Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)
Кафедра Безпеки інформаційних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка мобільного додатку для безпечного зберігання та обміну записами з використання Zero Knowledge Architecture

Виконав:
студент 2 курсу, групи БДІРм 20 -1
Мандич Д.Р.
(прізвище, ініціали)
Спеціальність 125 Кібербезпека
(код і повна назва спеціальності)
Освітня програма «Безпека державних інформаційних ресурсів»
(повна назва освітньої програми)
Керівник доцент Гріненко Т.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Халімов Г.З.
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Безпеки інформаційних технологій_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність_____125 Кібербезпека _____
(код і повна назва)

Освітня програма_____Безпека державних інформаційних ресурсів___
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Мандичу Дмитру Руслановичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи ____ Розробка мобільного додатку для безпечного зберігання та обміну записами з використання Zero Knowledge Architecture

затверджена наказом по університету від _08_ листопада__2021 р. № __1684 ст_

2. Термін подання студентом роботи до екзаменаційної комісії _13_грудня__2021 р.

3. Вихідні дані до роботи статистичні дані щодо кібер інцидентів, пов'язаних з мобільними додатками; методика оцінки ризиків OWASP_

4. Перелік питань, що потрібно опрацювати в роботі

- Аналіз літературних джерел за темою кваліфікаційної роботи;
- Аналіз загроз мобільним додаткам;
- Аналіз методів захисту мобільних додатків ;
- Розробка моделі захисту мобільних додатків;
- Розробка рекомендацій з забезпечення безпеки мобільних додатків;
- Дослідження Zero Knowledge Architecture;
- Реалізація мобільного додатку с використанням Zero Knowledge Architecture;

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _презентаційний матеріал у вигляді слайдів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	1.09.2021	Виконано
2	Аналіз літературних джерел за темою кваліфікаційної роботи	1.09.2021-1.10.2021	Виконано
3	Аналіз загроз мобільним додаткам	1.10.2021-22.10.2021	Виконано
4	Аналіз методів та засобів захисту мобільних додатків	22.10.2021-29.10.2021	Виконано
5	Розробка моделі захисту мобільних додатків	29.10.2021-6.11.2021	Виконано
6	Дослідження Zero Knowledge Architecture	6.11.2021-12.11.2021	Виконано
7	Реалізація мобільного додатку с використанням Zero Knowledge Architecture	12.11.2021-25.11.2021	Виконано
8	Оформлення пояснювальної записки	2.11.2021-13.12.2021	Виконано

Дата видачі завдання __1___вересня_____2021 р.

Студент _____
(підпис)

Керівник роботи _____ _____ доцент Гріненко Т.О._
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 89 сторінок, 1 таблиця, 10 рисунків, 23 джерела.

ВРАЗЛИВІСТЬ, ЗАГРОЗА, МОБІЛЬНИЙ ДОДАТОК, ОПЕРАЦІЙНА СИСТЕМА, ОБМІН ДАНИМИ

Об’єкт дослідження – забезпечення безпеки мобільних додатків для безпечного зберігання та обміну записами.

Предмет дослідження – архітектура Zero Knowledge Architecture, з використанням якої розроблений мобільний додаток, що забезпечує безпечне зберігання та обмін записами.

Мета роботи – аналіз загроз та вразливостей мобільних додатків; аналіз та дослідження методів та засобів захисту мобільних додатків, що забезпечують конфіденційність даних, безпечний доступ та керування ключами; аналіз та дослідження Zero Knowledge Architecture, що використовується для захисту мобільних додатків; розробка iOS додатку для безпечного зберігання та передачі повідомлень з використанням Zero Knowledge Architecture.

Надані результати аналізу вразливостей та загроз інформаційної безпеки мобільних додатків. Побудовано модель загроз для мобільних пристроїв. Проведений аналіз методів та засобів захисту додатків, що забезпечують конфіденційність даних, безпечний доступ та керування ключами. Побудовано модель захисту для мобільних додатків. Надані результати аналізу та дослідження архітектури Zero Knowledge Architecture, з використанням якої розроблений мобільний додаток, що забезпечує безпечне зберігання та обмін записами. Практичною частиною роботи є розроблений з використанням Zero Knowledge Architecture iOS-додаток для безпечного зберігання та передачі повідомлень.

