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

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Метод та засіб для управління паролями»

Виконав: студент 2 курсу групи 1БС-24м
спеціальності 125 Кібербезпека та захист інформації
_____ Максим ТКАЧУК

Керівник: к. т. н., доцент каф. ЗІ
_____ Віталій ЛУКІЧОВ
«19» грудня 2025 р.

Рецензент: к.т.н., доц. каф. ПЗ
_____ Володимир МАЙДАНЮК
«19» грудня 2025 р.

Допущено до захисту
В. о. зав. каф. ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«19» грудня 2025 р.

Вінниця ВНТУ – 2025 року

Міністерство освіти і науки України
Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека та захист інформації
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ЗІ,
д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
«24» 09 2025 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Максиму ТКАЧУКУ

1. Тема роботи: «Метод та засіб для управління паролями» керівник роботи: Віталій ЛУКІЧОВ, к. т. н., доцент кафедри ЗІ, затверджені наказом ректора ВНТУ від 24 вересня 2025 року №313.
2. Строк подання студентом роботи 19 грудня 2025 року
3. Вихідні дані до роботи:
   - Мова програмування; Dart.
   - Фреймворк; Flutter.
   - Підтримка операційних систем: IOS, Android.
4. Зміст текстової частини: Вступ. 1. Аналіз інформаційних джерел. 2. Розробка методу управління паролями. 3. Розробка засобу для управління паролями. 4. Економічна частина. Висновки. Перелік використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: загальний алгоритм роботи засобу, схема зберігання ключів у застосунку з використанням YubiKey, алгоритм методу перевірки паролів, узагальнений протокол роботи застосунку, механізм HMAC-SECRET з розширенням FIDO2, схема шифрування даних з використанням AES-256-GSM, дизайн головної сторінки застосунку після додавання паролю, сторінка створення та редагування паролів, результати інтеграційного тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Віталій ЛУКІЧОВ, к.т.н., доц. каф. ЗІ	25.09.25	18.12.25
2	Віталій ЛУКІЧОВ, к.т.н., доц. каф. ЗІ	25.09.25	18.12.25
3	Віталій ЛУКІЧОВ, к.т.н., доц. каф. ЗІ	25.09.25	18.12.25
4	О. ЛЕСЬКО, зав. каф. ЕПВМ, к.е.н., доц.	25.09.25	18.12.25

7. Дата видачі завдання 24.09.25

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	24.09.2025 – 26.09.2025	
2	Аналіз інформаційних джерел за напрямком магістерської кваліфікаційної роботи	27.09.2025 – 07.10.2025	
3	Науково-технічне обгрунтування	11.10.2025 – 22.10.2025	
4	Аналіз інформаційних джерел	23.10.2025 – 26.10.2025	
5	Аналіз та формування методу захисту паролів	27.10.2025 – 02.11.2025	
6	Розробка засобу захисту паролів	03.11.2025 – 10.11.2025	
7	Тестування розробленого засобу	10.11.2025 – 17.11.2025	
8	Розробка розділу економічного обгрунтування доцільності розробки	18.11.2025 – 22.11.2025	
9	Оформлення пояснювальної записки	23.11.2025 – 29.11.2025	
10	Попередній захист та доопрацювання МКР	29.11.2025 – 11.12.2025	
11	Перевірка на наявність текстових запозичень	12.12.2025 – 15.12.2025	
12	Представлення МКР до захисту, рецензування	16.12.2025 – 19.12.2025	
13	Захист МКР	19.12.2025 – 23.12.2025	

Студент _____ Максим ТКАЧУК

Керівник роботи _____ Віталій ЛУКІЧОВ

АНОТАЦІЯ

УДК 004.056

Магістерська кваліфікаційна робота складається з 79 сторінок формату А4, на яких є 24 рисунка, 9 таблиць, перелік використаних джерел містить 21 найменування.

Магістерська кваліфікаційна робота присвячена дослідженню методів підвищення рівня захищеності систем мобільного застосунку для захисту паролів та практичній розробці захищеного застосунку. В ході роботи проведено порівняльний аналіз сучасних мобільних застосунків для управління паролями, в результаті якого виявлено недоліки в існуючих підходах до захисту.

На основі проведеного аналізу запропоновано удосконалений підхід до організації захищеного обміну даними, що включає модифіковані алгоритми шифрування та автентифікації. Розроблено узагальнену архітектуру застосунку та спроектовано протокол взаємодії застосунку, який забезпечує підвищений рівень конфіденційності та цілісності даних.

Практична складова роботи полягає у реалізації кросплатформного мобільного застосунку з використанням фреймворку Flutter та мови програмування Dart. Виконано комплексне тестування розробленого програмного продукту, яке підтвердило ефективність запропонованих рішень щодо безпеки та стабільність функціонування застосунку.

Ключові слова: кібербезпека, цілісність, доступність, конфіденційність, захист даних, персональні дані, смартфон, мобільний застосунок, YubiKey, авторизація, криптографічний захист, Dart, Flutter.

ABSTRACT

UDC 004.056

The master's thesis consists of 79 pages in A4 format, containing 24 figures, 9 tables, and a list of 21 references.

The master's thesis is devoted to researching methods for improving the security of mobile application systems for password protection and the practical development of a secure application. In the course of the work, a comparative analysis of modern mobile applications for password management was carried out, which revealed shortcomings in existing approaches to protection.

Based on the analysis, an improved approach to organizing secure data exchange was proposed, which includes modified encryption and authentication algorithms. A generalized application architecture was developed and an application interaction protocol was designed to provide an increased level of data confidentiality and integrity.

The practical part of the work consists in the implementation of a cross-platform mobile application using the Flutter framework and the Dart programming language. Comprehensive testing of the developed software product has been carried out, confirming the effectiveness of the proposed security solutions and the stability of the application.

Keywords: cybersecurity, integrity, availability, confidentiality, data protection, personal data, smartphone, mobile application, YubiKey, authorization, cryptographic protection, Dart, Flutter.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні системи відіграють ключову роль у функціонуванні бізнесу, державних установ та суспільства загалом. З розвитком цифрових технологій зростає обсяг оброблюваних даних та кількість користувацьких облікових записів, що зумовлює підвищені вимоги до безпеки автентифікації. Однією з найбільш критичних загроз є компрометація паролів, що призводить до несанкціонованого доступу, витоків даних та фінансових збитків. Традиційні підходи до збереження паролів часто виявляються недостатньо стійкими перед сучасними методами атак, такими як перебір, словникові атаки чи злам хешів. Це підкреслює необхідність удосконалення методів захисту автентифікаційних даних із використанням криптографічних механізмів.

Об’єктом є процес захищеного менеджера пролів фізичних осіб.

Предметом дослідження є метод та засіб захищеного зберігання паролів користувачів. Запропонований підхід передбачає застосування сучасних алгоритмів шифрування та хешування, зокрема стійких криптографічних функцій, сольових значень і механізмів багаторівневого захисту, що забезпечує підвищений рівень безпеки при зберіганні та перевірці паролів.

Метою дослідження є підвищення захисту паролів шляхом інтеграції криптографічних алгоритмів та використання апаратного ключа YubiKey, які гарантують конфіденційність, цілісність та стійкість до сучасних атак.

Науковою новизною є метод для управління паролями, що дозволить покращити менеджмент власних даних та підвищить безпеку та контроль над паролями, за допомогою апаратного ключа YubiKey.

Практична цінність полягає у створенні інструменту, що може бути впроваджений у різні інформаційні системи для підвищення рівня безпеки користувацьких даних. Використання криптографічного підходу дозволяє мінімізувати ризики компрометації, зменшити наслідки витоків даних та забезпечити відповідність сучасним вимогам кібербезпеки.

Методами дослідження, використаними в магістерській роботі, є криптографічні алгоритми та об’єктно-орієнтовані методи програмування, що забезпечують розробку та інтеграцію програмного засобу для захисту паролів.

Апробація результатів була здійснена у вигляді тез, представлених на 2 конференціях:

– Всеукраїнська науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» [1];

– Всеукраїнська науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» [2].

1 АНАЛІЗ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1.1 Аналіз сучасних загроз безпеці паролів

У сучасному світі цифрових технологій безпека автентифікаційних даних, зокрема паролів, стає однією з найбільш актуальних проблем кібербезпеки. Зростання кількості облікових записів та підвищення складності атак створюють нові виклики для захисту інформації. Щороку збільшується кількість випадків компрометації паролів, що призводить до несанкціонованого доступу, витоків персональних даних та значних фінансових збитків. Зловмисники застосовують різноманітні методи атак – від простого перебору та словникових атак до складних багатовекторних технік, що дозволяють обходити класичні механізми захисту.

Однією з ключових проблем є недостатня ефективність традиційних методів зберігання та перевірки паролів. Використання застарілих алгоритмів хешування без додаткових механізмів захисту (наприклад, «солі») робить системи вразливими до атак із застосуванням потужних обчислювальних ресурсів. Це призводить до швидкої компрометації даних та великої кількості інцидентів, пов’язаних із зламом облікових записів.

Іншою важливою проблемою є масштабність сучасних інформаційних систем. Щодня створюються та обробляються мільйони паролів, що потребує ефективних механізмів збереження й управління ними. Аналіз таких даних ускладнюється високою швидкістю атак та використанням кіберзлочинцями новітніх інструментів автоматизації.

Не менш значущою є проблема адаптації нових засобів захисту до існуючих систем. Багато організацій досі використовують застарілі підходи до управління паролями, що ускладнює інтеграцію сучасних рішень. Проте, враховуючи зростання кількості та складності атак, впровадження більш надійних і масштабованих методів захисту стає критичною необхідністю.

Таким чином, основна проблема захисту паролів полягає у невідповідності традиційних методів сучасним загрозам. З розвитком кіберзагроз виникає потреба в нових підходах до збереження та аналізу паролів, які здатні

забезпечити стійкість до сучасних атак і гарантувати надійний рівень безпеки даних.

1.2 Криптографічні методи захисту паролів

Принципи захисту паролів ґрунтуються на використанні різних методів та інструментів, що забезпечують надійне зберігання й перевірку автентифікаційних даних, а також мінімізацію ризиків їх компрометації. Це важливий процес у сфері кібербезпеки, адже паролі залишаються одним із найпоширеніших засобів автентифікації користувачів. Недостатній рівень їх захисту може призвести до витоків інформації, несанкціонованого доступу та значних фінансових збитків.

Захист паролів починається зі збереження їх у захищеному вигляді. Найпоширенішими методами є:

– хешування паролів із використанням криптографічних алгоритмів (SHA-2, SHA-3, bcrypt, scrypt, Argon2 [3]), що перетворюють пароль у нереверсивний рядок;

– використання солі (salt), яка додається до пароля перед хешуванням, щоб унеможливити атаки з попередньо обчисленими веселковими таблицями (rainbow tables) [4];

– перець (pepper) – додаткове секретне значення, яке зберігається окремо й підвищує стійкість захисту [5];

– багаторівневе хешування, що забезпечує складність і стійкість до атак із застосуванням сучасних обчислювальних потужностей;

– шифрування паролів за допомогою AES-256-GCM – симетричний алгоритм шифрування з автентифікацією, який забезпечує не лише конфіденційність даних, але й захист їх цілісності через генерацію тегу автентифікації, що дозволяє виявляти будь-які несанкціоновані зміни зашифрованої інформації [6].

Зібрані дані про паролі мають зберігатися в уніфікованому й безпечному вигляді. Для цього використовують спеціалізовані менеджери паролів (LastPass, 1Password, KeePass, Bitwarden), які шифрують дані та надають зручний

інтерфейс для управління ними. Важливою складовою є нормалізація та стандартизація підходів до автентифікації, що передбачає застосування єдиних практик зберігання та захисту паролів відповідно до вимог OWASP та NIST.

Не менш важливим етапом є агрегація та фільтрація інцидентів, пов'язаних із використанням паролів. Системи безпеки аналізують спроби входу, виявляють масові невдалі авторизації, атаки методом перебору (brute force), словникові атаки чи підозрілу поведінку користувачів. Такі методи дозволяють зменшити кількість хибних тривог і зосередитися на справді небезпечних подіях.

Наступним кроком є кореляція подій, коли різні спроби компрометації паролів об'єднуються в єдиний сценарій для виявлення складних атак. Наприклад, численні невдалі входи з різних IP-адрес можуть свідчити про атаку методом brute force, а поєднання підозрілих входів із нетиповою активністю користувача це про компрометацію облікового запису.

Сучасні системи захисту паролів активно використовують також аналіз аномалій. Він полягає у виявленні нетипової поведінки користувачів і пристроїв, наприклад, авторизація з нової геолокації, різке збільшення кількості спроб входу або звернення до ресурсів, якими користувач раніше не користувався. Такі механізми зазвичай реалізуються за допомогою машинного навчання та поведінкової аналітики.

Для захисту паролів застосовуються і додаткові інструменти:

– багатофакторна автентифікація (MFA), що поєднує пароль із додатковим фактором (SMS, токен, біометрія);

– системи контролю доступу (IAM, PAM), що централізовано керують правами користувачів;

– захист при передачі паролів (TLS, SSL), що гарантує безпеку обміну даними.

Таким чином, ефективний захист паролів передбачає застосування комплексного підходу, який включає криптографічні методи, інструменти управління, аналіз поведінки та багаторівневі механізми автентифікації. Комбінація цих рішень дозволяє підвищити стійкість до сучасних атак і забезпечити надійний захист критичних даних.

1.3 Огляд існуючих методів та інструментів захисту паролів

У сучасному світі інформаційна безпека стала одним із ключових факторів для успішного функціонування організацій будь-якого масштабу. Кількість і складність кіберзагроз постійно зростає, що зумовлює потребу у впровадженні більш ефективних та комплексних методів захисту. Одним із критично важливих напрямів у цій сфері є криптографічні методи захисту паролів, які забезпечують конфіденційність, цілісність і стійкість до злому облікових даних користувачів.

Паролі залишаються основним механізмом автентифікації у більшості інформаційних систем. Водночас просте зберігання паролів у відкритому вигляді робить систему надзвичайно вразливою до витоків даних. Саме тому у сучасних системах автентифікації застосовуються криптографічні алгоритми хешування, сольові та перцеві значення, а також методи багатофакторного захисту.

Основна ідея полягає в тому, щоб зробити відновлення пароля з хешу або його підбір обчислювально неможливим. Це досягається завдяки застосуванню стійких криптографічних алгоритмів (SHA-2, SHA-3, bcrypt, scrypt, Argon2), що ускладнюють атаки типу brute-force або словникові атаки.

Основні переваги для криптографічних методів захисту паролів:

– Стійкість криптографічних алгоритмів – необхідність застосування сучасних, захищених алгоритмів хешування (bcrypt, Argon2, PBKDF2), які витримують атаки підбору.

– Продуктивність та безпека – баланс між швидкістю обчислення хешу та складністю, щоб не перевантажувати систему, але й не допустити легкого злому.

– Захист від атак на хеш-таблиці – використання солі та перцю, щоб унеможливити застосування rainbow tables.

– Зміна стандартів – необхідність своєчасного оновлення алгоритмів у зв'язку з розвитком квантових обчислень і появою нових методів злому.

– Масштабованість – забезпечення захисту паролів у великих системах з мільйонами користувачів без значного падіння продуктивності.

Основні недоліки для криптографічних методів захисту паролів:

– Використання застарілих алгоритмів – MD5, SHA-1 та навіть SHA-256 без додаткових заходів (сіль/ітерації) є небезпечними.

– Слабкі паролі користувачів – навіть найкращий алгоритм хешування не допоможе, якщо пароль надто простий.

– Неправильна реалізація – помилки програмістів (зберігання паролів у відкритому вигляді, неправильне використання криптобібліотек).

– Атаки грубої сили та словникові атаки – постійне удосконалення інструментів для підбору паролів, включно з GPU-фермами та хмарними сервісами.

– Загроза витоку даних – навіть захешовані паролі при витоку бази можуть стати об'єктом атак, якщо обраний алгоритм недостатньо стійкий.

– Людський фактор – повторне використання паролів на різних сервісах, що підвищує ризики при компрометації одного з них.

Використання машинного навчання. Виявлення аномалій у введенні пароля – аналіз швидкості та ритму натискання клавіш (keystroke dynamics); виявлення відхилень від звичної поведінки користувача.

Адаптивна автентифікація – ML-моделі можуть оцінювати ризик входу за контекстом (IP, геолокація, час доби, пристрій); у випадку підозрілої активності – підвищувати рівень захисту (наприклад, вимагати 2FA).

Передбачення слабких паролів – нейромережі використовуються для моделювання та прогнозування вірогідних паролів (це загроза для захисту, але й допомога для розробників у тестуванні стійкості); системи можуть попереджати користувачів про небезпеку вибраного пароля.

Автоматизація зловмисних атак (негативний аспект) – ML застосовують хакери для більш ефективного підбору паролів (генерація словників на основі стилю користувача); криптографічні алгоритми повинні враховувати майбутні загрози від AI.

Для кращого розуміння які існуючі менеджери паролів є максимально корисними для певного підприємства потрібно провести порівняння популярних рішень від різних компаній.

Менеджери паролів можна поділити на комерційні та безкоштовні. Комерційні рішення (Okta Identity Cloud, Azure Active Directory, Ping Identity) є потужними інструментами з широким функціоналом, проте вони коштують дорого, що може бути проблемою для невеликих організацій. Тому в даному аналізі пріоритет буде віддано рішенням якими може користуватись одна людина, таким як LastPass, 1Password, Bitwarden, та KeePass.

LastPass – хмарний менеджер паролів, який зберігає зашифровані сейфи користувачів і синхронізує їх між усіма пристроями. Використовує AES-256, PBKDF2-SHA256, сіль для кожного користувача [7]. Алгоритм роботи LastPass зображено на рисунку 1.1.

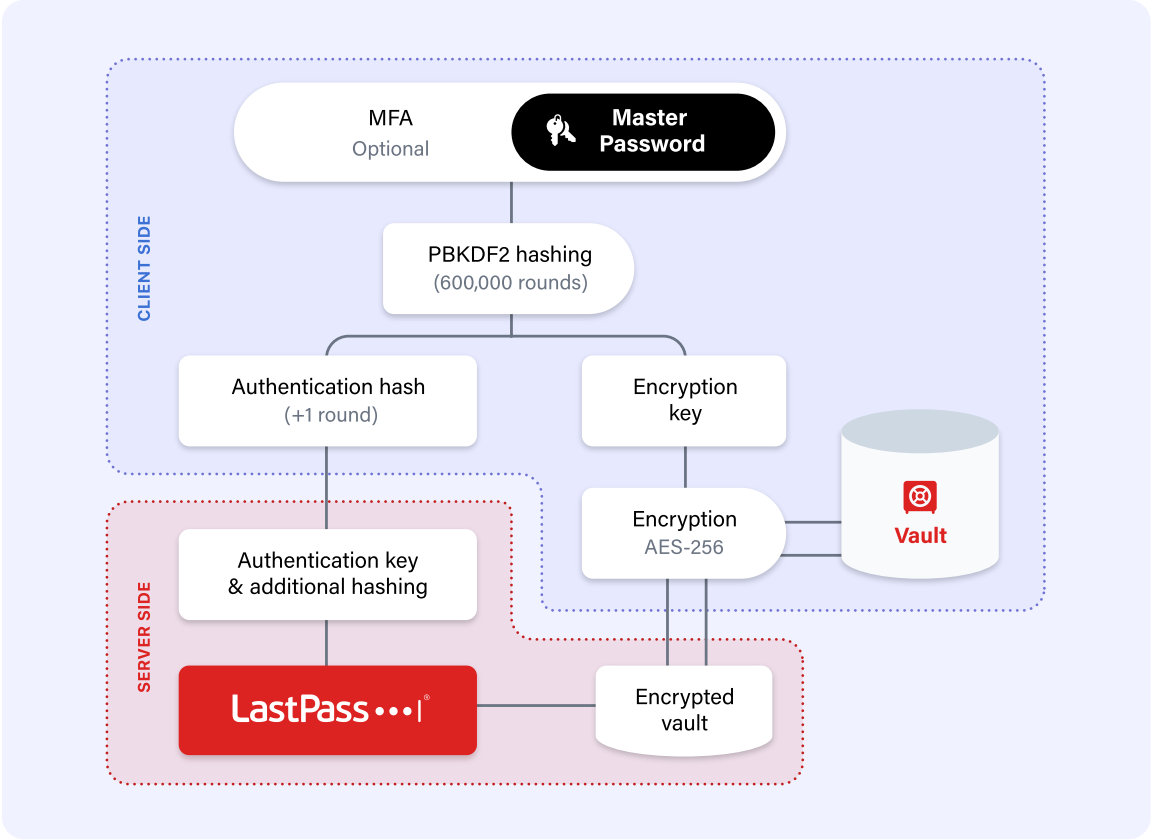


Рисунок 1.1 – Алгоритм роботи LastPass

LastPass як один із найбільш відомих менеджерів паролів на ринку пропонує широкий спектр переваг, що роблять його привабливим для масового користувача. Ключовою перевагою є повна кросплатформенність, яка забезпечує зручний доступ до сховища паролів із будь-якого пристрою, включаючи

операційні системи Windows та macOS на комп'ютерах, мобільні платформи IOS та Android на смартфонах і планшетах, а також всі популярні веббраузери через спеціалізовані розширення, що дозволяє користувачу отримати доступ до своїх облікових записів незалежно від того, яким пристроєм він користується в конкретний момент. Система пропонує високоякісне автоматичне автозаповнення форм входу та реєстрації на вебсайтах, що значно спрощує щоденну роботу з численними онлайн-сервісами та економить час користувача, а також забезпечує зручну інтеграцію з більшістю популярних вебресурсів через розпізнавання форм та інтелектуальне заповнення полів. Важливим елементом безпеки є вбудована підтримка двофакторної автентифікації, яка додає додатковий рівень захисту облікового запису користувача через вимогу підтвердження входу за допомогою другого фактора, такого як одноразовий код або біометрія. Крім основного функціоналу управління паролями, LastPass надає корисні додаткові можливості, включаючи безпечне збереження конфіденційних нотаток, реквізитів кредитних карток та іншої чутливої інформації, що перетворює його на універсальне сховище для всіх типів особистих даних.

Однак LastPass має значні недоліки, які критично впливають на його репутацію та рівень довіри користувачів. Найсерйознішою проблемою є історія кількох інцидентів з витоком даних, зокрема у 2015 та 2022 роках, коли зловмисники отримали несанкціонований доступ до інфраструктури компанії, що призвело до компрометації певних даних користувачів і значно підірвало довіру спільноти до безпеки платформи, оскільки навіть один такий інцидент є неприйнятним для сервісу, який позиціонується як захисник найбільш конфіденційної інформації користувачів. Хмарна модель зберігання даних, хоча і забезпечує зручність доступу з різних пристроїв, водночас створює підвищений ризик централізованих атак на сервери компанії, оскільки всі зашифровані сховища користувачів фізично знаходяться на інфраструктурі LastPass і теоретично можуть стати ціллю масштабних кібератак або компрометації через вразливості на рівні серверної інфраструктури чи операційних процесів компанії. Додатковим недоліком є суттєво обмежені можливості безкоштовної версії продукту, яка накладає жорсткі обмеження на кількість пристроїв, типи даних та

додаткові функції, фактично змушуючи користувачів переходити на платну підписку для отримання повноцінного досвіду використання менеджера паролів, що може бути критичним фактором для користувачів з обмеженим бюджетом або тих, хто не бажає здійснювати регулярні платежі за базовий функціонал безпеки.

1Password – це преміум-менеджер паролів із акцентом на простоту, дизайн та додаткові фактори безпеки. Паролі шифруються за допомогою AES-256, PBKDF2 та унікального “Secret Key” [8]. Алгоритм роботи 1Password зображено на рисунку 1.2.

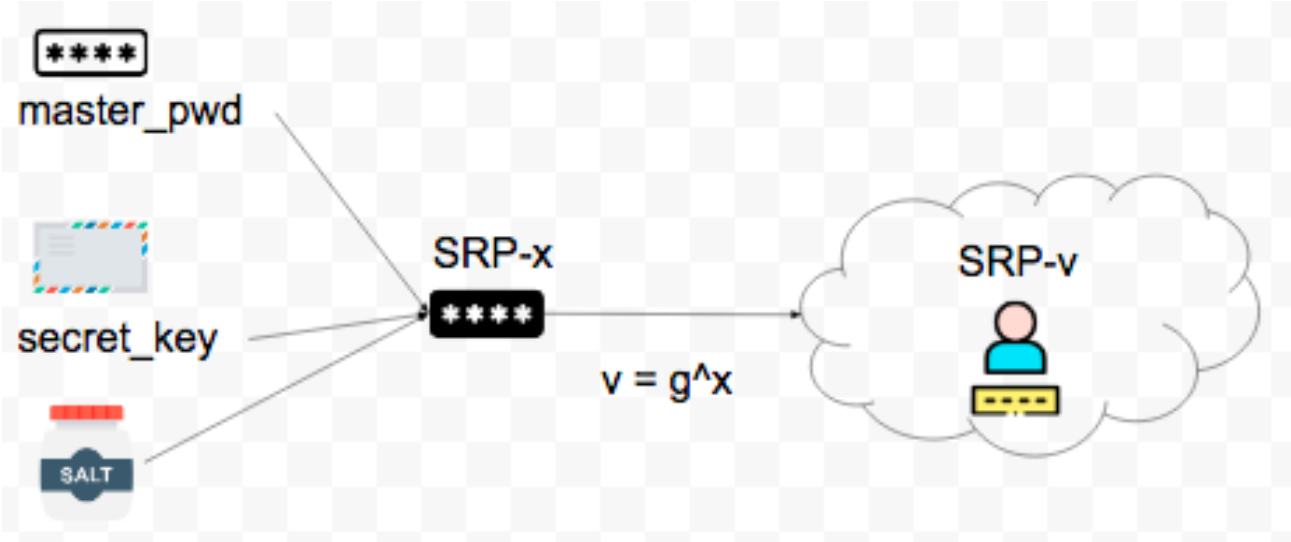


Рисунок 1.2 – Алгоритм роботи 1Password

Серед ключових переваг даного менеджера паролів слід виділити інтуїтивний користувацький інтерфейс, який спроектований з урахуванням потреб нетехнічних користувачів та дозволяє навіть людям без спеціальних знань у галузі інформаційної безпеки легко освоїти функціонал системи та ефективно управляти своїми обліковими записами. Важливою перевагою є реалізація багаторівневої системи захисту через двофакторну автентифікацію у поєднанні з власним Secret Key, який додає додатковий криптографічний рівень безпеки понад стандартний майстер-пароль, створюючи унікальну комбінацію, що робить несанкціонований доступ практично неможливим навіть у випадку компрометації одного з факторів. Система також пропонує гнучку підтримку

сімейних та корпоративних акаунтів, дозволяючи організувати спільне безпечне використання паролів у межах родини або робочої команди з розмежуванням прав доступу та централізованим управлінням, що особливо цінно для малого та середнього бізнесу. Крім того, менеджер має надійну репутацію на ринку кібербезпеки та за весь час свого існування не зазнав жодних серйозних зламів або витоків користувацьких даних, що підтверджує високу якість його криптографічної архітектури та відповідальний підхід розробників до безпеки.

Водночас система має певні недоліки, які можуть бути критичними для окремих категорій користувачів. По-перше, відсутність повністю безкоштовної версії з повним функціоналом обмежує доступність продукту для користувачів з обмеженим бюджетом, оскільки пропонується лише пробний період, після закінчення якого необхідно оформлювати платну підписку для продовження використання всіх можливостей системи. По-друге, застосування хмарної моделі зберігання даних, коли зашифрований сейф паролів зберігається на серверах розробника, може не підходити для користувачів з підвищеними вимогами до конфіденційності або тих, хто з принципових міркувань надає перевагу виключно локальному зберіганню даних без їхньої передачі на зовнішні сервери, навіть у зашифрованому вигляді. По-третє, вартість підписки на цей менеджер паролів є дещо вищою у порівнянні з основними конкурентами на ринку, що може змусити потенційних клієнтів обирати більш доступні за ціною альтернативи з подібним функціоналом, особливо якщо бюджетні обмеження є важливим фактором при виборі рішення для управління паролями.

Bitwarden – це менеджер паролів з відкритим кодом. Підтримує як хмарну синхронізацію, так і локальний хостинг. Застосовує AES-256, PBKDF2-SHA256, Argon2 (у нових версіях) [9]. Алгоритм роботи Bitwarden зображено на рисунку 1.3.

