Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інституту комп'ютерних наук та штучного інтелекту
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

В.о. зав. кафедрою КІСМіТ
Марина ЄСІНА
"Допущено до захисту"

« »_____2025р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Обґрунтування властивостей і побудова мобільного клієнта децентралізованої системи захищеного голосування»

оцінка «_____»

Керівник: д.т.н., проф. Олійников Р. В.

Голова ЕК
Мичуда Л.З.

Рецензент: доц. Родінко М. Ю.

Виконавець: студент групи КБ-41
Зуєв О. С.

Харків 2025

РЕФЕРАТ

Робота містить 57 сторінок, 2 таблиць, 1 рисунок, 4 додатку, 62 джерела.

Мета роботи. Обґрунтувати вимоги та розробити мобільний Android-клієнт децентралізованої системи захищеного голосування.

Методи дослідження і апаратура. Зроблено порівняння стандартів безпеки NIST SP 800-63 B та ISO/IEC 27001, здійснено порівняння алгоритмів Ed25519 і secp256r1, реалізовано прототип із використанням Android Keystore у TEE, Jetpack Compose, Firebase Auth та Firestore.

Результати роботи та їх новизна. Реалізовано модульну MVVM-архітектуру із шарами UI, ViewModel і Repository; впроваджено генерацію й захист ключів ECDSA P-256 у TEE Android Keystore; забезпечено цифрове підписання голосів приватним ключем через API Signature.

Рекомендації щодо використання результатів роботи. Інтегрувати клієнт із консорціумним блокчейном для децентралізації зберігання, додати certificate pinning і державну автентифікацію (BankID, «Дія»).

Значущість роботи та висновки. Запропоноване рішення підвищує довіру виборців завдяки прозорій моделі «читати може кожен, змінити — тільки власник», гарантує цілісність і невідкличність голосів у масштабованій системі.

У першому розділі наведено огляд сучасного стану та перспектив розвитку децентралізованих систем захищеного голосування. У другому розділі проаналізовано вимоги до безпеки та функціоналу мобільного клієнта у контексті розподіленого голосування. У третьому розділі обґрунтовано вибір архітектурних і технологічних рішень для реалізації Android-додатку, зокрема застосування MVVM, Jetpack Compose і TEE Android Keystore. У четвертому розділі описано практичну реалізацію прототипу мобільного клієнта та результати його функціонального й безпекового тестування.

Ключові слова: БЛОКЧЕЙН, КРИПТОГРАФІЯ, АВТЕНТИФІКАЦІЯ, ЦИФРОВИЙ ПІДПИС, ANDROID KEYSTORE, MVVM, FIREBASE, TEE.

ABSTRACT

The thesis comprises 57 pages, 2 tables, 1 drawing, 4 supplements, and 62 references.

Objective. To substantiate the requirements and develop a mobile Android client for a decentralized secure voting system.

Methods and Equipment. A comparison of the NIST SP 800-63B and ISO/IEC 27001 security standards was conducted, alongside an analysis of the Ed25519 and secp256r1 signature algorithms. A prototype was implemented using the Android Keystore in a Trusted Execution Environment (TEE), Jetpack Compose for the UI, Firebase Authentication, and Cloud Firestore.

Results and Novelty. A modular MVVM architecture was realized, comprising UI, ViewModel, and Repository layers. Generation and secure storage of ECDSA P-256 keys in the TEE Android Keystore were implemented, and vote signing with the private key via the Android Signature API was ensured.

Recommendations. Integrate the client with a consortium blockchain for decentralized data storage, and enhance security by adding certificate pinning and government-level authentication (BankID, Diia).

Significance and Conclusions. The proposed solution increases voter trust through a transparent “anyone can read, only the owner can modify” model, guaranteeing vote integrity and non-repudiation in a scalable system.

In the first chapter, an overview of the current state and future prospects of decentralized secure voting systems is presented. The second chapter analyzes security and functional requirements for the mobile client within a distributed voting context. The third chapter justifies the architectural and technological choices for the Android application, including MVVM, Jetpack Compose, and TEE Android Keystore. The fourth chapter describes the practical implementation of the mobile client prototype and reports the results of its functional and security testing.

Keywords: BLOCKCHAIN, CRYPTOGRAPHY, AUTHENTICATION, DIGITAL SIGNATURE, ANDROID KEYSTORE, MVVM, FIREBASE, TEE.

ЗМІСТ

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ЕЦП	– Електроний цифровий підпис
ІТС	– Інформаційно-телекомунікаційні системи
API	– Application Programming Interface
APK	– Android Application Package
AES	– Advanced Encryption Standard
GCM	– Galois/Counter Mode
DI	– dependency injection
ECC	– Elliptic Curve Cryptography
Ed25519	– Edwards25519
ECDSA	– Elliptic Curve Digital Signature Algorithm
HTTPS	– Hypertext Transfer Protocol Secure
IEC	– International Electrotechnical Commission
ISO	– International Organization for Standardization
JSON	– JavaScript Object Notation
MASVS	– Mobile Application Security Verification Standard
MVVM	– Model-View-ViewModel
NIST	– National Institute of Standards and Technology
OAuth	– Open Authorization
SDK	– Software Development Kit
SHA	– Secure Hash Algorithm
SHA1PRNG	– SHA-1 Pseudo-Random Number Generator
SP	– Special Publication
TEE	– Trusted Execution Environment
TLS	– Transport Layer Security
UID	– User Identifier
UUID	– Universally Unique Identifier
UI	– User Interface

ВСТУП

Сьогодні, коли все більше процесів переходить в онлайн, важливо перевести вибори в електронний формат. Головне – це забезпечити безпеку, захист особистих даних і довіру людей до результатів голосування. Блокчейн дає змогу створювати прозорі й незмінні списки голосів [62], але для роботи в мобільному додатку потрібні чіткі технічні рішення та обґрунтовані вимоги. Саме тому актуально розробити Android-додаток для децентралізованої системи захищеного голосування, який буде простим у використанні та надійно взаємодіяти з існуючими сервісами електронної ідентифікації.

Об’єктом дослідження є процеси забезпечення безпеки електронного голосування в децентралізованих мережах. Предметом – методи, алгоритми та архітектурні рішення для реалізації мобільного Android-клієнта, здатного проводити безпечну генерацію й зберігання криптографічних ключів, автентифікацію користувача та цифровий підпис голосів із використанням середовища TEE Android Keystore.

Метою даної кваліфікаційної роботи є обґрунтування вимог до захищеного мобільного клієнта та розробка його прототипу на платформі Android із застосуванням сучасних криптографічних стандартів і сервісів Firebase.

Завдання дослідження:

1) Проаналізувати сучасний стан та тенденції розвитку децентралізованих систем голосування із врахуванням рекомендацій NIST SP 800-63B [2] та ISO/IEC 27001 [4].
2) Дослідити та порівняти алгоритми Ed25519 і secp256r1 для цифрових підписів у контексті мобільних додатків [43,45].
3) Визначити архітектурні вимоги до Android-клієнта (MVVM, Jetpack Compose, Repository) та політики безпеки збірки й даних.

4) Реалізувати прототип клієнта з генерацією і захистом ECDSA P-256-ключів у TEE, автентифікацією через Firebase Auth та збереженням голосів у Firestore.
5) Провести функціональне й безпекове тестування розробленого прототипу.

У роботі використано методи аналізу й узагальнення нормативних документів і наукових публікацій, порівняльний аналіз криптографічних алгоритмів, методи проєктування програмного забезпечення (архітектура MVVM), прототипування з використанням Android Keystore у середовищі TEE, а також функціональне та безпекове тестування.

Було запропоновано та реалізовано модульний Android-клієнт із генерацією та захистом приватних ключів ECDSA P-256 у Android Keystore, який забезпечує цифрове підписання голосів із гарантією їхньої цілісності та невідкличності. Окрім того, розроблено набір рекомендацій щодо інтеграції клієнта з блокчейном, certificate pinning та державною автентифікацією (BankID, «Дія»).

Розроблені рішення можуть бути використані як основа для впровадження мобільного голосування у виборчих процесах державних і приватних організацій. Інтеграція з Firebase дозволяє швидко масштабувати систему, а застосовані криптографічні підходи гарантують високий рівень безпеки.

Робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел і додатків. У першому розділі наведено огляд сучасного стану та перспектив розвитку децентралізованих систем голосування. У другому – проведено аналіз вимог до захищеного голосування та мобільного клієнта. У третьому – обґрунтовано вибір архітектури та технологій. У четвертому – описано практичну реалізацію Android-клієнта та результати тестування. У висновках підбито підсумки та окреслено перспективи подальших досліджень.

1 АКТУАЛЬНИЙ СТАН І ПЕРСПЕКТИВИ РОЗВИТКУ ДЕЦЕНТРАЛІЗОВАНИХ СИСТЕМ ЗАХИЩЕННОГО ГОЛОСУВАННЯ

1.1 Актуальний стан і перспективи розвитку децентралізованих систем захищеного голосування

З 2017 по 2023 рік інтерес до децентралізованих систем голосування значно зріс, що відображає підвищену увагу до блокчейн-технологій як інструменту для забезпечення довіри у виборчих процесах. Цей тренд підтверджується дослідженнями, які фіксують зростання кількості розробок і пілотних проєктів у цій сфері [1]. Основна увага дослідників зосереджена на аспектах безпеки, прозорості та децентралізації, які є ключовими для створення надійних систем голосування.

Децентралізовані системи голосування приваблюють своєю здатністю підвищувати безпеку виборчих процесів. Використання блокчейну дозволяє захищати дані від зовнішніх атак і маніпуляцій завдяки криптографічним механізмам, які відповідають рекомендаціям NIST SP 800-63 щодо цифрової автентифікації [2]. Незмінність даних унеможливлює зміну голосів після запису. Прозорість таких систем забезпечує можливість публічної перевірки даних, що підвищує довіру виборців. Завдяки децентралізації усувається єдиний центр управління, це знижує ризик зловживань і підвищує стійкість системи до технічних збоїв. Важливим аспектом також є приватність виборців. Технології, такі як докази з нульовим розголошенням (ZKP), дозволяють захищати вибір виборця, одночасно забезпечуючи верифікованість голосів [3]. Крім того, ці системи здатні спрощувати процес голосування, роблячи його більш ефективним і доступним для учасників.

Проте впровадження децентралізованих системи голосування стикаються з низкою викликів, які призупиняють їх широке впровадження. Одним з яких є забезпечення приватності виборців при збереженні можливості верифікації голосів. Прозорість блокчейну ускладнює захист даних, що вимагає складних криптографічних рішень. Іншою проблема – масштабованість, системи на основі блокчейну часто мають обмеження шодо обробки великох кількості транзакцій.

Технічна складність – розробка, впровадження та підтримка систем такого плану вимагає значних ресурсів і експертизи. Зручність використання також відіграє важливу роль. Інтерфейси для виборців повинні бути інтуїтивно зрозумілими, щоб забезпечити широку участь серед усіх виборців.

Зрештою, відсутність чітких правових рамок і стандартів, таких як ISO/IEC 27001 для управління інформаційною безпекою [4], ускладнює інтеграцію цих технологій у державні виборчі процеси.

Деякі проєкти вже продемонтрували потенціал децентралізованих систем голосування, хоча більшість із них перебуває на стадії пілотних, або обмежених до використання.

Voatz – мобільний додаток для голосування, який використовувався у виборах США, зокрема для військових і закородонних виборців у штатах Західна Вірджинія, Юта і Колорадо. Система використовує біометричну автентифікацію (розпізнавання по обличчю та по відбитках пальців) і блокчейн для забезпечення безпеки та прозорості. Voatz пропонує портал для аудиту на основі блокчейну, адміністративну панель і аналітику в реальному часі. У 2018 році в Західній Вірджинії виборці з 29 країн використовували Voatz для голосування. Проте дослідження MIT у 2020 році виявило потенційні вразливості, такі як можливість перехоплення голосів або порушення конфіденційності, шо викликало суперечки щодо безпеки системи [5].

Agora – провела пілотне дослідження під час президентських виборів у С’єрра-Леоне в березні 2018 року. Спочатку повідомлялося, що це перші вибори на основі блокчейну, але Національна виборчка комісія (NEC) уточнила, шо Agora діяла лише як міжнародний спостерігач, записуючи голоси з окремих дільниць на власний блокчейн. Пілот продемонстрував можливість швидкого підрахунку голосів і забезпечення їх незмінності, але не був частиною офіційного виборчого процесу. Agora викоростовує власний блокчейн із унікальними механізмами безпеки та консенсусу, а також, децентралізовані додатки dapps для відстеження виборчих даних [6].

Luxoft – платформа e-Vote від Luxoft використовує блокчейн Hyperledger Fabric і Ethereum для створення масштабованої системи голосувань. Вона підтримує публічні аудити безпеки, забезпечує приватність та інтегрується з цифровими ідентифікаторами, такими як Zug digital ID та uPort. Система призначена для масового призначена для масового використання, але конкретні приклади впровадження обмежені [7].

Follow My Vote – платформа Pollaris від Follow My Vote є відкритою і фінансується через краудфаундинг. Вона надає прозору перевірки бюлетенів і аналітику в реальному часі, шо робить її корисною для оцінки настроїв виборців або членів організацій [7].

Polys – це відкрита хмарна система, розроблена для невеликих виборів, таких як студентські ради чи асоціації. Вона використовує блокчейн для забезпечення незмінності голосів і зниження організаційних витрат. Polys підтримує термінали для голосування та пропонує послуги з обслуговування [7].

Серед технологій, шо застосовуються в цих системах, найпоширенішою платформою є Ethereum (34,91%), за якою йдуть Hyperledger Fabric, Bitcoin і Multichain. Механізми консенсусу включають Proof of Work (PoW), Proof of Stake (PoS) та Proof of Authority (PoA). Для забезпечення безпеки та приватності використовуються докази з нульовим розголошенням (ZKP), гомоморфне шифрування, сліпі підписи та кільцеві підписи. Автентифікація базується на біометричних методах, цифрових індентифікаторах, що відповідає стандартам НД ТЗІ шодо захисту інформації [8].

1.2 Ключові проблемні аспекти існуючих рішень

Децентралізовані системи голосування на базі блокчейну мають значний потенціал для підвищення безпеки та прозорості виборчих процесів. Проте їхнє впровадження супроводжується низкою викликів, які обмежують їхнє використання, особливо у масштабних виборах. Розглянемо основні проблемні аспекти, які виділяють дослідники.

Масштабованість – є однією з найбільших перешкод. Блокчейн-платформи, такі як Ethereum, здатні обробляти лише близько 15 транзакцій за секунду, тоді як

національні вибори можуть генерувати мільйони транзакцій. Це призводить до затримок і високих витрат, що робить системи непрактичними для великих виборів [9]. Дослідники пропонують рішення, як шардинг, але вони ще не є стандартними.

Забезпечення приватності виборців також залишається складним завданням. Блокчейн за своєю природою прозорий, що ускладнює захист даних виборців. Технології, такі як докази з нульовим розголошенням (ZKP), можуть вирішити цю проблему, але їхня інтеграція є складною і не завжди ефективною [10]. Без належного захисту існує ризик розкриття голосів або ідентичності виборців.

Енергоефективність – є ще одним викликом. Консенсусні механізми, такі як Proof of Work (PoW), споживають значну кількість енергії, що робить їх дорогими та неекологічними. Це особливо проблематично для країн, які прагнуть зменшити екологічний вплив [11].

Технічна незрілість – блокчейн-технологій ускладнює їхнє впровадження. Обмежене розуміння технології серед громадськості та фахівців гальмує оцінку її потенціалу та довіру до систем [12]. Це вимагає значних зусиль для популяризації та навчання.

Безпека залишається критично важливою, а існуючі системи, такі як Voatz, показали вразливості, включаючи ризики перехоплення голосів або недостатню масштабованість для національних виборів [5]. Ці проблеми підкреслюють необхідність подальших досліджень для створення надійних рішень.

1.3 Модель загроз у децентралізованому голосуванні

Для забезпечення безпеки децентралізованих систем голосування необхідно чітко визначити модель загроз, яка охоплює потенційні ризики та атаки, що можуть вплинути на цілісність, конфіденційність або доступність системи. Модель загроз є основною для ідентифікації вразливостей і розробки відповідних контрзаходів. У децентралізованих системах голосування загрози можуть включати шахрайство з боку виборців, кібератаки на мережу або маніпуляції результатами. Основні категорії загроз систематизовані на основі міжнародних стандартів, таких як ISO/IEC 27001 та NIST SP 800-53, які визначають вимоги до інформаційної безпеки [4][18].

Для розуміння цих загроз наведено в таблиці Б.1 у Додатку Б, яка ілюструє основні категорії загроз у децентралізованих системах голосування, їх описи та приклади атак. Таблиця базується на рекомендаціях стандартів ISO та NIST [4][18].

Додатково виділяються специфічні загрози, такі як атаки Sybil, коли зловмисник створює фальшиві ідентичності для множинного голосування, та мережеві атаки, які можуть порушити доступ до системи [13]. Наприклад, атака DoS може зупинити голосування, якщо система не має достатнього захисту.

Для протидії цим загрозам дослідники пропонують використовувати криптографічні методи, такі як ZKP для приватності та надійні механізми консенсусу, як Proof of Stake (PoS), для підвищення безпеки та енергоефективності [10]. Крім того, системи повинні включати механізми моніторингу та аудиту для виявлення атак у реальному часі.

Модель загроз охоплює широкий спектр ризиків, від технічних до соціальних. Для створення надійної системи необхідно враховувати ці загрози на етапі проектування, використовуючи сучасні технології та стратегії безпеки.

