повідомлень **та** цифровий підпис для довгих повідомлень**.** Такий підхід дозволить забезпечити збалансовану швидкість і безпеку, оптимізуючи час обробки і знижуючи затримку відповіді для користувача.

ЕКОНОМІЧНА ЧАСТИНА

4.1 Технологічний аудит розробки

Конкурентоспроможність розкривається через систему якісних та економічних показників.

Для проведення оцінювання необхідно отримати оцінки експертів відповідно кожного критерія і обчислити середню арифметичну оцінку, яка визначатиме науково-технічний рівень та комерційний потенціал розробки.

Метою проведення технологічного аудиту є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності, тобто під час виконання магістерської кваліфікаційної роботи на тему «Метод та засіб автентифікації застосунків в операційній системі Android»

Експертами виступають:

1) Каплун Валентина Аполінаріївна — старший викладач кафедри ЗІ;
2) Хошаба Олександр Мирославович — к.т.н., доцент кафедри ПЗ;
3) Гарнага Володимир Анатолійович — к.т.н., доцент кафедри ЗІ;

Оцінювання проведено за показниками, що наведені в Додатку, оцінки цих показників, виражені в балах, представлені в таблицях 4.1 і 4.2.

Таблиця 4.1 – Показники ступеня новизни науково-дослідної роботи.

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	2	2	2
2. Ринкові переваги (наявність аналогів)	1	1	1
3. Ринкові переваги (ціна продукту)	3	3	2
4. Ринкові переваги (технічні властивості)	2	2	3
5. Ринкові переваги (експлуатаційні витрати)	3	2	3
6. Ринкові перспективи (розмір ринку)	2	3	2
7. Ринкові перспективи (конкуренція)	3	2	2
8. Практична здійсненність (наявність фахівців)	2	2	3
9. Практична здійсненність (наявність фінансів)	1	1	1
10. Практична здійсненність (необхідність нових матеріалів)	2	2	2
11. Практична здійсненність (термін реалізації)	3	3	3
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	28	27	28
Середньоарифметична сума балів СБс	27,6		

$$СБ_С = \frac{\sum_{i=1}^{3} СБ_i}{3} = \frac{28 + 27 + 28}{3} = 28$$

За результатами розрахунків, наведених в таблиці 4.1, робиться висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використовують рекомендації, наведені в табл. 4.2.

де:

$СБ_С$ – середньоарифметична сума балів

$СБ_i$ – сума балів і-го експерта

Таблиця 4.2 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вищий середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Досягнутий науково-технічний потенціал дорівнює середньому рівню науково-технічної розробки порівняно з аналогами існуючими на ринку за рахунок розширення функціональних можливостей.

Покращення функціональних можливостей нової науково-технічної розробки порівняно з аналогічними розробками за рахунок, існуючими в цей час на ринку відбулося за рахунок альтернативного підходу, який дозволяє враховувати невизначеності та обраховувати прогноз з тією ж точністю.

Для проведення оцінки конкурентоспроможності шляхом порівняння з аналогами потрібно визначити показники за якими буде проводитись порівняння. Так технічними показниками будуть точність моделі та швидкість обробки запитів. З економічних показників це лише ціна та вартість оновлення змін за певний період часу (2 роки).

Для обчислення одиничних параметричних індексів застосовується формула, як показано на приклад оцінки індексу зміни значення параметра «Точність моделі»:

$$q_{\text{зд}} = \frac{P_{\text{баз зд}}}{P_{\text{зд}}} = \frac{91}{84} = 1{,}1, \quad (4.2)$$

Таблиця 4.3 - Оцінки якісних показників

Параметр	Одиниця виміру	Аналог	Нова розробка	Індекс зміни значення параметра	Коефіцієнт вагомості
Технічні					
Точність моделі	%	84	91	1,1	0,7
Співвідношення шифрування/підпис	%	74	81	1,09	0,3
Економічні					
Ціна	грн	250000	190000	0,76	0,4
Вартість впровадження змін (рік)	грн	32000	19000	0,6	0,6

Для визначення групового параметричного індексу за технічними показниками конкурентоспроможності застосуємо наступну формулу:

$$\text{I}_{\text{ТП}} = \sum_{i=1}^{4} q_i * \alpha_i = 1.1 * 0.7 + 1{,}09 * 0.3 = 1{,}1 \quad (4.3)$$

Для визначення групового параметричного індексу за економічними показниками конкурентоспроможності застосуємо наступну формулу:

$$\text{I}_{\text{ЕП}} = \sum_{i=1}^{2} q_i * \beta_i = 0{,}8 * 0{,}4 + 0{,}6 * 0{,}6 = 0{,}68$$

Інтегральний показник конкурентоспроможності обчислюється за формулою:

$$\text{К}_{\text{ІНТ}} = \text{I}_{\text{НП}} * \frac{\text{I}_{\text{ТП}}}{\text{I}_{\text{ЕП}}} = 1 * \frac{1{,}1}{0{,}68} = 1.6$$

Отриманий інтегральний показник $\text{К}_{\text{ІНТ}} = 1.6$ свідчить про перспективність конкурентоспроможності даної розробки.

З огляду на оцінки експертів та оцінку конкурентоспроможності розробки, можна зробити висновок про високий науково-технічний рівень, комерційний

потенціал та перспективу конкурентоспроможності. В такому випадку можна зробити висновок про потенційну користь від науково-технічної розробки

4.2 Розрахунок витрат на впровадження МКР

Основні витрати потрібно розрахувати за такими статями:

- витрати на оплату праці;
- відрахування на соціальні заходи;
- паливо та енергія для науково-виробничих цілей;
- витрати на службові відрядження;
- спец устаткування для наукових (експериментальних) робіт;
- програмне забезпечення для наукових (експериментальних) робіт;
- витрати на роботи, які виконують сторонні підприємства, установи і організації;
- інші витрати;
- накладні (загальновиробничі) витрати.

4.2.1 Витрати на оплату праці

Основна заробітна плата дослідників

Витрати на основну заробітну плату досліднтків ($З_о$) розраховують відповідно до посадових оклладівпрацівників, за фомулою:

$$З_о = \sum_{i=1}^{k} \frac{(М_{пі} * t_i)}{T_p},$$

де k – кількість посад дослідників, залучених до процесу досліджень;

$М_{пі}$ – місячний посадовий оклад конкретного дослідника, грн;

t_i – кількість днів роботи конкретного дослідника, дн.;

$Т_р$ – середня кількість робочих днів в місяці, $Т_р$=24дні.