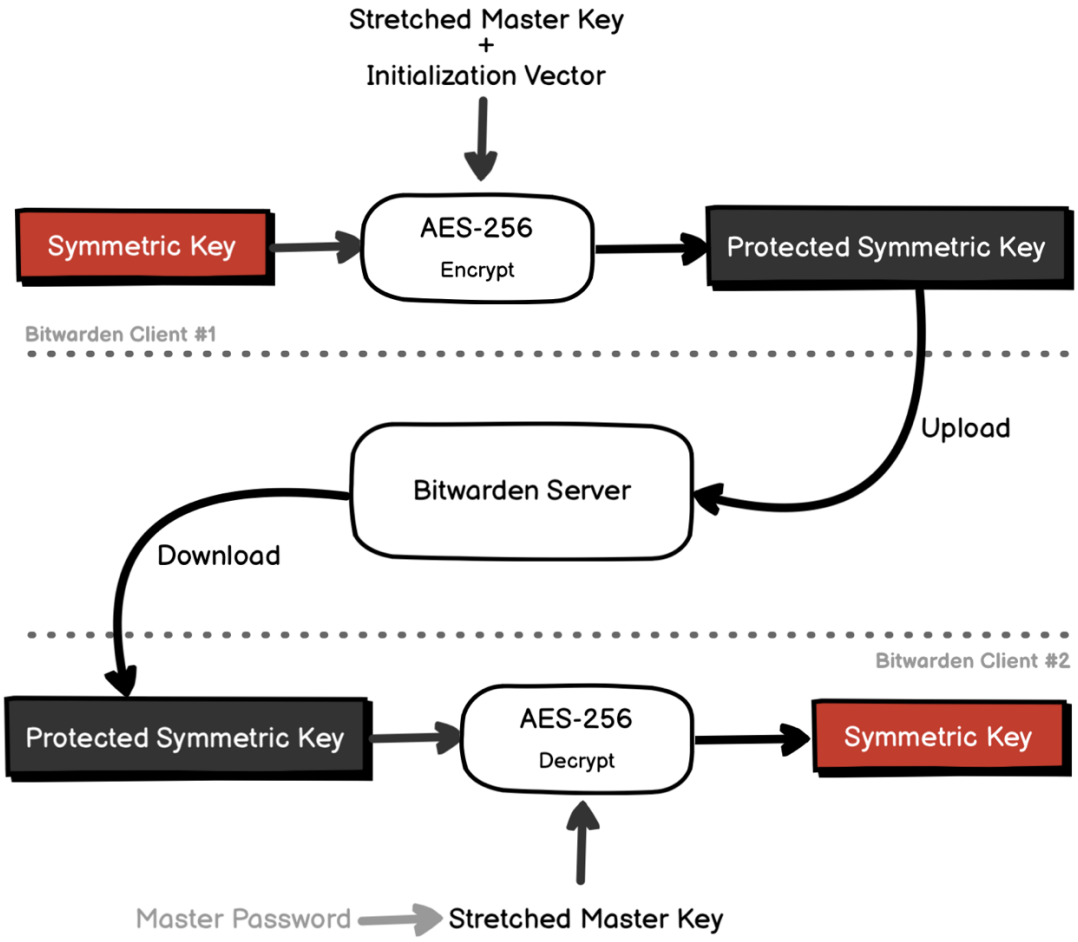


Рисунок 1.3 – Алгоритм роботи Bitwarden

Bitwarden виділяється серед конкурентів завдяки своїй гнучкості налаштувань, що робить його особливо привабливим для технічно підкованих користувачів та організацій з високими вимогами до безпеки. Найважливішою перевагою є повністю відкритий вихідний код (Open Source), який дозволяє будь-якому фахівцю з інформаційної безпеки або незалежній аудиторській компанії провести детальний аналіз коду на предмет потенційних вразливостей, бекдорів чи помилок у реалізації криптографічних алгоритмів, що забезпечує максимальну прозорість та підвищує довіру до продукту порівняно з закритими комерційними рішеннями. Унікальною можливістю є підтримка self-hosted розгортання, коли користувач або організація можуть встановити власний сервер Bitwarden на своїй інфраструктурі, отримуючи повний контроль над місцем фізичного зберігання даних та усуваючи необхідність довіряти зовнішньому постачальнику хмарних послуг, що критично важливо для компаній з суворими

вимогами до суверенітету даних або користувачів, які принципово не довіряють хмарним сервісам. Безкоштовна версія Bitwarden є надзвичайно щедрою та включає всі базові функції, необхідні більшості користувачів, зокрема необмежене сховище паролів, синхронізацію між пристроями, потужний генератор паролів з налаштуваними параметрами, та можливість безпечного обміну обліковими записами, що робить його доступним для широкої аудиторії без фінансових бар'єрів. Важливою перевагою є комплексна підтримка багатофакторної автентифікації через різноманітні методи, включаючи стандартний TOTP (Time-based One-Time Password) для генерації одноразових кодів, апаратні ключі безпеки YubiKey [10] для максимального рівня захисту, та інтеграцію з сервісом Duo для корпоративної двофакторної автентифікації, що дозволяє користувачам обрати оптимальний для них спосіб додаткового захисту облікового запису.

Попри численні переваги, Bitwarden має певні недоліки, які можуть вплинути на вибір користувача. Користувацький інтерфейс Bitwarden є менш візуально відполірованим та інтуїтивним порівняно з преміальними рішеннями на кшталт 1Password, оскільки команда розробників приділяє більше уваги функціональності та безпеці, ніж естетиці та UX-дизайну, що може створювати враження застарілості або складності для користувачів, які звикли до сучасних, елегантних інтерфейсів мобільних застосунків. Для користувачів-початківців або тих, хто не має технічного досвіду адміністрування серверів, процес налаштування власного self-hosted сервера Bitwarden може виявитися складним та časозатратним завданням, що вимагає знань у галузі DevOps, управління базами даних, налаштування SSL-сертифікатів та забезпечення безпеки серверної інфраструктури, через що багато користувачів можуть відмовитися від цієї переваги на користь простішого хмарного варіанту. Хоча безкоштовна версія Bitwarden є надзвичайно функціональною, деякі просунуті можливості, такі як розширені звіти про безпеку паролів, пріоритетна технічна підтримка, розширене збереження файлів та додаткові опції організації даних, доступні виключно у преміум-версії, проте слід зазначити, що вартість преміум-підписки

є значно нижчою порівняно з конкурентами, що робить ці обмеження менш критичними для більшості користувачів, які готові платити за додаткові функції.

KeePass – класичний офлайн-менеджер паролів, де всі дані зберігаються локально у зашифрованій базі. Шифрування: AES-256, ChaCha20, Twofish [11]. Алгоритм роботи KeePass зображено на рисунку 1.4.

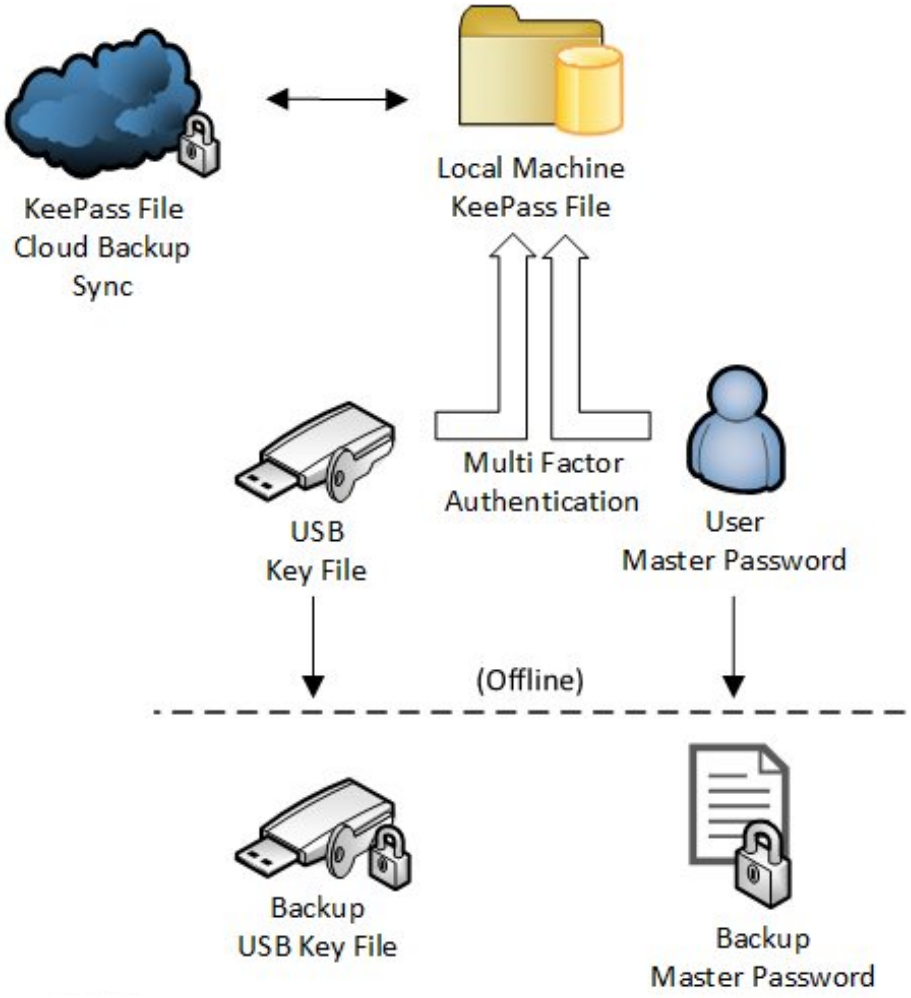


Рисунок 1.4 – Алгоритм роботи KeePass

KeePass є одним із найстаріших та найпопулярніших менеджерів паролів серед користувачів, які цінують максимальний контроль над своїми даними та прозорість програмного забезпечення. Найвагомішою перевагою є те, що KeePass є повністю безкоштовним рішенням з відкритим вихідним кодом, що дозволяє будь-якому користувачу або безпековому експерту переглянути, перевірити та навіть модифікувати код програми для власних потреб, усуваючи

будь-які питання щодо прихованих функцій, бекдорів чи потенційних вразливостей, а також гарантуючи відсутність комерційних обмежень або вимог до підписки. Система забезпечує найвищий рівень безпеки завдяки тому, що база даних з паролями зберігається виключно локально на пристрої користувача у вигляді зашифрованого файлу, що надає повний контроль над місцем розташування та методами резервного копіювання конфіденційної інформації без необхідності довіряти її зовнішнім хмарним сервісам або серверам розробників. Унікальною особливістю є можливість використання ключових файлів як додаткового фактора автентифікації, де для розблокування бази паролів необхідно не лише знати майстер-пароль, але й мати доступ до спеціального файлу, що може зберігатися на USB-накопичувачі або іншому окремому носії, створюючи багаторівневу систему захисту від несанкціонованого доступу навіть у випадку компрометації пароля. Завдяки відкритій архітектурі KeePass підтримує велику кількість плагінів та розширень, розроблених спільнотою, що дозволяє користувачам кастомізувати функціонал програми під свої специфічні потреби, додавати інтеграції з різними сервісами, покращувати інтерфейс або впроваджувати додаткові механізми безпеки відповідно до індивідуальних вимог.

Однак KeePass має суттєві недоліки, які обмежують його зручність використання для широкої аудиторії, особливо для нетехнічних користувачів. Найбільш проблемним аспектом є відсутність вбудованої зручної автоматичної синхронізації бази паролів між різними пристроями, що змушує користувачів вдаватися до ручного копіювання файлу бази даних або використовувати сторонні хмарні сервіси на кшталт Dropbox, Google Drive чи OneDrive для забезпечення синхронізації, при цьому користувач повинен самостійно стежити за актуальністю версій файлу на різних пристроях та розв'язувати потенційні конфлікти при одночасному редагуванні. Користувацький інтерфейс KeePass виглядає застарілим та базується на класичних принципах дизайну Windows-додатків початку 2000-х років, що робить його менш інтуїтивним та привабливим для користувачів, які звикли до сучасних мобільних застосунків з мінімалістичним дизайном та зручною навігацією, а відсутність полірованого

UX може відлякувати новачків, які шукають простий та зрозумілий інструмент для управління паролями. Мобільний досвід використання KeePass значно поступається конкурентам, оскільки офіційної версії для IOS та Android не існує, і користувачі змушені покладатися на сторонні порти та реалізації, такі як KeePassXC для десктопних платформ або KeePassDX та Keepass2Android для мобільних пристроїв, які розробляються різними командами, мають різний рівень функціональності та підтримки, що створює фрагментований екосистемний досвід та потенційні проблеми сумісності або безпеки при виборі неперевіреної реалізації.

Якщо проводити узагальнення всіх описаних вище систем то їх можна описати через наступну таблицю 1.1.

Таблиця 1.1 – Порівняння популярних менеджерів паролів

Менеджер паролів	Переваги	Недоліки
LastPass	Доступ з усіх пристроїв, автозаповнення, підтримка 2FA	Були витоки даних, ризик атаки на сервери
1Password	Зручний інтерфейс, додатковий “Secret Key”	Немає повністю безкоштовної версії, дорожчий за інші, лише хмарне зберігання
Bitwarden	Відкритий код, Self-hosted сервер, підтримка MFA (TOTP, YubiKey)	Інтерфейс менш сучасний, налаштування self-hosted складне
KeePass	Повний контроль над даними, Open Source Підтримка ключових файлів Плагіни та кастомізація	Немає зручної синхронізації, застарілий інтерфейс, обмежені мобільні можливості

Криптографічні методи захисту паролів є фундаментальною складовою сучасних систем інформаційної безпеки. Використання сольових і перцевих значень, стійких алгоритмів хешування та інтеграція з машинним навчанням дозволяє значно підвищити рівень захисту від атак. Проте ефективність таких систем напряму залежить від правильного налаштування криптографічних алгоритмів і політик безпеки в організації.

1.4 Порівняльний аналіз алгоритмів хешування та шифрування

У сучасних умовах розвитку інформаційних технологій та зростання кількості кіберзагроз використання надійних алгоритмів хешування та шифрування є ключовим елементом забезпечення інформаційної безпеки. Вибір алгоритму безпосередньо впливає на стійкість систем до атак, швидкість обробки даних та можливість масштабування. Хешування та шифрування реалізуються різними підходами, кожен з яких має власні переваги й обмеження.

Хеш-функції забезпечують одностороннє перетворення даних у фіксований розмір, що робить їх незворотними. Це дає змогу зберігати паролі та перевіряти цілісність даних. Серед відомих алгоритмів хешування можна виділити MD5 та SHA-1, які вважаються застарілими та вразливими до колізій. Більш надійними є алгоритми сімейства SHA-2, зокрема SHA-256 та SHA-512, проте вони є швидкими, тому потребують додаткових механізмів захисту. Спеціалізовані алгоритми для захисту паролів, такі як bcrypt, scrypt та Argon2, використовують механізми солі та ітерацій для підвищення стійкості.

До переваг хешування належить висока швидкість обчислення для базових алгоритмів, ефективність у перевірці цілісності даних, а також те, що адаптивні алгоритми на кшталт bcrypt та Argon2 захищають від атак брутфорсу. Водночас існують і недоліки: застарілі алгоритми не захищені від колізій, надмірна швидкість роботи робить їх уразливими до атак перебором, а також існує постійна потреба у оновленні стандартів у відповідь на нові загрози [12].

Симетричне шифрування використовує один ключ для шифрування і розшифрування даних. Найчастіше воно застосовується для захисту великих обсягів інформації. Сучасним стандартом є алгоритм AES з підтримкою ключів довжиною 128, 192 та 256 біт. Альтернативою AES є ChaCha20, який демонструє високу швидкість у програмному виконанні. Ще одним представником симетричного шифрування є Twofish, алгоритм із гнучкою структурою ключів.

Серед переваг симетричного шифрування виділяють високу швидкість обробки даних, простоту реалізації, а також придатність для потокового шифрування великих масивів інформації. До недоліків належить потреба у

надійному зберіганні ключів та менша безпечність у випадку компрометації ключа, оскільки один ключ використовується для обох операцій [13].

Асиметричне шифрування базується на парі ключів: публічному і приватному. Воно використовується для автентифікації, обміну ключами та створення цифрових підписів. Серед найпоширеніших алгоритмів виділяють RSA, який широко застосовується, проте є повільним, та ECC (криптографія на еліптичних кривих), що забезпечує високу стійкість при коротших ключах порівняно з RSA.

До переваг асиметричного шифрування належить підвищений рівень безпеки завдяки використанню двох ключів, можливість реалізації цифрових підписів, а також ефективне керування доступом у розподілених системах. Водночас існують і недоліки: низька швидкість у порівнянні із симетричними алгоритмами та високі обчислювальні витрати, що обмежує застосування асиметричного шифрування для обробки великих обсягів даних.

Для ефективного захисту інформації рекомендується комбінувати різні алгоритми залежно від конкретних потреб та сценаріїв використання. Хешування з використанням bcrypt або Argon2 доцільно застосовувати для безпечного зберігання паролів, оскільки ці алгоритми спеціально розроблені для протидії атакам перебором. Симетричне шифрування, зокрема AES або ChaCha20, оптимально підходить для захисту даних у процесі зберігання та передавання завдяки високій швидкості обробки. Асиметричне шифрування з використанням RSA або ECC найкраще використовувати для розподілу ключів та автентифікації користувачів у системах із високими вимогами до безпеки.

Порівняння алгоритмів хешування та шифрування наведено в таблиці 1.2.

Комплексний підхід до вибору криптографічних алгоритмів дозволяє створити багаторівневу систему захисту, яка ефективно протидіє сучасним кіберзагрозам. При цьому важливо враховувати не лише технічні характеристики алгоритмів, але й контекст їх використання, обчислювальні ресурси системи та потенційні моделі загроз. Постійний моніторинг розвитку криптографічних стандартів та своєчасне оновлення використовуваних алгоритмів є невід'ємною частиною забезпечення довгострокової інформаційної безпеки.

Таблиця 1.2 – Порівняння алгоритмів хешування та шифрування

Технологія	Призначення	Переваги	Недоліки	Приклад використання
MD5, SHA-1	Хешування даних	Висока швидкість	Вразливі до колізій	Старі системи, контроль цілісності
SHA-2, SHA-3	Сучасне хешування	Надійність, стандарт NIST	Швидкість → уразливість до перебору	Хешування файлів, цифрові підписи
bcrypt, Argon2	Захист паролів	Стійкі до brute-force, адаптивні	Повільніші у виконанні	Менеджери паролів, бази даних
AES, ChaCha20	Симетричне шифрування	Висока швидкість, надійність	Проблема зберігання ключів	VPN, дискове шифрування
RSA, ECC	Асиметричне шифрування та підписи	Висока безпека, цифрові підписи	Повільні, ресурсомісткі	SSL/TLS, PKI-системи

Таким чином, порівняльний аналіз показує, що жоден алгоритм не є універсальним: хешування доцільно застосовувати для захисту паролів, симетричне шифрування – для даних, а асиметричне – для автентифікації та обміну ключами.

1.5 Мови програмування та фреймворки

Для розробки мобільного застосунку для захисту паролів необхідно розглянути, мови програмування та фреймворки, щоб побачити, який підійде найліпше, тому у даному розділі було проаналізовано: Kotlin, Swift, React Native, Flutter.

Kotlin – це статично типізована мова програмування, що використовується переважно для розробки Android-застосунків. Вона була розроблена компанією JetBrains і офіційно підтримується Google як альтернатива Java для Android. Kotlin славиться своєю безпекою відносно помилок в часі виконання та більш виразною та компактною синтаксичною структурою порівняно з Java [14].

Swift – це мова програмування, розроблена Apple для IOS, macOS, watchOS та tvOS. Swift призначений для роботи з фреймворками Apple та має сучасний, чистий синтаксис. Він забезпечує безпеку типів та пам'яті, а також містить ряд особливостей, що спрощують написання коду, як-от управління пам'яттю без прямого втручання програміста [15].

React Native – це фреймворк для розробки крос-платформних мобільних застосунків, розроблений Facebook. Він дозволяє розробникам використовувати React разом з нативною платформою для створення застосунків для IOS та Android з однієї кодової бази. React Native використовує JavaScript для розробки, що робить його доступнішим для великої спільноти веб-розробників [16].

Flutter [17], розроблений Google, також є крос-платформним фреймворком для створення мобільних, веб- та настільних застосунків. Flutter використовує мову Dart [18] і надає велику кількість вбудованих віджетів, що дозволяє розробникам створювати візуально привабливі інтерфейси з відносною легкістю.

Таким чином, в таблиці 1.4 наведено порівняльна характеристика мов програмування та фреймворків для створення мобільних застосунків.

Таблиця 1.4 – Характеристика мов програмування та фреймворків

Особливості	Kotlin	Swift	React Native	Flutter
Платформа	Android (головно)	Apple's IOS and macOS	Крос-платформний (IOS, Android)	Крос-платформний (IOS, Android, Web, Desktop)
Мова	Kotlin	Swift	JavaScript	Dart
Тип	Мова програмування	Мова програмування	Фреймворк	Фреймворк
Виконання	Компілюється в байт-код	Компілюється в машинний код	Інтерпретується через JavaScript	Компілюється в машинний код
Розробка UI	Нативний UI	Нативний UI	Через JavaScript (Бібліотека компонентів)	Власна високоповий двигун для створення UI
Спільнота та підтримка	Велика спільнота, офіційна підтримка Google для Android	Широка підтримка від Apple, велика спільнота	Велика спільнота, багато ресурсів і бібліотек	Швидко зростаюча спільнота, активна підтримка від Google

Ця таблиця та аналіз демонструють, що вибір між цими технологіями залежить від конкретних потреб проекту, цільової платформи та особистих переваг розробника. Kotlin і Swift є відмінними варіантами для розробки нативних застосунків для Android та IOS відповідно, в той час як React Native і Flutter пропонують переваги крос-платформної розробки, дозволяючи розробникам використовувати одну кодову базу для створення застосунків на кілька платформ.

1.6 Постановка задачі

У сучасних умовах активного розвитку інформаційних технологій та глобальної цифровізації зростає кількість випадків несанкціонованого доступу до конфіденційних даних. Паролі залишаються одним із найбільш поширених засобів автентифікації користувачів, проте саме вони часто стають об’єктом атак, таких як підбір, перехоплення чи викрадення зловмисниками. Використання простих або скомпрометованих паролів значно знижує рівень захищеності інформаційних систем.

Для вирішення цієї проблеми необхідно розробити метод та засіб захисту паролів, що базується на сучасних криптографічних підходах. Таке рішення має забезпечувати:

– стійке збереження паролів за допомогою алгоритмів хешування з використанням солі та перцю;

– можливість шифрування паролів при їх передачі через мережеві канали;

– захист від атак словником, brute-force та rainbow tables;

– ефективність реалізації з урахуванням продуктивності системи;

– масштабованість для інтеграції з існуючими інформаційними системами.

Таким чином, постає задача розробити метод і програмний засіб для захисту паролів на основі криптографії, який поєднуватиме алгоритми хешування та шифрування, забезпечуючи високий рівень безпеки автентифікації користувачів в інформаційних системах.

2 РОЗРОБКА МЕТОДУ УПРАВЛІННЯ ПАРОЛЯМИ

2.1 Обґрунтування вибору криптографічних алгоритмів

Розробка мобільного застосунку для управління паролями вимагає ретельного підходу до вибору криптографічних алгоритмів, оскільки безпека користувацьких даних є критично важливою. Сучасні менеджери паролів повинні забезпечувати надійне шифрування, стійкість до атак та оптимальну продуктивність на мобільних пристроях з обмеженими ресурсами. Вибір криптографічних алгоритмів базується на аналізі загроз, вимогах до продуктивності та міжнародних стандартах безпеки.

Основними критеріями вибору криптографічних алгоритмів для застосунку є стійкість до сучасних методів криптоаналізу, підтримка на мобільних платформах, продуктивність та енергоефективність, відповідність міжнародним стандартам безпеки, наявність апаратного прискорення на сучасних процесорах, а також підтримка спільнотою та регулярні аудити безпеки. Для надійної безпеки використовується криптографічний набір з трьох різних алгоритмів, кожен з яких виконує своє специфічне завдання у забезпеченні комплексного захисту даних користувача.

Для шифрування локального сховища паролів в офлайн менеджері використовується AES-256-GCM, який є золотим стандартом симетричного шифрування у сучасній криптографії. AES з довжиною ключа 256 біт застосовують уряди, банки та всі надійні менеджери паролів завдяки його надзвичайній швидкості та абсолютній стійкості до атак грубої сили станом на 2024 рік. Апаратна підтримка AES на всіх сучасних процесорах забезпечує високу продуктивність навіть на мобільних пристроях з обмеженими ресурсами. Найважливішою частиною є режим GCM (Galois/Counter Mode), який забезпечує автентифіковане шифрування AEAD (Authenticated Encryption with Associated Data). Це означає, що алгоритм одночасно гарантує конфіденційність даних через шифрування та їхню цілісність через генерацію тегу автентифікації. Якщо зловмисник спробує змінити хоча б один біт у зашифрованому сховищі паролів, GCM негайно виявить це під час спроби розшифрування, і застосунок

відмовиться від доступу до даних, захищаючи користувача від атак на цілісність інформації.

Офлайн менеджер паролів використовує YubiKey як додатковий фактор захисту через протокол FIDO2 з розширенням HMAC-Secret, що забезпечує апаратну автентифікацію без необхідності підключення до інтернету. YubiKey генерує криптографічно стійкі секретні значення, які зберігаються виключно у захищеному апаратному модулі ключа і ніколи не можуть бути витягнуті або скопійовані. Під час початкової реєстрації застосунок генерує випадкове значення salt розміром 32 байти і надсилає його через NFC або USB до YubiKey, який застосовує функцію HKDF з власним приватним ключем та отриманим salt для генерації спільного секрету sharedSecret.

YubiKey шифрує згенерований sharedSecret за допомогою алгоритму ECDH і повертає його застосунку у захищеному вигляді, після чого мобільний додаток розшифровує та безпечно зберігає цей секрет у апаратному сховищі пристрою (Keychain на IOS або Keystore на Android) разом із відповідним salt. Кожного разу, коли користувач намагається розблокувати сховище паролів, застосунок генерує новий випадковий challenge розміром 32 байти і надсилає його до YubiKey через розширення hmac-secret, а ключ обчислює HMAC-SHA256 з використанням збереженого sharedSecret та отриманого challenge, повертаючи 32-байтний HMAC як доказ автентичності.

Для перетворення майстер-пароля користувача на криптографічний ключ шифрування застосовується Argon2id, сучасна функція деривації ключа, яка перемогла у міжнародному конкурсі Password Hashing Competition. Майстер-пароль не можна безпосередньо використовувати як 256-бітний ключ для AES-256-GCM, оскільки звичайні паролі є вразливими до словникових атак та швидкого перебору з використанням потужних обчислювальних ресурсів. Argon2id бере пароль користувача та випадково згенеровану криптографічну сіль, після чого виконує надзвичайно ресурсоємні обчислення, які вимагають значного часу процесора та, найважливіше, великого обсягу оперативної пам'яті.

Ця особливість робить атаки з використанням спеціалізованих відеокарт (GPU) або ASIC-чіпів надзвичайно неефективними та економічно недоцільними,

оскільки кожна спроба підбору пароля стає повільною та вимагає значних апаратних ресурсів. Варіант Argon2id поєднує захист від атак через кеш-тайминг (притаманний Argon2i) та максимальну стійкість до GPU-атак (притаманну Argon2d), забезпечуючи найкращий баланс безпеки для мобільних застосунків.

Параметри Argon2id налаштовуються таким чином, щоб деривація ключа займала приблизно 500-1000 мілісекунд на середньому мобільному пристрої, створюючи помітну затримку для користувача, але роблячи масовий перебір паролів практично неможливим.

Повний процес розблокування сховища з YubiKey. Коли користувач намагається отримати доступ до свого офлайн сховища паролів, система виконує комбіновану автентифікацію з використанням як майстер-пароля, так і YubiKey. Спочатку користувач вводить свій майстер-пароль, після чого застосунок витягує збережену криптографічну сіль зі сховища та пропускає комбінацію пароля і солі через функцію Argon2id для отримання першого компонента ключа шифрування. Одночасно застосунок генерує випадковий challenge та надсилає його до приєднаного YubiKey (через NFC або USB), який обчислює HMAC-SHA256 на основі збереженого у його апаратному модулі sharedSecret та отриманого challenge.

Застосунок самостійно обчислює очікуване значення HMAC на основі свого збереженого sharedSecret та порівнює його з отриманим від YubiKey значенням для підтвердження автентичності апаратного ключа. Після успішної верифікації обох факторів автентифікації, застосунок комбінує отриманий від Argon2id ключ та результат автентифікації YubiKey для формування фінального 256-бітного ключа шифрування, який використовується для розшифрування сховища паролів за допомогою AES-256-GCM. Режим GCM автоматично перевіряє тег автентифікації, гарантуючи, що сховище не було пошкоджене або модифіковане, і лише після успішної верифікації цілісності даних застосунок надає користувачу доступ до збережених паролів та облікових записів.

Така архітектура забезпечує максимальний рівень безпеки офлайн менеджера паролів, оскільки для злому сховища зловмиснику потрібно одночасно отримати три незалежні компоненти: знати майстер-пароль

користувача, мати фізичний доступ до YubiKey, та володіти зашифрованим файлом сховища. Навіть якщо одна з цих складових буде скомпрометована, дані залишаються захищеними завдяки багаторівневій системі криптографічного захисту, що робить такий підхід одним із найнадійніших методів захисту паролів у мобільних застосунках.