1.4 Модель зловмисника та рівні атаки

У децентралізованих системах захищеного голосування визначення моделі зловмисника є критично важливим для розуміння потенційних загроз і розробки ефективних заходів безпеки. Модель зловмисника описує, хто може атакувати систему, які в нього можливості та які цілі він переслідує. У контексті систем голосування на базі блокчейну зловмисники можуть намагатися порушити цілісність результатів, конфіденційність виборців або доступність системи. Для класифікації загроз використано стандарти НД ТЗІ, ISO та NIST.

Зловмисників можна поділити на кілька категорій залежно від їхньої ролі та намірів:

1) Зловмисні виборці: Це виборці, які можуть намагатися проголосувати кілька разів (подвійне голосування), продати свої голоси або порушити роботу системи. Наприклад, вони можуть використовувати вразливості в мобільному клієнті, щоб пройти процедуру автентифікації.

2) Зловмисні оператори вузлів: У консорціумному блокчейні, де вузли керуються різними організаціями, деякі оператори можуть бути нечесними. Вони можуть маніпулювати механізмом консенсусу, надавати неправдиві дані або співпрацювати з іншими зловмисниками, щоб змінити результати виборів.
3) Зовнішні атакуючі: Це хакери або групи, які не мають прямого доступу до системи, але можуть атакувати через мережу. Вони можуть здійснювати атаки відмови в обслуговуванні (DoS), перехоплювати дані або використовувати вразливості в програмному забезпеченні клієнта чи вузлів.
4) Інсайдери: До цієї групи належать розробники, адміністратори або виборчі чиновники з привілейованим доступом. Вони можуть впроваджувати бекдори, змінювати код або зловживати доступом для маніпуляцій.

Для оцінки загроз атаки поділяються на чотири рівні залежно від ресурсів і технічної складності. Таблиця Б.2, яка наведена у Додатку Б, ілюструє ці рівні відповідно до стандартів ISO та NIST [4][18], забезпечуючи структурований підхід до аналізу атак.

Безпека системи залежить від механізму консенсусу. Наприклад, у Proof of Work (PoW) зловмисник потребує понад 50% обчислювальної потужності для атаки, у Proof of Stake (PoS) – контроль більшості стейкованих токенів. У дозволених блокчейнах, таких як Practical Byzantine Fault Tolerance (PBFT), система може витримати до третини зловмисних вузлів. Модель зловмисника має враховувати максимальну кількість компрометованих вузлів, яку система може толерувати.

Крім того, у системах голосування важлива стійкість до специфічних атак, таких як порушення приватності виборців або примус до певного вибору. Для цього використовуються криптографічні методи, як-от докази з нульовим розголошенням (ZKP), які дозволяють перевіряти голоси без розкриття їхнього вмісту. Дослідження показують, що захист від таких загроз є ключовим для забезпечення довіри до системи [10].

Модель зловмисника охоплює різні типи атакуючих – від виборців до державних акторів – і визначає їхні можливості, щоб система могла протистояти атакам на всіх рівнях.

1.5 Перспективи розвитку та ключові напрями вдосконалення

Децентралізовані системи голосування на основі блокчейну мають потенціал змінити виборчі процеси, зробивши їх прозорішими та безпечнішими. Проте для їх широкого впровадження необхідно подолати низку технічних, соціальних і правових викликів. Нижче розглянуто ключові напрями вдосконалення та перспективи розвитку таких систем.

Масштабованість – поточні блокчейн-платформи, як-от Ethereum, обробляють лише 15–30 транзакцій за секунду, що недостатньо для національних виборів. Для вирішення цієї проблеми дослідники пропонують шардинг – поділ блокчейну на менші паралельні ланцюги – та протоколи другого рівня, які зменшують навантаження на основний ланцюг. Наприклад, технології типу Lightning Network можуть прискорити обробку транзакцій [14].

Конфіденційність та приватність – прозорість блокчейну може загрожувати приватності виборців, якщо голоси можна пов’язати з їхніми власниками. Для захисту конфіденційності використовуються ZKP, кільцеві підписи та гомоморфне шифрування. Ці методи дозволяють перевіряти голоси, не розкриваючи їхнього вмісту [10]. Подальші дослідження мають зосередитися на спрощенні цих технологій для масового використання.

Зручність використання – система повинна бути простою для всіх виборців, включаючи тих, хто не має технічних навичок. Мобільний клієнт має мати інтуїтивний інтерфейс, чіткі інструкції та підтримку різних пристроїв. Інтеграція з системами автентифікації, як-от Дія чи BankID, може спростити процес доступу і підвищити рівень безпеки [16][17]. Зручність є одним з ключових факторів успішного впровадження технології [10].

Безпека – незважаючи на вбудовані функції безпеки блокчейну, система вразлива до атак, таких як компрометація клієнтських додатків чи DoS-атаки. Регулярні аудити, тестування на проникнення та безпечне програмування є

необхідними. Також важливо розробити механізми реагування на атаки в реальному часі, наприклад, виявлення аномалій у транзакціях [14].

Енергоефективність – механізми консенсусу, як-от PoW, споживають багато енергії, що робить їх непрактичними для великих систем. Альтернативи, такі як PoS або Proof of Authority (PoA), є більш енергоефективними і краще підходять для консорціумних блокчейнів. Дослідники пропонують також гібридні моделі, як-от Proof of Vote (PoV), для підвищення ефективності [15].

Стандартизація та взаємодія – розробка міжнародних стандартів для систем голосування сприятиме їх інтеграції та обміну досвідом між країнами. Взаємодія з іншими системами, наприклад, для транскордонного голосування, також є перспективним напрямом.

Децентралізовані системи голосування, засновані на технології блокчейн, мають потенціал суттєво вдосконалити виборчі процеси завдяки забезпеченню прозорості, безпеки та підвищенню довіри з боку учасників. Проведений аналіз сучасного стану таких систем свідчить, що їхнє широке впровадження потребує значних удосконалень для подолання існуючих технічних обмежень.[4][18].

Виходячи з проведеного аналізу сучасного стану децентралізованих систем голосування, у рамках даної визначено низку задвань, спрямованих на створення більш надійних і ефективних рішень. По-перше, планується розробити модель масштабованої системи голосування на основі технології шардингу, це дозволить підвищити пропускну здатність блокчейн-мережі та забезпечить її ефективне функціонування під час голосувань чи виборів. По-друге, буде впроваджено механізми захисту конфіденційності, зокрема докази з нульовим розголошенням, для забезпечення приватності виборців без втрати прозорості процесу. Крім того, значна увага приділятиметься створенню зручного інтерфейсу мобільного клієнта, який буде інтуїтивно зрозумілим для користувачів із різним рівнем технічної підготовки.

Отже, децентралізовані системи голосувань на основі блокчейну мають потенціал суттєво покращити прозорість, безпеку і довіру до виборчих процесів. Однак їх широке впровадження залежить від вирішення ключових викликів, таких

як: масштабність, конфіденційність, зручність у використанні та безпека. Подальші дослідження та розробки мають бути спрямовані на створення ефективних рішень для подолання цих обмежень.

Завдання для подальшої роботи:

1) Провести глибокий літературний огляд та аналіз вимог до децентралізованих систем голосування: безпека, приватність, масштабованість та зручність мобільного клієнта.
2) Обґрунтувати вибір архітектурних рішень і ключових технологій (криптографічні протоколи, Android Keystore, UI-фреймворки) та спроектувати загальну архітектуру системи.
3) Розробити прототип мобільного клієнта: реалізувати механізми автентифікації, захисту й зберігання ключів, цифрового підпису голосів та взаємодії з блокчейном; провести функціональне й безпекове тестування.

2 АНАЛІЗ ВИМОГ ДО ДЕЦЕНТРАЛІЗОВАНИХ СИСТЕМ ЗАХИЩЕНОГО ГОЛОСУВАННЯ

2.1 Аналіз загальних вимог до систем захищеного голосування

Свобода виборців включає два основні аспекти: свободу виборців формувати свою думку та вільне висловлення цієї думки. Принцип вільних виборів вимагає, щоб весь виборчий процес відбувався без будь-якого насильства, примусу, тиску, маніпулятивного втручання чи інших впливів, які можуть здійснюватися державою, організацією або однією чи кількома особами. Виборець повинен мати можливість голосувати особисто та без будь-якого стороннього впливу. Дехто пропонує надавати виборцям посилання на сайти та застосунки політичних партій та груп інтересів і виступає за надання чітко розроблених меню, які дозволяють їм швидко знаходити потрібну їм інформацію перед тим, як проголосувати, щоб сприяти вигляду більш поінформованого та активного громадянина [19].

Не заперечуючи значення інформації та обговорення для можливості здійснення політичних прав, слід наголосити, що інтерактивне та інформативне використання/зручність сайту або застосунку електронного голосування може призвести до неналежного впливу виборців. Завдяки дизайну порталу або сайту, фактично організованого та контрольованого партією влади, інформація та вибір можуть бути спрямовані в певному напрямку. Більше того, часова близькість інформації та голосування може призвести до емоційних опцій [19].

Вимоги щодо вільних виборів застосовуються стосовно «онлайн-середовища» виборця. Свобода прийняття рішень також може бути порушена, якщо повідомлення кампанії відображається на екрані пристрою, поки виборець голосує за електронний бюлетень. В існуючих виборчих схемах заборонено розміщувати рекламу поблизу виборчої дільниці (поблизу неї). Таким чином, процедура електронного голосування повинна зробити технічно неможливим розміщення реклами політичних партій/кандидатів на веб-сайті або у застосунку електронного голосування [19].

Демократична легітимізація електронного голосування спирається на задоволення загальних критеріїв голосування демократичної виборчої системи. Це включає вільне вираження уподобань виборця, навіть шляхом голосування за недійсним або «білим» бюлетенем. Для збереження свободи рішення виборця необхідно забезпечити та гарантувати можливість свідомо подавати недійсний голос. Однак бажано попередити виборців перед тим, як опускати порожній бюлетень, щоб уникнути помилок через можливу відсутність технічних можливостей та досвіду [19].

Примус, купівля голосів та вимагання викликають серйозне занепокоєння у зв'язку з іншими методами дистанційного або заочного голосування. Там, де методи дистанційного голосування були конституційно визнані, регуляторні органи та судді надали великого значення засвідченню того, що виборець особисто заповнив бюлетень. Надання засвідчувальної декларації за допомогою цифрового підпису замість власноруч написаної декларації може слугувати інституційно еквівалентним та доцільним рішенням [19].

Також, щоб протистояти ризикам пов’язаним із таємницею голосування, секрет повинен бути фізично захищеним традиційними процедурами голосування, має бути гарантований під час подачі, передачі, отримання, збору та підрахунку голосів. Жоден із учасників процесу голосування (організатори, посадові особи виборчих комісій, виборці тощо) не повинен мати можливості пов'язати голос з ідентифікованим виборцем або навіть з групою виборців. Електронне голосування – це унікальна онлайн-«транзакція», за допомогою якої слід виключити можливість відстеження, зберігаючи при цьому перевірку автентичності поданого бюлетеня.

Подальшою вимогою є наявність чіткого та очевидного розмежування процедур реєстрації та автентифікації, а також передачі голосу, хоча поєднання безпечної автентифікації виборців, які мають право голосу, та гарантованої таємниці голосування здається складною вимогою, але досить досяжною у сьогоденні. Додатковою складністю є необхідність узгодження таємниці, з одного боку, та прозорості, можливості перевірки та перевірки, з іншого [19].

Таємниця голосування є фундаментальним принципом, що реалізується через приватність і безособовість акту голосування.

2.2 Вимоги до мобільного клієнта в контексті децентралізованої архітектури

Обробка даних виборців під час виборчої процедури повинна бути задумана та організована таким чином, щоб зберігати та гарантувати конфіденційність, цілісність та автентичність, а також дотримуватися відповідного законодавства, якщо таке є. На цьому етапі існує сильне перекриття між законодавством про захист даних та кількома конституційними положеннями, що стосуються конфіденційності та недоторканності приватного життя [19].

Персональні дані, пов'язані з виборчими процедурами, необхідні для них або отримані в результаті їх проведення, повинні використовуватися лише для визначеної мети, тобто для організації та проведення виборів, та у спосіб, сумісний з цією метою.

Несанкціонована передача або розкриття персональних даних третім сторонам заборонено. Збір та обробка даних підлягають обмеженням, що випливають з – у всьому світі, навіть з великими розбіжностями – визнаних принципів остаточності та пропорційності. Несанкціонований доступ та вторинне використання даних можуть перешкоджати виборцям здійснювати свої права і, як наслідок, становити загрозу для демократії. З принципу остаточності можна вивести зобов'язання вживати всіх технічних заходів для анонімізації всіх даних, що стосуються ідентифікованого або такого, що може бути ідентифікований, щойно цілі їх обробки, включаючи перевірку та контроль процедур, будуть досягнуті. Принцип пропорційності вимагає обробки даних, які є адекватними та релевантними для процедур електронного голосування, і вони не є надмірними щодо цієї мети [19].

Відповідно до вище зазначеної логіки, мобільний клієнт системи голосування має бути побудований з урахуванням принципів: «privacy-by-design» та «privacy-by-default» [57].

Усі логічні процеси застосунку, а саме: автентифікація, шифрування та передача даних повинні реалізовуватись саме на стороні клієнта, що зменшить

ризик витоку або маніпуляцій з боку серверної частини. Також, обробка персональних даних, таких як ідентифікаційна інформація, біометричні параметри або токени доступу, повинна виконуватись локально, з використанням захищених середовищ.

Щоб гарантувати захист голосу, застосунок має забезпечувати попереднє шифрування бюлетеня ще до його надсилання, і уникати будь-якого довготривалого зберігання чутливої інформації в пам’яті пристрою. Автоматичне очищення сесійних даних після завершення голосування є додатковим та влучним засобом запобігання несанкціонованому доступу у разі компрометації пристрою користувача.

Окрім технічних аспектів захисту, важливо також розглянути інтерфейсну частину, яка повинна бути інтуїтивно зрозуміла, доступна для осіб з різним рівнем цифрової грамотності та відповідати стандартам інклюзивності (зокрема WCAG). Гарно та вміло структурований інтерфейс допомагає знизити ризик випадкових помилок при голосуванні, а також запобігти розгортанню соціальної інженерії з боку сторонніх осіб, які можуть чинити тиск або вплив на виборця.

Особливу увагу слід приділити обмеженню демонстрації або доказу вибору, зробленого користувачем. Це дозволяє захистити свободу волевиявлення та мінімізує ризики примусу або підкупу. Усі дії в межах клієнта мають бути побудовані так, щоб виборець не мав змоги підтвердити третім особам свій голос, навіть якщо сам цього бажає [56].

2.3 Аналіз типових вразливостей і атак на клієнтські застосунки для голосування

Загалом, типовими вразливостями і атаками на застосунки для голосування можуть бути будь які звичайні атаки, які порушують – цілісність, конфіденційність та автентичність даних, у цьому випадку голосів виборців. Далі розглянемо основні вектори та напрями атак і вразливості які можуть бути застосовані.

2.3.1 Неправильне використання облікових даних

Агенти загроз, що використовують жорстко закодовані облікові дані та їх неправильне використання в мобільних додатках, можуть включати автоматизовані

атаки з використанням загальнодоступних або спеціально розроблених інструментів. Такі агенти потенційно можуть знаходити та використовувати жорстко закодовані облікові дані або використовувати слабкі місця, спричинені неправильним використанням облікових даних [20].

Після виявлення вище зазначених вразливостей зловмисник може використовувати жорстко закодовані облікові дані для отримання несанкціонованого доступу до конфіденційних функцій мобільного додатка. Шахраї також можуть зловживати обліковими даними, наприклад, отримуючи доступ через неправильно перевірені або збережені облікові дані, тим самим обходячи необхідність легітимного доступу [20].

Уникнення незахищеного керування обліковими даними передбачає невикористання жорстко закодованих облікових даних та належну обробку облікових даних користувачів [20].

Кроки, які потрібно виконувати, щоб протистояти зазначеній вразливості [20]:

1) Шифрування облікових даних під час передачі.

2) Використання безпечних, відкличних токенів доступу.

3) Впровадження надійних протоколів автентифікації.

4) Регулярне оновлення та заміна будь-яких використаних ключів або токенів.

2.3.2 Незахищена автентифікація/авторизація потенційно вразливі механізми автен або автор

Недостатньо хороші або відсутні схеми авторизації можуть потенційно дозволити зловмиснику виконати функціональність, на яку він не повинен мати права, використовуючи автентифікованого, але з нижчими привілеями користувача мобільного додатка. Цей ризик атаки ескалації привілеїв підвищується, коли рішення про авторизацію приймаються на мобільному пристрої, а не через віддалений сервер, сценарій, який часто може виникати через вимоги мобільного використання в автономному режимі [21].

Як тільки зловмисник зрозуміє вразливості в схемі автентифікації або авторизації, він може використати ці слабкі місця одним із двох способів. Вони

можуть або підробити, або обійти автентифікацію, безпосередньо надсилаючи запити на обслуговування на сервер мобільного додатка, обходячи будь-яку пряму взаємодію з мобільним додатком, або вони можуть увійти в додаток як легітимний користувач після успішного проходження контролю автентифікації, а потім примусово перейти до вразливої кінцевої точки для виконання адміністративних функцій. Обидва методи експлуатації зазвичай реалізуються за допомогою мобільного шкідливого програмного забезпечення на пристрої або ботнетах, що належать зловмиснику [21].

Щоб запобігти як незахищеній автентифікації, так і авторизації, вкрай важливо уникати слабких шаблонів та посилювати заходи безпеки, а саме [21]:

1) Уникнення слабких шаблонів.