ABSTRACT

Explanatory note to the qualification work: 89 pages, 1 table, 10 figures, 23 sources.

VULNERABILITY, THREAT, MOBILE APPLICATION, OPERATING SYSTEM, DATA EXCHANGE

The object of research is to ensure the security of mobile applications for secure storage and exchange of records.

The subject of research is Zero Knowledge Architecture for mobile application development.

Purpose – analysis and study of information security threats to mobile devices; analysis of methods of protection of mobile applications that ensure data confidentiality, secure access and management; analysis and research of Zero Knowledge Architecture to create recommendations for the protection of mobile applications; development of iOS application using Zero Knowledge Architecture for secure storage and transmission of messages.

The results of analysis and research of information security threats to mobile applications are presented. A threat model for mobile devices has been built. An analysis of application protection methods that ensure data privacy, secure access and key management. A model of protection for mobile applications has been built. The results of analysis and research of Zero Knowledge Architecture architecture for creating secure mobile applications are presented. The practical part of the work is developed using Zero Knowledge Architecture iOS-application for secure storage and transmission of messages.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

UID – унікальний ідентифікатор

APNs – Служба Apple Push Notification

IDS – Служба идентифікації Apple

ECDHE – Протокол обміну Діффі–Хеллмана на еліптичних кривих

ECID – Унікальний ідентифікатор пристрою

HSM – Апаратний модуль системи безпеки

ADEP – Apple Developer Enterprise Program

ZKA – Zero Knowledge Architectures

2FA – TwoFactor Authentication

TLS – Transport Layer Security

SQL – Structured Query Language

OWASP – Open Web Application Security Project

URL – Uniform Resource Locator

HTTP – HyperText Transfer Protocol

API – Application Programming Interface

ВСТУП

Розвиток інформаційних технологій призвів до того, що сучасні мобільні пристрої часто використовуються як додатковий офіс, розважальний центр та інструмент для споживання контенту. Сам пристрій може надати багато інформації про свого власника, адже в його пам'яті зберігаються: контакти колег, друзів і родичів з їх персональними даними; журнал викликів; корпоративне листування; Налаштувати точки доступу Wi-Fi в будинку власника; програми соціальних мереж (часто із збереженими паролями); банківські реквізити або мобільний SMS-банкінг, фотографії, відео, нотатки тощо.

Така концепція ділових і особистих даних призводить до того, що абстрактна цінність інформації перевищує ціну самого пристрою. Саме тому завдання захисту мобільного пристрою як від загроз, так і від втрати/збою є критичним.

Питання захисту мобільних додатків завжди було актуальним для розробників. Безпека продукції є важливою складовою її якості. І, незважаючи на стрімко зростаючий ринок мобільних додатків, рівень піратства відверто розчаровує. Безпека додатків важлива, оскільки сучасні програми часто доступні в різних мережах і підключені до хмари, що підвищує їх вразливість до загроз і порушень безпеки. Зростає потреба в безпеці не тільки на рівні мережі, а й на рівні самих програм, і цей підхід дає все більше переваг. Одна з причин полягає в тому, що атаки все частіше націлені на програми. Тестування безпеки програми виявляє вразливості на рівні програми, які допомагають запобігти таким атакам.Саме з цієї причини у даній роботі була зібрана інформація та проведено аналіз сучасних загроз персональним даним в мобільних додатках. А також проведений аналіз та дослідження архітектури Zero Knowledge Architecture для розробки мобільного додатку.

1 МОБІЛЬНІ ДОДАТКИ НА РИНКУ ТЕЛЕКОМУНІКАЦІЙ

1.1 Тенденції та перспективи ринку мобільних додатків

Ринок мобільних додатків зростає і розвивається. Ця величезна галузь розширюється з кожним днем. Різко зросла кількість розробників мобільних додатків і кількість самих додатків. Дохід, отриманий індустрією мобільних додатків, досяг значного рівня.