2.2 Метод генерації та зберігання ключів

Метод генерації та зберігання ключів у мобільному застосунку для менеджменту паролів передбачає використання криптографічних найкращих практик для забезпечення безпеки. Цей процес включає derivation ключів від master password, використання безпечних функцій derivation (KDF), шифрування даних та безпечне зберігання ключів у апаратно-захищених сховищах.

Процес генерації ключів починається з отримання головного пароля від користувача, який проходить через криптографічну функцію виведення ключа (Key Derivation Function). Найсучасніші менеджери паролів використовують такі алгоритми як PBKDF2, bcrypt, scrypt або Argon2, які спеціально розроблені для повільного обчислення ключів з метою захисту від атак перебору. Ці функції приймають master password разом із унікальною сіллю (salt) та виконують тисячі або навіть мільйони ітерацій хешування, що робить атаки методом грубої сили практично неможливими. Результатом цього процесу є криптографічно стійкий ключ шифрування, який використовується для захисту всіх збережених паролів користувача.

Зберігання згенерованих ключів здійснюється з використанням апаратно-захищених сховищ, таких як Keychain на IOS або Keystore на Android. Ці системи забезпечують додатковий рівень захисту через апаратну ізоляцію криптографічних операцій у спеціалізованих процесорах безпеки (Secure Enclave на IOS або Trusted Execution Environment на Android). Криптографічні ключі ніколи не зберігаються у відкритому вигляді в пам'яті застосунку чи файловій системі, а всі операції шифрування та дешифрування відбуваються всередині захищеного середовища. Додатково, сучасні системи підтримують біометричну автентифікацію, що дозволяє користувачам безпечно розблокувати доступ до

ключів за допомогою відбитка пальця або розпізнавання обличчя, при цьому біометричні дані ніколи не залишають апаратний модуль безпеки пристрою.

Найкращий метод – це не просто брати випадкові символи, а гарантувати наявність кожного типу символів, а потім перемішати їх. Алгоритм генерації безпечних паролів у мобільному менеджері передбачає надання користувачу повного контролю над параметрами створюваного пароля через інтуїтивний інтерфейс з налаштуваннями. Користувач може самостійно визначити довжину майбутнього пароля за допомогою зручного слайдера, який дозволяє обрати значення в діапазоні від 12 до 64 символів, забезпечуючи баланс між безпекою та зручністю використання на різних платформах та сервісах. Крім довжини, система надає можливість гнучкого налаштування складу символів через набір прапорців, що дозволяють включити або виключити різні категорії символів: великі літери латинського алфавіту від A до Z для додаткової складності, малі літери від a до z як базовий компонент пароля, цифри від 0 до 9 для числового різноманіття, а також спеціальні символи з набору !@#$%^&*()_+–=[]{}|;:',.<>?/~ для максимального підвищення ентропії та стійкості до атак підбору. На основі обраних користувачем параметрів алгоритм використовує криптографічно безпечний генератор псевдовипадкових чисел (CSPRNG) для створення пароля, який відповідає всім заданим критеріям та забезпечує високий рівень захисту облікових записів від несанкціонованого доступу.

Для роботи з менеджером було розглянуто 3 методи:

1) з використанням мобільного сховища;
2) з використанням звичайної флешки;
3) з використанням YubiKey;

Загальний алгоритм роботи засобу зображена на риснуку 2.1.

Рисунок 2.1 – Загальний алгоритм роботи засобу

Використанням мобільного сховища. В залежності від платформи мобільного пристрою (Android чи IOS) вибирається метод зберігання ключів. Більшість сучасних мобільних ОС мають вбудовані захищені сховища, такі як

Android Keystore або IOS Keychain, що дозволяють безпечно зберігати ключі без можливості їх експорту.

Для забезпечення безпечного зберігання ключів використовуються апаратно-захищені модулі, що запобігає витоку ключів навіть у разі компрометації ОС. Залежно від вимог до безпеки та продуктивності, необхідно налаштувати параметри KDF, наприклад, ітерації, пам'ять та паралелізм для Argon2id, щоб балансувати між швидкістю та стійкістю до brute-force атак.

Після всіх налаштувань та генерації ключів застосунок ініціалізує процес шифрування та зберігання паролів. Це критично важлива частина. Паролі, які ви згенерували, не можна хешувати. Хешування (Argon2id) – це одностороння функція. Використовується для перевірки майстер-пароля. Не можна "розхешувати" його, щоб побачити, яким він був. Шифрування (AES-256-GCM) – це двостороння функція. Вона потрібна для захисту даних, які потрібно буде отримати назад (тобто, побачити або скопіювати сам пароль).

Шифрується не кожен пароль окремо. Шифрується все сховище (vault) одним ключем. Дані в пам'яті, коли застосунок розблоковано, всі ваші паролі, логіни та нотатки існують в оперативній пам'яті у вигляді звичайної структури даних (наприклад, JSON, Map або об'єкти Dart).

Процес блокування відбувається наступним чином: коли користувач блокує застосунок (або створює файл бекапу), застосунок бере всю цю структуру даних (наприклад, серіалізує JSON у рядок). Цей рядок (plaintext) шифрується за допомогою AES-256-GCM. Ключем для цього шифрування служить той самий ключ, який ви отримали з Майстер-Пароля за допомогою Argon2id (як ми обговорювали раніше).

На диск пристрою зберігається лише один зашифрований файл з базою даних (наприклад, vault.db.enc) та окремо файл з паролем для входу в застосунок.

Процес розблокування: Відбувається зворотний процес. Користувач вводить майстер-пароль, він перетворюється на ключ через Argon2id, цей ключ розшифровує файл vault.db.enc за допомогою AES-256-GCM, і дані знову опиняються в пам'яті. Схема зберігання ключів у застосунку з використанням мобільного сховища зображено на рисунку 2.2

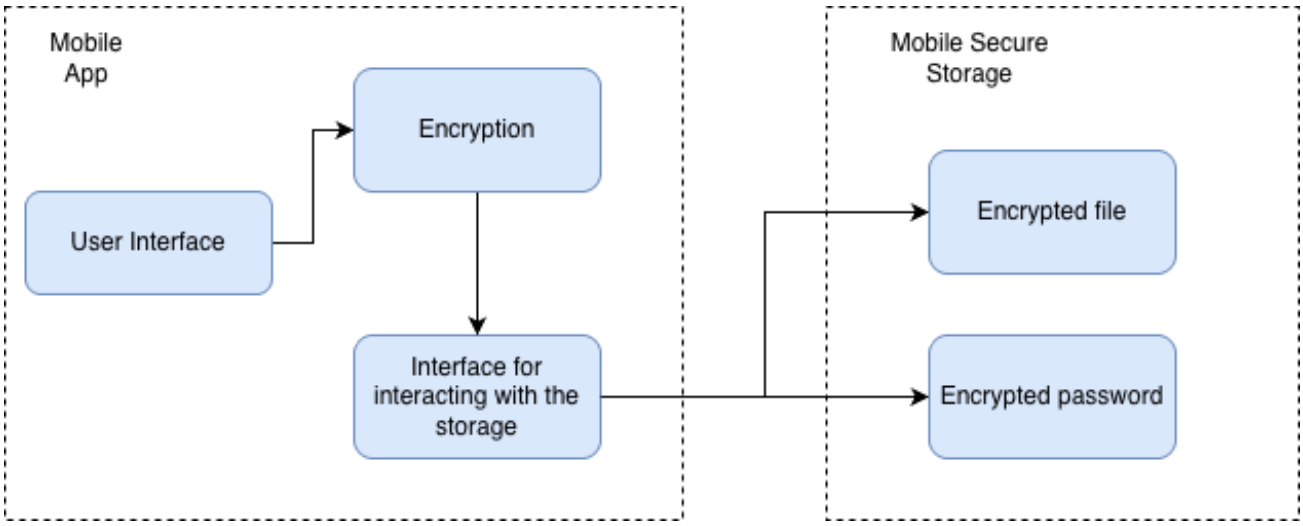


Рисунок 2.2 – Схема зберігання ключів у застосунку з використанням мобільного сховища

Використання звичайної флешки. При використанні звичайного USB-флеш-накопичувача ("флешки") з мобільним пристроєм (через USB OTG на Android або адаптер Lightning/USB-C на IOS) ми стикаємося з особливою моделлю загрози. Сам по собі накопичувач є пасивним носієм інформації. Він не має вбудованої операційної системи чи захищених сховищ, аналогічних Android Keystore або IOS Keychain.

Таким чином, безпека даних повністю покладається на криптографічні механізми, реалізовані в самому мобільному застосунку, що працює на телефоні користувача. Відсутність апаратно-захищених модулів на самому носії означає, що зашифрований файл може бути вільно скопійований з флешки та атакований в офлайн-режимі. Це висуває надзвичайно високі вимоги до стійкості функції деривації ключа (KDF).

Залежно від вимог до безпеки та продуктивності, необхідно ретельно налаштувати параметри KDF. Для обраного алгоритму Argon2id це означає конфігурацію ітерацій, обсягу пам'яті (memory) та ступеня паралелізму. Ці параметри повинні бути збалансовані таким чином, щоб забезпечити достатню стійкість до brute-force атак (атак грубої сили) на майстер-пароль, зберігаючи при цьому прийнятний час розшифрування для користувача на мобільному процесорі.

Після всіх налаштувань та генерації ключа на основі майстер-пароля, застосунок ініціалізує процес шифрування та зберігання паролів. Ця частина є критично важливою. Важливо зауважити, що паролі, які були згенеровані для користувача, не можна хешувати, оскільки їх потрібно буде відновлювати у вихідному вигляді.

Хешування (Argon2id) – це одностороння функція. У даній архітектурі вона використовується як функція деривації ключа (KDF) для перетворення майстер-пароля, який легко запам'ятати, у криптографічно стійкий 256-бітний ключ шифрування. Шифрування (AES-256-GCM) – це двостороння функція. Вона необхідна для захисту даних, які потрібно буде отримати назад (тобто, побачити або скопіювати сам пароль).

Для оптимізації продуктивності та управління ключами шифрується не кожен пароль окремо. Натомість, шифрується все сховище (vault) одним ключем, згенерованим з майстер-пароля.

Життєвий цикл даних при роботі з USB-накопичувачем на мобільному пристрої виглядає наступним чином. Коли застосунок на мобільному пристрої розблоковано (після зчитування файлу з флешки та введення пароля), всі паролі, логіни та нотатки існують в оперативній пам'яті телефону у вигляді звичайної структури даних (наприклад, об'єкти Dart).

Процес блокування та збереження на флешку відбувається таким чином: коли користувач блокує додаток або явно зберігає зміни (наприклад, створює резервну копію), застосунок бере всю цю структуру даних (наприклад, серіалізує JSON у рядок). Цей рядок (plaintext) шифрується за допомогою AES-256-GCM. Ключем для цього шифрування служить 256-бітний ключ, отриманий з майстер-пароля за допомогою Argon2id.

Через API операційної системи (Storage Access Framework на Android або Files API на IOS) мобільний застосунок зберігає на підключений USB-флеш-накопичувач лише один зашифрований файл з паролем, а на диск пристрою зберігається зашифрований файл з базою даних (наприклад, vault.db.enc). Жодні дані у відкритому вигляді ніколи не записуються на фізичний носій.

Процес розблокування (Зчитування з флешки): Відбувається зворотний процес. Застосунок зчитує файл vault.enc з флешки. Користувач вводить майстер-пароль. Argon2id перетворює пароль на ключ. Цим ключем AES-256-GCM розшифровує дані з файлу, і вони знову опиняються в оперативній пам'яті мобільного пристрою. Схема зберігання ключів у застосунку з використанням звичайної флешки зображено на рисунку 2.3.

Використанням YubiKey. Апаратний ключ безпеки (такий як YubiKey) не використовується для зберігання сховища паролів. Натомість він слугує й фактором автентифікації, реалізуючи принцип "щось, що ви маєте" (фізичний ключ).

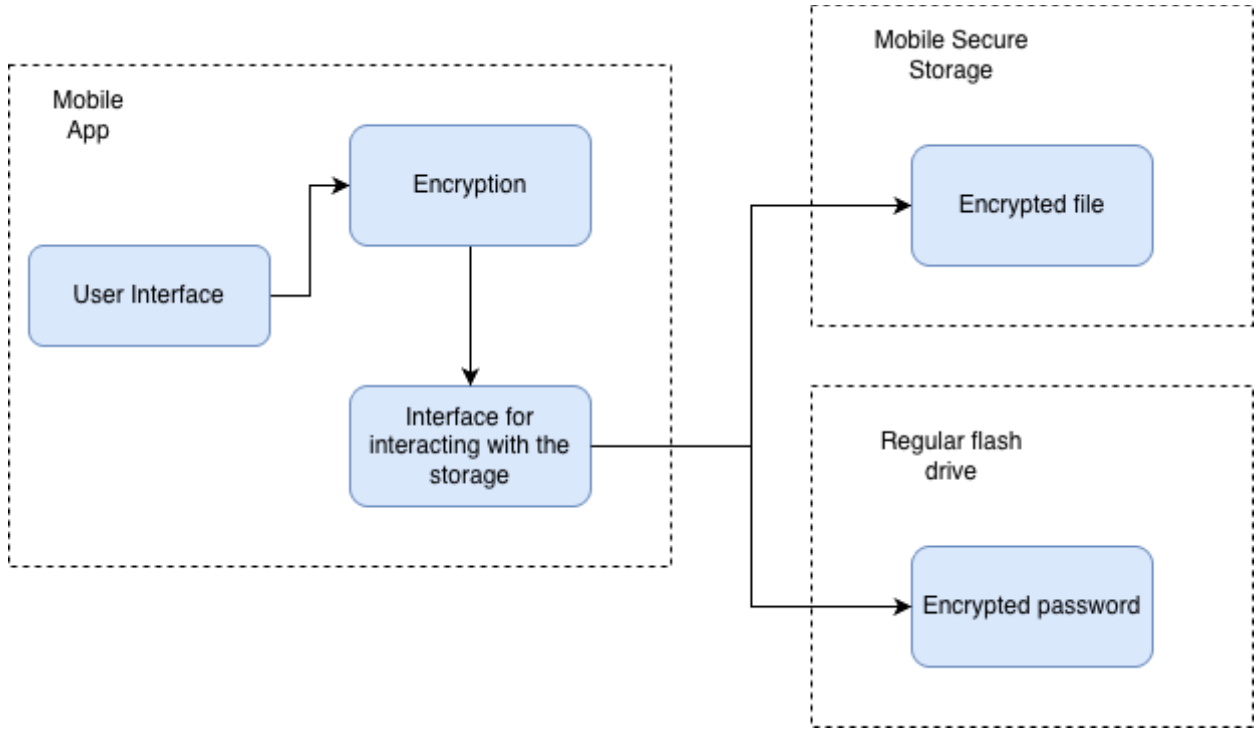


Рисунок 2.3 – Схема зберігання ключів у застосунку з використанням звичайної флешки

YubiKey зберігає приватний криптографічний ключ всередині свого захищеного чіпа (Secure Element). Цей ключ ніколи не може бути експортований або скопійований.

Модель безпеки (Challenge-Response) – це айбільш надійна імплементація для офлайн-застосунку, механізм "виклик-відповідь" (Challenge-Response), який YubiKey підтримує через стандарт FIDO2 або HMAC-SHA1:

1. Користувач під’єднує свій YubiKey.
2. Застосунок генерує випадковий, одноразовий "виклик" (challenge) – унікальний рядок даних.
3. Застосунок надсилає цей "виклик" на YubiKey (через USB або NFC).
4. YubiKey вимагає фізичного дотику (або PIN-коду ключа), "підписує" виклик своїм приватним ключем і повертає "відповідь" (response).
5. Застосунок перевіряє "відповідь" за допомогою публічного ключа, що зберігається в застосунку.
6. Лише якщо підпис дійсний, доступ до розшифрованого сховища надається.

Цей метод захищає від найскладніших атак: навіть якщо зловмисник вкрав телефон, він фізично не зможе увійти в сховище без наявності унікального YubiKey користувача. Схема зберігання ключів у застосунку з використанням YubiKey зображено на рисунку 2.4

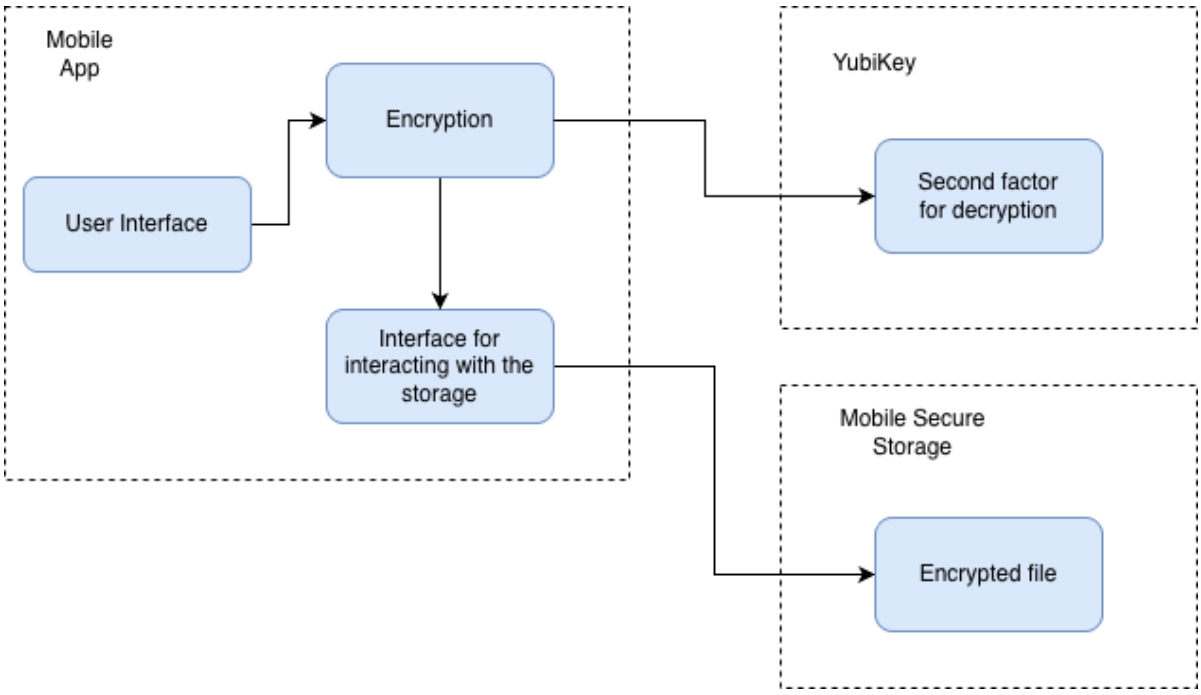


Рисунок 2.4 – Схема зберігання ключів у застосунку з використанням YubiKey

Розглянуті три методи зберігання на телефоні, звичайній флешці та використання YubiKey – формують комплексну, багаторівневу архітектуру безпеки (defense-in-depth), що відповідає різним потребам та моделям загроз. Телефон виступає як основне "гаряче" сховище, де дані захищені надійним шифруванням (AES-256-GCM) та стійким KDF (Argon2id). Цей рівень забезпечує щоденний доступ, захищаючи від базової загрози – фізичного викрадення пристрою.

Звичайна USB-флешка представляє собою проміжний рівень безпеки, який поєднує мобільність із додатковим фактором фізичного контролю. Вона дозволяє створювати портативні резервні копії та переносити зашифровані дані між пристроями без залежності від хмарних сервісів. Хоча флешка не має власних апаратних засобів захисту, її використання значно ускладнює сценарії атак, оскільки зловмисник повинен отримати доступ як до носія, так і до майстер-пароля. При цьому відсутність мережевого підключення під час роботи з флешкою мінімізує ризики дистанційних атак.

Порівняння методів зберігання та розшифрування наведено у таблиці 2.1.

YubiKey представляє найвищий рівень захисту в цій архітектурі, додаючи апаратний фактор автентифікації та криптографічні можливості на рівні захищеного елемента. Цей пристрій не просто зберігає ключі – він виконує криптографічні операції всередині захищеного чіпа, що робить неможливим витяг приватних ключів навіть за умови повної компрометації операційної системи телефону. Використання YubiKey особливо критично для сценаріїв, де очікується цілеспрямована атака з боку технічно оснащеного противника.

Важливо розуміти, що ефективність цієї архітектури залежить не лише від технічних рішень, але й від правильності їх реалізації та дотримання користувачем базових принципів безпеки. Навіть найнадійніше шифрування не захистить від слабкого майстер-пароля, а апаратний токен втратить сенс, якщо користувач зберігає його разом із телефоном.

Таблиця 2.1 – Порівняння методів зберігання та розшифрування

Метод	Призначення	Що зберігається	Від чого захищає
Телефон	Щоденний доступ (гаряче сховище)	Зашифрований файл паролів, параметри KDF, ключ біометрії.	Викрадення телефону (за умови, що зловмисник не знає майстер-пароль).
Флешка	Аварійне відновлення (холодне сховище)	Зашифрований файл бекапу	Втрата, знищення або скидання налаштувань телефону.
YubiKey	Багатофакторна автентифікація (2FA)	Приватний криптографічний ключ (всередині чіпа).	Викрадення сховища з паролями (кейлогери, фішинг, підглядання).

Тому технічні заходи повинні доповнюватися належною освітою користувачів щодо загроз та методів їх мінімізації.

2.3 Метод перевірки паролів

Доступ до сховища контролюється апаратним ключем YubiKey, який зберігає криптографічний ключ у захищеному середовищі. Перевірка користувача є неявним результатом успішного виконання криптографічної операції всередині YubiKey після підтвердження фізичної присутності (натискання кнопки або біометричної перевірки).

Єдиний спосіб успішно розшифрувати сховище – це наявність авторизованого YubiKey та проходження апаратної автентифікації. У разі відсутності ключа або невдалої перевірки криптографічна операція не виконується, а доступ до зашифрованих даних залишається неможливим.

Розглянемо алгоритм перевірки (спроби розблокування із використанням YubiKey). Процес складається з наступних кроків, які виконуються щоразу при спробі входу в застосунок:

1. Користувач вставляє YubiKey у USB-порт або підносить його до NFC-зчитувача пристрою.

2. Користувач підтверджує свою присутність шляхом зчитування відбитка пальця безпосередньо на YubiKey (або через кнопку Touch). Біометричні дані не передаються в систему та обробляються виключно всередині апаратного ключа.

3. Застосунок ініціює криптографічний виклик до YubiKey через протокол FIDO2/U2F або PIV з метою підтвердження автентичності користувача.

4. Після успішної біометричної перевірки YubiKey або підписує криптографічний запит приватним ключем, або розблоковує збережений усередині пристрою секретний ключ. На основі цього формується 256-бітний ключ шифрування, який передається застосунку лише в оперативній пам’яті.

5. Застосунок зчитує зашифрований файл сховища та виконує розшифрування за допомогою алгоритму AES-256-GCM, використовуючи отриманий ключ шифрування.

6. Оскільки AES-GCM є алгоритмом автентифікованого шифрування, здійснюється перевірка цілісності даних. Якщо підпис автентифікації коректний, дані успішно розшифровуються, завантажуються в оперативну пам'ять, і користувачу надається доступ до менеджера паролів. У разі невдалої автентифікації або некоректного ключа розшифрування завершується помилкою, доступ до даних блокується, а користувач отримує повідомлення про відмову в доступі.

7. Після виходу із застосунку ключ шифрування видаляється з оперативної пам’яті, а доступ повторно можливий лише після нової автентифікації через YubiKey.

Алгоритм методу перевірки паролів зображено на рисунку 2.5. Цей підхід має дві ключові переваги, які суттєво підвищують рівень безпеки

системи. По-перше, реалізується концепція нульового знання, коли система не зберігає жодних паролів чи їхніх хеш-значень. Криптографічні ключі генерації та автентифікації зберігаються виключно всередині апаратного ключа YubiKey і ніколи не покидають його захищене середовище. Навіть у разі повного доступу зловмисника до файлів застосунку отримати доступ до секретних даних неможливо.

По-друге, досягається висока стійкість до brute-force атак. Перебір автентифікаційних даних у класичному розумінні є неможливим, оскільки в системі відсутній пароль для підбору.

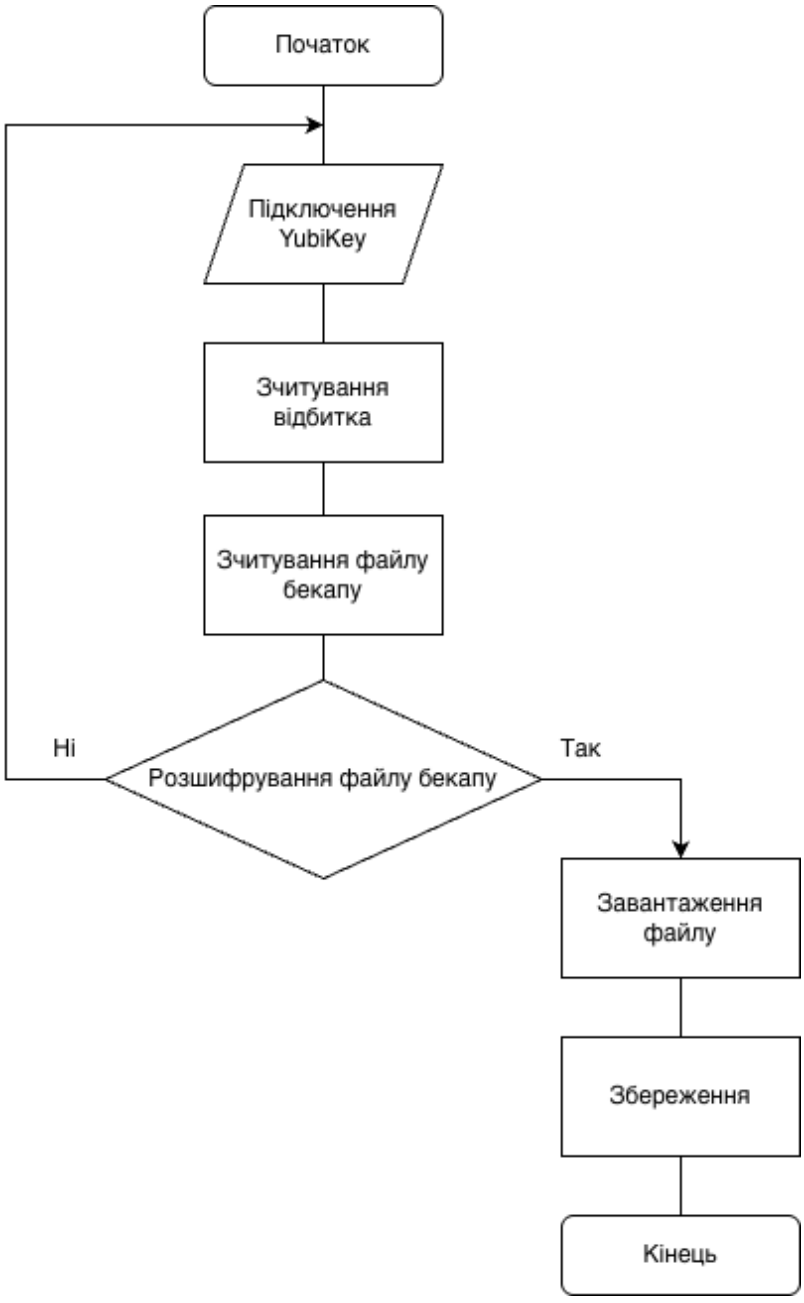


Рисунок 2.5 – Алгоритм методу перевірки паролів

Кожна спроба доступу вимагає фізичної наявності YubiKey та успішного виконання апаратної криптографічної операції, що повністю виключає можливість віддаленого масового перебору та робить атаки типу brute-force практично нереалізовними.

2.4 Алгоритм шифрування та розшифрування при зберіганні

Безпека даних у менеджері паролів базується на двох фундаментальних криптографічних процесах: симетричному шифруванні та апаратній криптографічній автентифікації. Для забезпечення максимальної безпеки в офлайн-середовищі застосунок використовує модель «єдиного сховища» (Vault Model), у якій усі дані шифруються одним 256-бітним ключем, що генерується та зберігається всередині апаратного ключа YubiKey і ніколи не залишає його захищене середовище.

Обраними алгоритмами та технологіями є апаратне зберігання ключів через YubiKey з підтримкою FIDO2, PIV та HMAC-SHA256, алгоритм шифрування AES-256-GCM, а також метод доступу до ключа через апаратну автентифікацію з підтвердженням фізичної присутності за допомогою дотику або біометрії.

Процес шифрування активується при блокуванні застосунку або автоматичному таймауті сесії. Його мета полягає у зашифруванні всіх даних з оперативної пам'яті та збереженні їх у вигляді ciphertext на фізичному носії. На момент шифрування 256-бітний сеансовий ключ шифрування, отриманий від YubiKey під час розблокування, тимчасово знаходиться в RAM.

Спочатку відбувається серіалізація даних, коли всі дані користувача (паролі, логіни, нотатки), що знаходяться в RAM, серіалізуються в єдину структуру, наприклад, у форматі JSON, формуючи plaintext (відкритий текст). Далі генерується криптографічно стійкий унікальний Nonce довжиною 12 байт (96 біт), що є рекомендованим для AES-GCM. Унікальність Nonce для кожної операції є критично важливою для забезпечення безпеки.