2) За можливістю виконувати усі запити на автентифікацію на стороні сервера.

3) Якщо необхідне зберігання даних на стороні клієнта, потрібно зашифрувати дані за допомогою ключа шифрування, безпечно отриманого з облікових даних користувача для входу.

4) Потрібно уникати використання підроблюваних значень для автентифікації користувача, включаючи ідентифікатори пристроїв або геолокацію.

2.3.3 Незахищена комунікація

Більшість сучасних мобільних програм обмінюються даними з одним або кількома віддаленими серверами. Коли відбувається передача даних, вона зазвичай проходить через мережу оператора мобільного пристрою та Інтернет. Агент загрози, який прослуховує дротовий зв'язок, може перехопити та змінити дані, якщо вони передані у відкритому тексті або за допомогою застарілого протоколу шифрування. Агенти загрози можуть мати різні мотиви, такі як крадіжка конфіденційної інформації, шпигунство, крадіжка особистих даних тощо [22].

Припустімо, що мережевий рівень не є безпечним і вразливим до прослуховування, то що ж потрібно запровадити, щоб він став безпечним [22].

1) Застосовання TLS для транспортних каналів, які мобільний додаток використовуватиме для передачі даних на сервер або веб-сервіс.

2) Використання надійних, стандартних для галузі наборів шифрів з відповідною довжиною ключів.

3) Під час циклів розробки потрібно уникати перевизначення методів перевірки TLS, щоб дозволити ненадійні сертифікати, натомість треба використовувати самопідписані сертифікати або локальний центр сертифікації розробки (CA).

2.3.4 Незахищене зберігання даних

Незахищене зберігання даних у мобільному додатку може привабити різних агентів загроз, які прагнуть використати вразливості та отримати несанкціонований доступ до конфіденційної інформації. До цих агентів загроз належать досвідчені зловмисники, які націлюються на мобільні додатки для вилучення цінних даних, зловмисні інсайдери в організації або команді розробників додатків, які зловживають своїми привілеями, державні суб'єкти, що здійснюють кібершпигунство, кіберзлочинці, які прагнуть фінансової вигоди шляхом крадіжки даних або викупу, розробники скриптів, які використовують готові інструменти для простих атак, брокери даних, які прагнуть використати незахищене сховище для продажу особистої інформації, конкуренти та промислові шпигуни, які прагнуть отримати конкурентну перевагу, а також активісти або хактивісти з ідеологічними мотивами [23].

Ці агенти загроз використовують такі вразливості, як слабке шифрування, недостатній захист даних, незахищені механізми зберігання даних та неправильне поводження з обліковими даними користувачів [23].

Як приклад, можна виділити інцидент пов’язаний із зазначеною вразливістю: під час аудиту мобільного застосунку Voatz, що використовувався на виборах у США, було виявлено серйозні недоліки, які могли дозволити стороннім особам змінювати або відстежувати голоси виборців. Дослідники з MIT відзначили використання застарілих криптографічних протоколів та можливість атак типу man-in-the-middle [55].

Розробникам мобільних додатків та організаціям вкрай важливо впроваджувати стійкі заходи безпеки, такі як надійне шифрування, безпечні методи

зберігання даних та дотримання найкращих практик безпеки мобільних додатків, щоб зменшити ризики, пов'язані з незахищеним зберіганням даних [23].

Окремо можна виділити [23]:

1) Механізми безпечного зберігання: збереження конфіденційних даних в безпечних місцях зберігання, недоступних для неавторизованих користувачів. Специфічні для платформи механізми безпечного зберігання, що надаються мобільною операційною системою, такі як Keychain (iOS) або Keystore (Android).

2) Перевірка вхідних даних та очищення даних: методи перевірки вхідних даних та очищення даних, щоб запобігти атакам ін'єкцій та забезпечити збереження лише дійсних та очікуваних даних. Потрібно перевіряти вхідні дані користувача, щоб зменшити ризик ін'єкції шкідливого коду або ненавмисного витоку даних.

2.4 Комлексний підхід до забезпечення безпеки мобільного клієнта

Безпека мобільного клієнта в системі електронного голосування – це не про окремі захисні механізми, а про цілу стратегію, що охоплює всі етапи взаємодії користувача з додатком. У попередніх підрозділах було розглянуто основні вразливості, які можуть становити загрозу для конфіденційності, цілісності та автентичності голосу. Але важливо не лише знати про ці ризики, а й розуміти, як їм ефективно протидіяти.

Комплексний підхід означає, що захист має бути реалізований на кількох рівнях одночасно: під час передачі даних, при зберіганні інформації, у процесі автентифікації користувача, у середовищі виконання додатка та навіть на рівні інтерфейсу. Крім того, постійний моніторинг, аудит та оновлення системи є обов’язковими елементами для підтримання належного рівня безпеки протягом усього життєвого циклу застосунку.

Також важливо враховувати людський фактор: користувач може використовувати зламаний пристрій, підключатися до незахищеної мережі чи натрапити на фальшивий додаток. Саме тому кожен захист повинен працювати незалежно від інших – як додатковий бар'єр, якщо попередній був обійдений, атаку на себе прийме наступний.

Щоб систематизувати ключові аспекти цього підходу, у таблиці В.1, яка наведена у Додатку В, наведено основні рівні безпеки мобільного клієнта, типові загрози для кожного з них, а також відповідні механізми захисту.

Такий підхід дозволяє будувати не просто функціональний, а по-справжньому захищений мобільний клієнт, здатний протистояти як технічним, так і соціальним загрозам. У наступному розділі буде розглянуто, які саме технології доцільно використати для реалізації цих механізмів у рамках обраної архітектури мобільного застосунку.

2.5 Висновки до розділу

У цьому розділі ми встановили, що мобільний клієнт децентралізованої системи електронного голосування повинен реалізувати комплекс вимог, які гарантуватимуть і технічну стійкість, і зручність для користувача. Зокрема, серед найважливіших для самого застосунку можна виділити:

1) шифрування даних у пам’яті та під час передачі задля конфіденційності й цілісності кожного голосу;
2) цифровий підпис кожної голосової транзакції з використанням ізольовано збережених у TEE Android Keystore ключів.

Разом із цим клієнт має забезпечувати просту інтеграцію зі сторонніми сервісами (Firebase – зараз, BankID та «Дія» – у перспективі) через стандартизовані API та дотримуватися принципів доступності і локалізації.

У межах загальної системи пріоритетними залишаються прозорість і масштабованість:

1) публічні логи транзакцій та відкриті API дадуть змогу зовнішньому аудиту,
2) шардинг і рішення другого рівня (включно з PoS чи Proof of Vote) забезпечать необхідну пропускну здатність та знизять енергоспоживання.
3) Не менш важливим є дотримання міжнародних стандартів (GDPR, ISO/IEC 27001, NIST SP 800-63) для юридичної та технічної відповідності.

Щоб протидіяти основним векторним атакам, система передбачає такі контрзаходи:

1) Man-in-the-Middle: TLS 1.2 + сертифікатний pinning;

2) Replay-атаки: унікальні nonce і часові мітки для кожної транзакції;
3) DoS: rate-limiting, масштабування через CDN;
4) Підміна коду клієнта: обфускація APK і перевірка його цілісності під час запуску;
5) Side-channel атаки: винесення криптоопераційу захищену зону TEE.

Лише така багаторівнева архітектура, що поєднує технічні й UX-рішення, дозволить досягти високого рівня безпеки, довіри та доступності.

Виходячи з проведеного аналізу, у наступному розділі буде виконано детальну проєктну роботу над клієнтською частиною:

1) сформулювати та структурувати вимоги до мобільного додатка,
2) обґрунтувати вибір архітектури та політик захисту збірки й доступу до даних
3) розглянути криптографічну підтримку на стороні клієнта: вибір алгоритмів, взаємодію з блокчейном і бібліотеки,
4) розробити концепцію інтерфейсу користувача з декларативним підходом і підготовкою до мультиплатформенності,

3 АНАЛІЗ І ВИБІР ТЕХНОЛОГІЙ ПОБУДОВИ МОБІЛЬНОГО КЛІЄНТА

3.1 Вимоги до клієнтської частини системи голосування

Розробка мобільного клієнта для децентралізованої системи захищеного голосування ставить перед собою комплексне завдання – забепечити не лише повноцінну функціональнісь, а й відповідність актуальним вимогам інформаційної безпеки. Такий застосунок має працювати в умовах, де довіра до результатів голосування формується не лише через наявність крипто-механізмів, а й завдяки прозорості, надійності та цілісності клієнтської реалізації. Отже, ще на етапі планування необхідно враховувати вимоги, як міжнародних, так і національних нормативів.

У рекомендаціях NIST SP 800-124 підкреслюється, шо мобільні пристрої частіше використовуються для доступу до критичних сервісів. У зв’язку із цим, особлива увага має приділятися контролю доступу, збереженню даних та обмеженню дозволів застосунків [24]. У національному контексті ці питання регламентуються нормативами НД ТЗІ, що зобов’язують розробників передбачати захист даних від несанкціонованого доступу і впроваджувати механізми автентифікації, контролю цілісності та логування подій, пов’язаних з обробкою чутливих даних [25].

Мобільний клієнт який розробляється у даній роботі, передбачає реалізацію повного циклу взаємодії користувача із системою голосування, починаючи з авторизації в системі та до відображення результатів і перевірки їх автентичності. На цьому етапі, зважаючи на обмеження інфраструктури, для автентифікації обрано сервіс Firebase Auth. Цей інструмент дозволяє швидко впровадити базову автентифікацію користувача з подальшою можливістю розширення до багатофакторної схеми. У перспективі розглядається інтеграція з BankID та Дія – державних провайдерів електронної автенціфікації, які базуються на протоколах Oauth 2.0 та OpenID Connect. Проте, ця частина наразі виходить за межі реалізації і потребує наявності серверної інфраструктури та фінансових витрат.

Після проходження автентифікації, мобільний клієнт має забезпечити створення унікальної пари криптографічних ключів для кожного користувача. Генерація відбувається на самому пристрої з використанням Android KeyStore – вбудованого сховища, яке ізолює приватні ключі від основого середовища виконання. Завдяки цьому, навіть при отриманні root-доступу до пристрою, сторонні програми не матимуть доступу до ключів, та не зможуть їх вилучити. Захист на апаратному рівні реалізується через TEE (Trusted Execution Environment), який виконує криптографічні операції у відокремленому середовищі, мінімізуючи ризики компрометації [26].

Важливо також приділити увагу захисту передачі даних. У межах поточної реалізації, комунікація клієнта із системою здійснюється через сервіси Firebase, рішення котрого використовує стандарти шифрування підкдючення (TLS 1.2 і вище) із “коробки” [29]. Таким чином, забезпечується захист каналу передавання даних від потенційного перехоплення чи модифікації. Оскільки вбудований Firebase SDK не дозволяє повноцінно реалізувати certificate pinning на стороні клієнта, ця функція розглядається як можливе доповнення у випадку переходу до власного API.

Останнім етапом взаємодії клієнта із системою є отримання підтвердження про участь у голосуванні. У межах поточної реалізації з використанням про участь у голосуванні. У межах поточної реалізації, результати зберігаються централізовано у Firestore, що дозволяє оперативно отримати інформацію про стан голосування. Перевірка узгодженості результатів між вузлами мережі блокчейну, як і перегляд інформації про конкретні ноди, є перспективною функціональністю, яка потребує переходу до децентралізованої архітектури та інтеграції з відповідними API вузлів.

3.1.1 Захист клієнтської збірки та політики доступу до даних

У процесі підготовки релізної збірки додатка важливо подбати і про захист коду. Для цього в Gradle-конфігурації проекту активовано параметр minifyEnabled та очищення невикористаних ресурсів, а також налаштування R8 (ProGuard). Обфускація байткоду унеможливлює декомпіляцію додатку, вилучення та аналіз логіки програми. Це значно ускладнює підбір імен класів та методів

зловмисниками[30], що прагнуть зрозуміти структуру клієнта голосування та виявити потенційно вразливі місця.

Цифровий підпис релізного пакета, це ще один важливий рівень захисту. Для публікації в Play Store застосунок збирається у форматі AAB та підписується ключем, який зберігається у сховищі Play App Signing. Це гарантує, що кінцевий користувач інсталює саме автентичний пакет, а спроби поширення змінених APK/AAB будуть відхилятися захистом Play Store, ще до того, як додаток буде встановлено на пристрій [31]. За потреби у процесі виконання можна здійснювати перевірку контрольного відбитка підпису (signatureFingerprint) і припиняти роботу у разі виявлення невідповідності.

Рівень політик безпеки даних доповнює клієнтську захищеність на серверній (BaaS) стороні. У Firebase Security Rules прописуються умови, за яких користувач може читати або записувати документи у Firestore, наприклад, доступ до колекції історії голосувань надається лише за наявності автентифікації (request.auth != null) та тільки до власних записів (request.auth.uid == userId), приклад наведено на рисунку 3.1. Таким чином, навіть у разі компрометації іншого API чи експлуатації вразливості на стороні клієнта, сервер відходить несанкціоновані запити [32].

```
match /VotesHistory/{userId} {
  allow read: if request.auth != null;
  allow write: if request.auth != null && request.auth.uid == userId;
}
```

Рисунок 3.1 – Приклад правил до коллекції Firestore

3.2 Аналіз і вибір архітектури клієнтського застосунку

Побудова мобільного клієнта в рамках проєкту децентралізованої системи захищеного голосування передбачає не лише впровадження окремих технологій, а й продуману архітектуру, яка забезпечить розподілення відповідальностей, полегшить тестування, подальше масштабування та адаптування під інші конпецпії. Під час вибору архітектури напрямок був визначений двома головними критеріями: по--перше, максимальна простота підтримки та розширення коду власними силами,

по-друге, вимоги до безпеки й відповідність рекомендаціям Google для Android-додатків.

З огляду на зазначені вимоги та власний досвід, для реалізації цілей роботи було обрано паттерн Model–View–ViewModel (MVVM). У рамках MVVM шар View відповідає виключно за відображення і взаємодію з користувачем, тоді як бізнес-логіка та обробка подій переходять до ViewModel. ViewModel, у свою чергу, спільно з репозиторіями формує єдиний інтерфейс доступу до даних – будь то виклики Firebase, локальна база Room або майбутнє звернення до блокчейн-нод. Такий поділ знижує кількість обсяг відповідальностей та навантаженності на UI, який не повинен містити жодної логіки, а лише спостерігати за потоками даних Flow і реагувати на їх зміни [35].

Наявність шару репозиторія над ViewModel вважається рекомендованим підходом від Google, присвячених кращім архітектурним практикам [33]. Репозиторій інкапсулює джерела даних і ховає деталі реалізації – це підвищує тесто-придатність. У юніт-тесті можна замінити реальний репозиторій мок-версією з фейковими даними, не втручаючись у логіку ViewModel. У нашому випадку буде два ключові репозиторії: AuthRepository, що працює з FirebaseAuth для входу/реєстрації та VotingRepository, який звертається до Firestore для отримання списків голосувань, зберігання результатів та створення своїх. Перехід до майбутньої роботи з блокчейном вимагатиме розширення VotingRepository новими методами, але реалізація для ViewModel не зміниться – це і є модульність архітектури.

Наступним важливим елементом є ін’єкція залежностей (dependency injection). З огляду на потреби безпечного та прозорого створення екземплярів сервісів, було обрано Hilt – легковаговий фреймворк від Google, який спрощує налаштування Dagger2 і автоматично генерує компоненти на етапі компіляції. За допомогою Hilt немає необхідності вручну описувати графи залежностей, достатньо анотувати класи @Inject та модулі @Module, а Hilt подбає про життєвий цикл об’єктів відповідно до вимог Activity, ViewModel чи Singleton [11]. Цей підхід узгоджується з рекомендаціями NIST SP 800-218, які стосуються побудови

безпечного програмного забезпечення, зокрема в аспектах управління конфігурацією, розділення відповідальностей між компонентами та контролю життєвого циклу [13].

Отже, вибір архітектури MVVM + Repository + Hilt + Coroutines + Compose формує модульну та тесто-придатну основу клієнта. Завдяки такому підходу, при подальшій інтеграції механізмів роботи з децентралізованою мережею блокчейну або при додаванні складних криптографічних протоколів, зміни обмежаться, в основному, репозиторіями та сервісами, а ViewModel і UI залишаться стабільними [10][12]. Це забезпечує відповідність принципам “separation of concerns” і “extendability”, які є загальновизнаними в розробці безпечного ПЗ згідно з OWASP MASVS [4] та ISO/IEC 27002 [14].

3.3 Криптографічна підтримка на стороні клієнта

Криптографічні функції мобільного клієнта базуватимуться на Android Keystore. Це API, що дозволяє генерувати та зберігати ключі у захищеному апаратному сховищі. Згідно з документацією Android, клас Android KeyStore забезпечує керування приватними ключами у “довіреному обладнанні” (наприклад, TEE чи Secure Element), що суттєво ускладнює їх викрадення з пристрою [18]. Починаючи з Android 7.0, Keystore обов’язково реалізується в апаратурі, тож навіть у випадку компрометації ОС, приватні ключі залишаться невилучальними [19]. При генерації ключів ми можемо скористатися KeyPairGenerator із провайдером "AndroidKeyStore" і вказати KeyGenParameterSpec, який дозволяє задати обмеження. Наприклад, за призначенням імені ключа та умою, що користувач автентифікаваний. Удосконалення API Keystore у Android дозволило додати підтримку симетричних алгоритмів (AES, HMAC) і гнучкі політики використання. Ключі можна обмежити для використання лише після введення PIN/пароля або лише для певних алгоритмів [18][19]. Таким чином, генерація ключів на Android може виконуватися безпечно в контейнері Keystore, із захистом у TEE або Secure Element, забезпечуючи апаратний рівень довіри та захищене зберігання ключів.