Гібридні моделі монетизації (такі як вбудована реклама та покупки в додатку) швидко набирають популярність у діловому світі [1]. Сьогодні мобільний інтернет став гострою потребою для багатьох користувачів. Два мобільних гіганта iOS і Android домінують на світовому ринку смартфонів. Згідно з дослідженням Gartner [2], 87,8% смартфонів, проданих у третьому кварталі 2016 року, були iOS. Цей показник на 3,1% більше, ніж рік тому. Частка ринку IOS – 11,5% (на 2% менше, ніж у 2015 році). І хоча ця цифра дуже мала для і без того величезної частки ринку, це зростання значно послаблює позиції інших гравців ринку. Windows, на яку припадає 0,4% усіх проданих смартфонів, посіла третє місце в гонці мобільних платформ з часткою 2,5% за рік. Крім того, Apple і Google є найбільшими і найпопулярнішими магазинами додатків. Сьогодні здається, що іншим учасникам ринку не варто мріяти зрівнятися з ними за діапазоном додатків і кількістю розробників.

Мобільний ринок є одним із сегментів онлайн-реклами, які швидко розвиваються. Мобільні пристрої стали доступнішими, вартість мобільного Інтернету знизилася, що не могло не вплинути на долю користувачів мобільного зв’язку.

У 2020 році користувачі по всьому світу активніше встановлювали додатки для бізнесу та освіти на смартфони та iPhone [5]. Стрибок викликаний пандемією COVID-19 і подальшою самоізоляцією. На це відзначили аналітики App Annie (глобального постачальника даних про мобільний ринок). Ігрові

програми займають друге місце за популярністю. Соціальні мережі та програми для потокового відео – на третьому. Будуть надбавки на доставку їжі, з фітнес-тренуваннями, на спостереження за здоров'ям, консультації лікаря та надбавки на фінанси.

Аналітики App Annie: час, який відзначається в додатках, а також більше. Як і в інших секторах, кілька основних розробників додатків отримують основну частину доходів від реклами. Всі інші розробники змушені задовольнятися залишками. За даними Klick Health, беззаперечним лідером є Facebook з 44,3% усіх показів мобільної реклами через його додатки. Далі в рейтингу по порядку: Alibaba, Google, Tencent, Twitter, Pandora і Yahoo.

Очевидно, що зростання ринку мобільних додатків продовжиться найближчим часом. Сьогодні iOS і Android є провідними мобільними операційними системами, технологічні гіганти Apple і Google володіють найбільшими магазинами мобільних додатків. Зрозуміло, що широко обговорювана революція мобільних додатків тільки починається.

1.2 Важливість захисту даних у мобільних додатках

При розробці мобільного додатку слід враховувати, що дані, якими оперує цей додаток, можуть представляти певний інтерес для третіх осіб. Ступінь їх цінності варіюється в широких межах, проте навіть найпростіша приватна інформація, наприклад: пароль у програмі, потребує роботи з її захистом. Це особливо важливо при поширенні мобільних додатків у всій сфері електронних послуг, включаючи фінансові, банківські, зберігання та передачу персональних даних[6].

Небезпечне зберігання даних - основним недолік, виявлений у 76% мобільних додатках. Найчастіше - це паролі, фінансова інформація, персональні дані та особисте листування. Зловмиснику майже не потрібен фізичний доступ

до смартфона, щоб отримати дані: 89% вразливостей можна реалізувати за допомогою шкідливого програмного забезпечення.

Проблема захисту мобільних пристроїв є у двох напрямках. Перший, суто прикладний технічний захист. Кількість інцидентів, пов'язаних саме з зараженням пристроїв, за останні роки істотно зросла[7]. Є кілька прикладів цього, які ілюструють масштаб проблеми:

– шкідливе програмне забезпечення CopyCat, яке транслює підроблені рекламні ролики, заразило понад 14 мільйонів пристроїв у всьому світі, принісши злочинцям близько 1 500 000 доларів лише за два місяці;

– всесвітньо відомий гурт Lazarus, можливо, причетний до зараження мобільних телефонів Samsung через закладки в мобільних додатках;

– за даними аналітиків Check Point, 89% організацій стали жертвами принаймні атак типу Man-in-the-Middle на мережі Wi-Fi.