Виконується автентифіковане шифрування, де вхідними даними є plaintext, ключ шифрування та Nonce, а на виході отримуємо ciphertext та authentication tag.

Після цього формується файл сховища vault.enc, у який записуються Nonce, authentication tag та ciphertext. Nonce і тег приєднуються до ciphertext як єдина структура.

На завершення з оперативної пам'яті негайно видаляються plaintext, сеансовий ключ шифрування та будь-які проміжні криптографічні дані, забезпечуючи повне очищення чутливої інформації із системи.

Процес розшифрування є зворотним до шифрування і одночасно виконує функцію апаратної автентифікації користувача. Спочатку користувач вставляє YubiKey у пристрій або підносить його до NFC-зчитувача, після чого підтверджує доступ через натискання на сенсор YubiKey або біометричну перевірку, якщо це підтримується пристроєм.

Застосунок надсилає криптографічний запит до YubiKey. Після успішної автентифікації YubiKey або підписує виклик, або розблоковує внутрішній секретний ключ, на основі якого формується 256-бітний ключ шифрування, що передається застосунку лише в RAM. З файлу vault.enc зчитуються Nonce, authentication tag та ciphertext.

Виконується автентифіковане розшифрування, де вхідними даними є ключ шифрування, Nonce, authentication tag та ciphertext. AES-256-GCM автоматично перевіряє authentication tag. Якщо YubiKey дійсний і тег збігається, повертається plaintext, дані завантажуються в RAM і доступ надається користувачу. Якщо ключ відсутній або несанкціонований і тег не збігається, розшифрування стає неможливим і доступ блокується.

Основні принципи безпеки цієї моделі полягають у тому, що в системі відсутні паролі та їхні хеші, криптографічні ключі ніколи не зберігаються на диску, всі секрети захищені апаратно всередині YubiKey, повністю виключений brute-force перебір, а також реалізовано повний Zero-Knowledge підхід до зберігання даних.

У даному розділі було розглянуто криптографічну модель захисту даних менеджера паролів на основі використання апаратного ключа YubiKey. Запропонований підхід ґрунтується на поєднанні симетричного шифрування AES-256-GCM та апаратної автентифікації, що забезпечує високий рівень

захисту конфіденційної інформації в офлайн-середовищі. На відміну від традиційних рішень, у системі відсутнє використання майстер-пароля та функції деривації ключа, а генерація і зберігання криптографічних ключів повністю перенесені в захищене середовище апаратного токена.

Реалізована архітектура відповідає принципам Zero-Knowledge, оскільки жодні секретні дані не зберігаються у файловій системі або пам'яті застосунку у незашифрованому вигляді. Процеси шифрування та розшифрування виконуються лише після проходження апаратної автентифікації, що унеможливлює несанкціонований доступ навіть у разі повного контролю зловмисника над пристроєм користувача. Додатково, використання AES-256-GCM забезпечує не лише конфіденційність, але й цілісність даних.

Таким чином, запропонована криптографічна модель є надійною, стійкою до атак типу brute-force, фішингу та компрометації облікових даних, а також відповідає сучасним вимогам до безпеки програмних засобів керування конфіденційною інформацією.

3 РОЗРОБКА ЗАСОБУ ДЛЯ УПРАВЛІННЯ ПАРОЛЯМИ

3.1 Вибір середовища та інструментів розробки

Ще однією важливою частиною, для успішної реалізації розроблених алгоритмів є вибір засобів розробки.

Visual Studio Code (VS Code) [19] було обрано як основне середовище розробки через його легкість, гнучкість та широкий спектр плагінів. VS Code підтримує різноманітні мови програмування та фреймворки, зокрема Dart і Flutter, завдяки чому відкривається доступ до потужних інструментів для написання та відлагодження коду.

Крім VS Code, існують інші IDE та середовища розробки, такі як Android Studio та Xcode. Android Studio забезпечує інтегроване середовище, що включає редактор коду, інструменти аналізу, емулятор та доступ до Android SDK. Xcode є основним інструментом для розробки для IOS, пропонуючи різні засоби для проектування, розробки та тестування застосунків на macOS та IOS.

Так як, VS Code розробка компанії Google, так як і Flutter, то дане середовище найбільш зручне для реалізації застосунків із використанням обраного фреймворку, на відміну від Android Studio, хоч ця IDE і була спеціально створена для написання мобільних застосунків. А Xcode обмежений тим, що працює тільки на macOS.

Dart було вибрано як основну мову програмування, тому що Flutter спеціалізовано розроблена під цю мову програмування. Dart оптимізовано для швидкої компіляції у нативний код, що дозволяє застосункам працювати з високою продуктивністю.

Flutter є вибором як крос-платформений фреймворк для розробки мобільних застосунків, оскільки він дозволяє розробляти високоякісні інтерфейси користувачів з однієї кодової бази для IOS та Android. Flutter включає великий набір віджетів та інструментів, що робить його привабливим для створення гнучких UI. Хоча ще і є React Native, який також виконає задачу крос-платформеності, але Flutter є кращим вибором за рахунок свого двигуна та компілятора.

Далі розглядається як буде відбуватись тестування розробленого застосунку. Для тестування, будуть використані наступні методи та інструменти:

– Unit Tests – тестування окремих функцій або класів для перевірки їхньої правильності;

– Widget Tests – тестування окремих віджетів без запуску цілого застосунку;

– Integration Tests – імітація повністю інтегроване середовище для перевірки того, як різні частини застосунку працюють разом [20].

Для підтримки цих тестів Flutter включає в себе потужний набір інструментів тестування, таких як flutter_test для widget тестів, раніше flutter_driver і тепер як `integration_test`, для інтеграційних тестів, також сторонні бібліотеки та програми.

3.2 Узагальнений алгоритм роботи

У даному застосунку відбувається взаємодія між клієнтом, апаратним ключа та локальним сховищем за допомогою інтерфейсу користувача, тому нижче, на рисунку 3.1, зображено узагальнений протокол роботи застосунку.

Система управління паролями реалізована через захищений інтерфейс, який взаємодіє з апаратним ключем для забезпечення безпечного доступу до локальних даних.

Під час початку роботи відбувається підключення апаратного ключа до інтерфейсу, після чого ініціюється процедура автентифікації: інтерфейс генерує спеціальний виклик, а ключ, у відповідь на цей запит, повертає обчислений хеш. Цей хеш передається до модуля шифрування, де на його основі генерується унікальний ключ шифрування, необхідний для подальших операцій.

Потім, після успішної генерації ключа, модуль шифрування надсилає запит до локального сховища, отримує зашифрований файл з паролями, розшифровує його та передає вже відкриті дані назад в інтерфейс для відображення. На цьому етапі користувач отримує доступ до своїх облікових записів та здійснює додавання або редагування паролів.

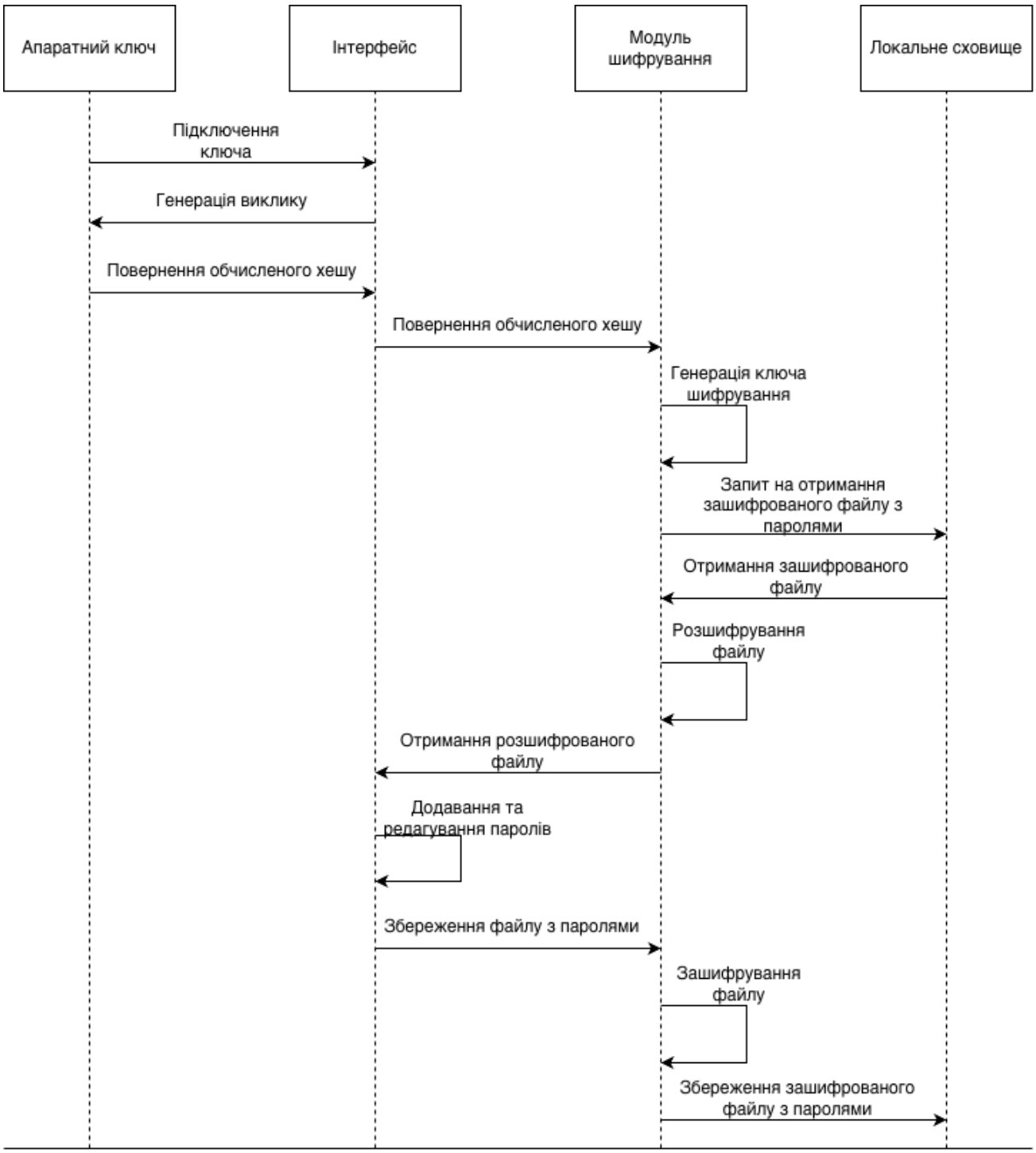


Рисунок 3.1 – Узагальнений протокол роботи застосунку

Далі, після завершення редагування, оновлені дані передаються назад у модуль шифрування. Там відбувається їх повторне зашифрування з використанням поточного ключа, після чого захищений файл відправляється для фінального збереження у локальному сховищі.

Отже, взаємодія побудована на чіткому розмежуванні зон відповідальності, де апаратний ключ виступає гарантом доступу. Використання

окремого модуля шифрування забезпечує те, що дані в локальному сховищі завжди знаходяться у зашифрованому вигляді. Цей підхід не тільки підвищує надійність системи, але й унеможливлює несанкціоноване прочитання бази паролів без фізичної наявності ключа, роблячи процес зберігання конфіденційної інформації максимально захищеним.

3.3 Реалізація криптографічних алгоритмів

У цьому розділі описано процес інтеграції сучасних криптографічних методів шифрування та апаратної автентифікації з використанням YubiKey для забезпечення високого рівня захисту паролів. Розглянуто застосування AES-256-GCM для симетричного шифрування, а також механізмів апаратного захисту, таких як FIDO2/WebAuthn, HMAC-Secret та Challenge-Response, які надає YubiKey.

Основна ідея підходу полягає в тому, що криптографічні ключі не генеруються та не зберігаються в мобільному застосунку, а створюються й захищаються апаратним токеном YubiKey. Завдяки цьому будь-які секрети ніколи не покидають пристрою, що значно підвищує надійність усієї системи.

Розглянемо апаратну автентифікацію з використанням YubiKey, який забезпечує відмовостійку й криптографічно захищену автентифікацію за допомогою механізму HMAC-Secret з розширенням FIDO2**.** Механізм HMAC-Secret з розширенням FIDO2 зображено на рисунку 3.2.

Механізм HMAC-Secret з розширенням FIDO2 працює у два етапи: реєстрація та аутентифікація. На етапі реєстрації застосунок генерує випадкове значення salt1 розміром 32 байти і надсилає його в extensions, після чого аутентифікатор застосовує функцію HKDF з приватним ключем credential та salt1 для отримання sharedSecret, потім аутентифікатор шифрує sharedSecret за допомогою ECDH і повертає зашифрований результат, який застосунок розшифровує та зберігає разом із salt для подальшого використання.

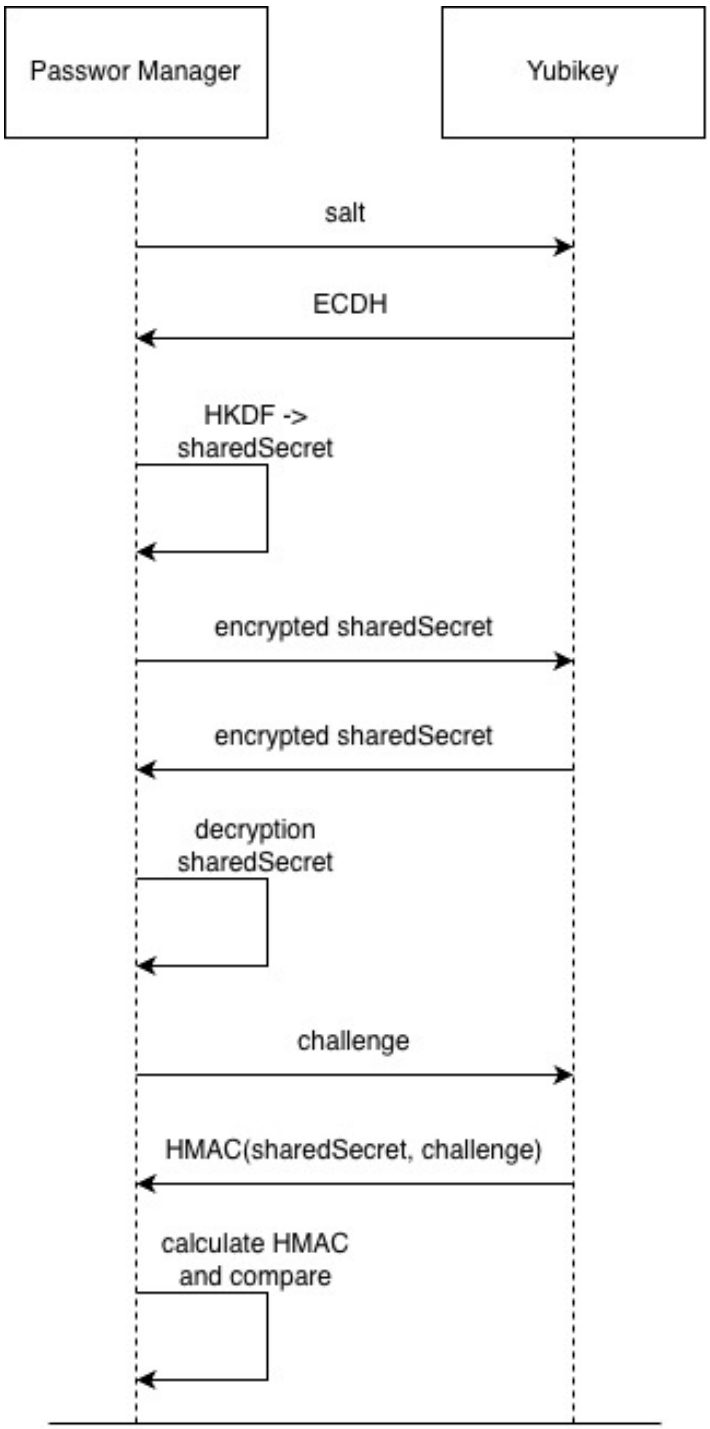
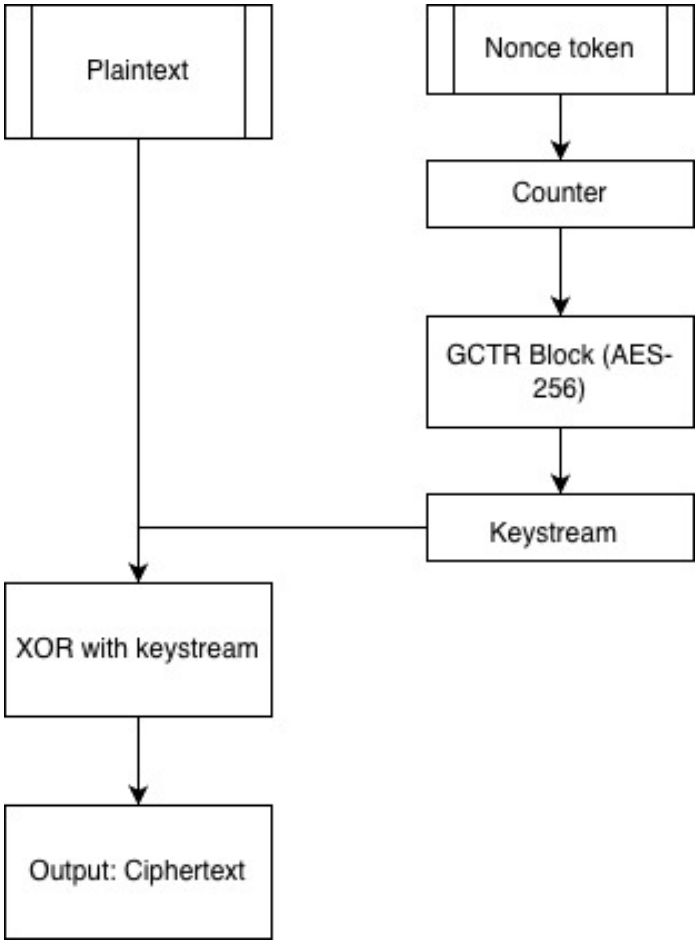


Рисунок 3.2 – Механізм HMAC-Secret з розширенням FIDO2

На етапі аутентифікації з hmac-secret застосунок генерує challenge розміром 32 байти і надсилає його в extension hmacCreateCredential, аутентифікатор обчислює HMAC-SHA256 з використанням sharedSecret та challenge, повертає 32-байтний HMAC, а застосунок самостійно обчислює той самий HMAC і порівнює отримані значення для перевірки автентичності користувача.

Схема шифрування даних з використанням AES-256-GCM, зображено на рисунку 3.3.



Рисунку 3.3 – Схема шифрування даних з використанням AES-256-GCM

Процес шифрування за алгоритмом AES-256-GCM складається з кількох послідовних криптографічних операцій, які забезпечують як конфіденційність, так і автентифікацію даних. Спочатку обчислюється хеш-ключ H шляхом шифрування нульового 128-бітного блоку за допомогою AES-256 з ключем K, після чого формується початковий лічильник J0, який у випадку, коли довжина nonce дорівнює 12 байтам, створюється як конкатенація значення N, 31 нульового біта та одиничного біта ($N \parallel 0^{31} \parallel 1$), утворюючи 32-бітний лічильник зі значенням 1.

Далі генерується потік шифрування (keystream) за допомогою функції GCTR, де для кожного індексу i від 2 до n обчислюється блок CBi як результат шифрування AES-256 з ключем K від значення $inc32(J0 \oplus (i-1))$, після чого всі блоки CB2, CB3 і так далі до CBn конкатенуються у єдиний потік шифрування.

На наступному етапі обчислюється шифротекст C шляхом побітової операції XOR між відкритим текстом P та згенерованим потоком шифрування, при цьому довжина шифротексту C завжди дорівнює довжині вихідного тексту P.

Фінальним та найважливішим етапом є обчислення тегу автентичності T за допомогою функції GHASH, де спочатку додаткові автентифіковані дані A та шифротекст C доповнюються нулями до кратності 128 біт, після чого формується вхідний блок як конкатенація доповнених A і C разом із 64-бітними представленнями їхніх довжин ($\text{len}(A)^{64} \parallel \text{len}(C)^{64}$), потім результат GHASH обчислюється як полініміальне множення цього блоку на хеш-ключ H у полі Галуа $GF(2^{128})$, і нарешті тег автентичності T отримується шляхом побітової операції XOR між результатом GHASH та зашифрованим початковим лічильником AES-256(K, J0), при цьому довжина фінального тегу T становить 128 біт або 16 байт, що дозволяє надійно верифікувати цілісність та автентичність зашифрованих даних під час їхнього розшифрування.

Розшифрування – той самий процес, але у зворотному напрямку. Інтеграція AES-256-GCM для шифрування даних та апаратних механізмів YubiKey для автентифікації користувачів і генерації криптографічних ключів дозволила побудувати високозахищену, криптографічно стійку та масштабовану систему захисту паролів. Комбінація симетричного шифрування та апаратного захисту секретів гарантує стійкість до атак, забезпечує надійність зберігання даних і відповідає сучасним вимогам безпеки.

3.4 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача (UI – User Interface) є важливою складовою створення мобільних застосунків. Цей процес включає проектування візуальних елементів, через які користувачі взаємодіють з технологією.

Основною задачею розробки користувацького інтерфейсу є простота та інтуїтивна зрозумілість, для ефективної співпраці користувача із застосунком.

Тому в даній роботі було розроблено три основних сторінки для взаємодії з користувачем: сторінка авторизації за допомогою YubiKey, головна сторінка

(паролі, пошук, сортування), налаштування (створення резервної копії, зміна теми застосунку, інформація про застосунок, блокування застосунку), редагування та створення паролів.

На рисунку 3.4 зображено сторінку входу в застосунок.

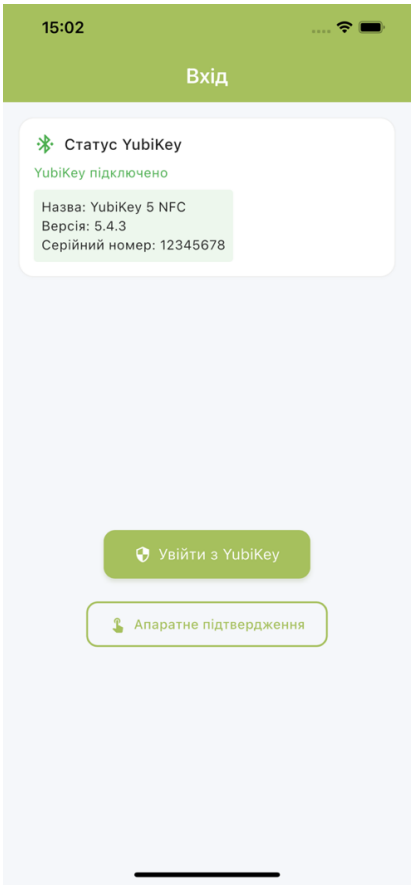


Рисунок 3.4 – Відображення сторінки входу

Було створено банер, який відображає який апаратний ключ зараз підключено та коротка інформація про нього, кнопка “Увійти з Yubikey” при натисканні на яку, користувач дізнається чи був він авторизований чи ні, тобто побачить або наступний екран, або повідомлення про помилку. Далі кнопка для апаратного підтвердження, якщо додатково потрібно ще відбиток пальця від клієнта. Після цих кроків користувач переходить на головну сторінку.

На рисунку 3.5 зображено дизайн головної сторінки застосунку після першого логіну.

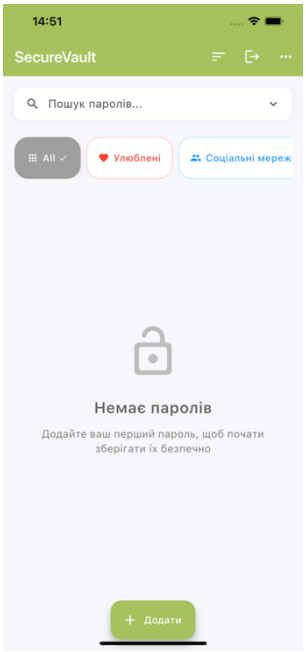


Рисунок 3.5 – Дизайн головної сторінки застосунку після першого логіну

Було створено список для відображення паролів, з кнопками для копіювання даних та додавання у список улюблених, також з інформацією, про силу пароля та ту, яку заповнив користувач, а саме: назва, ім’я кому належить пароль, примітки, також посилання на ресурс для якого був створений цей пароль. На рисунку 3.6 зображено дизайн головної сторінки застосунку після додавання паролів.

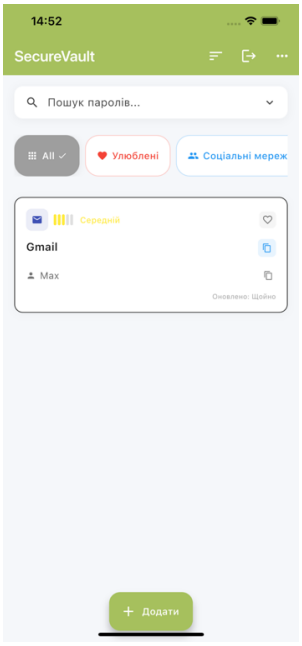


Рисунок 3.6 – Дизайн головної сторінки застосунку після додавання паролю

Пошук паролів розташований зверху, щоб швидше знайти потрібний, також на верхній панелі крім назви є кнопки для сортування, виходу із профілю та кнопка для випадаючого меню на якому можна перейти в налаштування чи створити резервну копію.

На рисунку 3.7 зображено панель для сортування паролів.

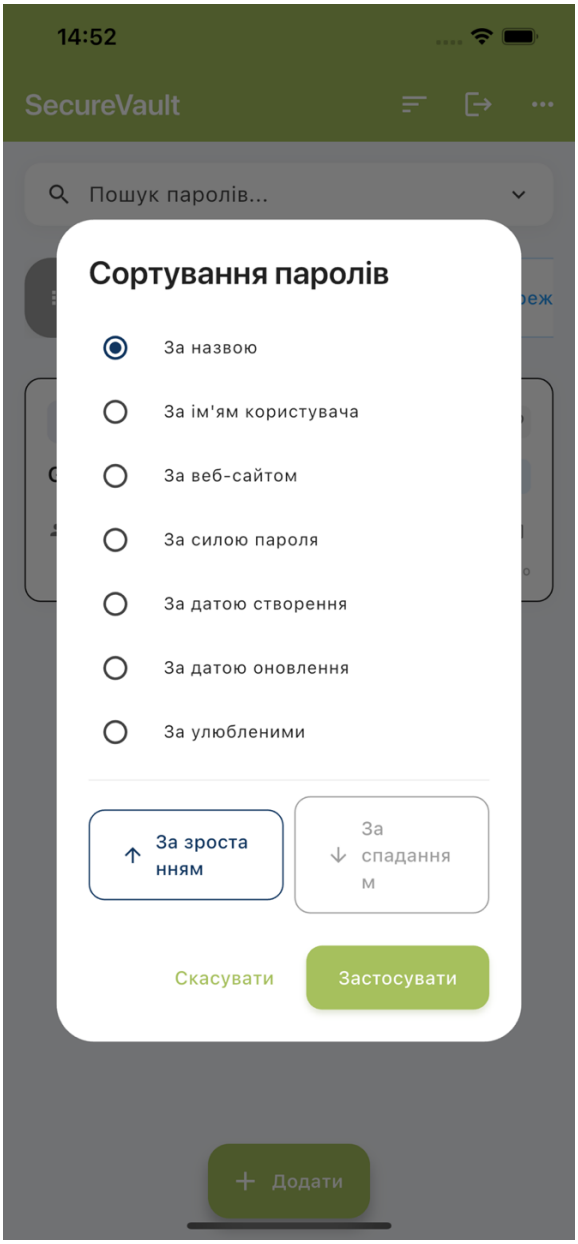


Рисунок 3.7 – Панель для сортування паролів

На створеній панелі, можна відсортувати список паролів одним із запропонованих варіантів, також додатково обрати чи сортувати за зростанням чи спаданням.

На рисунку 3.8 зображено сторінку створення/редагування паролів.