Для автентифікації запитів та цифрового підписування бюлетенів будуть використовуватися ці приватні ключі. Оскільки ключі генеруються і зберігаються

в Android Keystore, мобільний додаток може підписати дані без ризику витоку інформації. Це дозволяє гарантувати цілісність і достовірність запитів. Android Keystore забезпечує API для підписування даних (ECDSA чи RSA, залежно від алгоритму) [18]. Застосунок може, отримати об’єкт Signature, ініціалізувати його приватним ключем з Keystore та підписувати хеш повідомлення голосування на сервер. Самі приватні ключі залишаються невидимими для розробника й користувача – це сприяє захищеному зберіганню, оскільки ключі неможливо експортувати чи відтворити.

3.3.1 Вибір криптографічного алгоритму

При виборі алгоритму варто зважати на баланс безпеки, продуктивності та сумісності з майбутньою блокчейн-інтеграцією. Secp256k1 – це стандартна 256-бітна елептична крива, широко використовувана в криптовалютах Bitcoin та Ethereum, що забезпечує ~128-бітну криптостійкість [43]. Однак вона не входить до НД ТЗІ (ДСТУ 4145-2002) і не була рекомендована в українських стандартах. Натомість, Ed25519 (адоптована версія схеми EdDSA на базі Curve25519) демонструє декілька переваг – високу продуктивність, коротші відкриті ключі та підписи, стійкість до колізій і відсутність необхідності витрачати додаткові рандомні дані при підписанні [4]. Відповідно до сучасних вимог до цифрового підпису (не менше 128 біт безпеки, швидке підписання або перевірка та стійкість до сторонніх атак), Ed25519 добре підходить для мобільних застосунків голосування [44]. Крім того, радіотехнічні дослідження наголошують на потребі оновлення стандарту ДСТУ 4145-2002 і переході на нові алгоритми із високою продуктивністю [44]. Водночас, якщо в майбутньому передбачається взаємодія з існуючими блокчейнами на основі Bitcoin або Ethereum, логічним буде використання secp256k1, заради сумісності. Важливо також переконатися, що обрані алгоритми підтримуються вибраними бібліотеками, наприклад, Android Keystore має вбудовану підтримку ECDSA на NIST-кривих та RSA, але не підтримує Ed25519 “з коробки”. Можна, застосувати Ed25519 через сторонню бібліотеку Bouncy/Spongy або використовувати secp256k1, який може бути реалізований через сторонні сервіси.

3.3.2 Потенційна взаємодія з блокчейн

У майбутніх оновленнях клієнт може взаємодіяти з децентралізованим блокчейном, для запису результатів голосування чи перевірки прав виборців, тощо. Вибір криптографічних алгоритмів впливає на таку інтеграцію. Блокчейни на базі Bitcoin/Ethereum традиційно використовують ECDSA secp256k1, тож щоб користувачі могли використовувати наявні ключі або гаманці, має сенс забезпечити підтримку цієї кривої. Водночас, інші платформи підтримують EdDSA на Curve25519. Тому вибір може бути подвійним – або гнучка система з можливістю використання обох алгоритмів, або вибір одного, на основі цільового середовища. Важливо гарантувати, що відкриті ключі, які клієнтський застосунок ґенерує та надсилає, відповідають формату й вимогам блокчейн-протоколів, а також, що ключова генерація відповідає стандартам. За потреби, можна використовувати функції сумісності і конвертувати ключі формату AndroidKeystore у відомі блокчейн-формати.

3.3.3 Порівняння криптографічних бібліотек

У процесі впровадження криптографічного шару мобільного клієнта було досліджено кілька Java та Kotlin-бібліотек, здатних реалізувати як симетричне шифрування, так і цифрові підписи. Спочатку увагу привернув Bouncy Castle – класична й визнана в академічному середовищі бібліотека, яка мала б усе необхідне, але Android-платформа постачає власний урізаний пакет Bouncy, а просте оновлення версії зазвичай приводить до конфліктів класів. Саме тому виникла ідея використати його андроїд-специфічний відгалужену версію – Spongy Castle, де пакети перейменовані, що усуває проблему дублювання і дозволяє скористатися алгоритмами Bouncy без втручання в системний провайдер [45-46].

Однак у Google/BoringSSL виникла альтернатива під назвою Conscrypt. Це сучасний JCA/JCE-провайдер із нативними реалізаціями AES-GCM, RSA і ECDSA, оптимізованими під Android та без необхідності вбудовувати важкі Java-бібліотеки[47]. Згідно з OWASP MASVS-CRYPTO, Conscrypt сьогодні вважається пріоритетним вибором за замовчуванням для Android, адже він гарантує найвищу продуктивність і найменшу площу потенційних вразливостей

[49]. Разом із цим ми розглянули Google Tink – крос-платформну бібліотеку з високорівневим API, яка суттєво спрощує правильне використання криптографічних схем (AES-GCM, ECDSA, HMAC). Tink ідеально підходить для швидкої розробки прототипів і гарантує відповідність внутрішнім рекомендаціям Google [48].

Для реалізації підписів на кривих Ed25519 чи secp256k1 можемо комбінувати Android Keystore (для генерації ключів у TEE/StrongBox) із зовнішніми провайдерами. Spongy [26][41] – для Ed25519 [23], Conscrypt/Bouncy – для secp256k1 [45][47]. При цьому, рекомендації NIST SP 800-124 однозначно вимагають апаратне коріння довіри та захищене зберігання [24], а NIST SP 800-56A прописує мінімальні рівні безпеки ECC-кривих (не менше 112 біт) [50].

Паралельно перевірили відповідність рішення ISO/IEC 27002 та OWASP MASVS CRYPTO (заборона власноручних схем, використання лише перевірених API й заборона SHA1PRNG) [37][49]. Також було враховано національні вимоги НД ТЗІ, хоча ДСТУ 4145-2002, що ґрунтується на кривих Галуа, залишається чинним нормативом, його пряме застосування нині трапляється рідше; тому для забезпечення 128-бітової криптостійкості було обрано алгоритми Ed25519 та secp256k1, доповнюючи ними вимоги ДСТУ 7564/7624 [25, 51-53].

Таким чином, ми з’ясували, що Conscrypt разом із Android Keystore, для генерації ключів у TEE/StrongBox, може стати основою криптомодуля, а Spongy Castle і Google Tink забезпечать гнучкість у роботі з нестандартними кривими чи високорівневими сценаріями. Усі обрані бібліотеки та механізми відповідають сучасним міжнародним і українським вимогам безпеки, створюючи фундамент для подальшої блокчейн-інтеграції.

3.4 Інтерфейс користувача: декларативний підхід і підготовка до мультиплатформності

Розробка користувацького інтерфейсу в мобільному додатку голосування спирається на сучасний декларативний підхід, який дозволяє прямо пов’язувати дані програми з її візуальним представленням, автоматизуючи оновлення інтерфейсу при зміні стану. У цьому проєкті для побудови UI використано Jetpack

Compose – фреймворк, що надає змогу описувати компоненти інтерфейсу у вигляді функцій, пов’язаних з моделлю даних, без необхідності ручного управління XML-макетами чи View-ієрархією. Такий підхід зменшує дублювання коду, підвищує подальшу підтримку інтерфейсу й забезпечує коректне оновлення елементів UI у відповідь на зміну стану даних, що особливо актуально для динамічного контенту, такого як списки голосувань або результати в реальному часі [38].

Навігація між екранами реалізована за допомогою Jetpack Compose Navigation із підтримкою @Serializable аргументів, що робить навігацію типобезпечною і дозволяє зручно направляти користувача з однієї форми в іншу, передаючи складні об’єкти без необхідності ручної серіалізації в Bundle [36]. Отже, це не лише підвищує зручність розробки, а й знижує ризик помилок, пов’язаних із втратами даних при передачі між екранами.

Окрему увагу приділено типографії. Для відтворення українського інтерфейсу використовується шрифт e-Ukraine, розроблений Міністерством цифрової трансформації України та відкритий для вільного використання. Підключення цього шрифту в Jetpack Compose відбувається через FontFamily, що гарантує високу читаність тексту українською та відповідає державним вимогам до офіційних додатків [37].

Зауважимо, що такий вибір технологій закладає основу для подальшого розширення клієнта на інші платформи. Compose Multiplatform дає змогу використовувати ті самі декларативні компоненти UI на iOS та в десктопних середовищах (Kotlin/Native), суттєво спрощуючи розвиток крос-платформного рішення. Таким чином, у майбутньому можна повністю перевикористовувати візуальну частину та бізнес-логіку, підтримуючи єдиний інтерфейс на різних пристроях без дублювання коду.

3.5 Висновки до розділу

У цьому розділі було проаналізовано основні технології, які використовуються для побудови безпечного та надійного мобільного клієнта в межах децентралізованої системи електронного голосування. На основі вивчених

вимог і можливостей сучасних платформ було обґрунтовано вибір Firebase як базового інструменту для автентифікації, а також Android Keystore – для захищеної генерації та зберігання ключів.

Мобільний додаток повинен гарантувати: конфіденційність і цілісність кожного бюлетеня, автентичність виборця, зручність користування та стійкість до збоїв мережі. Умовно, це зводиться до чотирьох груп критеріїв: безпека, продуктивність, зручність використання, підтримуваність.

Модель MVVM у поєднанні з шаром Repository, Hilt-інжекцією залежностей та асинхронними корутинами Kotlin розділяє логіку, спрощує тестування й оновлення компонентів. Завдяки цьому:

1) Безпека – секретні ключі й криптооперації ізольовано в TEE Android Keystore, а чітка структура коду зменшує поверхню атаки.
2) Продуктивність – корутини усувають блокування інтерфейсу, дозволяючи обробляти підписи й передавання бюлетенів у фоні.
3) Зручність – Jetpack Compose дає змогу швидко будувати сучасний, адаптивний інтерфейс з підтримкою української та інших мов і з урахуванням базових принципів доступності для користувачів з особливими потребами.
4) Підтримуваність – чітка залежність між модулями полегшує внесення змін, тестування та підготовку до портування на інші платформи.

Що виконується «на стороні клієнта»:

1) Генерація і зберігання пар ключів у захищеній пам’яті;
2) Цифровий підпис бюлетенів алгоритмами Ed25519 / ECDSA-P-256;
3) Багатофакторна автентифікація через Firebase Auth із можливістю подальшого підключення BankID/«Дія»;
4) Перевірка цілісності APK і шифрування локальних кешів.
5) Якою має бути ідеальна система

У завершеному вигляді клієнт повинен «непомітно» для виборця виконувати всі критичні криптографічні операції, забезпечувати анонімність через Zero-Knowledge Proofs, синхронізуватися з консорціумним блокчейном та автоматично

масштабуватись під пікові навантаження. Саме такий рівень зрілості й буде ціллю практичної реалізації у розділі 4, де спроєктовані рішення трансформуються у робочий прототип з інтегрованим тестовим середовищем та первинними вимірами безпеки й продуктивності.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО КЛІЄНТА СИСТЕМИ

4.1 Архітектура клієнтського додатка

Мобільний клієнт системи реалізовано на платформі Android із використанням мови Kotlin. Загальна архітектура додатка побудована на багаторівневому підході з розподілом відповідальності між шарами інтерфейсу та даних. Застосовано патерн MVVM (Model-View-ViewModel) для чіткого відокремлення логіки бізнес-процесів від коду інтерфейсу. На рисунку Г.1 у Додатку Г схематично показано основні шари додатка: шар UI (інтерфейс користувача), опціональний проміжний шар (ViewModel/Domain), а також шар даних (робота з зовнішніми сервісами та сховищами).

У клієнтському додатку шар UI відповідає за відображення даних і взаємодію з користувачем (екрани авторизації, списки голосувань, форми для голосування та створення опитувань тощо). Він представлений Activity та Fragment (або відповідними компонентами Jetpack Compose), які забезпечують відображення елементів інтерфейсу та обробку подій користувача. Ці UI-компоненти мінімально містять логіку – основна логіка зосереджена у шарі ViewModel/Domain.

ViewModel (стан додатка) слугує посередником між UI та шаром даних. ViewModel отримує дані з репозиторіїв, зберігає стан інтерфейсу (наприклад, список доступних голосувань, деталі вибраного голосування, стан авторизації) та надає ці дані UI у зручному для відображення вигляді. Взаємодія відбувається за принципом *односпрямованого потоку даних*: дані течуть від шару даних до UI, а події (наприклад, натиснення кнопки "Проголосувати") – від UI до ViewModel і далі до шару даних. Такий підхід підвищує передбачуваність і керованість стану додатка [58].

Шар даних відповідає за взаємодію з зовнішніми сервісами та сховищами, зокрема, з платформою Firebase (Firestore, Auth тощо) та криптографічними функціями Android. Цей шар може бути реалізований у вигляді набору репозиторіїв (наприклад, AuthRepository, VotingRepository тощо), які інкапсулюють роботу з Firebase і локальним сховищем. Такий підхід спрощує зміну джерел даних і

полегшує тестування. Репозиторії взаємодіють із Firebase Firestore, Firebase Authentication і іншими компонентами, та надають ViewModel необхідні дані у вигляді асинхронних потоків Kotlin Flow.

В рамках проєкту Domain-шар не виділявся явно, оскільки бізнес-логіка відносно проста і зосереджена безпосередньо в репозиторіях та ViewModel. Однак загальні принципи побудови – такі як розподіл відповідальностей і Single Source of Truth – дотримано, кожен клас має чітко визначену зону відповідальності, а дані в додатку мають єдине достовірне джерело (наприклад, Firestore виступає джерелом правди для списку голосувань та результатів) [58].

Значну увагу приділено безпеці архітектури. Критично важливі операції, такі як генерування і зберігання криптографічних ключів, ізолюються на рівні системних сервісів Android (Keystore, про що детальніше в підрозділі 4.3). Таким чином, приватні криптографічні операції не здійснюються у шарі UI, що зменшує ризики витоку даних.

Отже, архітектура клієнтського додатка спроєктована з урахуванням принципів модульності, підтримки приватності користувачів та захищеності даних.

4.2 Авторизація користувача

Для доступу до функцій системи голосування користувач проходить автентифікацію за допомогою свого Google-акаунту. У додатку реалізовано інтеграцію з Firebase Authentication з методом Sign in with Google. З технічної точки зору, використано Firebase Auth у поєднанні з новим API Credential Manager (раніше відомим як Google Sign-In/Credentials API). Це забезпечує єдиний інтерфейс для автентифікації і вибору облікового запису Google на пристрої, що робить процес входу для користувача простим і захищеним [59].

При відкритті додатка користувачеві пропонується увійти через Google. Натискання кнопки "Увійти з Google" викликає Credential Manager, який відкриває стандартний нижній лист із вибором акаунтів Google, вже присутніх на пристрої (або можливістю додати новий обліковий запис). Після того, як користувач обирає свій обліковий запис, додаток отримує від Google маркер автентифікації (OAuth 2.0 ID Token). Цей маркер передається до Firebase Authentication для входу.

Важливим нюансом є використання OAuth Client ID проекту. Firebase вимагає вказати Web Client ID (ідентифікатор клієнта для OAuth 2.0) для перевірки ID Token від Google. Цей ідентифікатор зберігається у додатку як константа, але з метою безпеки його не закодовано жорстко в коді. Натомість застосовано файл налаштувань Gradle secret.properties (див. рис. Г.2 у Додатку Г), де зберігаються чутливі дані (такі як GOOGLE_WEB_CLIENT_ID). Під час збірки Gradle завантажує цей файл і підставляє ідентифікатор у відповідне поле (наприклад, у вигляді BuildConfig змінної або ресурсу рядка). Такий підхід узгоджується з рекомендаціями Google – не зберігати відкрито ключі API та інші секрети в репозиторії коду [60].

Для реалізації авторизації було додано необхідні залежності для Firebase Auth та Credential API у Gradle. Наприклад, використано бібліотеки androidx.credentials:credentials та credentials-play-services-auth для роботи з Credential Manager, а також com.google.firebase:firebase-auth для інтеграції з Firebase. Після налаштування Google Sign-In в консолі Firebase та Google Cloud Console: увімкнення методу входу Google та додавання SHA-ключів додатку (див. рис. Г.3-Г.5 у Додатку Г), проєкт отримав конфігураційний файл google-services.json, це можна побачити на рисунку Г.6 який наведено у Додатку Г, показано необхідні OAuth-параметри. Цей файл додано до проєкту, щоб Firebase SDK міг автоматично визначати параметри OAuth-клієнта.

Процес авторизації проходить наступні кроки:

1) Додаток формує запит Credential Manager на отримання облікових даних: створюється GetCredentialRequest із опцією Google (GetGoogleIdOption), приклад реалізації модуля наведено у додатку А. У цій опції зазначено serverClientId – ідентифікатор клієнта OAuth (взято з secret.properties або google-services.json) – та інші параметри (наприклад, setFilterByAuthorizedAccounts(true) для відображення лише обліковок, що вже використовувались для входу).