Захист персональних пристроїв – особиста справа кожного власника, але лише до тих пір, поки смартфон не стане причиною зараження комп'ютерів в мережі через корпоративний Wi-Fi, або ж не станеться крадіжка даних з самого пристрою. Важливо розуміти, що такий пристрій є структурою будь-якої частини організації, тому, як і інші її елементи, вони повинні бути відповідно захищені. . Завдання помітно ускладнюється тим, що вони не можуть постійно перебувати за фаєрволом, тому, який би він не був та які на ньому не були побудовані політики безпеки, поширити їх дію на пристрій, який періодично залишає корпоративну мережу, без додаткового рішення просто неможливо.

Інший аспект – захист корпоративної інформації, яку може містити мобільний пристрій. В силу того, що в такому пристрої вкрай складно розділити особисту інформацію користувача, посягання на яку здійснюється з точки зору законодавства, і інформацію про корпоративну точку, яку необхідно захистити з метою запобігання шкоди, потрібен особливий підхід до захисту цієї інформації.

Більшість сучасних рішень мають клієнт-серверну архітектуру. Клієнт працює під управлінням мобільної операційної системи; використовує це iOS

або iOS. Клієнтська частина завантажується на пристрій з магазину додатків – спеціалізованої майданчики, де розробники публікують свої системи. З точки зору звичайного користувача, встановлена програма для смартфонів – це і мобільний додаток, тому що саме він взаємодіє з оплатою: покупки не оплачуються, оплачуються поштою. Але в дійсності є ще один компонент, який прийнято називати сервером [8].

Серверна частина знаходиться на стороні розробника. Найчастіше її роль виконує саме програмне забезпечення, яке відповідає генерації та обробці контенту на сайті, як представлено на рисунку 1.1. Іншими словами – частина сервера – це веб-додаток, який взаємодіє з мобільним клієнтом через Інтернет за допомогою спеціального інтерфейсу (API). Сервер саме можна вважати головною частиною: інформація тут обробляється і зберігається; крім цього, він відповідає за синхронізацією даних пристроями.

Сучасні версії мобільних операційних систем мають різноманітні вбудовані механізми захисту. Так, за допомогою всіх встановлених програм дозволено працювати лише з файлами у власних домашніх каталогах, а права користувача не зберігають редагування будь-яких системних файлів. Незважаючи на це, помилки, допущені розробниками при проектуванні та написані коду мобільних додатків, призводять до вразливостей в захисті та відкривають нові можливості для кіберзлочинців.

Рисунок 1.1 – Клієнт-серверна взаємодія в мобільному додатку

Комплексне тестування безпеки мобільних додатків передбачає пошук уразливостей як клієнтської, так і серверної частини; крім того, не менш важливо оцінити безпеку каналів даних між ними. У даній роботі розглянуто всі ці аспекти. Також розглядаються питання загроз, які очікують користувачів, в тому числі викликаних взаємодією клієнтської та серверної частини мобільних додатків.

2 БЕЗПЕКА ДАНИХ У МОБІЛЬНИХ ДОДАТКІВ

2.1 Загрози і основні вразливості додатків

Мобільні пристрої добре захищені від дистаційних атак. Однак прямий тривалий фізичний доступ до пристрою (або його компонентів) значно покращує роботу злочинців із криміналістики (комп'ютерна криміналістична експертиза, розслідування кіберзлочинів – прикладна наука про розкриття злочинів, пов'язаних зі злочинами, дослідження цифрових доказів, методи пошуку, отримання та консолідації такі докази), що ризик втрати даних.

Прямий доступ до пристрою можна отримати незаконним шляхом, за допомогою пристроїв або легально, шляхом дотримання правових норм, які можуть включати видачу паролів доступу. Органи влади можуть скористатися своїм службовим становищем або ж діями користувача, що може привести до повної компрометації пристрою. Зловмисник може зробить атаку за допомогою методів соціальної інженерії, видаючи себе, наприклад, через представника правоохоронних органів, або підробивши судові документи, отримані в результаті доступу до мобільного пристрою.

Фізичний доступ до пристрою за допомогою засобів кіберзлочинців дозволяє не тільки отримати дані, а й отримати токени доступу до захищеного хмарного сховища, яке знаходиться в пам'яті. Дані з пристрою та з хмарного сховища, як окремо, так і разом, можуть представляти різні категорії та особисту інформацію, створюючи серйозну загрозу для конфіденційності користувачів [9].