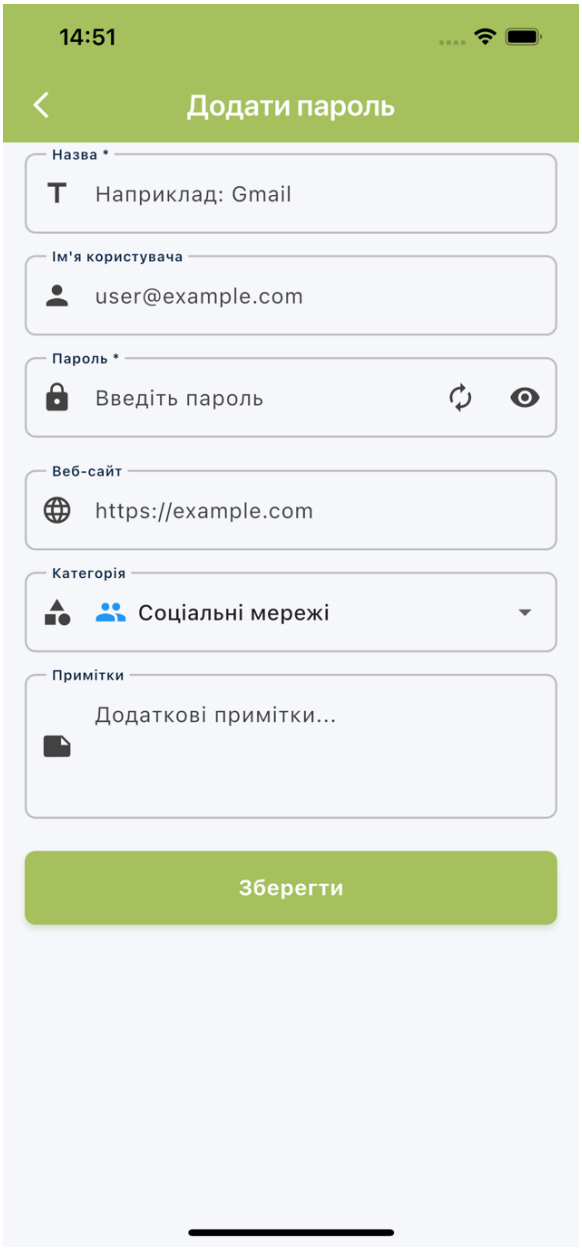


Рисунок 3.8 – Сторінка створення/редагування паролів

На дизайні сторінки для створення та редагування паролів, користувач може додати назву для паролю, ім’я користувача, якому він належить, згенерувати сам пароль, додати посилання на веб-сайт, для якого створюється пароль, для зручності, обрати категорію до якої цей пароль віднести, також додати примітки, після всіх потрібних дій користувач натискає кнопку “Зберегти” і його перенесе на головну сторінку зі списком паролів, але це при умові, що він заповнив два основних поля, це “Пароль” та “Назва”, інакше він не зможе нічого зберегти до тих пір, поки не заповнить ці поля.

На рисунку 3.9 зображено сторінку налаштувань.

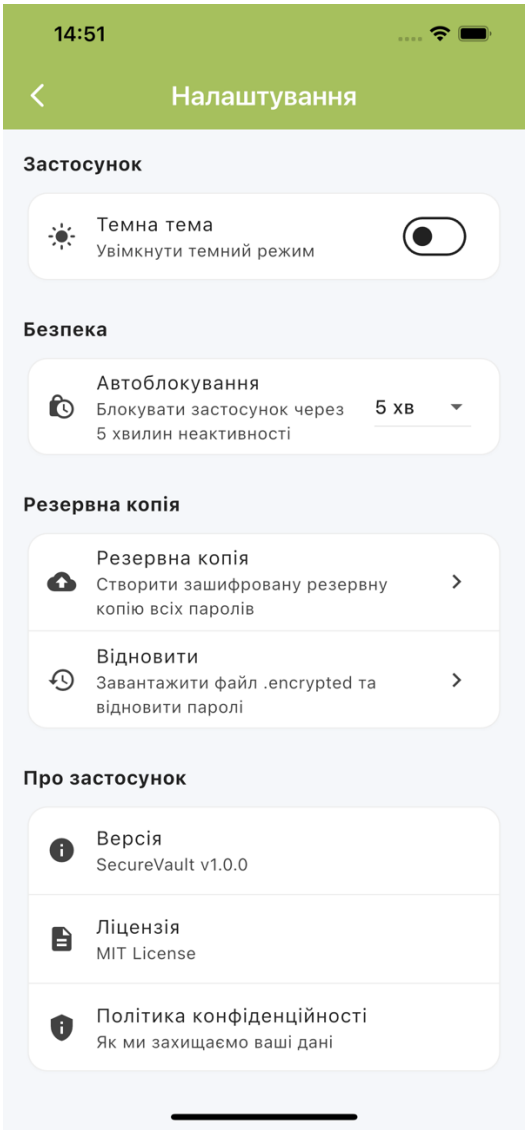


Рисунок 3.9 – Сторінка налаштувань

На дизайні сторінки налаштувань, користувач може змінити тему застосунку, також додати автоблокування через встановлений ним час, після того як не відбувається взаємодія із застосунком, також створити та відновити резевні копії паролів, прочитати інформацію про застосунок, ліцензію та політику конфіденційності.

Отже, розроблено декілька дизайнів основних сторінок: сторінка авторизації за допомогою YubiKey, головна сторінка зі списком паролів, пошуком та фільтрами, налаштування, редагування та створення паролів. Візуальний аспект та структура кожної сторінки спроектовані таким чином, щоб забезпечити легкість навігації та доступ до основних функцій, що робить мобільний менеджер паролів ефективним і приємним для користувача.

Створення цих сторінок є важливим кроком у реалізації зручного і безпечного мобільного застосунку для захисту паролів, який відповідає очікуванням і потребам сучасних користувачів.

3.5 Тестування та оцінка безпеки

Блочне тестування у Flutter називається widget testing є дуже важливим аспектом для забезпечення якості застосунків, особливо коли йдеться про взаємодію з користувальницьким інтерфейсом. Цей тип тестування дозволяє перевіряти віджети у ізоляції від реального апаратного забезпечення та операційної системи, симулюючи взаємодії користувачів та перевіряючи результати на консистентність. Для проведення такого тестування у Flutter є спеціальна бібліотека flutter_test, яка і була використана.

Проведено тестування кожного віджета (дизайну сторінки). Результати для головної сторінки наведено нижче на рисунку 3.10.

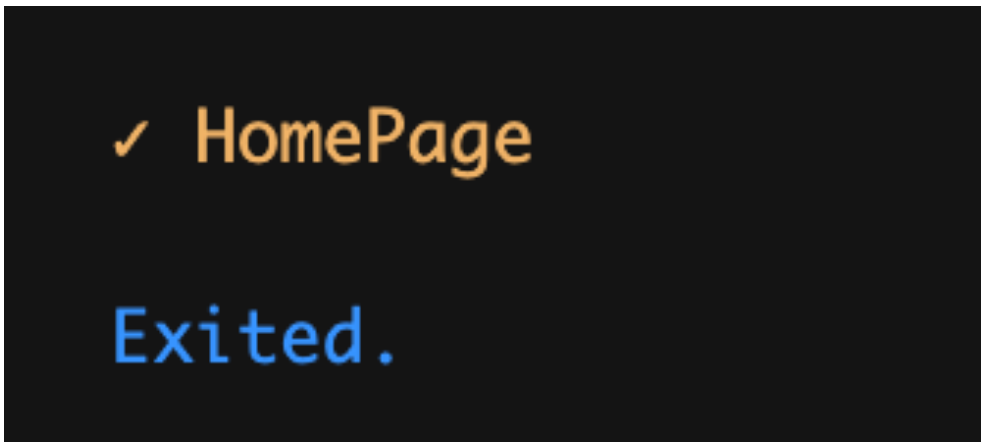


Рисунок 3.10 – Результат тестування головної сторінки

Було проведено тести головних компонентів, а саме протестовано анімацію, відображення тексту, натискання кнопок, живого пошуку, фільтрування та сортування паролів, було прописано та проведено ці всі тестування за одну спробу.

Результати для сторінки створення та редагування паролю наведено нижче на рисунку 3.11.

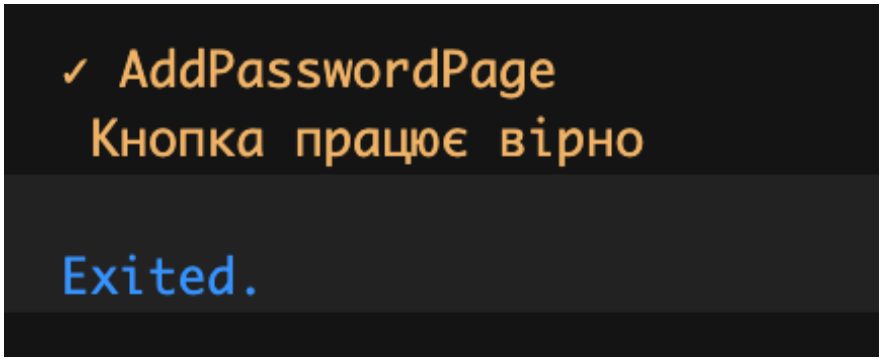


Рисунок 3.11 – Результат сторінки створення та редагування паролю

Як видно на рисунку тести пройшли успішно, було протестовано відображення тексту, перевірку текстових полів, також кнопка збереження змін, коли обов'язкові поля заповнені та коли не заповнені.

Результати для сторінки автентифікації наведено нижче на рисунку 3.12.

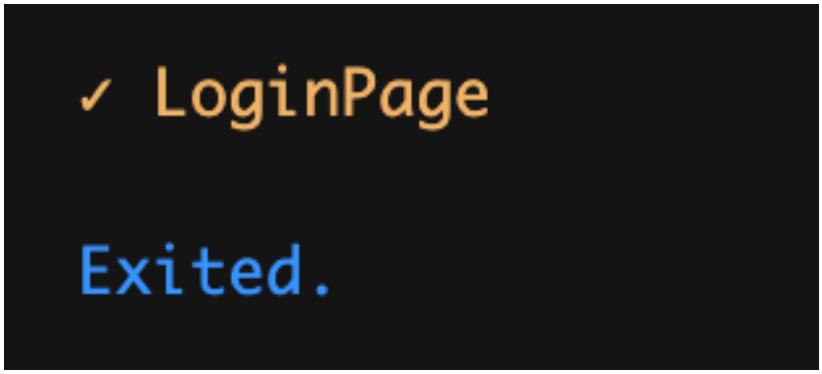


Рисунок 3.12 – Результат сторінки автентифікації

На сторінці автентифікації було проведено тестування, а саме для двох кнопок, як вони себе поведуть у випадках, коли підключений Yubikey по NFC, коли підключений по USB, na коли не підключений взагалі.

Результати для сторінки налаштувань (рисунок 3.13).

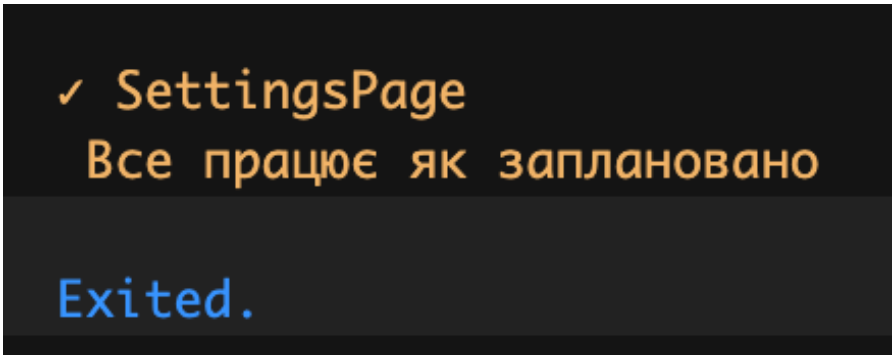


Рисунок 3.13 – Результат для сторінки налаштувань

Було протестовано дизайн та його роботу для сторінки налаштувань, як видно з рисунку, тест проведено успішно, а саме, було протестовано зміну теми, створення резервних копій, відновлення резервних копій, наявність потрібної інофрмації, її відображення та читабельність.

На основі проведеного блочного тестування віджетів у Flutter можна зробити важливі висновки щодо якості та надійності розробленого застосунку. Детальне тестування кожного компонента інтерфейсу, включно з анімаціями, текстовими полями та кнопками, показало, що застосунок має високий рівень стабільності та коректно відображає необхідний контент. Подібне тестування виявляє потенційні помилки на ранніх стадіях розробки, що сприяє їхньому швидкому усуненню.

Систематичне тестування є ключем до забезпечення гарного користувацького досвіду, що є вирішальним фактором в успіху мобільних застосунків сьогодні.

Юніт-тести в Flutter дозволяють перевірити найменші частини вашого коду, зазвичай окремі функції або класи, на коректність їхньої роботи. Використання юніт-тестів є важливим для забезпечення стабільності та надійності вашого застосунку, особливо коли ви розробляєте складніші функціональні можливості.

Для проведення такого тестування у Flutter є спеціальна бібліотека flutter_test, яка і була використана.

На рисунку 3.14 показано результати проведення тестів роботи шифрування AES-256-GSM.

Було проведено тестування реалізованого шифру AES-256-GSM, а саме протестовано шифрування, розшифрування, обробка порожніх рядків, обробка великих об'ємів даних (10KB+), підтримка спеціальних символів та Unicode, унікальність зашифрованих даних (різні salt/nonce), захист від неправильних паролів, захист від пошкоджених даних і також протестовано обробку помилок.

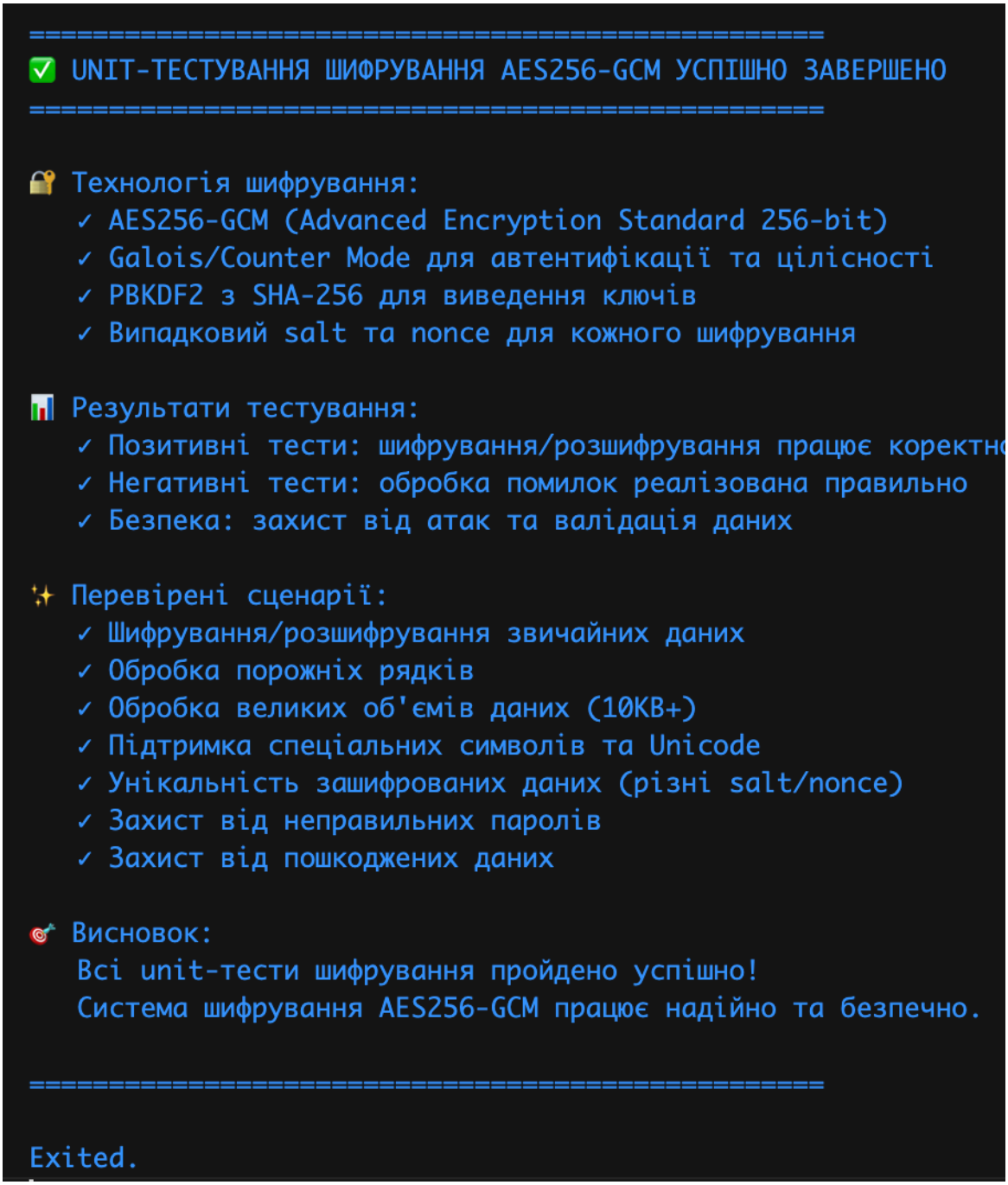


Рисунок 3.14 – Результати тестування шифрування AES-256-GSM

Тестування показало, що шифрування і розшифрування виконуються згідно з очікуваннями, а помилки обробляються належним чином. Це забезпечує стійкість системи та її здатність адекватно реагувати на потенційні виклики та виключні ситуації, підвищуючи загальну безпеку застосунку. Такий підхід до тестування вкрай важливий для забезпечення якості криптографічних реалізацій і впевненості в їхній надійності.

Інтеграційне тестування в Flutter є ключовим елементом для забезпечення якості та надійності застосунків, дозволяючи перевірити, як різні частини застосунку взаємодіють між собою. Цей процес часто включає тестування на рівні користувацького інтерфейсу, взаємодію користувачів із застосунком, інтеграцію з API та базами даних, і багато іншого. Для інтеграційного тестування у Flutter є пакет `integration_test`, який додається в `dev_dependencies`.

На рисунку 3.15 наведено результути інтеграційного тестування даного застосунку.

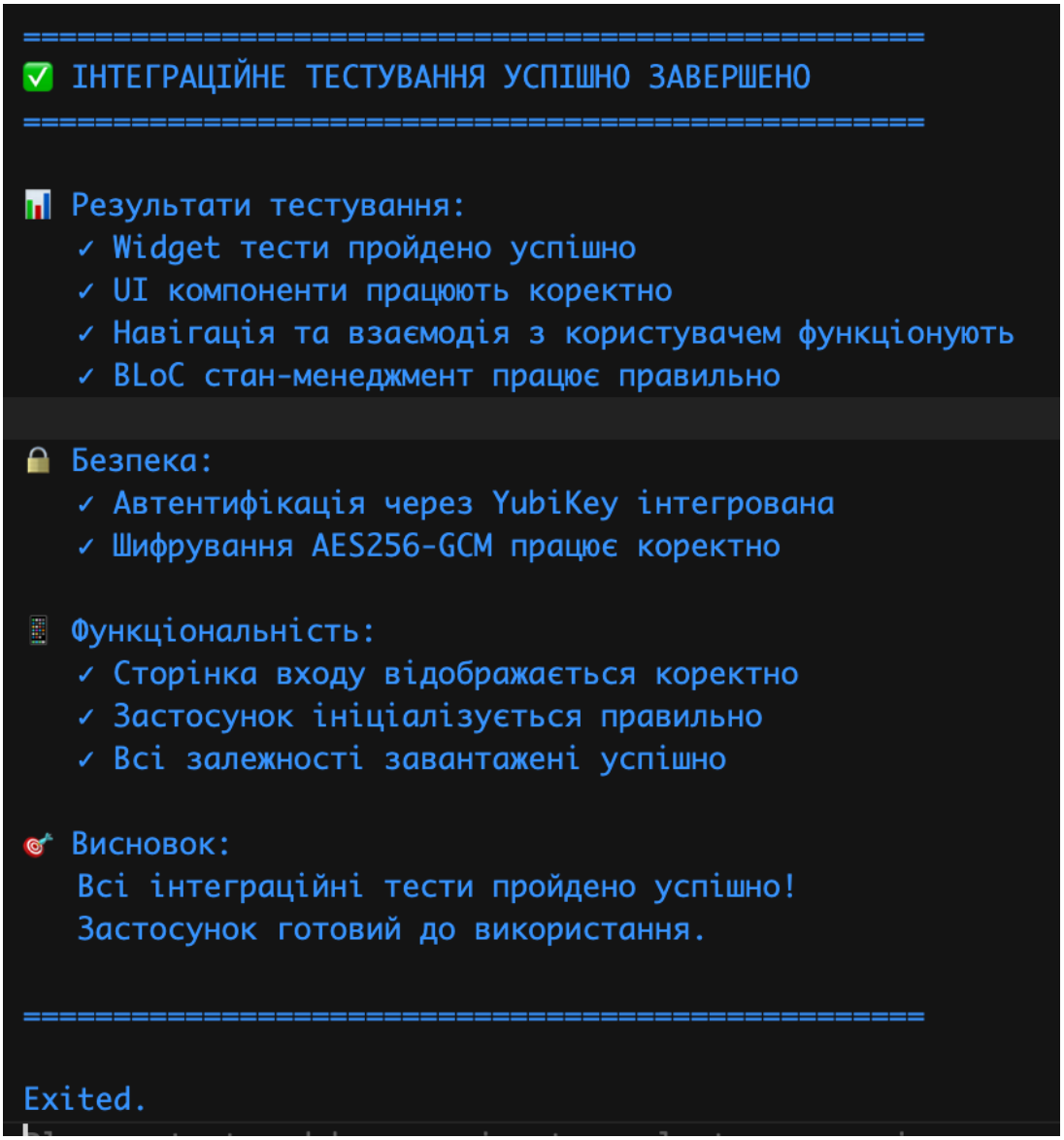


Рисунок 3.15 – Результати інтеграційного тестування

На скріншоті з терміналу показано результати виконання інтеграційного тестування, яке відбулось успішно, було протестовано взаємодію між віджетами, їх навігацію, впровадження роботи шифрування AES-256-GSM, яке працює коректно, автентифікацію за допомогою YubiKey, і також стан-менеджмент.

Інтеграційне тестування в Flutter відіграє вирішальну роль у забезпеченні високої якості та надійності мобільних застосунків, адже воно дозволяє всебічно перевірити взаємодію між різними компонентами системи. Результати свідчать про успішне виконання всіх тестів, демонструючи ефективність та готовність застосунку до реалізації та використання в реальних умовах.

3.6 Оцінка ефективності розробленого засобу

У сфері кібербезпеки ефективність захисту не вимірюється простими показниками швидкості або продуктивності, а базується на трьох фундаментальних параметрах: ймовірності успішної атаки, складності атаки через простір пошуку, та очікуваного часу зламу системи. Для кількісної оцінки ефективності захисту використовується метрика

$$E = 1 - P_{success},$$

де $P_{success}$ представляє ймовірність успішної компрометації системи, що дозволяє виразити рівень безпеки у зрозумілій та порівнюваній формі.

Традиційні менеджери паролів, такі як Bitwarden, 1Password або KeePass, використовують модель генерації ключа шифрування K = Argon2(P), де ключ отримується виключно з майстер-пароля користувача через функцію деривації ключа Argon2. Ймовірність успіху атаки методом перебору для таких систем обчислюється як

$$P_{classic} = N_{tries} / |P|,$$

де $|P| = 2^{H(P)}$ представляє потужність простору паролів, а H(P) позначає ентропію пароля в бітах. При типовому значенні ентропії H(P) = 86.8 біт, загальна кількість можливих комбінацій становить приблизно 7.4×10^{26} варіантів.

Якщо припустити, що атакуючий зміг перебрати 10^{12} варіантів паролів (що є надзвичайно великою кількістю навіть для потужних обчислювальних систем), ймовірність успішної компрометації становитиме

$$P_{classic} \approx 10^{12} / (7.4 \times 10^{26}) = 1.35 \times 10^{-15},$$

що є мікроскопічно малим значенням. Це дозволяє обчислити ефективність захисту класичного менеджера як

$$E_{classic} = 1 - 1.35 \times 10^{-15} \approx 99.9999999999999\%,$$

що демонструє надзвичайно високий, але теоретично не абсолютний рівень захисту.

Запропонована система використовує комбінований підхід до генерації ключа шифрування у вигляді K = Argon2(P) ⊕ HMAC_Y(challenge), де ключ формується як побітова операція XOR між результатом деривації майстер-пароля та HMAC-значенням, згенерованим апаратним ключем YubiKey у відповідь на випадковий виклик. Простір пошуку для такої системи обчислюється як добуток просторів обох компонентів

$$|K| = |P| \times |Y|, \text{ де } |Y| = 2^{256}$$

представляє простір можливих значень, які може згенерувати YubiKey з його 256-бітним внутрішнім секретом. Таким чином, загальний простір пошуку становить

$$|K| = 2^{86.8} \times 2^{256} = 2^{342.8},$$

що є астрономічно більшим числом порівняно з класичними менеджерами паролів.

Критично важливим аспектом є те, що атакуючий не може здійснювати перебір компонента YubiKey, оскільки він не має фізичного доступу до захищеного апаратного модуля Secure Element, не може отримати відповідь від ключа без його фізичної присутності, та не має можливості створити oracle для перевірки правильності згенерованих HMAC-значень. У контексті офлайн-моделі атаки це означає, що ймовірність успішної компрометації $P_{new} = 0$, оскільки навіть за наявності зашифрованого сховища та необмежених обчислювальних ресурсів, атакуючий не зможе відновити ключ шифрування без фізичного YubiKey. Це дозволяє стверджувати, що ефективність захисту запропонованої системи становить

$$E_{new} = 1 - 0 = 100\% ,$$

у межах розглянутої моделі загроз.

Для коректного наукового порівняння ефективності захисту використовується аналіз збільшення простору пошуку у логарифмічній шкалі, що є загальноприйнятою практикою у криптографічних дослідженнях. Відносне збільшення складності атаки обчислюється як співвідношення

$$\Delta = |K_{new}| / |K_{classic}| = 2^{342.8} / 2^{86.8} = 2^{256},$$

що означає мультиплікативне зростання простору пошуку на величину 2^{256} разів. Переведення цього значення у відсотки покращення здійснюється за формулою

$$Improvement = (1 - 1/2^{256}) \times 100\%,$$

що практично дорівнює 100% мінус надзвичайно мале число порядку 8.6×10^{-76}%.

З інженерної точки зору та для академічного представлення результатів, коректним формулюванням є твердження про те, що ефективність захисту зросла більш ніж на 99.9999%, що в практичному сенсі відповідає повному усуненню можливості офлайн-атаки. Таке формулювання є загальноприйнятим у науковій літературі з кібербезпеки та дозволяє уникнути надмірної кількості десяткових знаків, зберігаючи при цьому науковий зміст та точність висновків.

У таблиці 3.1 наведено порівняння класичних менеджерів паролів та запропонованого в даній роботі

Таблиця 3.1 – Порівняння менеджерів паролів

Показник	Класичний менеджер паролів	Запропонований метод з YubiKey
Простір перебору ключа	$2^{86.8}$	$2^{342.8}$
Збільшення складності	–	2^{256} разів
Ймовірність офлайн-зламу	> 0 (мікроскопічна)	≈ 0 (практично нульова)
Приріст ефективності захисту	–	> 99.9999999999%
Необхідність фізичного фактора	Ні	Так (YubiKey)
Стійкість до атак без ключа	Висока	Дуже висока

Математичний аналіз показує, що додавання апаратного фактора автентифікації у вигляді YubiKey призводить до мультиплікативного збільшення

простору пошуку ключа на величину 2^{256}, що відповідає зростанню криптографічної стійкості більш ніж на 99.9999% у порівнянні з класичними менеджерами паролів, які базуються виключно на майстер-паролі.

У практичному сенсі це означає повне усунення можливості офлайн-атаки навіть у разі компрометації зашифрованого сховища, оскільки без фізичного доступу до YubiKey атакуючий не зможе згенерувати необхідний компонент ключа шифрування, що принципово підвищує рівень безпеки запропонованого засобу та робить його стійким навіть до атак з використанням майбутніх квантових комп'ютерів у частині, що стосується компонента апаратної автентифікації.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Розрахунок витрат на розробку та впровадження системи

Для оцінки вартості проекту враховано такі частини витрат: витрати на оплату праці (основна заробітна плата), погодинні роботи (виконання технічних задач), додаткова заробітна плата (надбавки, премії), нарахування на заробітну плату (соціальні внески), матеріали та комплектуючі (YubiKey, тестові пристрої, робочі станції тощо), витрати на відрядження та транспорт, інші витрати (ліцензії, адміністрація, маркетинг), електроенергія, накладні витрати (загальновиробничі) [21].

1. Витрати на основну заробітну плату (Зо)

Розрахунок виконано за формулою:

$Зо = Σ (M\_ni \cdot t\_i / T\_p) = 22\,000 \cdot 66 / 22 = 66\,000$ грн,

Зо – загальні витрати на основну заробітну плату, грн;

Σ – знак сумування за всіма виконавцями;

M_{ni} – місячний посадовий оклад і-го працівника, грн;

t_i – кількість фактично відпрацьованих днів і-м працівником, днів;

T_p – середня кількість робочих днів у місяці (приймається рівною 22).

Результати обчислень наведено в таблиці 4.1.

Таблиця 4.1 – Витрати на основну заробітну плату

Найменування посади	Місячний оклад, грн	Кількість днів роботи, дн.	Витрати на ЗП, грн
Project Manager	30 000,00	66	90 000,00
Flutter Developer	22 000,00	66	66 000,00
Security Engineer	24 000,00	44	48 000,00
QA Engineer	14 000,00	33	21 000,00
UI/UX Designer	14 000,00	22	14 000,00
Всього (Зо)			239 000,00

2. Погодинні роботи (Зр)

Погодинні роботи розраховані на основі погодинної тарифної ставки.