2) Відображення вибору акаунта на рисунку Г.7, який наведено у Додатку Г. Credential Manager автоматично відображає стандартний інтерфейс вибору акаунта Google. Користувач обирає бажаний акаунт і підтверджує вхід.
3) Отримання маркера та автентифікація в Firebase. Після вибору акаунта Credential Manager повертає результат – об'єкт, що містить облікові дані Google (зокрема ID Token). Цей токен використовується для входу в Firebase через метод FirebaseAuth.signInWithCredential(...). Для цього ID Token обгортається в спеціальний Firebase AuthCredential (тип GoogleAuthProvider.getCredential(token, null)). Далі FirebaseAuth виконує автентифікацію і повертає інформацію про користувача (UID, email, ім'я тощо), Auth Credentials наведено у додатку А.
4) Завершення входу. Після успішної автентифікації користувача система може переходити до основного функціоналу. У додатку передбачено збереження деяких даних профілю користувача локально, наприклад ім'я або email (для відображення в інтерфейсі), хоча основним ідентифікатором користувача є його Firebase UID.

Використання Google, як провайдера автентифікації підвищує рівень безпеки, оскільки автентифікація відбувається через захищені сервіси Google. Додаток не оперує паролями користувача – він отримує лише перевірений маркер від Google. Firebase виступає довіреним посередником, який валідуючи токен, створює сесію користувача. Таким чином, ризики фішингу чи компрометації облікових даних зведені до мінімуму, а користувач отримує зручність єдиного входу. Після входу додаток переходить до головного екрану зі списком доступних голосувань, а UID автентифікованого користувача використовується для авторизації операцій у базі даних (через правила безпеки Firestore).

4.3 Генерація ключів та підпису

Однією з ключових вимог безпечної системи голосування є забезпечення криптографічної цілісності і перевірки дійсності голосів. У реалізованому клієнті кожен голос цифрово підписується користувачем. Для цього при першому запуску

додаток генерує пару ключів – приватний та відкрити ключ – і надалі використовує їх для підписування кожного голосу.

Реалізація виконана за використання Android Keystore, це підсистема, яка дозволяє зберігати криптографічні ключі в захищеному середовищі пристрою (апаратно ізольованому, наприклад у TEE). Важливо, що згенерований приватний ключ ніколи не покидає межі Keystore: криптографічні операції з ним виконуються всередині системного сервісу, а ключові дані недоступні додатку. Це означає, що навіть у разі компрометації пам’яті додатка зловмисник не зможе експортувати приватний ключ поза пристрій. Для генерації ключової пари застосовано алгоритм ЕЦП ECDSA на еліптичній кривій P-256 (також трапляються її синоніми secp256r1 та prime256v1) із використанням геш-функції SHA-256. Фрагмент коду, що реалізує створення такої пари ключів, наведено на рисунку Г.8, що наведено у Додатку Г.

Код генерує нову пару ключів (приватний та публічний) з аліасом "user_key_alias" у сховищі AndroidKeyStore. Параметр setUserAuthenticationRequired(false) означає, що для використання ключа не потрібна додаткова автентифікація (наприклад, відбиток пальця) – в контексті голосування це прийнятно, оскільки ми довіряємо автентифікації через Google і хочемо уникнути зайвого тертя для користувача під час голосування.

Згенерований приватний ключ залишається у захищеному сховищі, а публічний ключ можна експортувати для зберігання в базі даних.

Після генерації пари ключів клієнт кодує відкритий ключ у форматі DER (структура SubjectPublicKeyInfo), перетворює його на Base64-строку і записує у Firestore. Таким чином, у реалізації не створюється X.509-сертифікат і не застосовується формат PEM; зберігається саме Base64 відкритого ключа у двох згаданих локаціях, що спрощує схему даних і водночас гарантує перевіряємість цифрового підпису.

Коли користувач обирає варіант або декілька варіантів та натискає «Проголосувати», клієнт формує повідомлення для підпису у вигляді рядка <pollId>|<choiceId>|<timestamp> – тобто ідентифікатор голосування, ідентифікатор конкретного варіанта та обов’язкову поточну мітку часу (див. рис. Г.9 у Додатку

Г). Це забезпечує унікальність підпису й виключає можливість його повторного використання. Далі приватний ключ, збережений у Android Keystore, застосовується через клас Signature (алгоритм SHA256withECDSA) для створення підпису (див. рис. Г.10 у Додатку Г). Якщо голосування дозволяє кілька виборів, для кожного вибраного choiceId генерується окреме повідомлення та окремий підпис; відповідно у підколекції votes зберігається стільки записів, скільки обрано варіантів. Отриманий підпис і відкритий ключ у форматі Base64 додаються до об’єкта голосу й разом із міткою часу записуються до Firestore.

Отриманий підпис, разом з публічним ключем, додаються до об’єкта голосу, що буде відправлений у Firestore (див. рис. Г.11 у Додатку Г).

Після обчислення цифрового підпису клієнт формує документ підколекції voters, що розташована всередині конкретного опитування surveys/{surveyId}. Ідентифікатор цього документа складається з UID користувача та вибраного варіанта – наприклад, u123_c2. Таким чином сам UID не дублюється окремим полем, а інкапсульований безпосередньо в назві документа, чим спрощується пошук і водночас зберігається чітка відповідність «користувач-варіант».

Усередині документа зберігається чотири атрибути. Поле choiceId фіксує ідентифікатор обраного варіанта, а timestamp містить мітку часу подання голосу у мілісекундах від початку епохи. Два інші поля – це власне результат криптографічної операції: signature – рядок Base64, що містить підпис повідомлення surveyId|choiceId|timestamp, і publicKey – так само закодований у Base64 відкритий ключ виборця у форматі DER.

Якщо опитування дозволяє вибрати кілька варіантів, клієнт послідовно створює кілька таких документів, змінюючи лише суфікс choiceId у їхніх індентифікаторах. Така структура дає змогу перевіряти достовірність кожного голосу ізольовано: будь-який спостерігач, маючи дані документа, може відтворити початкове повідомлення та пересвідчитися, що підпис справді сформовано приватним ключем, відповідним до опублікованого відкритого ключа.

Структура даних у Firestore для коллекції surveys показана на рисунку Г.12, що наведено у Додатку Г:

Під час запису голосу додаток додає до документа і підпис (signature), і свій відкритий ключ (publicKey), тож кожен голос має криптографічне підтвердження. Відкритий ключ дає змогу будь-кому, хто має доступ, перевірити, що голос справді створений власником відповідного приватного ключа.

Перевірка підпису. Демонстраційна верифікація можлива прямо у клієнтському застосунку: для цього він бере publicKey, signature, а також поля surveyId, choiceId і timestamp, відтворює точний рядок-payload "<surveyId>|<choiceId>|<timestamp>", ініціалізує Signature.initVerify(publicKey) і викликає verify(signature). Успішний результат означає, що голос не підроблено.

Приватність при цьому не порушується: у Firestore зберігаються лише відкриті ключі й підписи, без прямого зв'язку із реальним ім'ям користувача. Адміністратор серверу не володіє приватними ключами, отже не може створити фальшивий голос. Це реалізує принцип децентралізації довіри – контроль над підписом належить виключно власнику пристрою, а верифікацію може виконати будь-яка зацікавлена сторона.

4.4 Робота з Firebase (Storage, Auth, Rules)

Основне сховище даних для системи – Cloud Firestore, який використано для зберігання інформації про голосування та самі голоси детальніше у таблиці 4.1. Взаємодія клієнтського додатка з Firestore відбувається через офіційний SDK Firebase для Android.

Таблиця 4.1 - Структура бази даних

Колекція / документ	Головні поля (реально присутні у класі SurveyDataItem)
surveys/{surveyId}	id, title, description, createdBy, creatorName, createdAt, expirationDate, isActive, isMultipleChoice, listOfCandidates (масив {name, userId, vote})
surveys/{surveyId}/voters/{uid}_{choiceId}(один документ = один вибір)	choiceId, timestamp, signature, publicKey
users/{userId}	displayName, publicKey64, createdSurveys (масив ID), participatedSurveys (масив ID)

Після успішної авторизації клієнт потрапляє на головний екран зі списком активних голосувань. Отримання цих голосувань реалізовагно шляхом запиту до Firestore (див. рис. Г.13 у Додатку Г). Отримуємо спочатку весь список і потім фільтруємо, по надібності. Наприклад по даті: завершені чи поточні. Зроблено це з метою оптимізації запитів та реактивного відображення на екранах.

Firestore підтримує realtime режим, для цього треба підписатися на оновлення колекції за допомогою слухача addSnapshotListener. Таким чином, якщо адміністратор або хтось створить нове голосування, користувачі побачать його в списку без перезапуску додатка (дані оновляться в реальному часі).

Саме таким підходом було реалізовано екран голосування – задля забезпечення найактуальніших даних, тобто якшо користувачі активно голосують, то поточний користувач буде бачити зміни голосування у режиму реального часу (див. рис. Г.14 у Додатку Г).

У розробленій системі зберігання даних на Firebase Firestore увесь контроль доступу покладається на правила безпеки, сформульовані у файлі firestore.rules. Їх логіка ґрунтується на чотирьох ідеях: відкритість перегляду, сувора автентифікація для будь-яких змін, однозначність авторства даних і відмова від руйнівних дій.

У колекції users кожен документ зі службовою інформацією про користувача доступний для читання будь-якому відвідувачу бази. Це забезпечує прозорість профілів, потрібну, наприклад, для відображення публічних імен чи аватарів у застосунку. Водночас створити або змінити власний профіль може лише той, чий uid збігається з ідентифікатором документа. Так гарантовано, що жоден користувач не зможе підмінити дані іншого. Операцію delete у цьому розділі заблоковано повністю: профілі зберігатимуться як архів навіть після деактивації облікового запису, що полегшує аудит і відкат змін.

Наступний логічний рівень – колекція surveys, у якій розміщуються всі опитування. Будь-який авторизований користувач має право ініціювати нове опитування, адже правило перевіряє лише наявність об’єкта request.auth. Після створення документ стає незмінним: операції update і delete прямо заборонені. Така незворотність важлива з погляду академічної доброчесності, бо виключає

можливість «підправити» запитання або варіанти відповідей після того, як учасники вже почали голосувати.

Кожен документ опитування містить підколекцію voters. Тут кожен голос записується окремим документом. Читати зміст цієї підколекції може будь-хто – отже, проміжні й фінальні результати завжди відкриті для огляду. Додати свій голос дозволено лише автентифікованому користувачеві; натомість повторно змінити чи видалити вже надісланий вибір нікому не вдасться. Правило не примушує іменувати документ так само, як uid користувача, але саме на цій угоді заснований клієнтський код. Документ з ідентифікатором currentUser.uid створюється методом create; якщо скринька вже заповнена, Firestore повертає помилку «PERMISSION_DENIED», таким чином запобігаючи повторному голосуванню без необхідності складних умов усередині правил.

З боку клієнта процес виглядає так: під час реєстрації опитування застосунок формує Map-об’єкт із полями title, options, multiple, createdBy тощо і відправляє його в колекцію surveys методом add, дозволяючи Firestore самостійно згенерувати surveyId. Для голосування користувач переходить у підколекцію voters конкретного опитування і створює документ із ключем, тотожним власному uid. Завдяки цьому ключу і заблокованим операціям update та delete кожен учасник залишає рівно один слід.

Підрахунок результатів відбувається на клієнтській стороні: після завершення голосування застосунок виконує простий get запит до підколекції voters, отримує набір документів і локально сумує вибрані варіанти. Це рішення зменшує складність бекенду й уникає додаткових транзакцій, хоча альтернативно можна було б підтримувати серверні лічильники через Cloud Functions, але це рішення потребує фінансових витрат.

Нарешті, варто згадати про офлайн-підтримку, притаманну Firestore. Якщо під час голосування зникне мережа, SDK збереже операцію у локальному кеші та синхронізує її, щойно з’явиться з’єднання. Таким чином жоден голос не загубиться через тимчасові мережеві збої. Помилки, пов’язані з порушенням правил,

обробляються на рівні клієнта: код PERMISSION_DENIED трансформується у зрозуміле повідомлення, що користувач уже взяв участь у цьому опитуванні.

Отже, описані правила безпеки реалізують рівновагу між відкритістю даних та суворим контролем їхніх змін (табл. 4.2). Завдяки прозорій моделі «читати може кожен, змінює лише власник» зберігається як академічна чесність процесу голосування, так і цілісність профілів, а одночасно мінімізується ризик випадкового чи зловмисного видалення інформації.

Таблиця 4.2 – Налаштування прав взаємодії із коллекціями

Колекція / дія	read	create	update	delete
users	усі	власник	власник	ніхто
surveys	усі	авториз.	—	—
voters	усі	авториз.	—	—

4.5 Участь у голосуванні

Після успішної автентифікації користувач одразу потрапляє на головний екран, де HomeViewModel через метод getSurveys() звертається до SurveyRepository. Репозиторій читає колекцію surveys, десеріалізує документи у SurveyDataItem, а ViewModel трансформує їх у спрощені об’єкти SurveyItemUi, що й потрапляють до інтерфейсу, вище зазначені класи навежені у додатку А. У потоці surveyListFlow UI спостерігає за цими даними й показує кожне опитування окремою карткою з назвою, датою закриття та піктограмою статусу.

Коли користувач обирає конкретне голосування, у VoteSurveyViewModel викликається setSurveyId(), вихідний код наведено у додатку А. Цей метод лише підставляє ідентифікатор у MutableStateFlow, після чого спрацьовує підписка observeSurveyById(): через addSnapshotListener ViewModel постійно отримує актуальний стан документа та миттєво оновлює інтерфейс. Якщо опитування підтримує єдиний вибір, інтерфейс відображає варіанти у вигляді радіокнопок; для мультивибору – чекбокси. Кнопка «Проголосувати» стає активною тільки тоді, коли у selectedOptions з’являється принаймні один елемент.

Натискання на кнопку запускає submitVote(). ViewModel делегує роботу PlaceVoteUseCase, передаючи surveyId і множину choiceIds. Усередині use-case формує payload "surveyId|choiceId|timestamp", підписує його приватним ключем, що зберігається в Android Keystore, кодує підпис та відкритий ключ у Base64 і, нарешті, викликає submitSignedVote() репозиторію. Той створює документ surveys/{surveyId}/voters/{uid}_{choiceId} і записує вибір, підпис і ключ. Під час запису інтерфейс блокує повторні натискання, а після успіху – виводить повідомлення «Ваш голос враховано» й робить форму недоступною для повторного голосування.

Якщо користувач спробує відкрити опитування вдруге, клієнт перед показом форми перевіряє, чи існує документ із його UID у підколекції voters. За наявності документа ViewModel одразу показує текст «Ви вже голосували» й не пропонує відправляти голос повторно. При цьому правила Firestore додатково гарантують, що голос із таким самим voteId не буде прийнято вдруге: повторна спроба завершиться помилкою PERMISSION_DENIED, яку додаток перехоплює й перетворює на дружнє повідомлення про помилку.

Поки опитування активне, жоден клієнт не може прочитати підколекцію voters: правила обмежують читання за умовою resource.data.expirationDate > request.time. Отже, ніхто не бачить проміжних результатів, а після дати закриття дані стають доступними для всіх авторизованих користувачів, і ViewModel може підрахувати підсумки локально, не звертаючись до хмарних функцій. У такий спосіб реалізовано баланс між прозорістю та приватністю: вибір кожного підписаний і перевіряємий, але не розкривається допоки голосування триває.

4.6 Створення голосування

Право ініціювати опитування має кожен автентифікований користувач. На головному екрані відображається плаваюча кнопка «+», яка відкриває екран CreateSurveyScreen. Його станом керує CreateSurveyViewModel, зазначено у додатку А. У ViewModel зберігаються surveyTitle, surveyDescription, прапорець isMultipleChoice і динамічний список candidates. Користувач вводить питання,

опис, додає щонайменше два варіанти відповіді та вибирає дату завершення (за замовчуванням – через 30 днів від поточної).

Натискання «Створити» стартує корутину createSurvey(). ViewModel перевіряє, що назва й опис не порожні, масив кандидатів містить мінімум два непорожні елементи, а дата завершення лежить у майбутньому. Після валідації формується екземпляр CreateSurveyDataItem, наведено у додатку А, і йому призначається UUID, у поля creatorId та creatorName підставляються дані FirebaseAuth, у listOfCandidates – пари {id, name}. Об’єкт передається до SurveyRepository.createSurvey(), який виконує set() за шляхом surveys/{id}.

Коли Firestore успішно зберігає документ, у ProfileRepository викликається addUserCreatedSurvey(id), що додає ідентифікатор новоствореного опитування до масиву createdSurveys у профілі користувача, цей клас та його методи було зазначено у додатку А. У відповідь UI показує toast «Голосування створено» і автоматично відкриває екран цього опитування, пропонуючи автору одразу проголосувати або повернутися до списку.

Усі перевірки доступу виконуються правилами Firestore: на рівні surveys/{surveyId} дозволений create, якщо request.auth != null, а update та delete заблоковані. Таке рішення спрощує модель безпеки: у демонстраційній версії адміністратор не потрібен, а будь-який користувач може ініціювати опитування.

Якщо виникне потреба обмежити коло авторів, достатньо додати перевірку request.auth.uid in ["UID1", "UID2"] або окрему роль.

Після створення документ одразу потрапляє в realtime-стрім колекції, тож усі клієнти майже миттєво бачать нове опитування в списку без явного оновлення. Свідомий мінімалізм інтерфейсу й перевірка даних на клієнті зменшують ризик помилок, а використання Firestore – потребу в окремому сервері. У результаті створення й публікація опитування займають лічені секунди й не перевантажують користувача зайвими кроками.

4.7 Висновки до розділу

У цьому розділі було детально описано практичну реалізацію мобільного клієнта системи "Децентралізована система захищеного голосування". Розглянуто

архітектурні рішення та ключові технічні аспекти, які забезпечують як функціональність, так і безпеку додатку. Наведемо основні підсумки та досягнення:

Архітектура додатка побудована на сучасних підходах (шари UI та даних, патерн MVVM), що забезпечило чистоту коду, його підтримуваність і масштабованість. Чітке розмежування відповідальностей дозволило легко інтегрувати сервіси Firebase та криптографічні операції.