Проведені розрахунки зведено до таблиці:

Таблиця 4.4 – Витрати на оплату праці дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Кількість днів роботи	Витрати на заробітну плату, грн
Керівник проекту	20000	833	24	20000

Програміст-розробник	15000	739	18	11000
Аналітик	14000	739	18	10250
Тестувальник	10000	416	24	10000
Всього				51250

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників (Зр) за відповідними найменуваннями робіт НДР на тему «Метод та засіб автентифікації застосунків в операційній системі Android» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i * t_i,$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду Сі можна визначити за формулою:

$$C_i = \frac{M_M * K_i * K_c}{T_p * t_{зм}}$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), грн. Приймемо $М_М$=7400 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середня кількість робочих днів в місяці, приблизно Тр = 24 дні;

$t_{зм}$ – тривалість зміни, год.

$$C_i = \frac{8000 \cdot 1{,}35 \cdot 1{,}8}{24 \cdot 8} = 101{,}25 \text{ грн.}$$

$$З_{рі} = 101{,}25 \cdot 9 = 911{,}25 \text{ грн.}$$

Таблиця 4.5 – Витрати на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год.	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, год.	Величина оплати на робітника, год.
Інсталяція програмного забезпечення	9	3	1,35	101,25	911,25
Налаштування програмних застосунків	5	4	1,5	112,5	562,5
Підготовка дослідження	12	4	1,5	112,5	1350
Тестування	6	3	1,35	101,25	607,5
Формування бази даних результатів дослідження	14	4	1,5	112,5	1575
Всього					5006,25

Додаткова заробітна плата дослідників та робітників. Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) * \frac{Н_{дод}}{100\%},$$

де Ндод – норма нарахування додаткової заробітної плати. Приймемо 10%.

$$З_{дод} = (51250 + 5006{,}25) * \frac{10\%}{100\%} = 5625{,}63 \text{ грн.}$$

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) * \frac{Н_{зп}}{100\%},$$

де Нзп – норма нарахування на заробітну плату. де Нзп = 22%.

$$З_н = (51250 + 5006{,}25 + 5625{,}63) * \frac{22\%}{100\%} = 13614{,}01 \text{грн.}$$

4.2.3 Сировина та матеріали

Витрати на матеріали на даному етапі проведення досліджень в основному пов'язані з використанням моделей елементів та моделювання роботи і досліджень за допомогою комп'ютерної техніки та створення експериментальних математичних моделей або програмного забезпечення, тому дані витрати формуються на основі витратних матеріалів характерних для офісних робіт.

Витрати на матеріали (М), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} H_j * Ц_j * K_j,$$

де Нj – норма витрат матеріалу j-го найменування;

n – кількість видів матеріалів;

Цj – вартість матеріалу j-го найменування, грн.;

Кj – коефіцієнт транспортних витрат, приймемо Кj = 1,13 .

Проведені розрахунки зведено до таблиці.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна, грн/шт.	Норма витрат, шт	Величина відходів, шт	Ціна відходів, грн/шт	Вартість витраченого матеріалу, грн.
Блокнот формату А5	60	4	0	0	264
Ручка кулькова, чорна	20	4	0	0	88
Всього					352

Отже, загальна кількість витрат на матеріали – 352 грн.

Розрахунок витрат на комплектуючі

Витрати на комплектуючі вироби відсутні, так як розробка не потребує використання додаткових елементів.

4.2.4 Програмне забезпечення для наукових робіт

Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum Ц_{іпрг} * С_{пргі} * К_і,$$

де $Ц_{іпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$С_{прг і}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (Кі = 1,10…1,12);

k – кількість найменувань програмних засобів.

Проведені розрахунки зведено до таблиці.

Таблиця 4.8 – Витрати на придбання програмних засобів

Найменування програмного засобу	Кількість, шт.	Ціна за шт, грн	Сума, грн.
Windows 10 PRO	1	7000	7770
Всього			7770

4.2.5 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} * \frac{t_{вик}}{12},$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Таблиця 4.9 – Амортизаційні відрахування

Найменування комплектуючих	Балансова вартість, грн.	Строк корисного використання	Термін використання, місяців	Амортизаційні відрахунки, грн.
Комплект(ноутбуків) Thomson Neo 14 (3663792023717) White	32770	3	1	910,27
Всього				910,27

Отже, сума амортизації – 910,27 грн.

4.2.6 Палисво та енергія для науково-виробних цілей

Витрати на силову електроенергію (Ве) можна розрахувати за формулою

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}$$

де W_{yi} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн, $Ц_e$ = 10,5 грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі}$ <1, $К_{впі}$ = 0,35;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$, $\eta_i = 0,8$.

Результати проведених розрахунків занесено до таблиці 5.9.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук Thomson №1	0,025	40	4,59
Ноутбук Thomson №2	0,03	54	7,44
Ноутбук Thomson №3	0,04	100	18,38
Ноутбук Thomson №4	0,05	40	9,19
Всього			39,6

Отже, витрати на електроенергію становлять 39,6 грн.

4.2.7 Службові відрядження

У службових відрядженнях немає потреби, тому кошти на них також не виділяються.

4.2.8 Витрати на роботи, які виконують сторонні підприємства, установи та організації

Зазначенні витрати не передбачені.

4.3 Інші витрати

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) * \frac{Н_{ів}}{100\%},$$

де $Н_{ів}$ – нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$$I_в = (51250 + 5006{,}25) * \frac{50\%}{100\%} = 28128{,}13 \text{ грн.}$$

Отже, інші витрати становлять 28128,13 грн.

4.3.1 Накладні (загальновиробничі) витрати

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \frac{Н_{нзв}}{100\%},$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 130%.

$$В_{нзв} = (51250 + 5006{,}25) * \frac{130\%}{100\%} = 73133{,}13 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи на тему «Метод та засіб автентифікації застосунків в операційній системі Android» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}.$$

$$В_{заг} = 51250 + 5006{,}25 + 5625{,}63 + 13614{,}01 + 352 + 7770 + 910{,}27 + 39{,}6 + 28128{,}13 + 73133{,}13 = 185829{,}02$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою

$$ЗВ = \frac{В_{заг}}{\eta}$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

ЗВ=185829,02/0,85=218622,38

4.3.2 Оцінювання важливості на наукової значимості науководослідної роботи фундаментального чи пошукового характеру

Для оцінки економічної доцільності виведення цієї науково-технічної розробки на ринок слід врахувати такі аспекти, як тривалість впровадження, прогнозований термін отримання прибутку для інвесторів, рівень попиту та його основні споживачі, а також ринкову ціну аналогічних рішень із подібними можливостями.