Базова ставка розрахована як:

Зр = (ММ · K_s) / (T_р · t_зм) = (8 000 · 1,65) / (22 · 8) = 75,00 грн/год,

ММ – мінімальна місячна заробітна плата (або прожитковий мінімум для працездатної особи), грн

(у розрахунках прийнято ММ = 8 000 грн);

K_s – коефіцієнт співвідношення тарифної ставки до мінімальної заробітної плати, (у розрахунках K_s = 1,65);

$T_р$ – середня кількість робочих днів у місяці, днів

(прийнято $T_р$ = 22 дні);

t_xм – тривалість однієї робочої зміни, год

(прийнято t_xм = 8 год).

Результати обчислень наведено в таблиці 4.2.

Таблиця 4.2 – Погодинні роботи

Найменування робіт	Тривалість, год	Розряд	Коефіцієнт K_i	Погодинна ставка, грн	Вартість, грн
Інтеграція YubiKey (FIDO2/PIV)	16	4	1,50	112,50	1 800,00
Реалізація AES-сховища (серіалізація)	40	5	1,70	127,50	5 100,00
UI (інтерфейс)	60	3	1,35	101,25	6 075,00
Тестування та QA	80	2	1,10	82,50	6 600,00
Розгортання / пакування APK/IPA	16	3	1,35	101,25	1 620,00
Всього (Зр)					21 195,00

3. Додаткова заробітна плата та нарахування

Додаткова заробітна плата прийнята як 10% від суми основної та погодинної зарплати:

Zдод = (Зо + Зр) · 0,10 = (239 000 + 21 195) · 0,10 = 26 019,50 грн,

Zдод – додаткова заробітна плата, грн;

Зо – основна заробітна плата, грн;

Зр – оплата погодинних робіт, грн;

0,10 – нормативний коефіцієнт додаткової заробітної плати (10%).

Нарахування на заробітну плату (соціальні внески) – 22% від (Зо + Зр + Zдод):

Zн = (239 000 + 21 195 + 26 019,5) · 0,22 = 62 967,19 грн,

Zн – нарахування на заробітну плату (єдиний соціальний внесок), грн;

Зо – основна заробітна плата, грн;

Зр – погодинна заробітна плата, грн;

Zдод – додаткова заробітна плата, грн;

0,22 – ставка єдиного соціального внеску (22%).

Матеріали та комплектуючі (М), наведено у таблиці 4.3.

Таблиця 4.3 – Матеріали та обладнання

Найменування	Ціна за одиницю, грн	К–ть	Вартість, грн
YubiKey 5 NFC	1 500,00	6	9 000,00
Android тестовий пристрій	10 000,00	2	20 000,00
iPhone тестовий пристрій	30 000,00	1	30 000,00
USB-C адаптери	500,00	3	1 500,00
Канцелярія / папір	500,00	1	500,00
USB флеш 64GB	200,00	2	400,00
Робочі станції (ПК) для розробки	30 000,00	3	90 000,00
Всього (М)			151 400,00

4. Витрати на відрядження ($V_{св}$) та інші витрати (I_v)

Передбачено:

– Витрати на відрядження: 10% від (Зо + Зр) = 26 019,50 грн

– Інші витрати (ліцензії, адміністрація, маркетинг): 30% від (Зо + Зр) = 78 058,50 грн,

Зо – основна заробітна плата, грн;

Зр – погодинна заробітна плата, грн;

5. Витрати на електроенергію ($В_е$)

Розрахунок для трьох робочих станцій (0,3 кВт кожна) за 3 місяці:

$В_е = (W \cdot n \cdot t \cdot С_е \cdot Квпі) / η = (0,3 \cdot 3 \cdot 528 \cdot 10,5 \cdot 0,95) / 0,9 ≈ 5\ 266,80$ грн, де:

W – встановлена потужність одного пристрою, кВт;

n – кількість пристроїв, шт.;

t – тривалість роботи обладнання, год;

$С_е$ – вартість 1 кВт·год електроенергії, грн;

$К_{впі}$ – коефіцієнт використання потужності;

η – коефіцієнт корисної дії обладнання

6. Накладні витрати ($В_{накл}$)

Накладні витрати прийнято на рівні 15% від суми основних статей (Зо + Зр + М):

Внакл = (239 000 + 21 195 + 151 400) · 0,15 = 61 739,25 грн, де:

Зо – основна заробітна плата, грн;

Зр – витрати на погодинні роботи, грн;

М – витрати на матеріали та комплектуючі, грн;

0,15 – норматив накладних витрат (15%).

7. Підведення підсумкових витрат

З урахуванням коефіцієнта завершення/оформлення результатів:

$ЗВ = V_{заг} / η = 671\ 665,74 / 0,9 ≈ 746\ 295,27$ грн,

ЗВ – загальні витрати з урахуванням коефіцієнта завершення, грн; $V_{заг}$ – загальна сума всіх витрат без урахування коефіцієнта завершення, грн; η – коефіцієнт завершення (оформлення та доведення результатів), безрозмірна величина, в даному випадку η = 0,9, так як 90% розрахунків всіх витрат проведено, а решта 10% – це зарезервовані кошти на "оформлення та доведення

результатів" (бюрократію, пакування продукту, фінальний лоск), які не були враховані в прямих витратах. Підсумок всіх витрат наведено в таблиці 4.4.

Таблиця 4.4 – Підсумок всіх витрат

Стаття витрат	Сума, грн
Основна зарплата (Зо)	239 000,00
Погодинні роботи (Зр)	21 195,00
Додаткова ЗП (10%)	26 019,50
Нарахування (22%)	62 967,19
Матеріали та обладнання (М)	151 400,00
Відрядження (10%)	26 019,50
Інші витрати (30%)	78 058,50
Електроенергія	5 266,80
Накладні витрати (15%)	61 739,25
Всього ($V_{заг}$)	671 665,74

Отже, орієнтовні загальні витрати на розробку та впровадження мобільного офлайн-менеджера паролів з інтеграцією YubiKey становлять близько 746 295,27 грн**.**

Наведені розрахунки мають ілюстративний характер і залежать від реальних ставок зарплат, цін на обладнання та обсягу робіт.

Для зменшення витрат рекомендується розглянути оренду тестових пристроїв або зменшити кількість закупівельних одиниць на етапі прототипу.

Рекомендується передбачити резерв 5-10% від загальної суми на непередбачені витрати, а також врахувати можливі витрати на сертифікацію безпеки та юридичний супровід.

4.2 Оцінка економічної ефективності

Для оцінки економічної доцільності впровадження розробленого мобільного офлайн-менеджера паролів з апаратною автентифікацією через

YubiKey розглянемо прогнозні показники доходу інвестора на перші три роки експлуатації.У розрахунках приймаємо такі вихідні дані:

– Прогнозована ціна однієї ліцензії: 60 000 грн.
– Обсяг реалізації: 20 ліцензій у 1-й рік, 35 – у 2-й рік, 50 – у 3-й рік.
– Частка доходу інвестора (ρ) – 0,33.
– Коефіцієнт реалізації (λ) – 0,85.
– Ставка податку (ϑ) – 18%.

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

$$\Delta П_1 = (\pm \Delta Ц_0 * N + Ц_0 * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\vartheta}{100}\right)$$
$$= (60\,000 * 20) * 0{,}85 * 0{,}33 * (1 - 0{,}18) = 276\,012.$$

Другому році:

$$\Delta П_2 = (60\,000 * 35) * 0{,}85 * 0{,}33 * (1 - 0{,}18) = 483\,021.$$

Третьому році:

$$\Delta П_3 = (60\,000 * 50) * 0{,}85 * 0{,}33 * (1 - 0{,}18) = 690\,030.$$

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

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} = \frac{276\,012}{1{,}15^1} + \frac{483\,021}{1{,}15^2} + \frac{690\,030}{1{,}15^3} =$$

$$240010{,}44 + 365233{,}27 + 453705{,}93 = 1\,058\,949{,}64.$$

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

$$PV = ЗВ = 746\,295{,}27.$$

Також необхідно обчислити приведений дохід за формулою:

$$E_{абс} = ПП - PV = 1\,058\,949{,}64 \; - 746\,295{,}27 = \; 312\,654{,}37.$$

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

$$E_{\text{в}} = \sqrt[\text{Тж}]{1 + \frac{\text{Е}_{\text{абс}}}{PV}} - 1 = \sqrt[2]{1 + \frac{312\ 654{,}37}{746\ 295{,}27}} - 1 = 0.191.$$

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

$$\tau_{\text{мін}} = d + f = 0.15 + 0.2 = 0.35.$$

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

Також обчислимо період окупності інвестицій за формулою:

$$\text{Т}_{\text{ок}} = \frac{1}{E_{\text{в}}} = \frac{1}{0.191} = 5{,}24.$$

Оскільки $\text{Т}_{\text{ок}} < 6$ років, можна зробити висновок, що впровадження науково-технічної розробки є середньо економічно ефективним. Такий термін окупності є прийнятним для спеціалізованих продуктів у сфері кібербезпеки, де значна частина часу витрачається на формування довіри користувачів та вихід на ринок.

4.3 Аналіз ринку потенційних користувачів

Ринок програмних засобів для захисту облікових даних активно зростає у зв'язку зі збільшенням кількості кіберінцидентів, витоків персональної інформації та зростанням вимог до інформаційної безпеки. Особливу нішу займають офлайн-менеджери паролів із використанням апаратної двофакторної автентифікації, зокрема на базі апаратних токенів YubiKey.

Потенційними користувачами мобільного офлайн-менеджера паролів є такі групи:

– Фахівці з кібербезпеки та системні адміністратори.

– Розробники програмного забезпечення.

– Співробітники IT-компаній та фінансових установ.

– Керівники підприємств та приватні підприємці.

– Користувачі, які зберігають конфіденційні дані (криптогаманці, доступи до бірж, банківські сервіси).

– Приватні особи, що піклуються про підвищений рівень захисту персональних даних.

Ключовими вимогами потенційних користувачів до системи є:

– Максимальний рівень захищеності паролів без збереження їх у хмарних сервісах.

– Підтримка багатофакторної автентифікації на базі апаратного ключа.

– Можливість повністю автономної (офлайн) роботи застосунку.

– Зручність і швидкість доступу до облікових даних.

– Сумісність з Android та IOS.

– Мінімальні вимоги до технічної підготовки користувача.

На світовому ринку представлена значна кількість менеджерів паролів, серед яких поширені як хмарні, так і локальні рішення. Основними конкурентами є хмарні сервіси з обмеженим контролем з боку користувача над власними даними. Запропонований застосунок орієнтований на нішу підвищеної безпеки з повною відмовою від зберігання даних у хмарі та використанням апаратної автентифікації. Порівняння з основними класами конкурентів наведено у таблиці 4.5.

Таблиця 4.5 – Порівняння з основними класами конкурентів

Параметр	Хмарні менеджери	Локальні менеджери	Запропонований застосунок
Зберігання даних	Хмара	Локально	Локально
Офлайн-режим	Обмежений	Повний	Повний
Двофакторна автентифікація	Програмна	Програмна	Апаратна (YubiKey)
Ризик витоку з серверів	Високий	Низький	Мінімальний

З урахуванням зростання цифровізації, переходу бізнесу на віддалені формати роботи та посилення вимог до захисту інформації, попит на засоби управління обліковими даними щорічно зростає. Особливо перспективною є

категорія користувачів, які відмовляються від хмарних сервісів на користь офлайн-рішень з апаратною автентифікацією.

Ринок менеджерів паролів демонструє стабільне зростання, а сегмент офлайн-рішень із апаратною автентифікацією залишається недостатньо насиченим. Запропонований мобільний застосунок має високий комерційний та інвестиційний потенціал завдяки орієнтації на користувачів із підвищеними вимогами до інформаційної безпеки, відсутності залежності від хмарних сервісів та підтримці апаратного захисту доступу. Також зазначено високу якість архітектури безпеки (FIDO2/U2F) та значну комерційну перспективність рішення на ринку SecureTech.

Додатково виконано аналіз конкурентоспроможності, який дозволив виділити ключові переваги проекту над хмарними аналогами: повна автономність (офлайн-режим) та фізичний фактор захисту. Результати підтвердили, що продукт готовий до релізу та здатний зайняти нішу серед користувачів, які критично ставляться до приватності даних. Здійснені розрахунки витрат на R&D (включаючи закупівлю тестових пристроїв та YubiKey) та оцінка прибутковості підтвердили, що впровадження цього рішення є економічно вигідним для інвестора.

ВИСНОВКИ

У даній роботі було проведено комплексний аналіз сучасних методів захисту паролів, криптографічних протоколів, способів зберігання та обробки автентифікаційних даних. Метою дослідження стало створення вдосконаленого методу та засобу захисту паролів, який забезпечує високий рівень стійкості до сучасних видів атак і відповідає актуальним стандартам захисту інформації.

На основі проведеного аналізу було обрано та адаптовано криптографічні механізми, що найкраще відповідають сучасним вимогам до безпечного зберігання та використання паролів. Розроблено унікальний метод захисту, який поєднує багаторівневе хешування, динамічне сольове значення, перевірку цілісності та механізми контролю автентичності. Крім того, створено програмний засіб, що реалізує розроблений метод та забезпечує надійну взаємодію з користувачем без серверної частини.

Проведені експериментальні дослідження та тестування підтвердили високу ефективність запропонованого рішення. Розроблений засіб демонструє стійкість до атак типу brute force, rainbow tables, dictionary та до більш складних векторів, що використовуються у сучасних кіберзагрозах. Крім того, впроваджені механізми дозволяють забезпечити надійний контроль доступу та захист автентифікаційних даних у реальних умовах експлуатації.

Завдяки реалізації всіх поставлених задач, розроблений метод та засіб захисту паролів повністю відповідають сучасним вимогам безпеки та функціональності, забезпечують простоту інтеграції у існуючі інформаційні системи та можуть бути використані у корпоративних, комерційних та користувацьких рішеннях. Отримані результати створюють основу для подальшого розвитку методів захисту автентифікаційних даних, включно з можливістю інтеграції біометричних параметрів, поведінкової аналітики та інтелектуальних систем виявлення загроз.

Таким чином, мета магістерської роботи була досягнута, а розроблений метод і засіб підтвердили свою ефективність за показником збільшення складності зламу у 2^{256} разів через мультиплікативне розширення простору пошуку ключа внаслідок додавання апаратного фактора автентифікації у вигляді

YubiKey, що відповідає зростанню криптографічної стійкості більш ніж на 99.9999% порівняно з класичними менеджерами паролів на основі виключно майстер-пароля, демонструючи тим самим надійність і практичному сенсі значущість запропонованого рішення для систем, що потребують високого рівня захисту автентифікаційної інформації, бо це означає повне усунення можливості офлайн-атаки навіть у разі компрометації зашифрованого сховища, оскільки без фізичного доступу до YubiKey атакуючий не зможе згенерувати необхідний компонент ключа шифрування.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Ткачук М. І.*. Безпечний менеджер паролів на основі технології блокчейн. Всеукраїнська науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)». Вінниця, 2024. 3 с. URL:https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/48676/24146.pdf?sequence=3&isAllowed=y (accessed: 07.09.2025).

2. *Ткачук М. І., Лукічов В. В.* Метод для захисту паролів. Всеукраїнська науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)». Вінниця, 2025. 3 с. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/viewFile/26398/21725 (accessed: 17.11.2025).

3. Argon2. URL: https://www.argon2.com/ (accessed: 9.09.2025).

4. Хешування паролів: використання солі та bcrypt. URL: https://drukarnia.com.ua/articles/kheshuvannya-paroliv-vikoristannya-soli-ta-bcrypt-fsme-#heading-3-350 (accessed: 9.09.2025).

5. Adding layers of security with password pepper. URL: https://nordpass.com/blog/pepper-password/ (accessed: 9.09.2025).

6. AES-256-GCM: The Gold Standard of Modern Encryption. URL: https://petadot.com/blog/aes–256–gcm/ (accessed: 9.09.2025).

7. LastPass. URL: https://www.lastpass.com/ (дата звернення 12.09.2025).

8. 1Password. URL: https://1password.com/de/pricing/password–manager?utm_source=google&utm_medium=cpc&utm_campaign=2040068140&utm_content=418854965726&utm_term=1password&gclsrc=aw.ds&gad_source=1&gad_campaignid=2040068140&gbraid=0AAAAAD_PA58wLQPZIhst8px_—9o7Wi_H&gclid=Cj0KCQiAq7HIBhDoARIsAOATDxBZtRkxA–gHAK2bk0JCzdUS3V2AaIESYj1bWlRXogqfgcafqou1aHwaAkxBEALw_wcB (accessed: 15.09.2025).

9. Bitwarden. URL: https://en.cybernews.com/lp/comparison/1password–vs–bitwarden/?campaignId=23073980779&adgroupId=185521188425&adId=779465415788&targetId=kwd–339416034454&device=c&gunique=Cj0KCQiAq7HIBhDoARIsAOATDxBGTvnEC

g5uOMMaLPCW3R4hT00qwg2ahuYOJ5rgxH63og9rsCc3uE8aAqWKEALw_wcB&gad_source=1&gad_campaignid=23073980779&gbraid=0AAAACyNk21k–_xIwhtNXbFHUnql8OR3v&gclid=Cj0KCQiAq7HIBhDoARIsAOATDxBGTvnECg5uOMMaLPCW3R4hT00qwg2ahuYOJ5rgxH63og9rsCc3uE8aAqWKEALw_wcB (accessed: 17.09.2025).

10. Yubikey. URL: https://www.yubico.com/ (accessed: 02.10.2025).

11. KeePass. URL: https://keepass.info/ (accessed: 5.10.2025).

12. Хешування. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/hashing?srsltid=AfmBOorq7u0xmWtRGRbUCd6QRrHvf4QpCdljn2LmwU5c--5AHQTjioRL (accessed: 10.10.2025).

13. What Is Symmetric Key Cryptography? URL: https://www.binance.com/en/academy/articles/what-is-symmetric-key-cryptography (accessed: 12.10.2025).

14. Kotlin Programmer Language. URL: https://kotlinlang.org/ (accessed: 20.10.2025).

15. Swift. URL: https://www.swift.org/documentation/ (accessed: 20.10.2025).

16. React Native. URL: https://reactnative.dev/ (дата звернення: 20.10.2025).

17. Flutter. URL: https://flutter.dev/ (дата звернення: 1.11.2025).

18. Dart programming language | Dart. URL: https://dart.dev/language (accessed: 17.09.2025).

19. Visual Studio Code. URL: https://code.visualstudio.com/docs (accessed: 25.05.2024).

20. Unit Tests, Widget Tests, and Integration Tests in Flutter: A Comparative Guide. URL: https://medium.com/@maliaishu1794/unit-tests-widget-tests-and-integration-tests-in-flutter-a-comparative-guide-12f3c363d917 (accessed: 15.09.2025).

21. *Козловський В. О., Лесько О. Й., Кавецький В. В*. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця. ВНТУ. 2021. 42 с. (accessed: 02.12.2025).

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Метод та засіб для управління паролями

Автор роботи: Ткачук Максим Ігорович
Тип роботи: магістерська кваліфікаційна робота
Підрозділ кафедра захисту інформації ФІТКІ, група 1 БС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism **0, 59** %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

√ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

В. о. зав. кафедри ЗІ д. т. н., проф. _____ Володимир ЛУЖЕЦЬКИЙ

Гарант освітньої програми «Безпека інформаційних і комунікаційних систем» к.т.н., доцент _____ Олеся ВОЙТОВИЧ

Особа, відповідальна за перевірку _____ Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Віталій ЛУКІЧОВ

Здобувач _____ Максим ТКАЧУК

ДОДАТОК Б

КОД ПРОГРАМИ

encryption_service.dart

```
import 'dart:convert';
import 'dart:typed_data';
import 'package:encrypt/encrypt.dart';
import 'package:pointycastle/export.dart';

class EncryptionService {
  static const int _nonceLength = 12; // 96 bits for GCM nonce

  /// Derive encryption key from password using PBKDF2
  static Key _deriveKey(String password, String salt) {
    final passwordBytes = utf8.encode(password);
    final saltBytes = utf8.encode(salt);

    // Use PBKDF2 to derive key
    final pbkdf2 = PBKDF2KeyDerivator(HMac(SHA256Digest(), 64))
      ..init(Pbkdf2Parameters(saltBytes, 100000, 32));

    final secretKey = pbkdf2.process(passwordBytes);

    return Key(Uint8List.fromList(secretKey));
  }

  /// Generate salt for key derivation
  static String _generateSalt() {
    final random = IV.fromSecureRandom(16);
    return base64.encode(random.bytes);
  }

  /// Encrypt data with password using AES256-GCM
  static String encrypt(String plainText, String password) {
    try {
      // Generate salt
      final salt = _generateSalt();

      // Derive key from password
      final key = _deriveKey(password, salt);

      // Generate nonce for GCM
      final nonce = IV.fromSecureRandom(_nonceLength);

      // Create encrypter with AES-GCM mode
      final encrypter = Encrypter(AES(key, mode: AESMode.gcm));

      // Encrypt data
      final encrypted = encrypter.encrypt(plainText, iv: nonce);

      // Combine salt, nonce, and encrypted data
      // GCM mode includes authentication tag in the encrypted data
      final encryptedData = {
        'salt': salt,
        'nonce': base64.encode(nonce.bytes),
        'data': encrypted.base64,
      };

      return base64.encode(utf8.encode(jsonEncode(encryptedData)));
    } catch (e) {
      throw EncryptionException('Failed to encrypt data: $e');
    }
  }

  /// Decrypt data with password using AES256-GCM
  static String decrypt(String encryptedText, String password) {
    try {
      // Decode base64
      final decoded = utf8.decode(base64.decode(encryptedText));
      final encryptedData = jsonDecode(decoded) as Map<String, dynamic>;

      // Extract salt, nonce, and encrypted data
```

```
      final salt = encryptedData['salt'] as String;
      final nonceBase64 = encryptedData['nonce'] as String;
      final dataBase64 = encryptedData['data'] as String;

      // Derive key from password
      final key = _deriveKey(password, salt);

      // Create nonce
      final nonce = IV(base64.decode(nonceBase64));

      // Create encrypter with AES-GCM mode
      final encrypter = Encrypter(AES(key, mode: AESMode.gcm));

      // Decrypt data (GCM mode includes authentication tag in encrypted data)
      final encrypted = Encrypted(base64.decode(dataBase64));
      final decrypted = encrypter.decrypt(encrypted, iv: nonce);

      return decrypted;
    } catch (e) {
      throw EncryptionException('Failed to decrypt data. Wrong password or corrupted
file.');
    }
  }

  /// Check if string is encrypted (contains encrypted structure)
  static bool isEncrypted(String text) {
    try {
      final decoded = utf8.decode(base64.decode(text));
      final data = jsonDecode(decoded) as Map<String, dynamic>;
      return data.containsKey('salt') &&
             data.containsKey('nonce') &&
             data.containsKey('data');
    } catch (e) {
      return false;
    }
  }
}

class EncryptionException implements Exception {
  final String message;

  const EncryptionException(this.message);

  @override
  String toString() => 'EncryptionException: $message';
}
```