Якщо взяти статистику OWASP 2018, можно дослідити, на які категорії вразливостей поділяється аудит мобільного додатка [10]. Вразливості в мобільних додатках, що представленно в списку OWASP TOP 10, тільки для мобільних пристроїв, як представлено на рисунку 2.1.

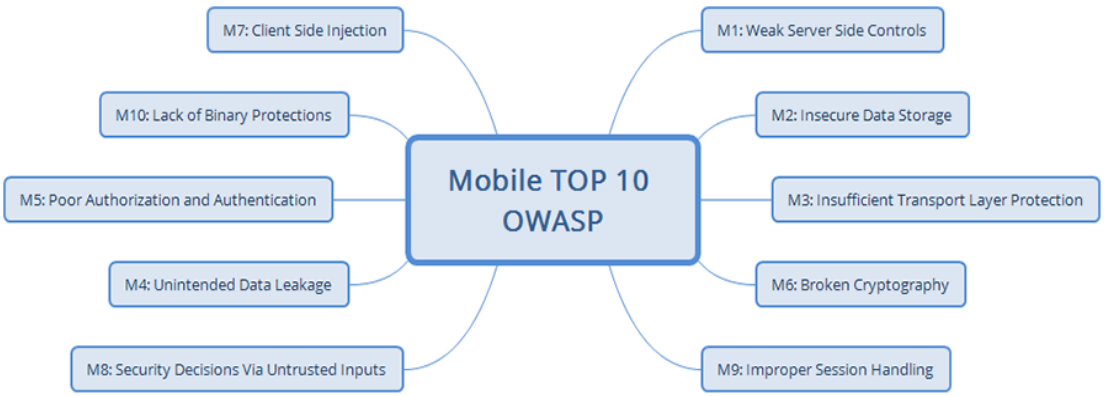

Рисунок 2.1 – Список вразливостей мобільних додатків за версією OWASP

Так як у мобільних додатках спосіб роботи по типу клієнт–серверної архітектури, то запити надходять на сервер. Розглянемо більш детально загрози мобільних додатків. Майже всі досліджувані програми знаходяться під загрозою з боку хакерів. Найпоширенішою проблемою мобільних додатків є небезпечне зберігання даних [11]. Найпоширенішим сценарієм є пошкодження зловмисним програмним забезпеченням, яке, швидше за все, збільшиться на пристроях з адміністративними привілеями (root або джейлбрейк). Однак шкідливі програми можуть самостійно збільшити права. Наприклад, троян-шпигун ZNIU використовує цю функцію для використання експлойтів до вразливості Dirty COW (CVE-2016–5195). Потрапивши на пристрій жертви, вірус може запросити дозвіл на доступ до даних користувача, отримати дозвіл на передачу даних зловмисникам.

2.2 Конфіденційність даних мобільних пристроїв

Практично всі користувачі мобільних пристроїв конфіденційності: електронна пошта, повідомлення, біометричні дані, фото та відеоконтент, документи, місцезнаходження, паролі, історія браузера тощо. Які є найбільш конфіденційними і які методи захисту використовуються на даний момент,

розглянемо нижче. Фахівці з форензіки виділяють найбільш пріоритетні категорії конфіденційних даних для злочинців [12]:

- IMEI, MEID / ESN і інші дані абонента мережі;
- контакти, адресна книга, календар, замітки;
- журнали вхідних і вихідних дзвінків;
- SMS, MMS, миттєві повідомлення;
- файли: аудіо, відео, документи;
- електронна пошта;
- активність браузера: історія, закладки;
- GPS і геолокаційні дані;
- соціальні мережі: акаунти, контент;
- SIM / UICC, провайдер, IMSI, MSISDN.

Слід зазначити, що цей список даних не обов’язково відображає загрозу конфіденційності користувачів. Деякі можуть мати терміни придатності, а деякі – коротку інформацію про інших людей. У міру поширення Інтернету та покращення можливостей машинного навчання деякі дані витягуються зі списку, що може призвести до порушення конфіденційності цілої групи користувачів.