Для прикладу наведемо обчислення прибутку інвестора в першому році після реалізації:

$$\Delta П_1 = (\pm \Delta Ц_0 * N + Ц_0 * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\vartheta}{100}\right) = (225000 * 4 + 200000 * 0) * 0{,}8333 * 0{,}3 * (1 - 0{,}18) = 184492{,}6 \quad (4.15)$$

$$\Delta П_2 = (\pm \Delta Ц_0 * N + Ц_0 * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\vartheta}{100}\right) = (225000 * 4 + 251000 * 5) * 0{,}8333 * 0{,}3 * (1 - 0{,}18) = 441757{,}3 \quad (4.16)$$

$$\Delta П_3 = (\pm \Delta Ц_0 * N + Ц_0 * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\vartheta}{100}\right) = (225000 * 4 + 251000 * 10) * 0{,}8333 * 0{,}3 * (1 - 0{,}18) = 699022 \quad (4.17)$$

$$\Delta П_4 = (\pm \Delta Ц_0 * N + Ц_0 * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\vartheta}{100}\right) = (225000 * 4 + 251000 * 15) * 0{,}8333 * 0{,}3 * (1 - 0{,}18) = 956287 \quad (4.18)$$

Далі необхідно розрахувати приведену вартість всіх чистих прибутків ПП за формулою:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} = \frac{184492{,}6}{1{,}14^1} + \frac{441757{,}3}{1{,}14^2} + \frac{699022}{1{,}14^3} + \frac{956287}{1{,}14^4} = 1539776$$

Необхідно розрахувати також початкові інвестиції PV. Обчислення проводиться за формулою:

$$PV = k_{розр} * ЗВ = 2 * 218622{,}38 = 437245$$

Також необхідно обчислити приведений дохід за формулою:

$$E_{\text{абс}} = \text{ПП} - PV = 1539776 - 437245 = 1102531$$

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність $E_{\text{в}}$ або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій за формулою:

$$E_{\text{в}} = \sqrt[\text{Тж}]{1 + \frac{\text{E}_{\text{абс}}}{PV}} - 1 = \sqrt[\text{з}]{1 + \frac{1102531}{437245}} - 1 = 0{,}52$$

Для визначення бар'єрної ставки, з якою необхідно порівняти внутрішню економічну дохідність, використовується формула:

$$\tau_{\text{мін}} = d + f = 0.12 + 0.23 = 0.35$$

Оскільки внутрішня економічна дохідність перевищує мінімальну, потенційний інвестор можу бути зацікавлений в даній розробці.

Також обчислимо період окупності інвестицій за формулою:

$$\text{T}_{\text{ок}} = \frac{1}{E_{\text{в}}} = \frac{1}{0{,}52} = 1{,}9$$

Оскільки $\text{T}_{\text{ок}} < 3$ років, можна зробити висновок, що впровадження науково-технічної розробки є економічно ефективним.

4.4Висновки

Отже, науково-технічна розробка залучає увагу інвесторів завдяки значному потенціальному доходу і прийнятному періоду повернення інвестицій.

Для детального аналізу економічної доцільності проведено оцінювання науково-технічного рівня та комерційних перспектив розробки трьома кваліфікованими експертами згідно з установленими критеріями. Середній бал, отриманий в результаті, дорівнює 28, що свідчить про знаходження науково-технічного рівня і комерційного потенціалу розробки на середньому рівні. Крім того, було здійснено аналіз конкурентоспроможності, який виявив певні переваги за технічними показниками та економічними характеристиками. Цей аналіз підтвердив, що розробка конкурентоспроможна і готова до виведення на ринок.

Додатково виконано розрахунки витрат на проведення науково-дослідницької роботи та аналізу економічної ефективності розробки при її імплементації в умовах підприємства. Згідно з проведеними розрахунками, впровадження цієї розробки виявилося економічно вигідним, що додатково підтверджує її привабливість для інвестицій.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи було проведено дослідження автентифікаційних понять, технологій, методів та їх застосування, а також аналіз версій Android, в яких відбулися зміни в механізмах автентифікації. Були розглянуті ключові методи шифрування, зокрема DES, 3DES, AES, RSA, ECDSA, а також техніки хешування MD5 та SHA-256.

Метою роботи була розробка Android-додатку спрямована на вдосконалення процесів автентифікації і забезпечення безпеки даних. Основні задачі роботи були виконані, а саме:

1. Розроблено методики автентифікації, заснованих на шифруванні і цифрових підписах для додатку в ОС Android;
2. Розроблено додаток за допомогою таких технологій, як Xamarin, що дало змогу значно знизити витрати на підтримку та розвиток додатку, забезпечуючи при цьому доступ до необхідних нативних функцій Android;
3. Розроблено сервер за допомогою Python Flask для побудови API, який обробляє запити від клієнтського додатка.
4. Для оцінки продуктивності методів автентифікації було проведено серію експериментів з різними довжинами повідомлень.;
5. Порівняльний аналіз двох методик.

При порівнянні функцій, можна зробити висновки, що для коротких повідомлень, краще використовувати автентифікацю з використанням парольної захисту з шифруванням даних, оскільки він демонструє стабільний час обробки на сервері і меншу затримку відповіді для клієнта, а для довгих повідомлень краще використовувати цифровий підпис повідомлення, оскільки забезпечує кращу продуктивність на сервері та підвищує рівень безпеки завдяки автентифікації відправника і перевірці цілісності повідомлення.

Для оптимізації швидкості та безпеки в майбутньому планується впровадити комбінований підхід, який поєднує використання шифрування для коротких повідомлень та цифрового підпису для повідомлень більшого обсягу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Схема автентифікації з використанням ідентифікатора. URL: https://studfile.net/preview/7788223/page:4/. (дата звернення 10.09.2024).
2. Digital Identity" Phillip J. Windley. 54p. (дата звернення 11.09.2024).
3. "Comparative Study of Various Authentication Techniques in Cloud Computing". "International Journal of Cloud Applications and Computing". 81p. URL: https://www.academia.edu/74264772/Comparative_Analysis_of_Various_Authentication_Techniques_in_Cloud_Computing?uc-sb-sw=6385756. (дата звернення 15.09.2024).
4. "Two-factor authentication for the IoT". "IoT Magazine". 83p. URL: https://www.researchgate.net/publication/329113965_Two-Factor_Authentication_for_IoT_With_Location_Information. (дата звернення 15.09.2024).
5. "Multi-factor Authentication". Markus Jakobsson. 38p. URL: https://www.researchgate.net/publication/334890864_Multi-Factor_Authentication_Modeling. (дата звернення 15.09.2024).
6. Андроїд (або Android). URL: https://gsmhub.com.ua/glossary/androyid. (дата звернення 19.09.2024).
7. Android, T-Mobile G1. URL: https://www.cnet.com/reviews/t-mobile-g1-review/. (дата звернення 19.09.2024).
8. Android 5.0 Lollipop. URL: https://developer.android.com/about/versions/lollipop. (дата звернення 19.09.2024).
9. Android 6.0 Marshmallow. URL: https://www.android.com/versions/marshmallow-6-0/. (дата звернення 19.09.2024).
10. Android 7.0 Nougat. URL: https://www.android.com/new-features-on-android/. (дата звернення 19.09.2024).
11. Android 10. URL: https://www.android.com/android-10/. (дата звернення 19.09.2024).