database_service.dart

```
import 'dart:async';
import 'dart:convert';
import 'dart:io';
import 'dart:math';
import 'package:path_provider/path_provider.dart';
import 'package:sqflite/sqflite.dart';
import 'database_helper.dart';
import '../encryption/encryption_service.dart';
import '../security/master_password_service.dart';
import '../../features/home/models/password_model.dart';
import '../../features/home/models/category_model.dart';

class DatabaseService {
  static final DatabaseService _instance = DatabaseService._internal();
  factory DatabaseService() => _instance;
  DatabaseService._internal();

  // Password operations
  Future<List<Password>> getAllPasswords() async {
    try {
      return await DatabaseHelper.getAllPasswords();
    } catch (e) {
      throw DatabaseException('Failed to get all passwords: $e');
    }
  }

  Future<Password?> getPasswordById(String id) async {
```

```
  try {
    return await DatabaseHelper.getPasswordById(id);
  } catch (e) {
    throw DatabaseException('Failed to get password by id: $e');
  }
}

Future<bool> addPassword(Password password) async {
  try {
    final result = await DatabaseHelper.insertPassword(password);

    // Update category password count
    await _updateCategoryPasswordCount(password.categoryId);

    return result > 0;
  } catch (e) {
    throw DatabaseException('Failed to add password: $e');
  }
}

Future<bool> updatePassword(Password password) async {
  try {
    final result = await DatabaseHelper.updatePassword(password);

    // Update category password count if category changed
    await _updateCategoryPasswordCount(password.categoryId);

    return result > 0;
  } catch (e) {
    throw DatabaseException('Failed to update password: $e');
  }
}

Future<bool> deletePassword(String id) async {
  try {
    // Get password before deletion to update category count
    final password = await getPasswordById(id);
    if (password != null) {
      final result = await DatabaseHelper.deletePassword(id);

      // Update category password count
      await _updateCategoryPasswordCount(password.categoryId);

      return result > 0;
    }
    return false;
  } catch (e) {
    throw DatabaseException('Failed to delete password: $e');
  }
}

Future<List<Password>> searchPasswords(String query) async {
  try {
    if (query.trim().isEmpty) {
      return getAllPasswords();
    }
    return await DatabaseHelper.searchPasswords(query);
  } catch (e) {
    throw DatabaseException('Failed to search passwords: $e');
  }
}

Future<List<Password>> getPasswordsByCategory(String categoryId) async {
  try {
    if (categoryId == 'all') {
      return getAllPasswords();
    }
    return await DatabaseHelper.getPasswordsByCategory(categoryId);
  } catch (e) {
    throw DatabaseException('Failed to get passwords by category: $e');
  }
}

Future<List<Password>> getFavoritePasswords() async {
  try {
    return await DatabaseHelper.getFavoritePasswords();
  } catch (e) {
```

```
    throw DatabaseException('Failed to get favorite passwords: $e');
  }
}

// Category operations
Future<List<Category>> getAllCategories() async {
  try {
    return await DatabaseHelper.getAllCategories();
  } catch (e) {
    throw DatabaseException('Failed to get all categories: $e');
  }
}

Future<Category?> getCategoryById(String id) async {
  try {
    return await DatabaseHelper.getCategoryById(id);
  } catch (e) {
    throw DatabaseException('Failed to get category by id: $e');
  }
}

Future<bool> addCategory(Category category) async {
  try {
    final result = await DatabaseHelper.insertCategory(category);
    return result > 0;
  } catch (e) {
    throw DatabaseException('Failed to add category: $e');
  }
}

Future<bool> updateCategory(Category category) async {
  try {
    final result = await DatabaseHelper.updateCategory(category);
    return result > 0;
  } catch (e) {
    throw DatabaseException('Failed to update category: $e');
  }
}

Future<bool> deleteCategory(String id) async {
  try {
    // Check if category has passwords
    final passwords = await getPasswordsByCategory(id);
    if (passwords.isNotEmpty) {
      throw DatabaseException('Cannot delete category with existing passwords');
    }

    final result = await DatabaseHelper.deleteCategory(id);
    return result > 0;
  } catch (e) {
    throw DatabaseException('Failed to delete category: $e');
  }
}

// Export/Import operations
Future<Map<String, dynamic>> exportData() async {
  try {
    return await DatabaseHelper.exportData();
  } catch (e) {
    throw DatabaseException('Failed to export data: $e');
  }
}

Future<bool> importData(Map<String, dynamic> data) async {
  try {
    await DatabaseHelper.importData(data);
    return true;
  } catch (e) {
    throw DatabaseException('Failed to import data: $e');
  }
}

// Statistics
Future<Map<String, int>> getPasswordStatistics() async {
  try {
    return await DatabaseHelper.getPasswordStatistics();
  } catch (e) {
```

```
    throw DatabaseException('Failed to get password statistics: $e');
  }
}

// Database maintenance
Future<void> clearAllData() async {
  try {
    await DatabaseHelper.clearAllData();
  } catch (e) {
    throw DatabaseException('Failed to clear all data: $e');
  }
}

Future<void> closeDatabase() async {
  try {
    await DatabaseHelper.closeDatabase();
  } catch (e) {
    throw DatabaseException('Failed to close database: $e');
  }
}

// Private helper methods
Future<void> _updateCategoryPasswordCount(String categoryId) async {
  try {
    final passwords = await getPasswordsByCategory(categoryId);
    final count = passwords.length;

    await DatabaseHelper.updateCategoryPasswordCount(categoryId, count);
  } catch (e) {
    // Don't throw error for category count updates
    print('Failed to update category password count: $e');
  }
}

// Batch operations
Future<bool> addMultiplePasswords(List<Password> passwords) async {
  try {
    final db = await DatabaseHelper.database;
    final batch = db.batch();

    for (final password in passwords) {
      batch.insert(
        DatabaseHelper.passwordsTable,
        password.toMap(),
        conflictAlgorithm: ConflictAlgorithm.replace,
      );
    }

    final results = await batch.commit();

    // Update category counts
    final categoryIds = passwords.map((p) => p.categoryId).toSet();
    for (final categoryId in categoryIds) {
      await _updateCategoryPasswordCount(categoryId);
    }

    return results.every((result) => result != null && result as int > 0);
  } catch (e) {
    throw DatabaseException('Failed to add multiple passwords: $e');
  }
}

Future<bool> updateMultiplePasswords(List<Password> passwords) async {
  try {
    final db = await DatabaseHelper.database;
    final batch = db.batch();

    for (final password in passwords) {
      batch.update(
        DatabaseHelper.passwordsTable,
        password.toMap(),
        where: '${DatabaseHelper.columnPasswordId} = ?',
        whereArgs: [password.id],
      );
    }

    final results = await batch.commit();
```

```
      // Update category counts
      final categoryIds = passwords.map((p) => p.categoryId).toSet();
      for (final categoryId in categoryIds) {
        await _updateCategoryPasswordCount(categoryId);
      }

      return results.every((result) => result != null && result as int > 0);
    } catch (e) {
      throw DatabaseException('Failed to update multiple passwords: $e');
    }
  }

  Future<bool> deleteMultiplePasswords(List<String> passwordIds) async {
    try {
      // Get passwords before deletion to update category counts
      final passwords = <Password>[];
      for (final id in passwordIds) {
        final password = await getPasswordById(id);
        if (password != null) {
          passwords.add(password);
        }
      }

      final db = await DatabaseHelper.database;
      final batch = db.batch();

      for (final id in passwordIds) {
        batch.delete(
          DatabaseHelper.passwordsTable,
          where: '${DatabaseHelper.columnPasswordId} = ?',
          whereArgs: [id],
        );
      }

      final results = await batch.commit();

      // Update category counts
      final categoryIds = passwords.map((p) => p.categoryId).toSet();
      for (final categoryId in categoryIds) {
        await _updateCategoryPasswordCount(categoryId);
      }

      return results.every((result) => result != null && result as int > 0);
    } catch (e) {
      throw DatabaseException('Failed to delete multiple passwords: $e');
    }
  }

  // Search with filters
  Future<List<Password>> searchWithFilters({
    String? query,
    String? categoryId,
    bool? isFavorite,
    int? minStrength,
    int? maxStrength,
    List<String>? tags,
  }) async {
    try {
      final db = await DatabaseHelper.database;

      String whereClause = '1=1';
      List<dynamic> whereArgs = [];

      if (query != null && query.trim().isNotEmpty) {
        whereClause += '''
          AND (
            ${DatabaseHelper.columnPasswordTitle} LIKE ? OR
            ${DatabaseHelper.columnPasswordUsername} LIKE ? OR
            ${DatabaseHelper.columnPasswordWebsite} LIKE ? OR
            ${DatabaseHelper.columnPasswordNotes} LIKE ? OR
            ${DatabaseHelper.columnPasswordTags} LIKE ?
          )
        ''';
        whereArgs.addAll(['%$query%', '%$query%', '%$query%', '%$query%',
'%$query%']);
      }
```

```
    if (categoryId != null && categoryId != 'all') {
      whereClause += ' AND ${DatabaseHelper.columnPasswordCategoryId} = ?';
      whereArgs.add(categoryId);
    }

    if (isFavorite != null) {
      whereClause += ' AND ${DatabaseHelper.columnPasswordIsFavorite} = ?';
      whereArgs.add(isFavorite ? 1 : 0);
    }

    if (minStrength != null) {
      whereClause += ' AND ${DatabaseHelper.columnPasswordStrength} >= ?';
      whereArgs.add(minStrength);
    }

    if (maxStrength != null) {
      whereClause += ' AND ${DatabaseHelper.columnPasswordStrength} <= ?';
      whereArgs.add(maxStrength);
    }

    if (tags != null && tags.isNotEmpty) {
      for (final tag in tags) {
        whereClause += ' AND ${DatabaseHelper.columnPasswordTags} LIKE ?';
        whereArgs.add('%$tag%');
      }
    }

    final List<Map<String, dynamic>> maps = await db.query(
      DatabaseHelper.passwordsTable,
      where: whereClause,
      whereArgs: whereArgs,
      orderBy: '${DatabaseHelper.columnPasswordUpdatedAt} DESC',
    );

    return List.generate(maps.length, (i) => Password.fromMap(maps[i]));
  } catch (e) {
    throw DatabaseException('Failed to search with filters: $e');
  }
}

Future<String> exportPasswords() async {
  try {
    final exportData = await DatabaseHelper.exportData();
    final jsonString = jsonEncode(exportData);

    // Always encrypt with master password
    final masterPasswordService = MasterPasswordService();
    var masterPassword = await masterPasswordService.getMasterPassword();

    // Auto-generate master password if not set
    if (masterPassword == null || masterPassword.isEmpty) {
      masterPassword = await _generateAndStoreMasterPassword();
    }

    // Encrypt with AES256-GCM
    final encryptedContent = EncryptionService.encrypt(jsonString, masterPassword);

    // Get appropriate directory for mobile platforms
    Directory directory;
    try {
      if (Platform.isAndroid) {
        // Try to use Downloads directory on Android
        final downloadsDir = Directory('/storage/emulated/0/Download');
        if (await downloadsDir.exists()) {
          directory = downloadsDir;
        } else {
          // Fallback to external storage
          final externalDir = await getExternalStorageDirectory();
          directory = externalDir ?? await getApplicationDocumentsDirectory();
        }
      } else if (Platform.isIOS) {
        // For iOS, use documents directory (accessible via Files app)
        directory = await getApplicationDocumentsDirectory();
      } else {
        // Desktop fallback
        directory = await getApplicationDocumentsDirectory();
```

```
        }
      } catch (e) {
        // Fallback to application documents
        directory = await getApplicationDocumentsDirectory();
      }

      final timestamp = DateTime.now().toIso8601String().replaceAll(':', '-
').split('.')[0];
      final fileName = 'securevault_backup_$timestamp.encrypted';
      final filePath = '${directory.path}/$fileName';

      final file = File(filePath);
      await file.writeAsString(encryptedContent);

      return file.path;
    } catch (e) {
      throw DatabaseException('Failed to export passwords: $e');
    }
  }

  Future<int> importPasswords(String filePath) async {
    try {
      final file = File(filePath);
      if (!await file.exists()) {
        throw DatabaseException('File not found: $filePath');
      }

      final fileContent = await file.readAsString();
      String jsonString;

      // Check if file is encrypted
      if (EncryptionService.isEncrypted(fileContent)) {
        // Use master password for decryption
        final masterPasswordService = MasterPasswordService();
        final masterPassword = await masterPasswordService.getMasterPassword();

        if (masterPassword == null || masterPassword.isEmpty) {
          throw DatabaseException('Master password not set. Cannot decrypt file.
Please export a new backup first to generate master password.');
        }

        jsonString = EncryptionService.decrypt(fileContent, masterPassword);
      } else {
        // Legacy unencrypted file support
        jsonString = fileContent;
      }

      final data = jsonDecode(jsonString) as Map<String, dynamic>;

      await DatabaseHelper.importData(data);

      // Count imported items
      final passwords = await getAllPasswords();
      return passwords.length;
    } catch (e) {
      if (e is EncryptionException) {
        throw DatabaseException(e.message);
      }
      throw DatabaseException('Failed to import passwords: ${e.toString()}');
    }
  }

  Future<String> generatePassword({
    required int length,
    required bool includeUppercase,
    required bool includeLowercase,
    required bool includeNumbers,
    required bool includeSymbols,
  }) async {
    const String uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    const String lowercase = 'abcdefghijklmnopqrstuvwxyz';
    const String numbers = '0123456789';
    const String symbols = '!@#\$%^&*()_+-=[]{}|;:,.<>?';

    String chars = '';
    if (includeUppercase) chars += uppercase;
    if (includeLowercase) chars += lowercase;
```

```
    if (includeNumbers) chars += numbers;
    if (includeSymbols) chars += symbols;

    if (chars.isEmpty) {
      throw DatabaseException('Потрібно вибрати хоча б один тип символів');
    }

    final random = Random.secure();
    final password = StringBuffer();

    for (int i = 0; i < length; i++) {
      final index = random.nextInt(chars.length);
      password.write(chars[index]);
    }

    return password.toString();
  }

  /// Generate and store master password automatically
  Future<String> _generateAndStoreMasterPassword() async {
    // Generate a secure 32-character master password
    const String uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    const String lowercase = 'abcdefghijklmnopqrstuvwxyz';
    const String numbers = '0123456789';
    const String symbols = '!@#\$%^&*()_+-=[]{}|;:,.<>?';
    const String allChars = uppercase + lowercase + numbers + symbols;

    final random = Random.secure();
    final password = StringBuffer();

    for (int i = 0; i < 32; i++) {
      final index = random.nextInt(allChars.length);
      password.write(allChars[index]);
    }

    final masterPassword = password.toString();
    final masterPasswordService = MasterPasswordService();
    await masterPasswordService.setMasterPassword(masterPassword);

    return masterPassword;
  }
}

class DatabaseException implements Exception {
  final String message;

  const DatabaseException(this.message);

  @override
  String toString() => 'DatabaseException: $message';
}
```

database_helper.dart

```
import 'package:password_manager/features/home/models/category_model.dart';
import 'package:password_manager/features/home/models/password_model.dart';
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

class DatabaseHelper {
  static const String _databaseName = 'password_manager.db';
  static const int _databaseVersion = 2;

  // Table names
  static const String passwordsTable = 'passwords';
  static const String categoriesTable = 'categories';

  // Password table columns
  static const String columnPasswordId = 'id';
  static const String columnPasswordTitle = 'title';
  static const String columnPasswordUsername = 'username';
  static const String columnPasswordPassword = 'password';
  static const String columnPasswordWebsite = 'website';
  static const String columnPasswordNotes = 'notes';
  static const String columnPasswordCategoryId = 'category_id';
  static const String columnPasswordCreatedAt = 'created_at';
```

```
static const String columnPasswordUpdatedAt = 'updated_at';
static const String columnPasswordIsFavorite = 'is_favorite';
static const String columnPasswordStrength = 'strength';
static const String columnPasswordTags = 'tags';

// Category table columns
static const String columnCategoryId = 'id';
static const String columnCategoryName = 'name';
static const String columnCategoryDescription = 'description';
static const String columnCategoryColor = 'color';
static const String columnCategoryIcon = 'icon';
static const String columnCategoryPasswordCount = 'password_count';
static const String columnCategoryCreatedAt = 'created_at';
static const String columnCategoryUpdatedAt = 'updated_at';

static Database? _database;

static Future<Database> get database async {
  _database ??= await _initDatabase();
  return _database!;
}

static Future<Database> _initDatabase() async {
  final databasesPath = await getDatabasesPath();
  final path = join(databasesPath, _databaseName);

  return await openDatabase(
    path,
    version: _databaseVersion,
    onCreate: _onCreate,
    onUpgrade: _onUpgrade,
  );
}

static Future<void> _onCreate(Database db, int version) async {
  // Create passwords table
  await db.execute('''
    CREATE TABLE $passwordsTable (
      $columnPasswordId TEXT PRIMARY KEY,
      $columnPasswordTitle TEXT NOT NULL,
      $columnPasswordUsername TEXT NOT NULL,
      $columnPasswordPassword TEXT NOT NULL,
      $columnPasswordWebsite TEXT,
      $columnPasswordNotes TEXT,
      $columnPasswordCategoryId TEXT NOT NULL,
      $columnPasswordCreatedAt INTEGER NOT NULL,
      $columnPasswordUpdatedAt INTEGER NOT NULL,
      $columnPasswordIsFavorite INTEGER NOT NULL DEFAULT 0,
      $columnPasswordStrength INTEGER NOT NULL DEFAULT 0,
      $columnPasswordTags TEXT
    )
  ''');

  // Create categories table
  await db.execute('''
    CREATE TABLE $categoriesTable (
      $columnCategoryId TEXT PRIMARY KEY,
      $columnCategoryName TEXT NOT NULL UNIQUE,
      $columnCategoryDescription TEXT,
      $columnCategoryColor TEXT NOT NULL,
      $columnCategoryIcon TEXT NOT NULL,
      $columnCategoryPasswordCount INTEGER NOT NULL DEFAULT 0,
      $columnCategoryCreatedAt INTEGER NOT NULL,
      $columnCategoryUpdatedAt INTEGER NOT NULL
    )
  ''');

  // Insert predefined categories
  await _insertPredefinedCategories(db);
}

static Future<void> _onUpgrade(Database db, int oldVersion, int newVersion) async {
  // Handle database upgrades if needed
  if (oldVersion < 2) {
    // Version 2: Fix column names and date formats
    // Drop and recreate tables with correct schema
    await db.execute('DROP TABLE IF EXISTS $passwordsTable');
```

```
    await db.execute('DROP TABLE IF EXISTS $categoriesTable');
    await _onCreate(db, newVersion);
  }
}

static Future<void> _insertPredefinedCategories(Database db) async {
  final predefinedCategories = PredefinedCategories.categories;

  for (final category in predefinedCategories) {
    await db.insert(
      categoriesTable,
      category.toMap(),
      conflictAlgorithm: ConflictAlgorithm.replace,
    );
  }
}

// Password operations
static Future<List<Password>> getAllPasswords() async {
  final db = await database;
  final List<Map<String, dynamic>> maps = await db.query(
    passwordsTable,
    orderBy: '$columnPasswordUpdatedAt DESC',
  );
  return List.generate(maps.length, (i) => Password.fromMap(maps[i]));
}

static Future<Password?> getPasswordById(String id) async {
  final db = await database;
  final List<Map<String, dynamic>> maps = await db.query(
    passwordsTable,
    where: '$columnPasswordId = ?',
    whereArgs: [id],
  );

  if (maps.isNotEmpty) {
    return Password.fromMap(maps.first);
  }
  return null;
}

static Future<int> insertPassword(Password password) async {
  final db = await database;
  return await db.insert(
    passwordsTable,
    password.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}

static Future<int> updatePassword(Password password) async {
  final db = await database;
  return await db.update(
    passwordsTable,
    password.toMap(),
    where: '$columnPasswordId = ?',
    whereArgs: [password.id],
  );
}

static Future<int> deletePassword(String id) async {
  final db = await database;
  return await db.delete(
    passwordsTable,
    where: '$columnPasswordId = ?',
    whereArgs: [id],
  );
}

static Future<List<Password>> searchPasswords(String query) async {
  final db = await database;
  final List<Map<String, dynamic>> maps = await db.query(
    passwordsTable,
    where: '''
      $columnPasswordTitle LIKE ? OR
      $columnPasswordUsername LIKE ? OR
      $columnPasswordWebsite LIKE ? OR
```

```
      $columnPasswordNotes LIKE ? OR
      $columnPasswordTags LIKE ?
    ''',
    whereArgs: [
      '%$query%', '%$query%', '%$query%', '%$query%', '%$query%'
    ],
    orderBy: '$columnPasswordUpdatedAt DESC',
  );
  return List.generate(maps.length, (i) => Password.fromMap(maps[i]));
}

static Future<List<Password>> getPasswordsByCategory(String categoryId) async {
  final db = await database;
  final List<Map<String, dynamic>> maps = await db.query(
    passwordsTable,
    where: '$columnPasswordCategoryId = ?',
    whereArgs: [categoryId],
    orderBy: '$columnPasswordUpdatedAt DESC',
  );
  return List.generate(maps.length, (i) => Password.fromMap(maps[i]));
}

static Future<List<Password>> getFavoritePasswords() async {
  final db = await database;
  final List<Map<String, dynamic>> maps = await db.query(
    passwordsTable,
    where: '$columnPasswordIsFavorite = ?',
    whereArgs: [1],
    orderBy: '$columnPasswordUpdatedAt DESC',
  );
  return List.generate(maps.length, (i) => Password.fromMap(maps[i]));
}

// Category operations
static Future<List<Category>> getAllCategories() async {
  final db = await database;
  final List<Map<String, dynamic>> maps = await db.query(
    categoriesTable,
    orderBy: '$columnCategoryName ASC',
  );
  return List.generate(maps.length, (i) => Category.fromMap(maps[i]));
}

static Future<Category?> getCategoryById(String id) async {
  final db = await database;
  final List<Map<String, dynamic>> maps = await db.query(
    categoriesTable,
    where: '$columnCategoryId = ?',
    whereArgs: [id],
  );

  if (maps.isNotEmpty) {
    return Category.fromMap(maps.first);
  }
  return null;
}

static Future<int> insertCategory(Category category) async {
  final db = await database;
  return await db.insert(
    categoriesTable,
    category.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}

static Future<int> updateCategory(Category category) async {
  final db = await database;
  return await db.update(
    categoriesTable,
    category.toMap(),
    where: '$columnCategoryId = ?',
    whereArgs: [category.id],
  );
}

static Future<int> deleteCategory(String id) async {
```

```
  final db = await database;
  return await db.delete(
    categoriesTable,
    where: '$columnCategoryId = ?',
    whereArgs: [id],
  );
}

static Future<int> updateCategoryPasswordCount(String categoryId, int count) async {
  final db = await database;
  return await db.update(
    categoriesTable,
    {
      columnCategoryPasswordCount: count,
      columnCategoryUpdatedAt: DateTime.now().millisecondsSinceEpoch,
    },
    where: '$columnCategoryId = ?',
    whereArgs: [categoryId],
  );
}

// Database maintenance
static Future<void> closeDatabase() async {
  if (_database != null) {
    await _database!.close();
    _database = null;
  }
}

static Future<void> clearAllData() async {
  final db = await database;
  await db.delete(passwordsTable);
  await db.delete(categoriesTable);
  await _insertPredefinedCategories(db);
}

// Export/Import operations
static Future<Map<String, dynamic>> exportData() async {
  final passwords = await getAllPasswords();
  final categories = await getAllCategories();

  return {
    'passwords': passwords.map((p) => p.toMap()).toList(),
    'categories': categories.map((c) => c.toMap()).toList(),
    'exported_at': DateTime.now().toIso8601String(),
    'version': _databaseVersion,
  };
}

static Future<void> importData(Map<String, dynamic> data) async {
  final db = await database;

  // Clear existing data
  await db.delete(passwordsTable);
  await db.delete(categoriesTable);

  // Import categories
  final categoriesData = data['categories'] as List<dynamic>?;
  if (categoriesData != null) {
    for (final categoryData in categoriesData) {
      await db.insert(
        categoriesTable,
        Map<String, dynamic>.from(categoryData),
        conflictAlgorithm: ConflictAlgorithm.replace,
      );
    }
  }

  // Import passwords
  final passwordsData = data['passwords'] as List<dynamic>?;
  if (passwordsData != null) {
    for (final passwordData in passwordsData) {
      await db.insert(
        passwordsTable,
        Map<String, dynamic>.from(passwordData),
        conflictAlgorithm: ConflictAlgorithm.replace,
      );
```

```
      }
    }
  }

  // Statistics
  static Future<Map<String, int>> getPasswordStatistics() async {
    final db = await database;

    final totalPasswords = Sqflite.firstIntValue(
      await db.rawQuery('SELECT COUNT(*) FROM $passwordsTable')
    ) ?? 0;

    final favoritePasswords = Sqflite.firstIntValue(
      await db.rawQuery('SELECT COUNT(*) FROM $passwordsTable WHERE
$columnPasswordIsFavorite = 1')
    ) ?? 0;

    final weakPasswords = Sqflite.firstIntValue(
      await db.rawQuery('SELECT COUNT(*) FROM $passwordsTable WHERE
$columnPasswordStrength < 3')
    ) ?? 0;

    return {
      'total': totalPasswords,
      'favorites': favoritePasswords,
      'weak': weakPasswords,
    };
  }
}
```

home_page.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:share_plus/share_plus.dart';
import 'package:password_manager/features/home/bloc/home_event.dart';
import 'package:password_manager/features/home/bloc/home_state.dart';
import
'package:password_manager/features/home/widgets/add_password_floating_button.dart';
import 'package:password_manager/features/settings/presentation/settings_page.dart';
import '../../../core/app_bar/custom_app_bar.dart';
import '../../../core/scaffold/custom_scaffold.dart';
import '../../../core/app.dart';
import '../../../core/security/app_lock_service.dart';
import '../bloc/home_bloc.dart';
import '../widgets/password_list_widget.dart';
import '../widgets/search_bar_widget.dart';
import '../widgets/category_filter_widget.dart';
import 'add_password_page.dart';
import 'edit_password_page.dart';
import 'password_details_page.dart';
import '../../login/presentation/login_page.dart';

class HomePage extends StatefulWidget {
  final AppThemeNotifier? themeNotifier;

  const HomePage({super.key, this.themeNotifier});

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> with WidgetsBindingObserver {
  final TextEditingController _searchController = TextEditingController();
  final AppLockService _lockService = AppLockService();
  String _selectedCategory = 'All';

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);
    _lockService.updateActiveTime();
    context.read<HomeBloc>().add(LoadPasswords());
  }

  @override
  void dispose() {
```

```
    WidgetsBinding.instance.removeObserver(this);
    _searchController.dispose();
    super.dispose();
  }

  @override
  void didChangeAppLifecycleState(AppLifecycleState state) {
    super.didChangeAppLifecycleState(state);
    if (state == AppLifecycleState.paused || state == AppLifecycleState.inactive) {
      _lockService.updateActiveTime();
    } else if (state == AppLifecycleState.resumed) {
      _lockService.updateActiveTime();
      // Check if app should be locked
      if (_lockService.isLocked && mounted) {
        // Logout user and return to login page
        _logoutUser();
      }
    }
  }

  void _logoutUser() {
    // Navigate to login page and clear navigation stack
    Navigator.of(context).pushAndRemoveUntil(
      MaterialPageRoute(
        builder: (context) => LoginPage(themeNotifier: widget.themeNotifier),
      ),
      (route) => false,
    );
    // Reset lock service state
    _lockService.updateActiveTime();
  }

  @override
  Widget build(BuildContext context) {
    return BlocListener<HomeBloc, HomeState>(
      listener: (context, state) {
        if (state is PasswordsExported) {
          _showExportSuccessDialog(context, state.filePath);
        } else if (state is PasswordsImported) {
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(
              content: Text('Імпортовано ${state.importedCount} паролів'),
              backgroundColor: Colors.green,
            ),
          );
        }
      },
      child: CustomScaffold.blue(
      appBar: CustomAppBar.green(
        title: 'SecureVault',
        actions: [
            BlocBuilder<HomeBloc, HomeState>(
              builder: (context, state) {
                if (state is HomeLoaded) {
                  return IconButton(
                    icon: Stack(
                      children: [
                        const Icon(Icons.sort),
                        if (state.sortType != PasswordSortType.title ||
!state.sortAscending)
                          Positioned(
                            right: 0,
                            top: 0,
                            child: Container(
                              width: 8,
                              height: 8,
                              decoration: const BoxDecoration(
                                color: Colors.orange,
                                shape: BoxShape.circle,
                              ),
                            ),
                          ),
                      ],
                    ),
                    tooltip: 'Сортування',
                    onPressed: () => _showSortDialog(context, state),
                  );
```

```
                }
                return const SizedBox.shrink();
              },
            ),
            IconButton(
              icon: const Icon(Icons.logout),
              onPressed: () => _showLogoutDialog(context),
            ),
            PopupMenuButton<String>(
                iconColor: Colors.white, // Білий колір для іконки три крапки
              onSelected: (value) => _handleMenuAction(context, value),
              itemBuilder: (context) => [
                  PopupMenuItem(
                  value: 'settings',
                  child: Row(
                    children: [
                        Icon(
                          Icons.settings,
                          color: Theme.of(context).textTheme.bodyLarge?.color,
                        ),
                        const SizedBox(width: 8),
                      Text('Налаштування'),
                    ],
                  ),
                ),
                  PopupMenuItem(
                  value: 'backup',
                  child: Row(
                    children: [
                        Icon(
                          Icons.backup,
                          color: Theme.of(context).textTheme.bodyLarge?.color,
                        ),
                        const SizedBox(width: 8),
                      Text('Резервна копія'),
                    ],
                  ),
                ),
              ],
            ),
          ],
        ),
        body: Column(
          children: [
            SearchBarWidget(
              controller: _searchController,
              onChanged: (value) {
                context.read<HomeBloc>().add(SearchPasswords(value));
              },
            ),
            const SizedBox(height: 8),
            CategoryFilterWidget(
              selectedCategory: _selectedCategory,
              onCategorySelected: (category) {
                setState(() {
                  _selectedCategory = category;
                });
                context.read<HomeBloc>().add(FilterByCategory(category));
              },
            ),
            const SizedBox(height: 16),
            Expanded(
              child: BlocBuilder<HomeBloc, HomeState>(
                builder: (context, state) {
                  if (state is HomeLoading) {
                    return const Center(
                      child: CircularProgressIndicator(),
                    );
                  } else if (state is HomeLoaded) {
                    return PasswordListWidget(
                      passwords: state.passwords,
                      onPasswordTap: (password) => _showPasswordDetails(context,
password),
                      onPasswordEdit: (password) => _editPassword(context, password),
                      onPasswordDelete: (password) => _deletePassword(context,
password),
                    );
```

```
                } else if (state is HomeError) {
                  return Center(
                    child: Column(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: [
                        const Icon(
                          Icons.error_outline,
                          size: 64,
                          color: Colors.red,
                        ),
                        const SizedBox(height: 16),
                        Text(
                          'Помилка завантаження',
                          style: Theme.of(context).textTheme.headlineSmall,
                        ),
                        const SizedBox(height: 8),
                        Text(
                          state.message,
                          style: Theme.of(context).textTheme.bodyMedium,
                          textAlign: TextAlign.center,
                        ),
                        const SizedBox(height: 16),
                        ElevatedButton(
                          onPressed: () =>
context.read<HomeBloc>().add(LoadPasswords()),
                          child: const Text('Повторити'),
                        ),
                      ],
                    ),
                  );
                }
                return const SizedBox.shrink();
              },
            ),
          ),
          AddPasswordFloatingButton(
        onPressed: () => _addPassword(context),
      ),
        ],
      ),
      // floatingActionButton:
      ),
    );
  }

  void _showLogoutDialog(BuildContext context) {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        title: const Text('Вихід'),
        content: const Text('Ви впевнені, що хочете вийти?'),
        actions: [
          TextButton(
            onPressed: () => Navigator.of(context).pop(),
            child: const Text('Скасувати'),
          ),
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
              Navigator.of(context).pushAndRemoveUntil(
                MaterialPageRoute(
                  builder: (context) => const LoginPage(),
                ),
                (route) => false,
              );
            },
            child: const Text('Вийти'),
          ),
        ],
      ),
    );
  }

  void _handleMenuAction(BuildContext context, String action) {
    switch (action) {
      case 'settings':
        Navigator.of(context).push(
```

```
          MaterialPageRoute(
            builder: (context) => SettingsPage(themeNotifier: widget.themeNotifier),
          ),
        );
        break;
      case 'backup':
        _showBackupDialog(context);
        break;
    }
  }

  void _showBackupDialog(BuildContext context) {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        title: const Text('Резервна копія'),
        content: const Text('Створити зашифровану резервну копію всіх паролів? Файл
буде зашифрований автоматично.'),
        actions: [
          TextButton(
            onPressed: () => Navigator.of(context).pop(),
            child: const Text('Скасувати'),
          ),
          ElevatedButton(
            onPressed: () {
              Navigator.of(context).pop();
              context.read<HomeBloc>().add(const ExportPasswords());
            },
            child: const Text('Створити'),
          ),
        ],
      ),
    );
  }

  void _showExportSuccessDialog(BuildContext context, String filePath) async {
    // Try to share the file
    try {
      await Share.shareXFiles(
        [XFile(filePath)],
        text: 'Резервна копія SecureVault',
      );
    } catch (e) {
      // If share fails, show dialog with file path
      if (mounted) {
        showDialog(
          context: context,
          builder: (context) => AlertDialog(
            title: const Text('Резервну копію створено'),
            content: Column(
              mainAxisSize: MainAxisSize.min,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                const Text('Файл збережено:'),
                const SizedBox(height: 8),
                SelectableText(
                  filePath,
                  style: const TextStyle(fontSize: 12, color: Colors.grey),
                ),
                const SizedBox(height: 16),
                const Text('Ви можете знайти файл за цим шляхом.'),
              ],
            ),
            actions: [
              TextButton(
                onPressed: () => Navigator.of(context).pop(),
                child: const Text('Закрити'),
              ),
            ],
          ),
        );
      }
    }
  }

  void _addPassword(BuildContext context) async {
    await Navigator.of(context).push(
```