Реалізовано захищену авторизацію користувача через Google (Firebase Authentication), що спрощує вхід і усуває потребу в окремих паролях. Використання Credential API гарантує зручний і безпечний вибір облікового запису.

Впроваджено генерацію і використання криптографічних ключів на пристрої для цифрового підпису голосів. Кожен голос завірений підписом і не може бути непомітно змінений. Приватний ключ користувача захищено апаратними засобами (Android Keystore), що унеможливлює його компрометацію без фізичного доступу до пристрою.

Сховище даних Firebase Firestore служить надійною базою для збереження голосувань і голосів. Налаштовані правила безпеки, які забезпечують авторизований доступ і дотримання правил "1 користувач = 1 голос". Структура даних дозволяє ефективно отримувати як списки активних голосувань, так і результати, при цьому зберігаючи приватність виборців.

Клієнт надає повний цикл функціональності: участь у голосуванні (з усіма перевірками та зручностями для користувача) та створення нового голосування (з гнучким налаштуванням параметрів). Це робить систему децентралізованою – будь-який користувач може ініціювати опитування, а не лише центральний адміністратор.

Виконано налаштування зосереджені на безпеці клієнта: застосовано ProGuard/R8 для обфускації коду, приховано конфіденційні дані (ключі API) з репозиторію, додаток підписано власним криптографічним ключем. Ці заходи

зменшують ризики, пов’язані з компрометацією клієнтської частини (наприклад, зловмисної модифікації).

Інтегровано аналітику та моніторинг: Firebase Analytics надає статистику використання (що важливо для подальшого розвитку проекту і розуміння поведінки аудиторії), а Crashlytics дає змогу оперативно отримувати інформацію про помилки і збої, що виникають у користувачів, підвищуючи стабільність додатка.

Інтерфейс користувача ретельно продуманий і реалізований у відповідності до сучасних стандартів. Він забезпечує легкість навігації, зрозумілість на кожному кроці (будь то голосування чи створення опитування) та приємний візуальний вигляд. Це підтверджує, що складна технологічно система може бути представлена користувачу у дружній формі.

Реалізований мобільний клієнт успішно демонструє можливість поєднання децентралізованих методів забезпечення довіри (криптографія на клієнті) з хмарними технологіями (Firebase) для побудови безпечної системи електронного голосування. Всі поставлені задачі практичної реалізації були виконані: від авторизації і до відображення результатів. Отриманий додаток пройшов тестування на коректність функціонування та безпеку (в межах доступних ресурсів) і може слугувати прототипом реальної системи електронного голосування з підвищеною приватністю та цілісністю даних. Система готова до подальших випробувань і, за потреби, масштабування або доопрацювання специфічних вимог.

ВИСНОВКИ

У процесі виконання дипломної роботи було детально досліджено вимоги до мобільного клієнта децентралізованої системи захищеного голосування та розроблено прототип на платформі Android. У другому розділі проведено глибокий аналіз загальних вимог до систем захищеного голосування — цілісності, конфіденційності, доступності, анонімності та аудиту — а також типових вразливостей клієнтських застосунків: неправильного використання облікових даних, ненадійної автентифікації та авторизації, незахищеної комунікації й зберігання даних. На основі цього аналізу запропоновано комплексний багаторівневий підхід до безпеки: генерація й захист ключів у Trusted Execution Environment, використання шифрованих каналів із certificate pinning, обфускація коду та захищена політика зберігання.

Особлива увага була приділена вибору архітектурних і технологічних рішень у третьому розділі. Ці вимоги лягли в основу архітектури мобільного клієнта, реалізованого за патерном MVVM із чітким розділенням UI, ViewModel і Repository. Для побудови інтерфейсу застосовано Jetpack Compose для декларативного опису UI та підготовки до мультиплатформності, а залежності організовано через Hilt. У процесі оцінки технологій було порівняно бібліотеки SpongyCastle і Conscrypt, що дозволило обрати оптимальні рішення з урахуванням продуктивності та розміру APK. Вибір Android Keystore у середовищі Trusted Execution Environment та використання стандартів шифрування, таких як SHA-256, забезпечують апаратний захист приватних даних користувачів.

Подальше тестування реалізованої системи підтвердило правильність обраних криптографічних підходів, зокрема забезпечення прозорості процесу голосування через цифрові підписи кожного голосу та верифікацію за допомогою стандартного API Signature.initVerify. Це дозволяє будь-якій зацікавленій стороні в будь-який час перевірити автентичність поданих голосів, не порушуючи при цьому анонімність виборців.

Але, окрім чисто технічних аспектів, розробка також передбачала впровадження додаткових заходів для підвищення стійкості системи до зовнішніх атак. Так, інтеграція клієнта в блокчейн і застосування механізмів certificate pinning дозволяють значно підвищити безпеку та забезпечити повну децентралізацію зберігання результатів.

У результаті було доведено доцільність використання мобільних платформ для організації безпечного голосування, що відкриває нові можливості для інтеграції цієї технології в реальні виборчі процеси. Запропоноване рішення може бути адаптоване під інші операційні системи, що забезпечує його універсальність і масштабованість.

Значущість роботи полягає в комплексному підході до забезпечення безпеки на всіх рівнях: від криптографічної стійкості до зручності інтерфейсу користувача. Подальші дослідження повинні бути спрямовані на вирішення проблеми масштабованості, підвищення продуктивності системи, а також забезпечення більшої анонімності голосування через застосування таких технологій, як шардинг і Zero-Knowledge Proofs.

Отже, розроблена система створює надійний фундамент для впровадження електронного голосування в реальних умовах, сприяючи підвищенню довіри виборців до виборчого процесу та відкриваючи нові можливості для вдосконалення технологій у цій галузі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Allen, D. W. E. (2019). Governing the entrepreneurial discovery of blockchain applications. [Електронний ресурс]. – Режим доступу: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2919170
2. National Institute of Standards and Technology. (2020). Digital Identity Guidelines: Authentication and Lifecycle Management (NIST SP 800-63B). [Електронний ресурс]. – Режим доступу: https://doi.org/10.6028/NIST.SP.800-63b
3. Goldwasser, S., Micali, S., & Rackoff, C. (1989). The knowledge complexity of interactive proof systems. SIAM Journal on Computing, 18(1), 186–208. [Електронний ресурс]. – Режим доступу: https://doi.org/10.1137/0218012
4. International Organization for Standardization. (2013). ISO/IEC 27001:2013 Information technology – Security techniques – Information security management systems – Requirements. [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/54534.html
5. Specter, M. A., Koppel, J., & Weitzner, D. J. (2020). The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in U.S. Federal Elections. In 29th USENIX Security Symposium (USENIX Security 20). [Електронний ресурс]. – Режим доступу: https://www.usenix.org/conference/usenixsecurity20/presentation/specter
6. Agora Blockchain Team. (2018, March 22). Agora Official Statement Regarding Sierra Leone Election. Medium. [Електронний ресурс]. – Режим доступу: https://medium.com/agorablockchain/agora-official-statement-regarding-sierra-leone-election-7730d2d9de4e
7. Moralis Academy. (2023). Top Blockchain Voting System Projects [online]. [Електронний ресурс]. – Режим доступу: https://academy.moralis.io/blog/top-blockchain-voting-system-projects
8. Державна служба спецзв'язку та захисту інформації України. (2015). Нормативні документи з технічного захисту інформації (НД ТЗІ).

[Електронний ресурс]. – Режим доступу: https://cip.gov.ua/ua/news/perelik-dokumentiv-sistemi-tekhnichnogo-zakhistu-informaciyi-nd-tzi

9. Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., & Capkun, S. (2016). On the security and performance of proof of work blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 3–16). [Електронний ресурс]. – Режим доступу: https://doi.org/10.1145/2976749.2978341
10. Hajian Berenjestanaki, M., Barzegar, H. R., El Ioini, N., & Pahl, C. (2024). Scalability Assessment of EVM-Compatible Blockchains for E-Voting. In 6th International Congress on Blockchain and Applications (BLOCKCHAIN 2024), Lecture Notes in Networks and Systems, 1256. Springer. [Електронний ресурс]. – Режим доступу: https://doi.org/10.1007/978-3-031-81927-8
11. Vranken, H. (2017). Sustainability of bitcoin and blockchains. Current Opinion in Environmental Sustainability, 28, 1–9. [Електронний ресурс]. – Режим доступу: https://doi.org/10.1016/j.cosust.2017.04.011
12. Yli-Huumo, J., Ko, D., Choi, S., Park, S., & Smolander, K. (2016). Where is current research on blockchain technology? A systematic review. PLOS ONE, 11(10), e0163477. [Електронний ресурс]. – Режим доступу: https://doi.org/10.1371/journal.pone.0163477
13. Park, S., et al. (2021). Going from bad to worse: from Internet voting to blockchain voting. Journal of Cybersecurity, 7(1), 1–15. [Електронний ресурс]. – Режим доступу: https://doi.org/10.1093/cybsec/tyaa025
14. Khan, K. M., et al. (2020). Investigating performance constraints for blockchain based secure e-voting system. Future Generation Computer Systems, 105, 13–26. [Електронний ресурс]. – Режим доступу: https://doi.org/10.1016/j.future.2019.11.005
15. Li, K., et al. (2020). PoV: An Efficient Voting-Based Consensus Algorithm for Consortium Blockchains. Frontiers in Blockchain, 3:11. [Електронний ресурс]. – Режим доступу: https://doi.org/10.3389/fbloc.2020.00011

16. Міністерство цифрової трансформації України. (2023). Інтегрована система електронної ідентифікації. Офіційний портал електронних послуг. [Електронний ресурс]. – Режим доступу: https://se.diia.gov.ua/integrated-system-of-electronic-identification
17. Національний банк України. (2024). Специфікація для банків-ідентифікаторів системи BankID НБУ, версія 2.0 (10 травня 2024). [Електронний ресурс]. – Режим доступу: https://bank.gov.ua/admin_uploads/article/Specification_BankID_NBU_Bank_v_2.0.2024.05.10.pdf
18. National Institute of Standards and Technology. (2020). Security and Privacy Controls for Information Systems and Organizations (NIST SP 800-53 Rev. 5). [Електронний ресурс]. – Режим доступу: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf
19. Electronic Voting: Constitutional and Legal Requirements and Their Technical Implications. (2002). ResearchGate. [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/226590438_Electronic_Voting_Constitutional_and_Legal_Requirements_and_Their_Technical_Implications
20. OWASP. (2023). M1 – Improper Credential Usage. OWASP Mobile Top 10. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-mobile-top-10/2023-risks/m1-improper-credential-usage.html
21. OWASP. (2023). M3 – Insecure Authentication & Authorization. OWASP Mobile Top 10. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-mobile-top-10/2023-risks/m3-insecure-authentication-authorization.html
22. OWASP. (2023). M5 – Insecure Communication. OWASP Mobile Top 10. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-mobile-top-10/2023-risks/m5-insecure-communication.html
23. OWASP. (2023). M9 – Insecure Data Storage. OWASP Mobile Top 10. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-mobile-top-10/2023-risks/m9-insecure-data-storage.html

24. National Institute of Standards and Technology. (2013). Guidelines for Managing the Security of Mobile Devices in the Enterprise (NIST SP 800-124 Rev. 1). [Електронний ресурс]. – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-124/rev-1/final
25. Адміністрація Держспецзв'язку України. Методичні матеріали з побудови систем ТЗІ. [Електронний ресурс]. – Режим доступу: https://duikt.edu.ua/wp-content/uploads/2022/03/Zahalni-vymohy-do-IT-system-zakhyst.pdf
26. Android Developers. Android Keystore system. [Електронний ресурс]. – Режим доступу: https://developer.android.com/training/articles/keystore
27. OWASP Mobile Security Testing Guide. Certificate Pinning. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-mobile-security-testing-guide/latest/Testing-Defense-Mechanisms/01-Testing-Certificate-Pinning
28. Android Developers. Security library. [Електронний ресурс]. – Режим доступу: https://developer.android.com/topic/security
29. Firebase. Secure data transmission. [Електронний ресурс]. – Режим доступу: https://firebase.google.com/support/privacy#data_security
30. Android Developers. Shrink, obfuscate, and optimize your app. [Електронний ресурс]. – Режим доступу: https://developer.android.com/studio/build/shrink-code
31. Google Play Console Help. App signing overview. [Електронний ресурс]. – Режим доступу: https://support.google.com/googleplay/android-developer/answer/9842756
32. Firebase. Get started with Security Rules for Cloud Firestore. [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/firestore/security/get-started
33. Android Developers. Architecture recommendations. [Електронний ресурс]. – Режим доступу: https://developer.android.com/topic/architecture/recommendations
34. Google. Hilt dependency injection. [Електронний ресурс]. – Режим доступу: https://developer.android.com/training/dependency-injection/hilt-android

35. Google. Architectural guidance: MVVM. [Електронний ресурс]. – Режим доступу: https://developer.android.com/jetpack/guide
36. National Institute of Standards and Technology. (2022). Secure Software Development Framework (SSDF) Version 1.1 (NIST SP 800-218). [Електронний ресурс]. – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-218/final
37. ISO/IEC. (2022). ISO/IEC 27002:2022 Information security, cybersecurity and privacy protection – Information security controls. [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/75652.html
38. Google. Jetpack Compose overview. [Електронний ресурс]. – Режим доступу: https://developer.android.com/jetpack/compose
39. Google. Navigation in Compose. [Електронний ресурс]. – Режим доступу: https://developer.android.com/jetpack/compose/navigation
40. Міністерство цифрової трансформації України. e-Ukraine font. [Електронний ресурс]. – Режим доступу: https://github.com/dash-incubator/e-Ukraine
41. Android Open Source Project. Keystore security features. [Електронний ресурс]. – Режим доступу: https://source.android.com/docs/security/features/keystore
42. Android Open Source Project. Android Enterprise Security Whitepaper 2018. [Електронний ресурс]. – Режим доступу: https://source.android.com/static/docs/security/overview/reports/Google_Android_Enterprise_Security_Whitepaper_2018.pdf
43. Nakov, N. (n.d.). Elliptic Curve Cryptography (ECC). CryptoBook. [Електронний ресурс]. – Режим доступу: https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc
44. Security of modified digital public-key signature EdDSA. (2019). ResearchGate. [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/337702674_Securuty_of_modified_digital_public-key_signature_EdDSA
45. Medium. Elliptic Curve Key Pair Generation for Virtually Every Possible Standard Curve. Bouncy Castle (cryptography). [Електронний ресурс]. – Режим доступу:

https://billatnapier.medium.com/elliptic-curve-key-pair-generation-for-virtually-every-possible-standard-curve-c1ddaa12a974

46. GitHub. Spongy Castle repository. [Електронний ресурс]. – Режим доступу: https://github.com/rtyley/spongycastle
47. GitHub. Conscrypt repository. [Електронний ресурс]. – Режим доступу: https://github.com/google/conscrypt
48. Google Online Security Blog. (2018). Introducing the Tink cryptographic software library. [Електронний ресурс]. – Режим доступу: https://security.googleblog.com/2018/08/introducing-tink-cryptographic-software.html
49. OWASP. Mobile Application Security Verification Standard (MASVS). [Електронний ресурс]. – Режим доступу: https://mas.owasp.org/MASVS/
50. National Institute of Standards and Technology. (2019). Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography (NIST SP 800-56A Rev. 3). [Електронний ресурс]. – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final
51. ISO/IEC. (2022). ISO/IEC 27002:2022 Information security, cybersecurity and privacy protection – Information security controls. [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/75652.html
52. ДСТУ 4145-2002. (2002). Криптографічний захист інформації. Електронний цифровий підпис. [Електронний ресурс]. – Режим доступу: https://zakon.isu.net.ua/sites/default/files/norm_docs/dstu_4145-2002.pdf
53. ДСТУ 7564:2014. (2014). Криптографічний захист інформації. Хеш-функції. [Електронний ресурс]. – Режим доступу: https://zakon.isu.net.ua/sites/default/files/norm_docs/dstu_7564-2014.pdf
54. ДСТУ 7624:2014. (2014). Криптографічний захист інформації. Блокові шифри. [Електронний ресурс]. – Режим доступу: https://zakon.isu.net.ua/sites/default/files/norm_docs/dstu_7624-2014.pdf

55. Massachusetts Institute of Technology. (2020). Voting Voatz App Hack Issues. MIT News. [Електронний ресурс]. – Режим доступу: https://news.mit.edu/2020/voting-voatz-app-hack-issues-0213
56. International Association for Cryptologic Research. (2002). Efficient hashing and signatures from modular arithmetic. IACR ePrint 2002/165. [Електронний ресурс]. – Режим доступу: https://eprint.iacr.org/2002/165.pdf
57. DataGrail. (2021). Privacy by Design, Privacy by Default. [Електронний ресурс]. – Режим доступу: https://www.datagrail.io/blog/data-privacy/privacy-by-design-privacy-by-default/
58. Android Developers. Typical architecture overview. [Електронний ресурс]. – Режим доступу: https://developer.android.com/topic/architecture#layered
59. Android Developers. Sign in with Google using Credential Manager SIWG. [Електронний ресурс]. – Режим доступу: https://developer.android.com/identity/sign-in/credential-manager-siwg
60. Google Maps Platform. Secrets Gradle Plugin. [Електронний ресурс]. – Режим доступу: https://developers.google.com/maps/documentation/places/android-sdk/secrets-gradle-plugin
61. Firebase. Google Sign-In for Android. [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/auth/android/google-signin
62. National Institute of Standards and Technology. (2018). Blockchain Technology Overview (NISTIR 8202). [Електронний ресурс]. – Режим доступу: https://doi.org/10.6028/NIST.IR.8202