Використовується поняття «особистість» використання програм і контексту. Зловмисника не завжди цікавлять імена користувачів, але якщо ви розробляєте додаток для бронювання приватного літака, навряд чи хтось із клієнтів захоче, щоб сторонні знали, що вони можуть скористатися вашими послугами. Отже, до особистих даних відносяться:

- токен, за допомогою якого робляться запити до API;
- номер кредитної картки;
- ім'я та прізвище;
- номер телефону;
- пошта;
- пароль.

Більшість даних не зберігається в програмі. Єдиним винятком, мабуть, є токен - унікальний набір символів для конкретного клієнта, за допомогою якого клієнтська сторона (додаток) робить усі запити до сервера. Якщо зловмисник зробить запит із цього токена, він використає законного користувача.

Токени є невизначеними та тимчасовими: невизначені дійсні, доки користувач не буде змушений вийти з системи, і в їх час є термін дії, який потрібно оновлювати. Останні працюють у банківських програмах.

Токен зберігається в Keychain. Це зашифрована база даних для зберігання метаданих та конфіденційної інформації від Apple.

Технічно Keychain може зберігати не тільки токен, а й паролі, пошту, номер телефону. Але для цього рекомендується обробляти дані, використовуючи «сіль», і шифрувати їх відразу після отримання. Таким чином, зловмисник, який зламав брелок, знайде цей хеш. Він може створити таблицю часто використовуваних паролів та їх хешів для порівняння з цим хешем. Якщо користувач використовує пароль без «солі», він існував у хеш-таблиці зловмисників, він був скомпрометований.

Додавання «солі» збільшує складність атаки. [6] це, наприклад, поєднання електронної пошти та пароля користувача з «сіллю», щоб створити хеш, а також ще важче отримати пароль.

Для взаємодії з сервером додаток і сервер використовуватимуть одну «сіль». Це дозволяє створювати хеші однаково і порівнювати два хеші, щоб перевірити ідентичність. Взаємодія додатків безпосередньо з Keychain ускладнюється, особливо в мові програмування Swift. Потрібно використовувати Apple Security Framework, який переважно написаний на мові C. Можна уникнути використання низькорівневого API, запозичуючи Swift оболонку GenericKeychain від Apple.

Відповідальність мобільного розробника за безпеку даних в тому, щоб всі потрібні дані зберігалися в Keychain.

Третя особа може отримати доступ до даних:

– в момент їх передачі;

– підглянувши або просто знаючи Apple ID і, відповідно, Keychain. Така можливість стала реальною завдяки уразливості, виявленої в жовтні 2017 року розробником Феліксом Краузе. 30 рядків коду в додатку відтворюють системне спливаюче вікно, що запрошує Apple ID. Щоб перевірити, запитується чи Apple ID системою або додатком, Краузе порадив згорнути програму, і якщо разом з ним згорнеться і вікно, то воно явно переслідує нехороші цілі;

– зайшовши в Developer Center і iTunes Connect через Apple ID. Але двухфакторна автентифікація вимагає вводити пароль від Apple ID і чотиризначний код, що ускладнює несанкціонований доступ до цих служб. Навіть отримавши доступ, зловмисник зможе управляти додатком (виставляти ціни, видалити), а не самими даними.

Якщо дані передаються між додатком і сервером через протокол HTTP, то вони передаються в нешифрованому вигляді. Як правило, спілкування мобільного додатка з сервером відбувається через HTTPS, що забезпечує шифрування [13].

Щоб сервер міг спілкуватися через зашифрований канал, сервер має бути аутентифікований. Для цього йому призначається програма цифрового сертифіката, яка включає ім’я сервера, назву центру сертифікації та сервер відкритих ключів. Перш ніж передавати номер, програма порівнює номер із сертифікатом з уже збереженими, і таким чином перевіряє, що сервер є автентичним. Після підтвердження сеанс ключа створюється, трафік шифрується.

Атаки MITM реалізуються шляхом заміни сертифікатів, як представлено на рисунку 2.2. Принцип атаки полягає в тому, щоб змусити програму вважати, що сертифікат зловмисника дійсний.