```
    MaterialPageRoute(
      builder: (context) => const AddPasswordPage(),
    ),
  );
  // Reload passwords after returning from add page
  if (mounted) {
    context.read<HomeBloc>().add(LoadPasswords());
  }
}

void _showPasswordDetails(BuildContext context, dynamic password) {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => PasswordDetailsPage(password: password),
    ),
  );
}

void _editPassword(BuildContext context, dynamic password) {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => EditPasswordPage(password: password),
    ),
  );
}

void _deletePassword(BuildContext context, dynamic password) {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      title: const Text('Видалення пароля'),
      content: Text('Видалити пароль для "${password.title}"?'),
      actions: [
        TextButton(
          onPressed: () => Navigator.of(context).pop(),
          child: const Text('Скасувати'),
        ),
        ElevatedButton(
          onPressed: () {
            Navigator.of(context).pop();
            context.read<HomeBloc>().add(DeletePassword(password.id));
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.red,
            foregroundColor: Colors.white,
          ),
          child: const Text('Видалити'),
        ),
      ],
    ),
  );
}

void _showSortDialog(BuildContext context, HomeLoaded state) {
  PasswordSortType selectedSortType = state.sortType;
  bool ascending = state.sortAscending;

  showDialog(
    context: context,
    builder: (context) => StatefulBuilder(
      builder: (context, setDialogState) => AlertDialog(
        title: const Text('Сортування паролів'),
        content: SingleChildScrollView(
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              _buildSortOption(
                context,
                'За назвою',
                PasswordSortType.title,
                selectedSortType,
                (value) => setDialogState(() => selectedSortType = value),
              ),
              _buildSortOption(
                context,
                'За ім\'ям користувача',
                PasswordSortType.username,
```

```
                  selectedSortType,
                  (value) => setDialogState(() => selectedSortType = value),
                ),
                _buildSortOption(
                  context,
                  'За веб-сайтом',
                  PasswordSortType.website,
                  selectedSortType,
                  (value) => setDialogState(() => selectedSortType = value),
                ),
                _buildSortOption(
                  context,
                  'За силою пароля',
                  PasswordSortType.strength,
                  selectedSortType,
                  (value) => setDialogState(() => selectedSortType = value),
                ),
                _buildSortOption(
                  context,
                  'За датою створення',
                  PasswordSortType.dateCreated,
                  selectedSortType,
                  (value) => setDialogState(() => selectedSortType = value),
                ),
                _buildSortOption(
                  context,
                  'За датою оновлення',
                  PasswordSortType.dateUpdated,
                  selectedSortType,
                  (value) => setDialogState(() => selectedSortType = value),
                ),
                _buildSortOption(
                  context,
                  'За улюбленими',
                  PasswordSortType.favorite,
                  selectedSortType,
                  (value) => setDialogState(() => selectedSortType = value),
                ),
                const Divider(height: 24),
                Row(
                  children: [
                    Expanded(
                      child: OutlinedButton.icon(
                        onPressed: () => setDialogState(() => ascending = true),
                        icon: Icon(
                          Icons.arrow_upward,
                          size: 18,
                          color: ascending ? Theme.of(context).primaryColor :
Colors.grey,
                        ),
                        label: const Text('За зростанням'),
                        style: OutlinedButton.styleFrom(
                          foregroundColor: ascending ? Theme.of(context).primaryColor
: Colors.grey,
                          side: BorderSide(
                            color: ascending ? Theme.of(context).primaryColor :
Colors.grey,
                          ),
                        ),
                      ),
                    ),
                    const SizedBox(width: 8),
                    Expanded(
                      child: OutlinedButton.icon(
                        onPressed: () => setDialogState(() => ascending = false),
                        icon: Icon(
                          Icons.arrow_downward,
                          size: 18,
                          color: !ascending ? Theme.of(context).primaryColor :
Colors.grey,
                        ),
                        label: const Text('За спаданням'),
                        style: OutlinedButton.styleFrom(
                          foregroundColor: !ascending ? Theme.of(context).primaryColor
: Colors.grey,
                          side: BorderSide(
```

```
                              color: !ascending ? Theme.of(context).primaryColor :
Colors.grey,
                            ),
                          ),
                        ),
                      ),
                    ],
                  ),
                ],
              ),
            ),
            actions: [
              TextButton(
                onPressed: () => Navigator.of(context).pop(),
                child: const Text('Скасувати'),
              ),
              ElevatedButton(
                onPressed: () {
                  context.read<HomeBloc>().add(SortPasswords(selectedSortType,
ascending));
                  Navigator.of(context).pop();
                },
                child: const Text('Застосувати'),
              ),
            ],
          ),
        ),
      );
    }

    Widget _buildSortOption(
      BuildContext context,
      String title,
      PasswordSortType sortType,
      PasswordSortType selectedSortType,
      Function(PasswordSortType) onSelected,
    ) {
      return RadioListTile<PasswordSortType>(
        title: Text(title),
        value: sortType,
        groupValue: selectedSortType,
        onChanged: (value) {
          if (value != null) {
            onSelected(value);
          }
        },
        dense: true,
        contentPadding: EdgeInsets.zero,
      );
    }
}
```

ІЛЮСТРАТИВНА ЧАСТИНА

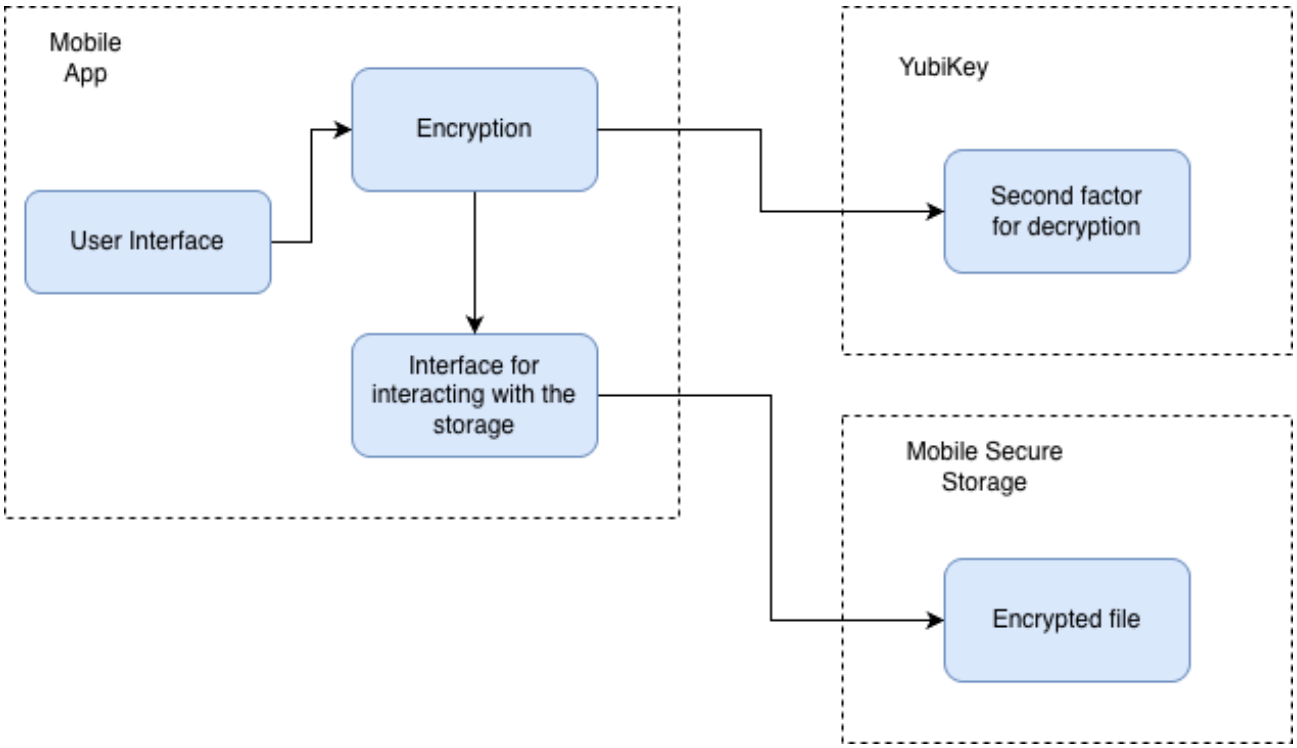
МЕТОД ТА ЗАСІБ ДЛЯ УПРАВЛІННЯ ПАРОЛЯМИ

Виконав: студент 1БС-24м
спеціальності 125 Кібербезпека та захист інформації
_____ Максим ТКАЧУК

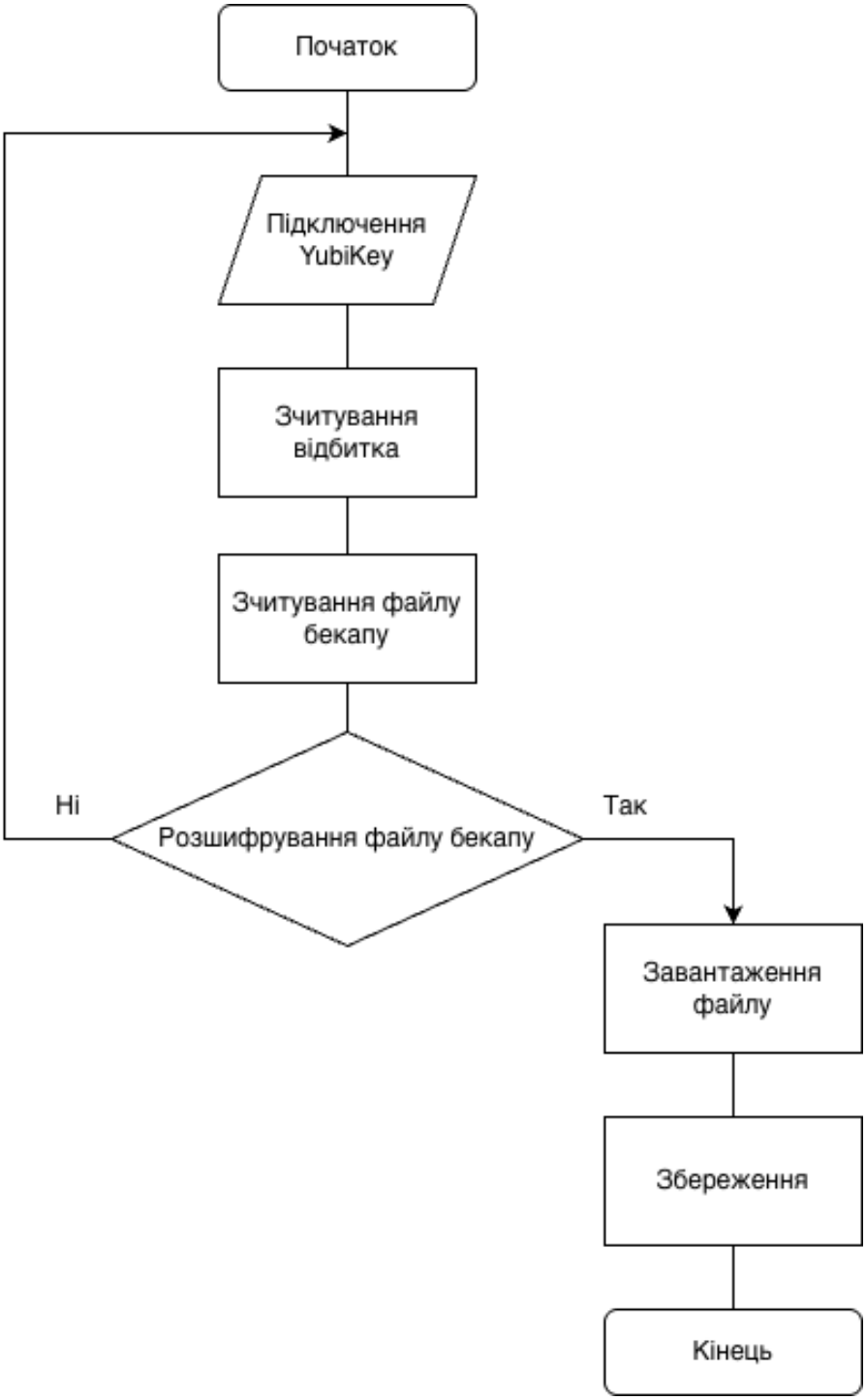
Керівник: к. т. н., доцент каф. ЗІ
_____ Віталій ЛУКІЧОВ
«19» грудня 2025 р.

ЗАГАЛЬНИЙ АЛГОРИТМ РОБОТИ ЗАСОБУ

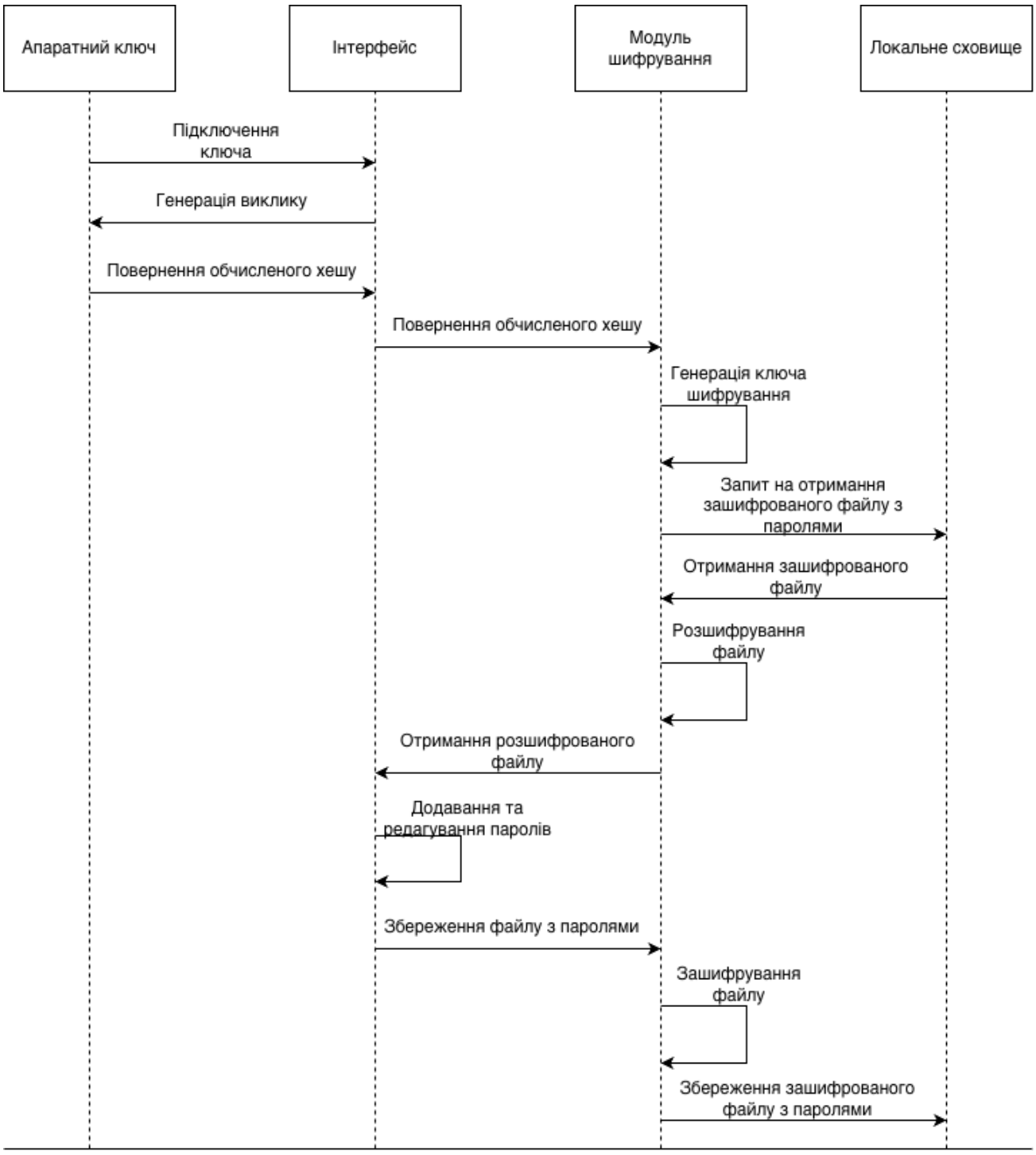
СХЕМА ЗБЕРІГАННЯ КЛЮЧІВ У ЗАСТОСУНКУ З ВИКОРИСТАННЯМ YUBIKEY



АЛГОРИТМ МЕТОДУ ПЕРЕВІРКИ ПАРОЛІВ



УЗАГАЛЬНЕНИЙ ПРОТОКОЛ РОБОТИ ЗАСТОСУНКУ



МЕХАНІЗМ HMAC-SECRET З РОЗШИРЕННЯМ FIDO2

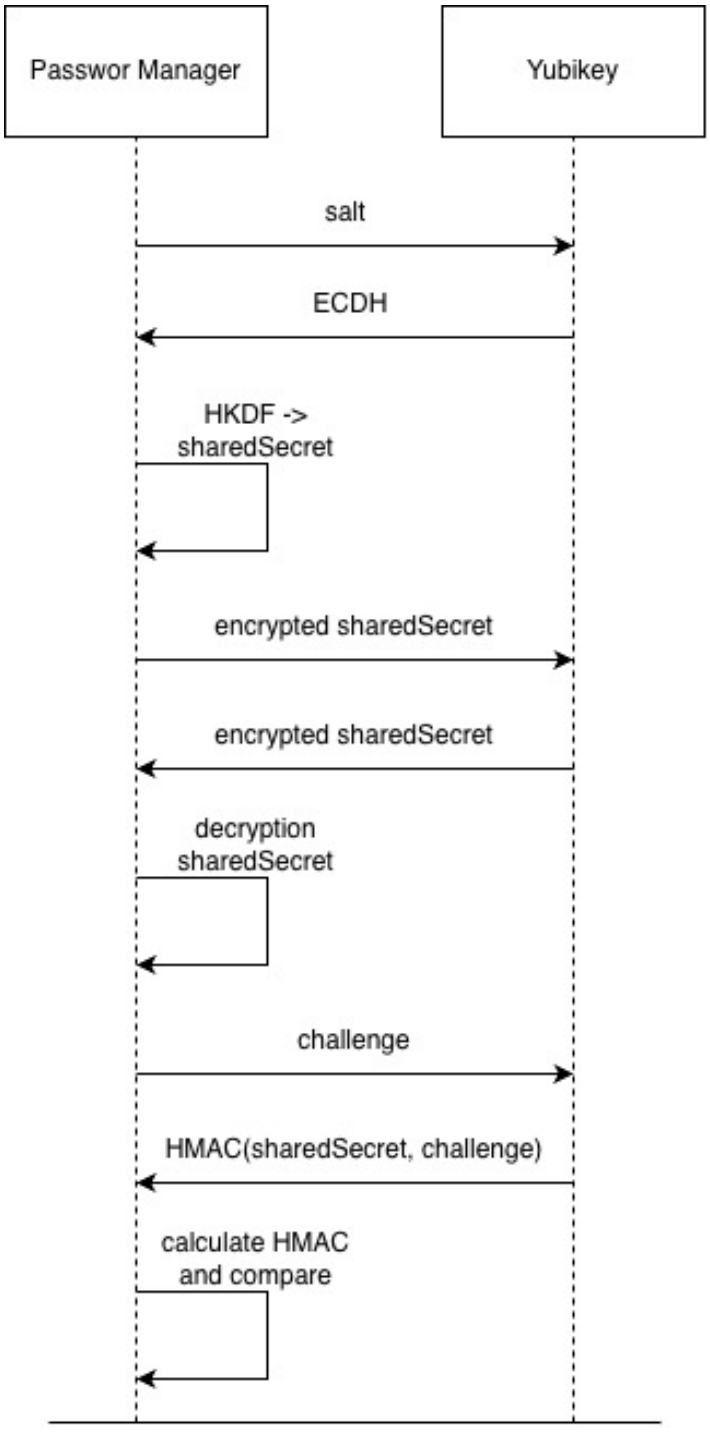
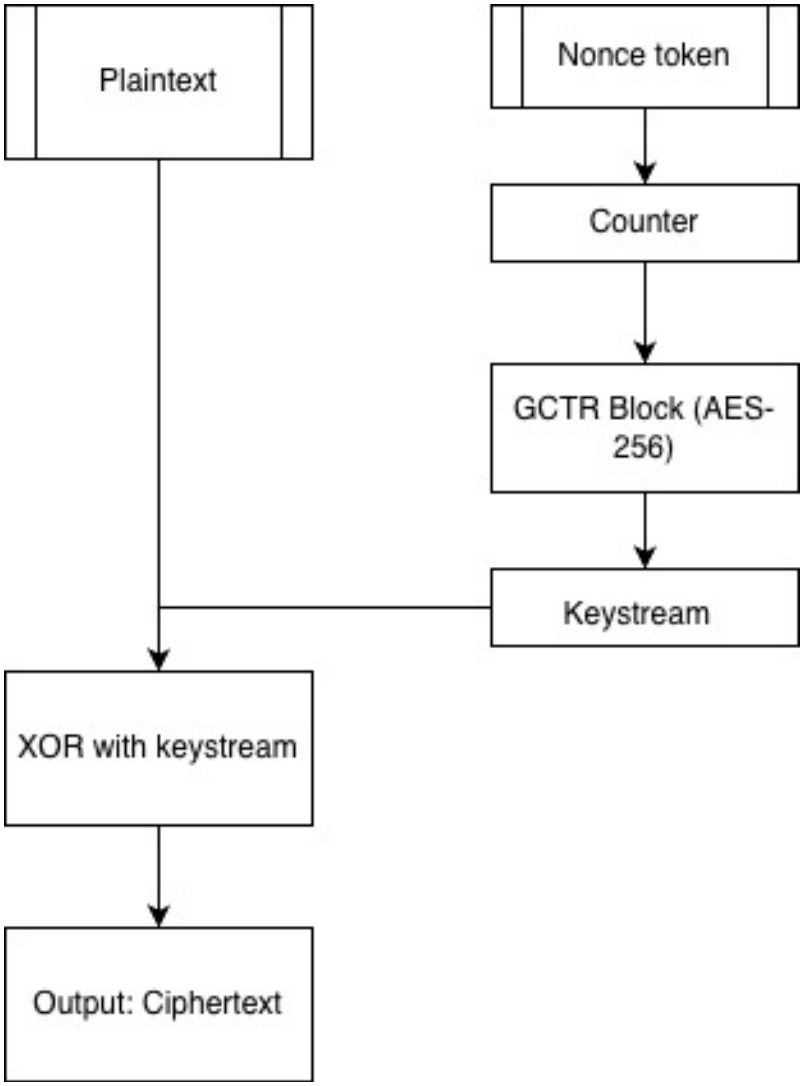
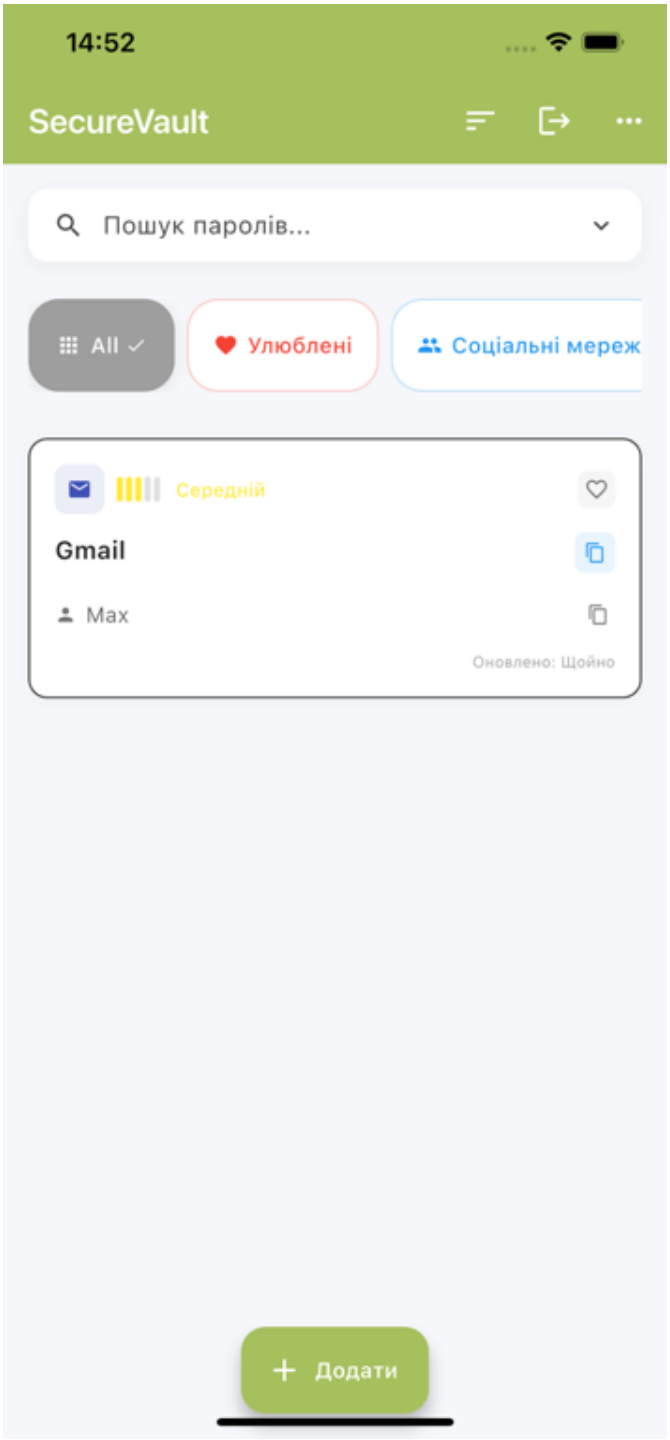


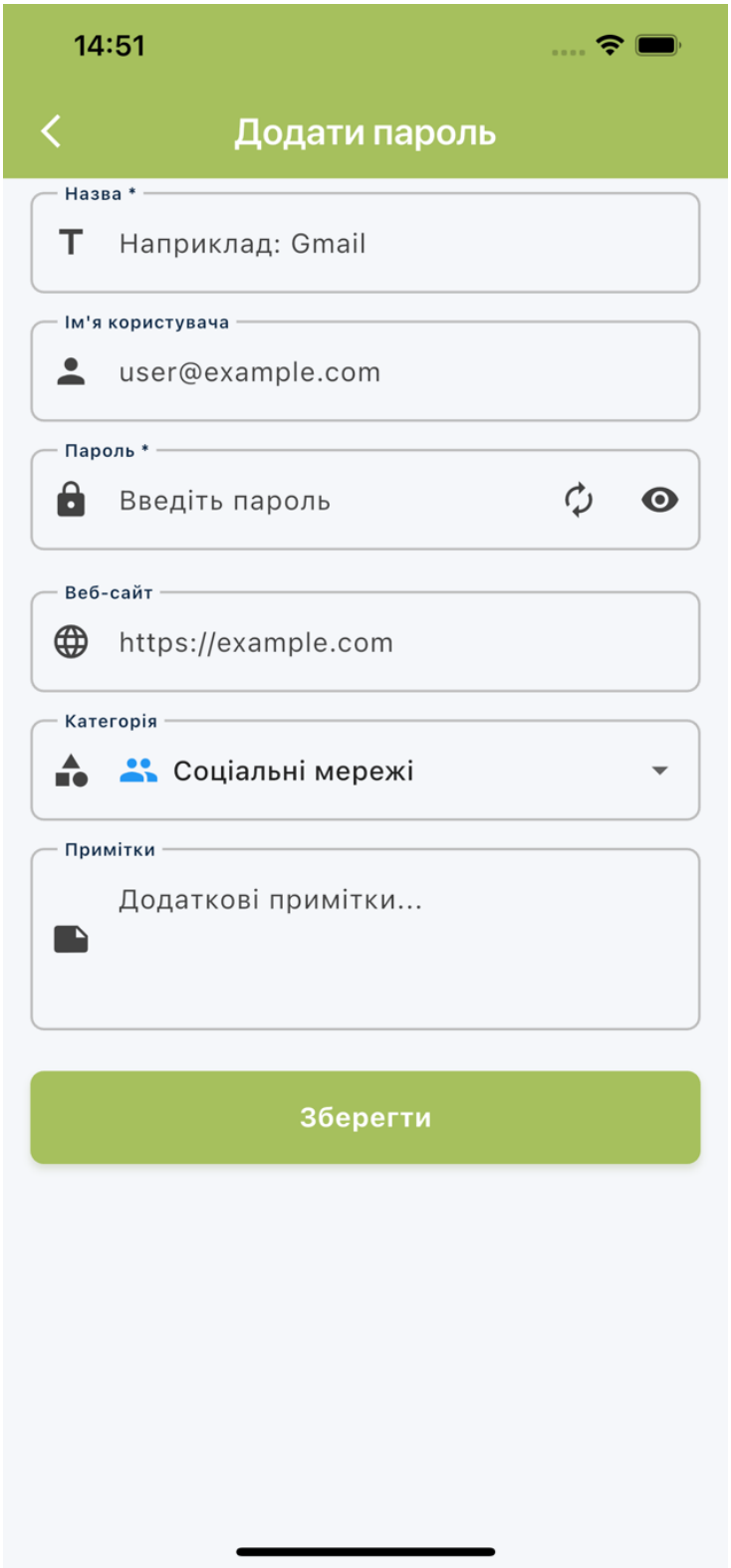
СХЕМА ШИФРУВАННЯ ДАНИХ З ВИКОРИСТАННЯМ AES-256-GCM



ДИЗАЙН ГОЛОВНОЇ СТОРІНКИ ЗАСТОСУНКУ ПІСЛЯ ДОДАВАННЯ ПАРОЛЮ



СТОРІНКА СТВОРЕННЯ ТА РЕДАГУВАННЯ ПАРОЛІВ



РЕЗУЛЬТАТИ ІНТЕГРАЦІЙНОГО ТЕСТУВАННЯ