ДОДАТОК А

ВИХІДНИЙ КОД РОЗРОБЛЕНОГО ЗАСТОСУНКУ

Повна версія застонуку знаходиться за посиланням:

https://github.com/slowwylx/DVote

А.1 ВИХІДНИЙ КОД КЛАСУ HOMEVIEWMODEL

```
@HiltViewModel
class HomeViewModel @Inject constructor(
    private val surveyRepository: SurveyRepository,
) : ViewModel() {

    private val _surveyListFlow =
MutableStateFlow<List<SurveyItemUi>>(emptyList())
    val surveyListFlow = _surveyListFlow.asStateFlow()

    init{
        getSurveyList()
    }

    private fun getSurveyList() {
        viewModelScope.launch(Dispatchers.IO) {
            val surveyList = surveyRepository.getSurveys()
            val surveyItemList = surveyList.map { survey ->
                SurveyItemUi(
                    id = survey.id,
                    title = survey.title,
                    description = survey.description,
                    createdBy = survey.createdBy,
                    createdAt = survey.createdAt,
                    expirationDate = survey.expirationDate,
                    isActive = survey.isActive,
                    totalVotes = survey.totalVotes,
                    totalParticipants = survey.totalVotes
                )
            }
            _surveyListFlow.value = surveyItemList
        }
    }

}
```

А.2 МОДУЛЬ АВТОРИЗАЦІЇ GOOGLE

```
@Module
@InstallIn(SingletonComponent::class)
object GoogleAuthModule {

    @Provides
    @Singleton
```

```
    fun provideGoogleSignInRequest(): GetCredentialRequest {
        return GetCredentialRequest.Builder()
            .addCredentialOption(provideGoogleSignInOption())
            .build()
    }

    @Provides
    @Singleton
    fun provideCredentialManager(@ApplicationContext context:
Context): CredentialManager {
        return CredentialManager.create(context)
    }

    private fun provideGoogleSignInOption() :
GetSignInWithGoogleOption{
        return
GetSignInWithGoogleOption.Builder(BuildConfig.WEB_CLIENT_ID).build()
    }
}
```

А.3 ОТРИМАННЯ AUTH CREDENTIALS

```
if (credential.type ==
GoogleIdTokenCredential.TYPE_GOOGLE_ID_TOKEN_CREDENTIAL) {
    try {
        val googleIdTokenCredential =
GoogleIdTokenCredential.createFrom(credential.data)
        val credentialResult =
GoogleAuthProvider.getCredential(googleIdTokenCredential.idToken,
null)
        AuthResult.Success(credentialResult)
    } catch (e: GoogleIdTokenParsingException) {
        e.printStackTrace()
        AuthResult.Error(e.localizedMessage)
    }
} else {
    AuthResult.Error("something_went_wrong")
}
```

А.4 ВИХІДНИЙ КОД КЛАСУ SURVEY REPOSITORY

```
class SurveyRepository @Inject constructor(
    private val firestore: FirebaseFirestore,
) {

    suspend fun getSurveys(): List<SurveyDataItem> {
        return firestore.collection("surveys")
            .get()
            .await()
            .documents
            .map { it.toObject(SurveyDataItem::class.java)!! }
    }
```

```
    suspend fun getMySurveys(userId: String):
List<CreateSurveyDataItem> {
        return firestore.collection("users")
            .whereEqualTo("createdBy", userId)
            .get()
            .await()
            .documents
            .map { it.toObject(CreateSurveyDataItem::class.java)!! }
    }

    fun observeSurveyById(surveyId: String): Flow<SurveyDataItem> =
callbackFlow {
        val docRef =
firestore.collection("surveys").document(surveyId)
        val listener = docRef.addSnapshotListener { snapshot, error
->
            if (error != null) {
                close(error)
                return@addSnapshotListener
            }
            snapshot?.toObject(SurveyDataItem::class.java)
                ?.let { trySend(it).isSuccess }
        }
        awaitClose { listener.remove() }
    }

    suspend fun createSurvey(survey: CreateSurveyDataItem) {

firestore.collection("surveys").document(survey.id).set(survey).awai
t()
    }

    suspend fun submitSignedVote(
        surveyId: String,
        userId: String,
        choiceId: String,
        timestamp: Long,
        signature: String,
        publicKey: String
    ) {
        val voteData = mapOf(
            "choiceId"  to choiceId,
            "timestamp" to timestamp,
            "signature" to signature,
            "publicKey" to publicKey
        )
        val voteDocId = "${userId}_$choiceId"

        firestore
            .collection("surveys")
            .document(surveyId)
            .collection("voters")
            .document(voteDocId)
```

```
            .set(voteData)
            .await()
    }

}
```

А.5 СУТНІСТЬ SURVEYDATAITEM

```
data class SurveyDataItem(
    val id: String = "",
    val title: String = "",
    val description: String = "",
    val createdBy: String = "",
    val createdAt: String = "",
    val expirationDate: String = "",
    val isActive: Boolean = true,
    val multipleChoice: Boolean = false,
    val listOfCandidates: List<Participant> = emptyList(),
    val totalVotes: Long = 0L
){
    data class Participant(
        val userId: String = "",
        val vote: Long = 0L,
        val name: String = ""
    )
}
```

А.6 СУТНІСТЬ SURVEYITEMUI

```
data class SurveyItemUi(
    val id: String = "",
    val title: String = "",
    val description: String = "",
    val createdBy: String = "",
    val createdAt: String = "",
    val expirationDate: String = "",
    val isActive: Boolean = true,
    val totalVotes: Long = 0L,
    val totalParticipants: Long = 0L,
)
```

А.7 ВИХІДНИЙ КОД КЛАСУ VOTESURVEYVIEWMODEL

```
@HiltViewModel
class VoteSurveyViewModel @Inject constructor(
    private val surveyRepository: SurveyRepository,
    private val votingUseCase: PlaceVoteUseCase
) : ViewModel() {

    private val surveyIdFlow = MutableStateFlow("")

    private val _uiState =
MutableStateFlow<SurveyViewUiState>(SurveyViewUiState.Loading)
    val uiState: StateFlow<SurveyViewUiState> =
```

```
_uiState.asStateFlow()

    private val _selectedOptions =
MutableStateFlow<Set<String>>(emptySet())
    val selectedOptions = _selectedOptions.asStateFlow()

    init {
        surveyIdFlow
            .filter { it.isNotEmpty() }
            .flatMapLatest { id ->
                surveyRepository
                    .observeSurveyById(id)
                    .catch { e ->
                        _uiState.value =
SurveyViewUiState.Error(e.localizedMessage ?: "Unknown error")
                    }
            }
            .onEach { survey ->
                _uiState.value = SurveyViewUiState.Success(survey)
                _selectedOptions.value = emptySet()
            }
            .launchIn(viewModelScope)
    }


    fun onOptionToggled(optionId: String) {
        when (val state = _uiState.value) {
            is SurveyViewUiState.Success -> {
                val current = _selectedOptions.value.toMutableSet()
                if (state.survey.multipleChoice) {
                    if (!current.add(optionId))
current.remove(optionId)
                } else {
                    current.clear()
                    current.add(optionId)
                }
                _selectedOptions.value = current
            }
            else -> Unit
        }
    }

    fun setSurveyId(surveyId: String){
        surveyIdFlow.value = surveyId
    }

    fun submitVote() = viewModelScope.launch(Dispatchers.IO) {
        val state = uiState.value
        if (state is SurveyViewUiState.Success &&
selectedOptions.value.isNotEmpty()) {
            votingUseCase(state.survey.id, selectedOptions.value)
            _selectedOptions.value = emptySet()
        }
```

```
    }
}
```

А.8 ВИХІДНИЙ КОД КЛАСУ PLACEVOTEUSECASE

```
class PlaceVoteUseCase @Inject constructor(
    private val surveyRepository: SurveyRepository,
    private val firebaseAuth: FirebaseAuth,
    private val keyStoreService: KeyStoreService,
    private val profileRepository: ProfileRepository
) {

    suspend operator fun invoke(
        surveyId: String,
        choiceIds: Set<String>
    ): VoteResult = withContext(Dispatchers.IO) {
        val uid = firebaseAuth.currentUser?.uid
            ?: return@withContext VoteResult.Error("User not
authenticated")

        if (choiceIds.isEmpty()) {
            return@withContext VoteResult.Error("No choices
selected")
        }

        val now = System.currentTimeMillis()
        val pubBytes =
keyStoreService.generateKeyPairIfNeeded().public.encoded
        val pubBase64 = Base64.encodeToString(pubBytes,
Base64.NO_WRAP)

        var count = 0
        for (choiceId in choiceIds) {
            val payload =
"$surveyId|$choiceId|$now".toByteArray(Charsets.UTF_8)
            val sigBytes = keyStoreService.signData(payload)
            val signature = Base64.encodeToString(sigBytes,
Base64.NO_WRAP)

            surveyRepository.submitSignedVote(
                surveyId  = surveyId,
                userId    = uid,
                choiceId  = choiceId,
                timestamp = now,
                signature = signature,
                publicKey = pubBase64
            )
            count++
        }

        profileRepository.addUserVotedSurvey(surveyId)

        VoteResult.Success("Successfully voted for $count choices")
    }
```

```
    sealed class VoteResult {
        data class Success(val message: String) : VoteResult()
        data class Error(val errorMessage: String) : VoteResult()
    }
}
```

А.9 ВИХІДНИЙ КОД КЛАСУ CREATESURVEYVIEWMODEL

```
@HiltViewModel
class CreateSurveyViewModel @Inject constructor(
    private val repository: SurveyRepository,
    private val profileRepository: ProfileRepository,
    private val firebaseAuth: FirebaseAuth,
) : ViewModel() {

    private val _uiState =
MutableStateFlow<CreateSurveyUiState>(CreateSurveyUiState.None)
    val uiState = _uiState.asStateFlow()

    val surveyTitle = MutableStateFlow("")
    val surveyDescription = MutableStateFlow("")
    val candidates = MutableStateFlow<List<Map<String,
String>>>(emptyList())
    val isMultipleChoice = MutableStateFlow(false)


    fun addCandidate() {
        val newCandidate = mapOf(
            "id" to UUID.randomUUID().toString(),
            "name" to ""
        )
        candidates.value = candidates.value + newCandidate
    }

    fun removeCandidate(index: Int) {
        candidates.value = candidates.value
            .toMutableList()
            .also { it.removeAt(index) }
    }

    fun updateCandidate(index: Int, newName: String) {
        candidates.value = candidates.value.mapIndexed { i,
candidate ->
            if (i == index) {
                mapOf(
                    "id" to (candidate["id"]!!),
                    "name" to newName
                )
            } else candidate
        }
    }
```

```
    fun createSurvey() {
        viewModelScope.launch(Dispatchers.IO) {
            _uiState.value = CreateSurveyUiState.Loading

            val title = surveyTitle.value
            val description = surveyDescription.value
            val validCandidates = candidates.value.filter {
it["name"]?.isNotBlank() == true }
            val createdAt = LocalDate.now().toString()
            val expirationDate =
LocalDate.now().plusDays(30).toString()
            val isMultipleChoice = isMultipleChoice.value
            val userName = firebaseAuth.currentUser?.displayName ?:
"Anonym User"
            val userId = firebaseAuth.currentUser?.uid ?:
return@launch

            if (title.isNotBlank() && description.isNotBlank() &&
validCandidates.isNotEmpty()) {
                val item = CreateSurveyDataItem(
                    id = UUID.randomUUID().toString(),
                    title = title,
                    description = description,
                    creatorName = userName,
                    creatorId = userId,
                    createdAt = createdAt,
                    expirationDate = expirationDate,
                    listOfCandidates = validCandidates.map {
candidate ->
                        CreateSurveyDataItem.Participant(
                            userId = candidate["id"] ?:
UUID.randomUUID().toString(),
                            name = candidate["name"] ?: ""
                        )
                    },
                    isMultipleChoice = isMultipleChoice,
                )

                val surveyId = repository.createSurvey(item)
                profileRepository.addUserCreatedSurvey(item.id)
                _uiState.value = CreateSurveyUiState.Success
            } else {
                _uiState.value = CreateSurveyUiState.Error("Please
fill in all fields and add candidates.")
            }
        }
    }
}
```

А.10 СУТНІСТЬ CREATESURVEYDATAITEM

```
data class CreateSurveyDataItem(
    val id: String,
    val title: String,
    val description: String,
    val creatorName: String,
    val creatorId: String,
    val createdAt: String,
    val expirationDate: String,
    val isMultipleChoice: Boolean,
    val listOfCandidates: List<Participant>,
){
    data class Participant(
        val userId: String = "",
        val vote: Long = 0L,
        val name: String = ""
    )
}
```

A.11 ВИХІДНИЙ КОД КЛАСУ PROFILEREPOSITORY

```
class ProfileRepository @Inject constructor(
    private val firestore: FirebaseFirestore,
    private val auth: FirebaseAuth,
    private val keyStoreService: KeyStoreService
) {

    suspend fun initUserData() {
        val userName = auth.currentUser?.displayName
        val userId = auth.currentUser?.uid ?: return
        val pubKeyBytes = getKeyPair().public.encoded
        val pubKeyBase64 =
            android.util.Base64.encodeToString(pubKeyBytes,
android.util.Base64.NO_WRAP)
        firestore.collection("users").document(userId).set(
            hashMapOf(
                "displayName" to userName,
                "publicKey64" to pubKeyBase64,
            )
        ).await()
    }

    suspend fun addUserCreatedSurvey(surveyId: String) {
        val userId = auth.currentUser?.uid ?: return
        firestore.collection("users").document(userId).update(
            "createdSurveys", FieldValue.arrayUnion(surveyId)
        ).await()
    }

    suspend fun addUserVotedSurvey(surveyId: String) {
        val userId = auth.currentUser?.uid ?: return
        firestore.collection("users").document(userId).update(
            "participatedSurveys", FieldValue.arrayUnion(surveyId)
        ).await()
```

```
    }

    fun getKeyPair(): KeyPair {
        return keyStoreService.generateKeyPairIfNeeded()
    }

}
```

ДОДАТОК Б

ТАБЛИЦІ, ЯКІ ЗАЗНАЧЕНІ У ПЕРШОМУ РОЗДІЛІ

Таблиця Б.1 – Основні категорії загроз у децентралізованих системах голосування

Категорія загроз	Опис	Приклади атак
Автентифікація та право на участь	Забезпечення того, що лише виборці з правом голосу можуть проголосувати один раз.	Подвійне голосування, голосування неуповноважених осіб.
Приватність	Захист особистих даних виборця та конфіденційності його голосу.	Зв’язування бюлетенів з виборцями, перехоплення даних.
Стійкість до примусу	Запобігання примусу чи підкупу виборців.	Примус до розкриття голосу, продаж голосів.
Цілісність і надійність	Захист від фальсифікації голосів чи системи.	Введення шкідливого коду, маніпуляція даними.
Виявлення і моніторинг	Виявлення атак у реальному часі.	Непомічені атаки, збої системи.
Об’єктивність	Забезпечення нейтральності системи.	Упередженість, вплив на вибір виборців.
Перевірка і точність	Можливість перевірки правильності запису та підрахунку голосів.	Помилки підрахунку, неможливість верифікації.
Доступність	Захист від атак, що переривають голосування.	Атаки відмови в обслуговуванні (DoS).

Таблиця Б.2 – Класифікація рівнів атаки на мобільний клієнт голосування

Рівень атаки	Опис	Приклади
Рівень 0 (Базовий)	Атаки, які виконуються особами з мінімальними ресурсами та знаннями.	Подвійне голосування, прості мережеві атаки (наприклад, перехоплення пакетів).

Рівень 1 (Проміжний)	Атаки, що вимагають технічного досвіду та помірних ресурсів.	Використання відомих вразливостей, соціальна інженерія, зараження пристроїв виборців шкідливим ПЗ.
Рівень 2 (Просунутий)	Складні атаки від добре фінансованих груп з великою обчислювальною потужністю.	Компрометація кількох вузлів, масштабні DoS-атаки, спроби зламати криптографію.
Рівень 3 (Державний)	Атаки, спонсоровані державами з величезними ресурсами.	Координовані атаки на мережу, контроль значної частини вузлів, вплив на фізичну інфраструктуру.