ДОДАТКИ

Додаток А

Код основних функціональних можливостей засобу

Код для запуску сервера:

```
import time
import csv
from flask import Flask, request, jsonify
import base64
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import hashlib
from ecdsa import VerifyingKey, SECP256k1, BadSignatureError
from ecdsa.util import sigdecode_der
import os


script_dir = os.path.dirname(os.path.abspath(__file__))

log_path = os.path.join(script_dir, 'log.csv')

app = Flask(__name__)

password = "testpassword"

encryption_key = hashlib.sha256(password.encode()).digest()
print(encryption_key)  # Это будет 32 байта


def log_to_csv(timestamp, processing_time, result, status,textLange):
    processing_time_str = str(processing_time).replace('.', ',')
    with open(log_path, mode='a', newline='') as file:
        writer = csv.writer(file, delimiter=';')
        writer.writerow([timestamp, textLange, processing_time_str, result, status])


@app.route('/process_encrypted_message', methods=['POST'])
def process_encrypted_message():
    start_time = time.time()
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())

    try:
        # Отримуємо зашифроване повідомлення
        data = request.get_json()
        encrypted_message = base64.b64decode(data.get("message", ""))

        # Виймаємо IV (перші 16 байт) і зашифроване повідомлення (частина, що залишилася)
        iv = encrypted_message[:16]
        ciphertext = encrypted_message[16:]

        # Дешифруємо повідомлення з використанням AES
        cipher = AES.new(encryption_key, AES.MODE_CBC, iv)
        decrypted_message = unpad(cipher.decrypt(ciphertext), AES.block_size).decode()

        # Підрахунок символів 'a'
        count_a = decrypted_message.count("a")
        processing_time = time.time() - start_time
        textLange = len(decrypted_message)
        # Логуємо результат у файл
        log_to_csv(timestamp, processing_time, count_a, "encrypted success",textLange)
```

```
        return jsonify({
            "count_a": count_a,
            "processing_time": processing_time
        })

    except Exception as e:
        processing_time = time.time() - start_time
        # Логуємо помилку
        log_to_csv(timestamp, processing_time, str(e), "error",0)
        return jsonify({"error": "Message could not be decrypted"}), 400

client_public_key = None

@app.route('/upload_public_key', methods=['POST'])
def upload_public_key():
    global client_public_key
    try:
        data = request.get_json()
        public_key_pem = data.get("public_key")
        client_public_key = VerifyingKey.from_pem(public_key_pem)
        # Перетворимо PEM-ключ на байтовий рядок
        #public_key_bytes = VerifyingKey.from_pem(public_key_pem).to_string()
        #client_public_key = VerifyingKey.from_string(public_key_bytes, curve=SECP256k1)
        print('Public key received and stored successfully')
        print('Public key (PEM):\n' + public_key_pem)
        return jsonify({"status": "Public key received and stored successfully"})
    except Exception as e:
       return jsonify({"error": str(e)}), 400


@app.route('/process_signed_message', methods=['POST'])
def process_signed_message():
    global client_public_key
    if client_public_key is None:
        return jsonify({"error": "Public key not set"}), 400

    start_time = time.time()
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())

    try:
        data = request.get_json()
        message = data.get("message", "").encode()
        signature = base64.b64decode(data.get("signature", ""))

            # Логуємо повідомлення та підпис у байтовому поданні
        print(f"Received message: {message.decode()}")
        print(f"Message bytes: {list(message)}")
        print(f"Received signature (base64): {data.get('signature', '')}")

            # Спробуємо перевірити підпис
        try:
         client_public_key.verify(signature, message)
         result = "Signature is valid"
         status = "success"
        except BadSignatureError:
         result = "Signature verification failed"
         status = "failed"

        processing_time = time.time() - start_time

            # Підрахунок символів 'a'
        message_str = message.decode('utf-8')
        count_a = message_str.count("a")
        textLange = len(message_str)
        processing_time = time.time() - start_time
```

```
            # Логуємо результат у файл
        log_to_csv(timestamp, processing_time, count_a, "signed success",textLange)

        print(f"Signature verification: {'success'}")


        return jsonify({
                    "count_a": count_a,
                    "processing_time": processing_time
                })

    except Exception as e:
        processing_time = time.time() - start_time
        log_to_csv(timestamp, processing_time, str(e), "error",0)
        return jsonify({"error": "An error occurred"}), 400

@app.route('/process_signed_message_ohneLog', methods=['POST'])
def process_signed_message_ohneLog():
    global client_public_key
    if client_public_key is None:
        return jsonify({"error": "Public key not set"}), 400

    start_time = time.time()
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())

    try:
        data = request.get_json()
        message = data.get("message", "").encode()
        signature = base64.b64decode(data.get("signature", ""))

        # Спробуємо перевірити підпис
        try:
            client_public_key.verify(signature, message)
            result = "Signature is valid"
            status = "success"
        except BadSignatureError:
            result = "Signature verification failed"
            status = "failed"

        processing_time = time.time() - start_time

        # Підрахунок символів 'a'
        message_str = message.decode('utf-8')
        count_a = message_str.count("a")
        textLange = len(message_str)
        processing_time = time.time() - start_time

        # Логуємо результат у файл
        log_to_csv(timestamp, processing_time, count_a, "signed success",textLange)

        return jsonify({
                "count_a": count_a,
                "processing_time": processing_time
            })

    except Exception as e:
        processing_time = time.time() - start_time
        log_to_csv(timestamp, processing_time, str(e), "error",0)
        return jsonify({"error": "An error occurred"}), 400

if __name__ == '__main__':
    app.run(debug=True)
```

Код для генерації ключа:

```
from ecdsa import SigningKey, SECP256k1
from cryptography.fernet import Fernet

# Генерація приватного ключа (закритий ключ)
private_key = SigningKey.generate(curve=SECP256k1)
private_key_bytes = private_key.to_string().hex()
print("Private Key:", private_key_bytes)

# Генерація публичного ключа (відкритий ключ) на основі приватного
public_key = private_key.get_verifying_key()
public_key_bytes = public_key.to_string().hex()
print("Public Key:", public_key_bytes)


# Генерація нового 32-байтного ключа
encryption_key = Fernet.generate_key()
print(encryption_key.decode())  # Виводить base64 строку для використання
```