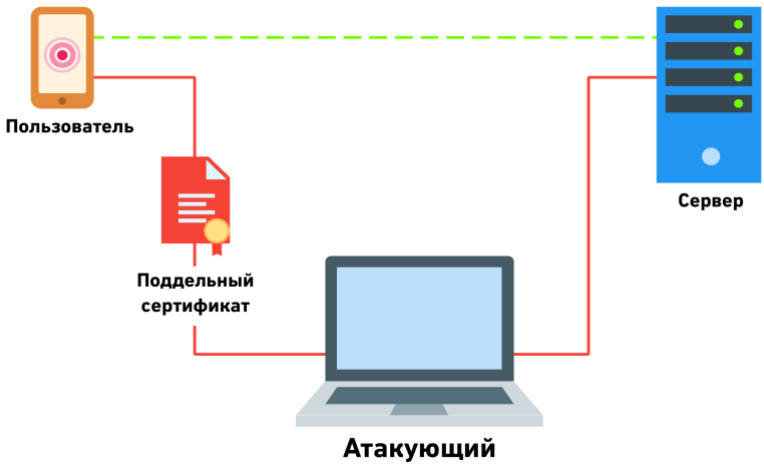

Рисунок 2.2 – Атака Man–in–the–middle

Атака відбувається по одній з чотирьох технік [13]:

– додавання сертифіката в сховище сертифікатів користувача в додатку. Це так званий експлоїт (шкідливий код, що використовує вразливість), коли атакуючий впроваджує в додаток користувача сертифікат і може встати в зашифрований канал і красти дані;

– модифікація, коли сертифікат вшивається в сам додаток. Це можна зробити навіть не на стадії розробки, а коли додаток заражено.

– обхід SSL pinning. Всі сучасні програми використовують SSL pinning як рішення від MITM. Сенс в тому, що сертифікат безпосередньо вшивається в код, а не зберігається в телефоні, де його можна підмінити, і видозмінюється до невпізнання. Сертифікат зберігає працездатність, але стає недоступним для зловмисника, навіть якщо той зробить реверс–інжиніринг. Але SSL pinning можна обійти, якщо підключитися безпосередньо до працюючому застосуванню на телефоні, яке зберігається в оперативній пам'яті;

– реверс–інжиніринг додатків і вивчення алгоритму верифікації сертифіката. Буває, що механізм перевірки сертифікату в додатку написаний розробниками або використовує якісь бібліотеки. Реверс–інжиніринг допомагає вивчити вихідний код і зрозуміти, як бібліотека перевіряє сертифікат, знайти вразливості в перевірці і їх використанні.

Реверс–інжиніринг проводиться в дві стадії:

– Зловмисник встановлює додаток, як звичайний користувач, знаходить уразливості і відтворює їх на своєму пристрої;

– Зловмисник пише експлоїт і доставляє цей експлоїт в пристрій: через віруси, шкідливі посилання, скачування.

У більшості випадків зловмисники використовують перший тип атаки, пов'язаний з логічними помилками в перевірці сертифікатів. Її механізм є в мобільному додатку, і в більшості випадків він реалізований з помилками, які дозволяють впровадити експлоїт.

Розглянемо тепер засоби запобігання перехопленням.

Для початку, будь–який мобільний додаток має створюватися за стандартами. Серед таких стандартів – звернення додатків до користувача з проханням відкрити йому доступ до камери, мікрофона, фотографій. Без цієї функції програми відхиллять на стадії перевірки в App Store.

Наступний важливий етап в житті даних – це їх передача з програми на сервер. Щоб ніхто не зміг перехопити їх в цей момент, використовується:

– токен, який отримує додаток в момент авторизації користувача;

– зашифроване з'єднання на стороні сервера (SSL–сертифікат);

– HTTPS–протокол. У HTTPS–каналі використовується протокол TLS. Розробники самі не займаються шифруванням, вони простоналаштовують HTTPS, а решту магію робить протокол. SSL–сертифікат і HTPPS–протокол стали стандартом. Але в травні 2018 року компанія Solar Security, що створює продукти і сервіси для захисту інформаційної безпеки, знайшла вразливості в 16 мобільних додатках для аренди автомобіля. Причинами можливого викрадення акаунтів користувачів і витоку даних представники компанії

назвали слабкі алгоритми хешування, небезпечну реалізацію SSL і використання незахищеного протоколу передачі даних HTTP.