ДОДАТОК В

ТАБЛИЦІ, ЯКІ ЗАЗНАЧЕНІ У ДРУГОМУ РОЗДІЛІ

Таблиця В.1 - Узагальнення методів захисту мобільного клієнта за рівнями

Рівень захисту	Типові загрози	Методи захисту
Передача даних	Перехоплення трафіку, підміна запитів	TLS 1.2+, certificate pinning, уникнення довіри до самопідписаних сертифікатів
Зберігання даних	Витік конфіденційих даних, доступ до кешу/локальної БД	Android Keystore, iOS Keychain, шифрування даних, обмеження доступу до файлів
Автентифікація та доступ	Несанкціонований вхід, brute-force, використання зламаних токенів	MFA, OTP, Firebase Auth, контроль сесій, відмова від жорстко закодованих паролів
Середовище виконання	Використання зламаних пристроїв, емуляторів, reverse engineering	Root/Jailbreak detection, obfuscation, flag_secure, перевірка підпису APK
Код та логіка додатка	Ін’єкції, модифікації, компрометація логіки голосування	ProGuard/R8, валідація даних, цілісність хешів, перевірка контрольних сум
Інтерфейс користувача (UX)	Фішинг, неправильне введення даних, маніпуляції візуально	Простий дизайн, валідація форм, блокування скриншотів, повідомлення про помилки
Моніторинг і аудит	Відсутність виявлення атак, пізнє реагування	Журналювання подій, автоматичний аналіз поведінки

ДОДАТОК Г

СКРІНШОТИ ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ

Рисунок Г.1 – Архітектурна діаграма трирівневої архітектури (UI, domain, data)

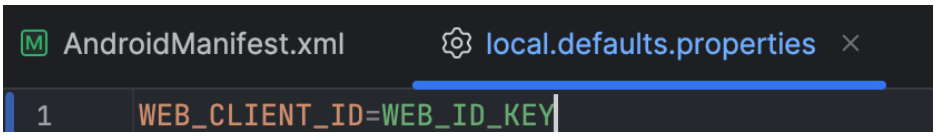

Рисунок Г.2 – Ініціалізація Web-ClienId у файлах gradle secret

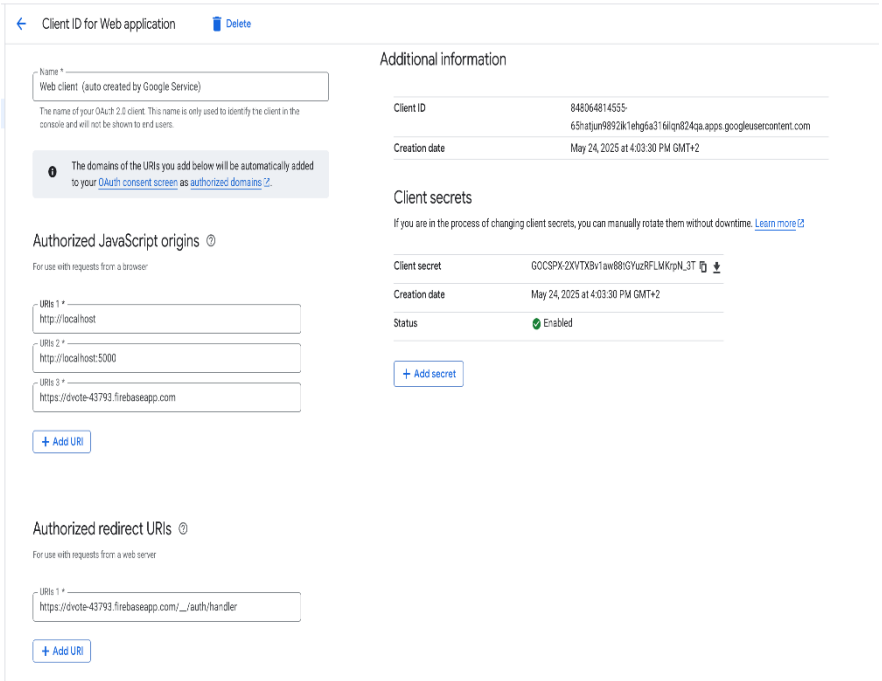

Рисунок Г.3 – Результат доданих SHA ключів до консолі Firebase

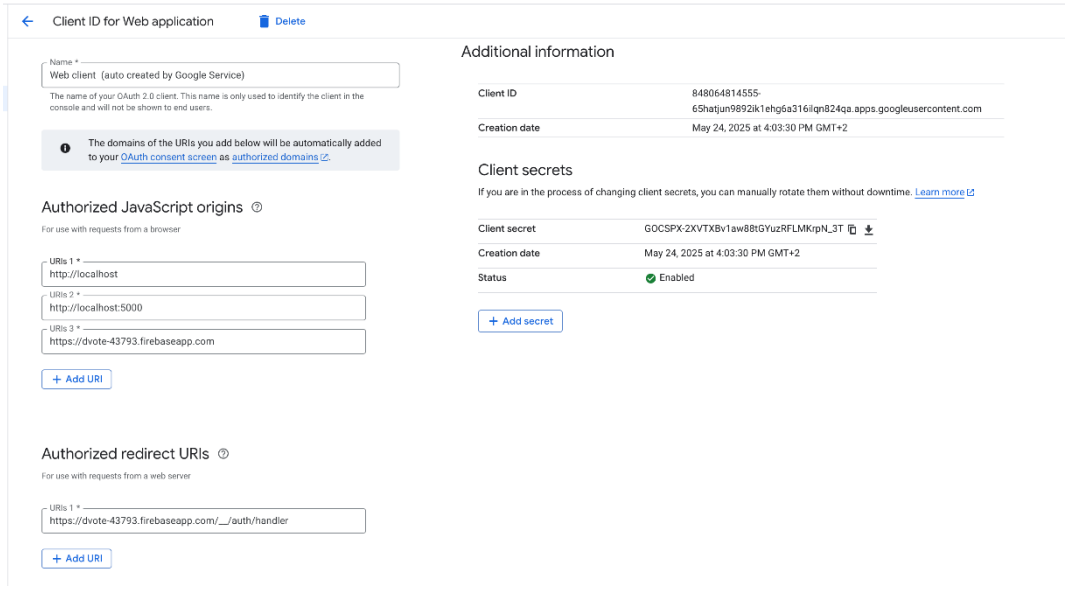

Рисунок Г.4 – Результат доданих SHA ключів до консолі google cloud

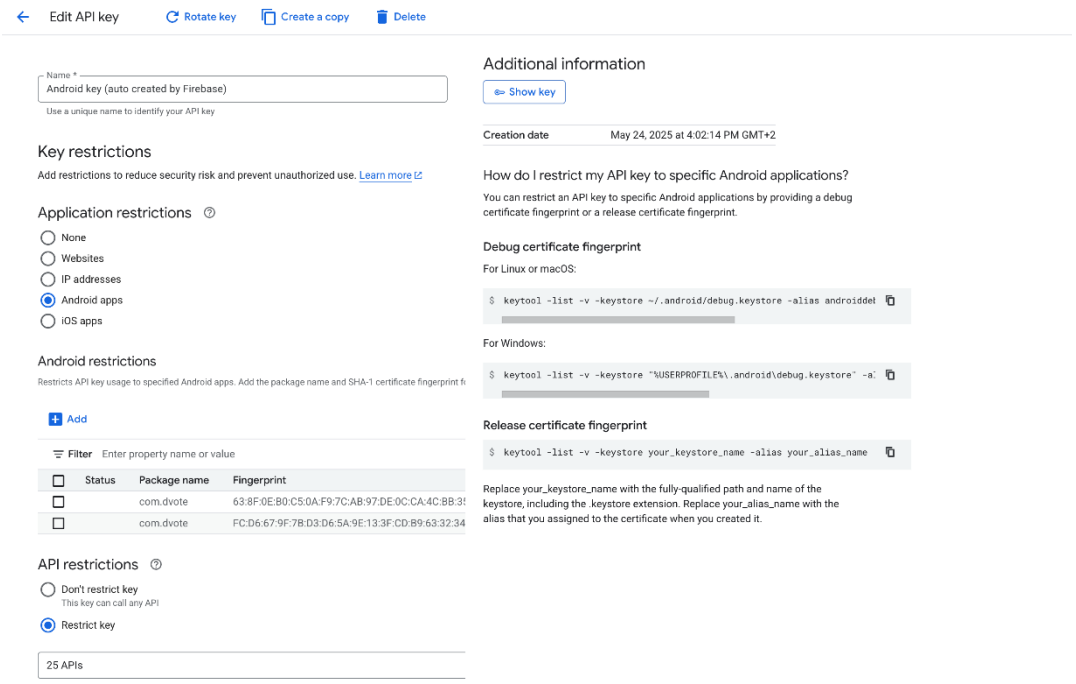

Рисунок Г.5 – Результат доданих SHA ключів до консолі Firebase

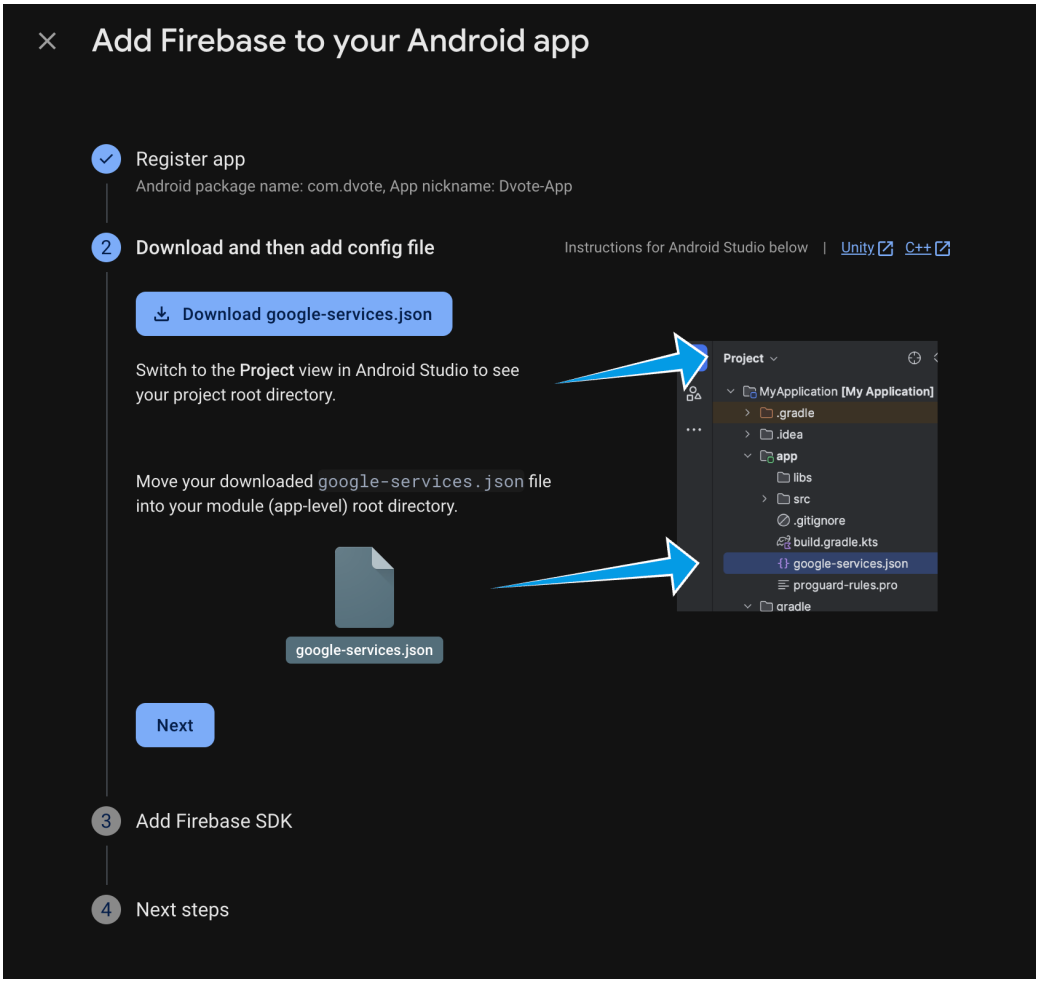

Рисунок Г.6 – отриманий файл конфігурацій firebase

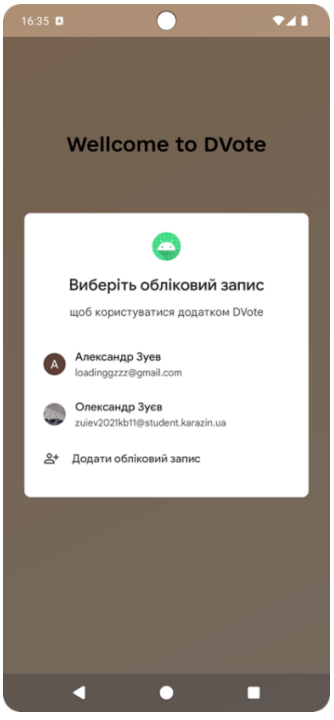

Рисунок Г.7 – Виклик модального вікна авторізації.

```
    private const val KEY_ALIAS = "user_key_alias"
    private const val ANDROID_KEYSTORE = "AndroidKeyStore"
    private const val KEY_ALGORITHM = "secp256r1"


    override fun generateKeyPairIfNeeded(): KeyPair {
        val keyStore = KeyStore.getInstance(ANDROID_KEYSTORE).apply {
            load(null)
        }

        if (!keyStore.containsAlias(KEY_ALIAS)) {
            val keyGen = KeyPairGenerator.getInstance(
                KeyProperties.KEY_ALGORITHM_EC,
                ANDROID_KEYSTORE
            )
            val spec = KeyGenParameterSpec.Builder(
                KEY_ALIAS,
                KeyProperties.PURPOSE_SIGN or KeyProperties.PURPOSE_VERIFY
            )
                .setAlgorithmParameterSpec(ECGenParameterSpec(KEY_ALGORITHM))
                .setDigests(KeyProperties.DIGEST_SHA256)
                .setUserAuthenticationRequired(false)
                .build()

            keyGen.initialize(spec)
            keyGen.generateKeyPair()
        }

        val entry = keyStore.getEntry(KEY_ALIAS, null) as KeyStore.PrivateKeyEntry
        return KeyPair(entry.certificate.publicKey, entry.privateKey)
    }
```

Рисунок Г.8 – Функція генерації ключа

```
val now = System.currentTimeMillis()
val pubBytes = keyStoreService.generateKeyPairIfNeeded().public.encoded
val pubBase64 = Base64.encodeToString(pubBytes, Base64.NO_WRAP)

var count = 0
for (choiceId in choiceIds) {
    val payload = "$surveyId|$choiceId|$now".toByteArray(Charsets.UTF_8)
    val sigBytes = keyStoreService.signData(payload)
    val signature = Base64.encodeToString(sigBytes, Base64.NO_WRAP)

    surveyRepository.submitSignedVote(
        surveyId  = surveyId,
        userId    = uid,
        choiceId  = choiceId,
        timestamp = now,
        signature = signature,
        publicKey = pubBase64
    )
    count++
}

profileRepository.addUserVotedSurvey(surveyId)
```

Рисунок Г.9 – Формування повідолмення підпису

```kotlin
override fun signData(data: ByteArray): ByteArray {
    val privateKey = generateKeyPairIfNeeded().private
    val sig = Signature.getInstance("SHA256withECDSA")
    sig.initSign(privateKey)
    sig.update(data)
    return sig.sign()
}
```

Рисунок Г.10 – Функція створення підпису голосування

```kotlin
suspend fun submitSignedVote(
    surveyId: String,
    userId: String,
    choiceId: String,
    timestamp: Long,
    signature: String,
    publicKey: String
) {
    val voteData = mapOf(
        "choiceId"  to choiceId,
        "timestamp" to timestamp,
        "signature" to signature,
        "publicKey" to publicKey
    )
    val voteDocId = "${userId}_$choiceId"

    firestore
        .collection("surveys")
        .document(surveyId)
        .collection("voters")
        .document(voteDocId)
        .set(voteData)
        .await()
}
```

Рисунок Г.11 – Надсилання підписаного голосу користувача

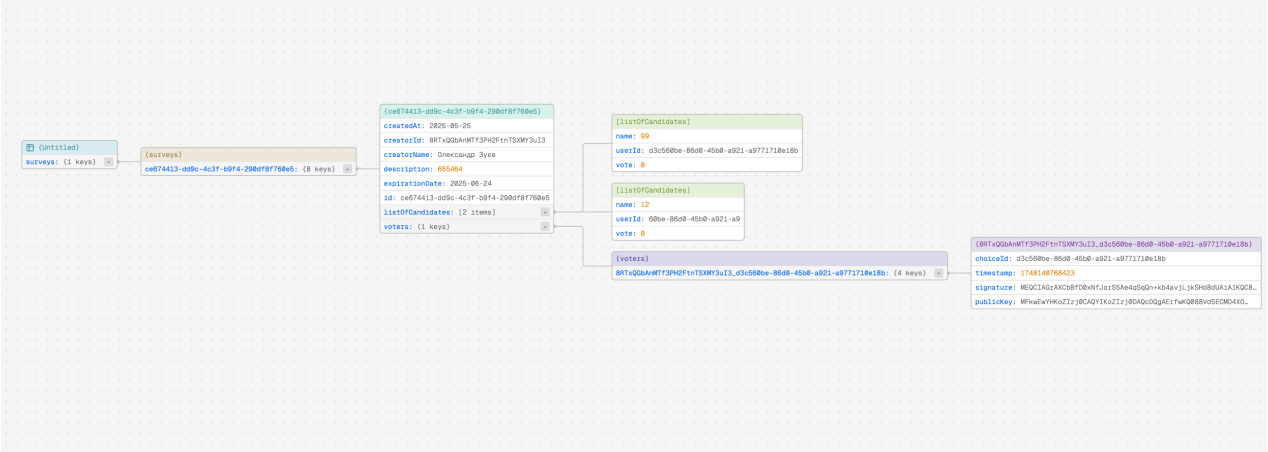

Рисунок Г.12 – Надсилання підписаного голосу користувача

```
suspend fun getSurveys(): List<SurveyDataItem> {
    return firestore.collection("surveys")
        .get()
        .await()
        .documents
        .map { it.toObject(SurveyDataItem::class.java)!! }
}
```

Рисунок Г.13 – Отримання списку з голосуваннями

```
fun observeSurveyById(surveyId: String): Flow<SurveyDataItem> = callbackFlow {
    val docRef = firestore.collection("surveys").document(surveyId)
    val listener = docRef.addSnapshotListener { snapshot, error ->
        if (error != null) {
            close(error)
            return@addSnapshotListener
        }
        snapshot?.toObject(SurveyDataItem::class.java)
            ?.let { trySend(it).isSuccess }
    }
    awaitClose { listener.remove() }
}
```

Рисунок Г.14 – Отримання списку з голосуваннями