Код для запуску додатку:

```
using System;
using Android.App;
using Android.OS;
using Android.Runtime;
using Android.Views;
using AndroidX.AppCompat.App;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;
using System.Net.Http;
using System.Text;
using Newtonsoft.Json;
using Android.Widget;

using Org.BouncyCastle.OpenSsl;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Crypto.Signers;
using Org.BouncyCastle.Asn1.Sec;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Generators;
using Org.BouncyCastle.Crypto.EC;
using Org.BouncyCastle.Asn1;


namespace AutentifikationTest
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme.NoActionBar", MainLauncher = true)]


    public class MainActivity : AppCompatActivity
    {

        private HttpClient _client = new HttpClient();
        private string serverUrl = "http://10.0.2.2:5000";
        string message = "In the heart of a bustling city";

        private static readonly byte[] encryptionKey = GenerateKey("testpassword");
```

```
    public static byte[] GenerateKey(string password)
    {
        using (SHA256 sha256 = SHA256.Create())
        {
            return sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
        }
    }

    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        Xamarin.Essentials.Platform.Init(this, savedInstanceState);
        SetContentView(Resource.Layout.activity_main);

        // Знаходимо кнопку її ID
        Button button1 = FindViewById<Button>(Resource.Id.button1);
        // Додаємо обробник події натискання
        button1.Click += Button1_Click;

        // Находим кнопку по ее ID
        Button button3 = FindViewById<Button>(Resource.Id.button3);
        // Добавляем обработчик события нажатия
        button3.Click += Button3_Click;


        Button button4 = FindViewById<Button>(Resource.Id.button4);
        button4.Click += Button4_Click;


        Button button2 = FindViewById<Button>(Resource.Id.button2);
        button2.Click += Button2_Click;

        _ = UploadPublicKey();  // Надсилання відкритого ключа під час запуску

    }

    public override bool OnCreateOptionsMenu(IMenu menu)
    {
        MenuInflater.Inflate(Resource.Menu.menu_main, menu);
        return true;
    }

    public override bool OnOptionsItemSelected(IMenuItem item)
    {
        int id = item.ItemId;
        if (id == Resource.Id.action_settings)
        {
            return true;
        }

        return base.OnOptionsItemSelected(item);
    }



    public override void OnRequestPermissionsResult(int requestCode, string[] permissions, [GeneratedEnum]
Android.Content.PM.Permission[] grantResults)
    {
        Xamarin.Essentials.Platform.OnRequestPermissionsResult(requestCode, permissions, grantResults);

        base.OnRequestPermissionsResult(requestCode, permissions, grantResults);
    }
```

```
private async Task<string> SendMessageWithTimeMeasurement(string message)
{
    string endpoint = "/process_encrypted_message";
    DateTime startTime = DateTime.Now; //Вимірюємо час перед відправкою

    var encryptedMessage = EncryptMessage(message);
    var jsonMessage = JsonConvert.SerializeObject(new { message = encryptedMessage });

    var content = new StringContent(jsonMessage, Encoding.UTF8, "application/json");

    var response = await _client.PostAsync(serverUrl + endpoint, content);
    var responseContent = await response.Content.ReadAsStringAsync();

    string responseBody = "";
    if (response.IsSuccessStatusCode)
    {
        responseBody = await response.Content.ReadAsStringAsync();
    }
    else
    {
        return "Ошибка сервера";
    }

    DateTime endTime = DateTime.Now;
    var timeDifference = (endTime - startTime).TotalMilliseconds;

    Console.WriteLine($"Response: {responseContent}");
    return responseBody+" "+ timeDifference.ToString();
}

private string EncryptMessage(string message)
{
    using (var aes = Aes.Create())
    {
        aes.Key = encryptionKey;
        aes.GenerateIV();

        using (var encryptor = aes.CreateEncryptor(aes.Key, aes.IV))
        using (var ms = new MemoryStream())
        {
            ms.Write(aes.IV, 0, aes.IV.Length);
            using (var cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write))
            using (var writer = new StreamWriter(cs))
            {
                writer.Write(message);
            }
            return Convert.ToBase64String(ms.ToArray());
        }
    }
}


private async void Button1_Click(object sender, System.EventArgs e)
{
    Toast.MakeText(this, "Test started!", ToastLength.Short).Show();

    string nMess = "";
    string s = "";
    for (int i = 0; i < 10000; i++)
    {
        nMess += message;
    }
```

```
        for (int j = 0; j < 10; j++)
        {
            s = await SendMessageWithTimeMeasurement(nMess);
        }

        // Знаходимо елемент TextView та оновлюємо його текст
        var resultsTextView = FindViewById<TextView>(Resource.Id.resultsTextView);
        resultsTextView.Text = s;
    }

    private async void Button2_Click(object sender, System.EventArgs e)
    {
        // по 10 раз
        Toast.MakeText(this, "Test started!", ToastLength.Short).Show();
        string nMess = "";
        string s;
        var resultsTextView = FindViewById<TextView>(Resource.Id.resultsTextView);
        resultsTextView.Text = "Тест з шифруванням \n";
        for (int x = 0; x < 10; x++)
        {
            for (int i = 0; i < 10000; i++)
            {
                nMess += message;
            }
            resultsTextView.Text += "Тест " + nMess.Length + " символів \n";
            for (int j = 0; j < 10; j++)
            {
                s = await SendMessageWithTimeMeasurement(nMess);
            }

        }
        resultsTextView.Text += "Тест завершено";

    }


    private string SignMessage(string message)
    {
        var signer = new ECDsaSigner();
        signer.Init(true, keyPair.Private);
        byte[] messageBytes = Encoding.UTF8.GetBytes(message);
        var signature = signer.GenerateSignature(messageBytes);

        // Створюємо DER-підпис
        var seq = new DerSequence(new DerInteger(signature[0]), new DerInteger(signature[1]));
        byte[] derSignature = seq.GetEncoded();

        return Convert.ToBase64String(derSignature); // Повертаємо DER-підпис у форматі Base64
    }


    private static readonly AsymmetricCipherKeyPair keyPair = GenerateKeyPair();

    public static AsymmetricCipherKeyPair GenerateKeyPair()
    {
        // Отримуємо параметри SECP256k1 через CustomNamedCurves
        var ecParams = CustomNamedCurves.GetByOid(SecObjectIdentifiers.SecP256k1);
        var domainParams = new ECDomainParameters(ecParams.Curve, ecParams.G, ecParams.N, ecParams.H,
ecParams.GetSeed());

        // Створюємо генератор ключів та ініціалізуємо його параметрами SECP256k1
        var keyGen = new ECKeyPairGenerator();
```

```
        var keyGenParam = new ECKeyGenerationParameters(domainParams, new
Org.BouncyCastle.Security.SecureRandom());
        keyGen.Init(keyGenParam);

        return keyGen.GenerateKeyPair();
    }

    private async Task<string> SendSignedMessageWithTimeMeasurement(string message, string endpoint,bool log)
    {

        DateTime startTime = DateTime.Now;

        // Підписуємо повідомлення
        var signedMessage = SignMessage(message);
        var signatureBase64 = signedMessage;

        // Створюємо JSON з даними повідомлення та підпису
        var jsonMessage = JsonConvert.SerializeObject(new { message = message, signature = signatureBase64 });
        var content = new StringContent(jsonMessage, Encoding.UTF8, "application/json");
        var response = await _client.PostAsync(serverUrl + endpoint, content);
        var responseContent = await response.Content.ReadAsStringAsync();

        DateTime endTime = DateTime.Now;
        var timeDifference = (endTime - startTime).TotalMilliseconds;
        if (log)
        {
            Console.WriteLine($"Sending message: {message}");
            Console.WriteLine($"Message bytes: {string.Join(", ", Encoding.UTF8.GetBytes(message))}");
            Console.WriteLine($"Generated signature (base64): {signatureBase64}");
            Console.WriteLine($"Generated signature (bytes): {string.Join(", ", signedMessage)}");
            Console.WriteLine($"Response: {responseContent}");
            Console.WriteLine($"Time taken: {timeDifference} ms");
        }

        return responseContent + " " + timeDifference.ToString();
    }

    private async Task UploadPublicKey()
    {
        var publicKey = (ECPublicKeyParameters)keyPair.Public;
        string publicKeyPem;

        using (var sw = new StringWriter())
        {
            var pemWriter = new PemWriter(sw);
            pemWriter.WriteObject(publicKey);
            pemWriter.Writer.Flush();
            publicKeyPem = sw.ToString();
        }

        Console.WriteLine("Uploading public key (PEM):");
        Console.WriteLine(publicKeyPem);

        var jsonMessage = JsonConvert.SerializeObject(new { public_key = publicKeyPem });
        var content = new StringContent(jsonMessage, Encoding.UTF8, "application/json");

        var response = await _client.PostAsync(serverUrl + "/upload_public_key", content);
        var responseContent = await response.Content.ReadAsStringAsync();

        if (response.IsSuccessStatusCode)
        {
            Console.WriteLine("Public key uploaded successfully.");
        }
```

```
            else
            {
                Console.WriteLine("Failed to upload public key: " + responseContent);
            }
        }



        private async void Button3_Click(object sender, System.EventArgs e)
        {
            // 1 раз з логуванням
            Toast.MakeText(this, "Test started!", ToastLength.Short).Show();
            string nMess = "";
            string s = "";
            for (int i = 0;i<10000;i++)
            {
                nMess += message;
            }
            string endpoint = "/process_signed_message";
            for (int i = 0; i < 10; i++)
            {
                s = await SendSignedMessageWithTimeMeasurement(nMess, endpoint, true);
            }
            var resultsTextView = FindViewById<TextView>(Resource.Id.resultsTextView);
            resultsTextView.Text = s;
        }

        private async void Button4_Click(object sender, System.EventArgs e)
        {
            // по 10 раз
            Toast.MakeText(this, "Test started!", ToastLength.Short).Show();
            string nMess = "";
            string endpoint = "/process_signed_message_ohneLog";
            string s;
            var resultsTextView = FindViewById<TextView>(Resource.Id.resultsTextView);
            resultsTextView.Text = "Тест з підписом \n";
            for (int x = 0; x < 10; x++)
            {
                for (int i = 0; i < 10000; i++)
                {
                    nMess += message;
                }
                resultsTextView.Text += "Тест " + nMess.Length + " символів \n";
                for (int j = 0; j < 10; j++)
                {
                    s = await SendSignedMessageWithTimeMeasurement(nMess, endpoint, false);
                }

            }
            resultsTextView.Text += "Тест завершено\n";

        }

    }
}
```

Додаток В

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Метод та засіб автентифікації застосунків в операційній системі Android

Автор роботи: Швець Владислав Вікторович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ

Керівник Гарнага Володимир Анатолійович, к.т.н., доцент

Показники звіту подібності

Turnitin	
Оригінальність	97 %
Загальна схожість	3 %

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи.

Автор роботи _____ (підпис) Швець В.В. (прізвище, ініціали)

Особа, відповідальна за перевірку _____ (підпис) Валентина КАПЛУН (прізвище, ініціали)

Керівник роботи _____ (підпис) Гарнага В.А. (прізвище, ініціали)

Додаток В

ІЛЮСТРАТИВНА ЧАСТИНА

«Метод та засіб автентифікації застосунків в операційній системі Android»

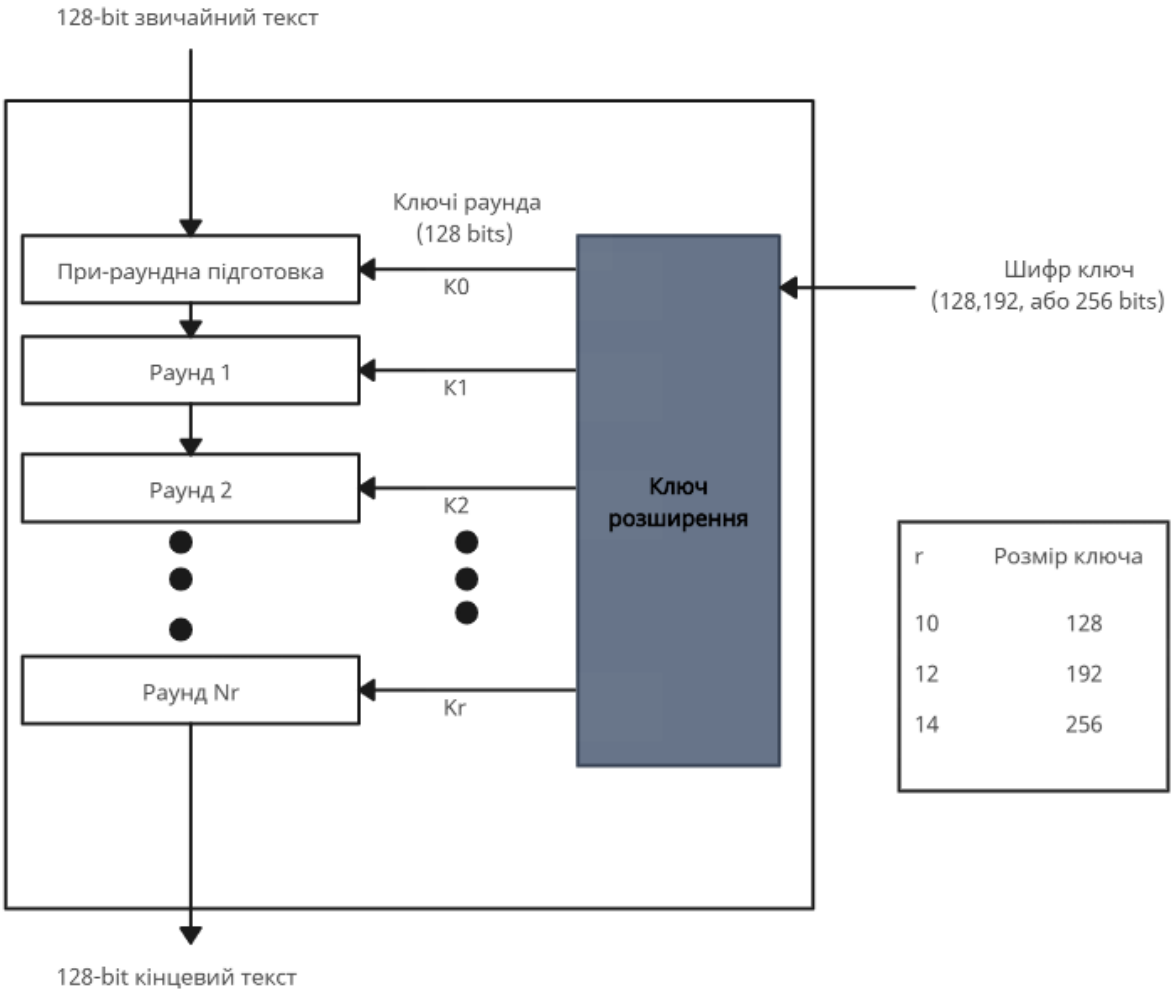
Виконав студент 2-го курсу, групи
1БС-23м
спеціальності 125 - Кібербезпека
та захист інформації
_____ Владислав ШВЕЦЬ
16.12 2024 р.

Керівник: к.т.н., доцент кафедри ПЗ
_____ Володимир ГАРНАГА
16.12 2024 р.

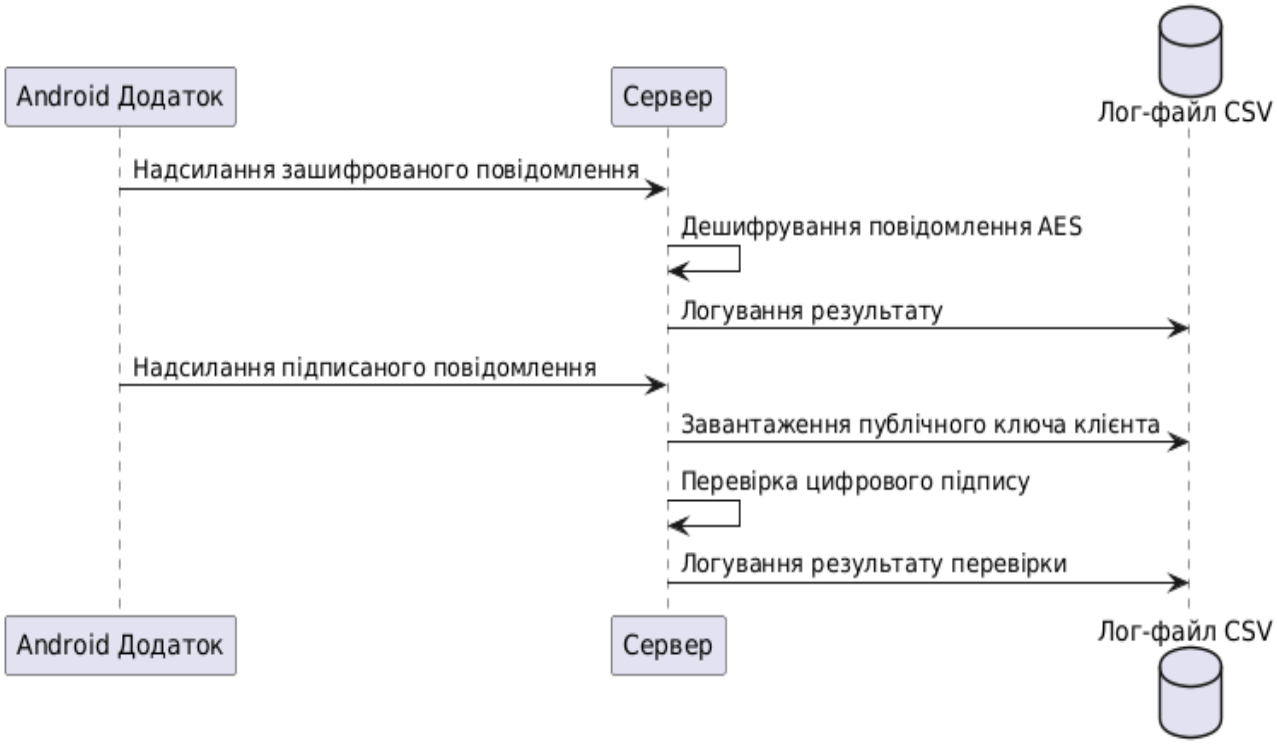
АНАЛІЗ ЗАСОБІВ ЗАХИСТУ АВТЕНТИФІКАЦІЇ

Версія Android	Рік випуску	Основні нововведення в автентифікації
Android 5.0 Lollipop	2014	Factory Reset Protection (FRP)
Android 6.0 Marshmallow	2015	API для сканерів відбитків пальців
Android 7.0 Nougat	2016	Покращення API відбитків пальців, функція Direct Boot
Android 9.0 Pie	2018	Впровадження API BiometricPrompt
Android 10	2019	Розширена підтримка біометричних технологій, включаючи розпізнавання обличчя
Android 11	2020	Дозволи на використання датчиків на разовій основі, автоматичне скидання дозволів
Android 12	2021	Удосконалена система управління дозволами, покращена біометрична автентифікація
Android 13	2022	Уніфікована платформа для біометричних технологій, підтримка Passkeys
Android 14	2023	Підвищена точність і безпека біометричної автентифікації, удосконалення NFC і Bluetooth

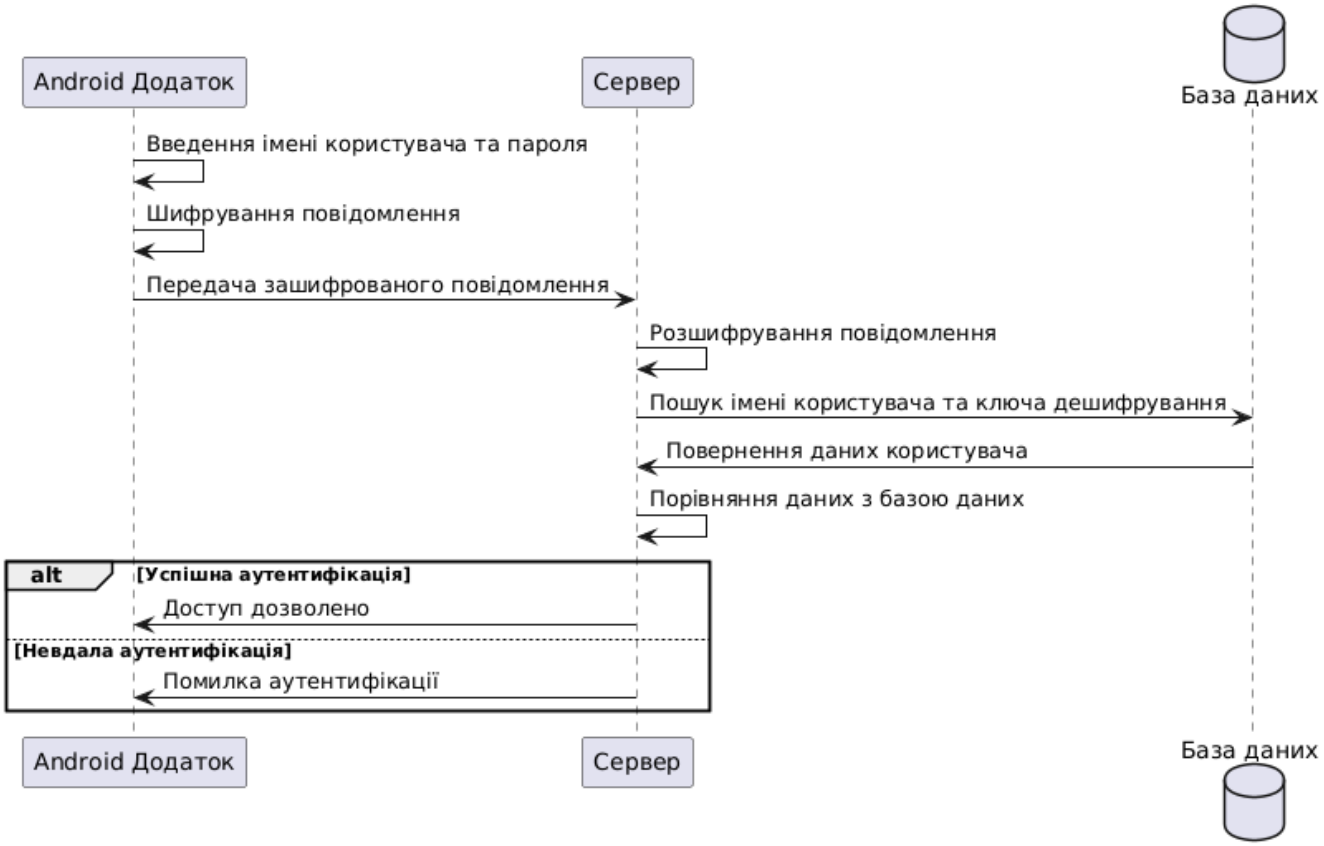
МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ ЗА ДОПОМОГОЮ ШИФРУВАННЯ



УЗАГАЛЬНЕНИЙ АЛГОРИТМ РОБОТИ ЗАСОБУ

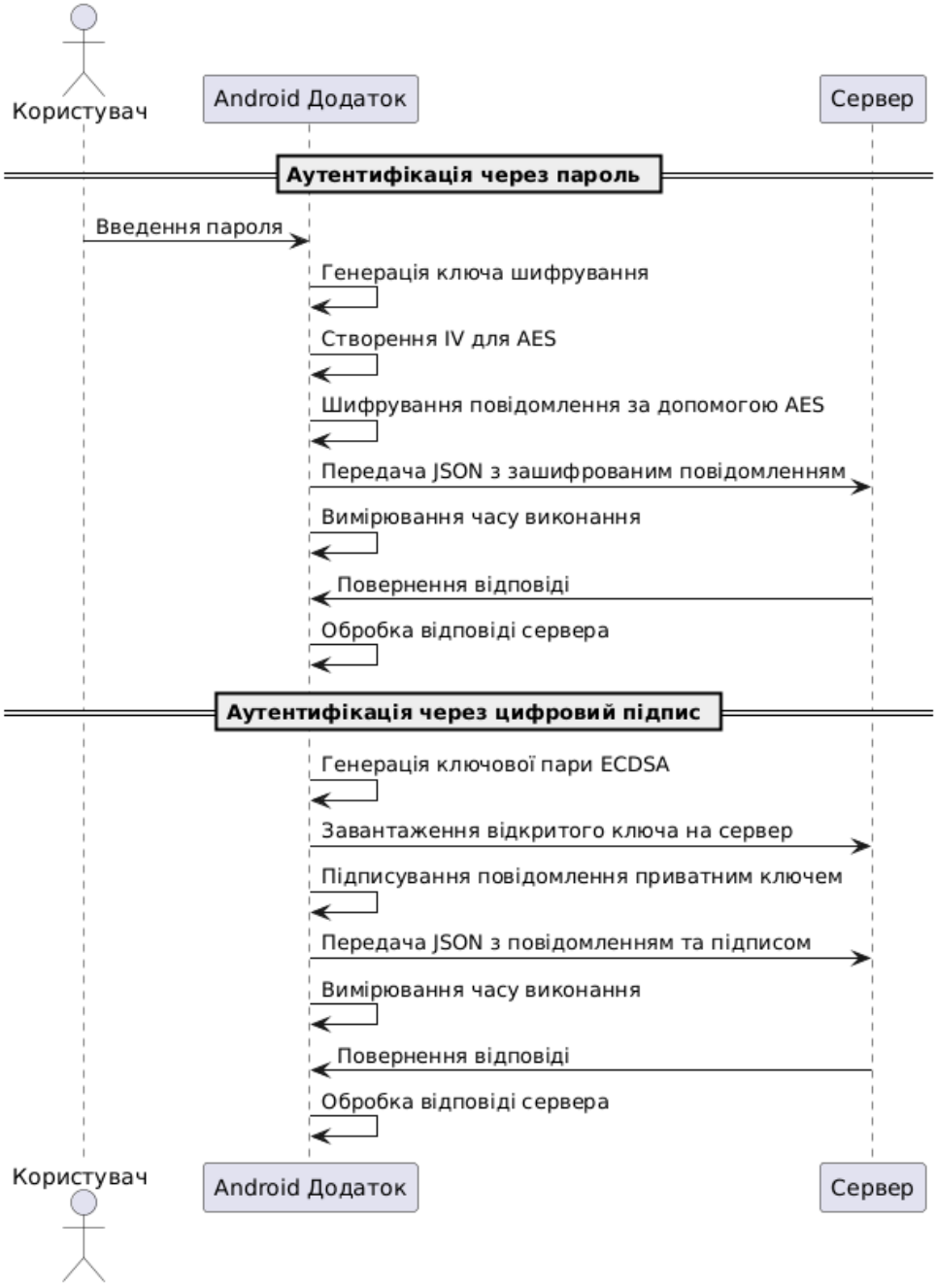


АЛГОРИТМ ЗАХИСТУ ШИФРУВАННЯ ТА ПЕРЕДАЧІ ШИФРОВАНИХ ПОВІДОМЛЕНЬ



АЛГОРИТМ ГЕНЕРАЦІЇ КЛЮЧОВОЇ ПАРИ

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ РОЗРОБКИ



РЕЗУЛЬТАТ ТЕСТУВАННЯ