Touch ID і Face ID – ідентифікація по відбитку пальця і лиця. Apple стверджує, що можливість збігу відбитків пальців – 1:50000, а параметрів особи – 1:1000000. Така низька статистика збігів доповнюється ще й тим, що біометричні дані не зберігаються на пристрої в чистому вигляді – вони перетворюються в математичну модель, яка не піддається розшифровці.

Паролі та відбитки (вірніше, їх математичні уявлення) знаходяться під захистом системи Secure Enclave, і тільки вона має до неї доступ. Secure Enclave ізольована від iOS, тому дані не будуть використані операційною системою та програмами, не потраплять на сервера Apple або в сховище iCloud. В цьому власників запевняє офіційний опис технології.

У серпні 2017 року, коли iOS 11 перебувала в бета-версії, спеціаліст із інформаційної безпеки під псевдонімом xerub опублікував ключ для розшифровки мікропрограмного забезпечення співпроцесора Secure Enclave Processor. За допомогою цього ключа ви можете отримати доступ до коду програмного забезпечення, на якому працює система Secure Enclave, але сама система залишається недоступною. Експерт сподівається, що вразливість допоможе посилити безпеку співпроцесора.

Face ID можна перевірити та підтвердити лише на фізичному пристрої. Роботу Touch ID можна грати в симуляторі xCode, починаючи з дев'ятої версії. Структура локальної аутентифікації використовується для ідентифікації користувача за допомогою Face ID та Touch ID.

Якщо користувач не проходить біометричну ідентифікацію або скасовує її, йому пропонується ввести код пароля програми, який зберігається в Keychain.

Щоб захистити ім'я в iOS, можна ввести два параметри: textview та textfield. Найчастіше для введення пароля, імені та прізвища використовується textfield. Він налаштовується за допомогою деяких методів: для пароля необхідно виставляти параметр Secure Entry, щоб перекривати пароль точками;

якщо важливі ім'я та прізвище, то можна відключити автокорекцію (відключити поле autocorrectiontype) і система не буде їх запам'ятовувати і індексувати.

Для додатків, які містять особисті дані користувачів і які можна потайки підглянути використовується метод розмивання екрану програми при згортанні. Продемонстровано на рисунку 2.3.

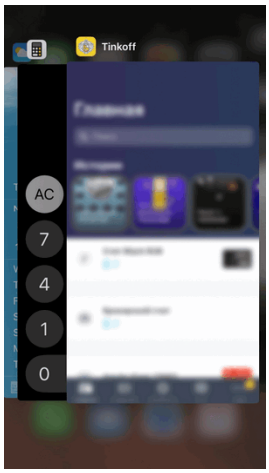

Рисунок 2.3 – Метод розмивання екрану програми при згортанні

Коли користувач згортає програму, операційна система застосовує до нього цей метод. Як результат, в режимі мультизадачності користувач бачить екран додатка розмитим, але його стан зберігається.

Отже, резюмуючи вищесказане, для того, щоб дані користувачів додатка залишалися в безпеці, розробникам додатків необхідно:

– зберігати дані в Keychain, обробляти всі дозволи на доступ до контактів, фото і т. п.;

– прописувати адресу сервера, до якого потрібно звертатися через HTTPS;

– налаштовувати доступ через Touch ID, Face ID і через код–пароль, що зберігається в Keychain, якщо палець не зчитується;

– передбачити безпечне введення даних через textfield і textview і обробляти його за правилами (вводити закритим без сторонніх символів)

– використовувати розмивання екрану з особистими даними при згортанні додатка;

– використовувати регулярні оновлення, що стосуються безпеки, використовуються, якщо в додатку виявлена помилка, яка стосується безпеки системи; якщо змінюється законодавство країн, в яких користуються додатком; ЄС чи Apple випускає оновлення, що стосується політики безпеки програми.

2.3 Модель загроз для мобільних операційних систем

Модель загроз для для мобільних операційних систем наведено в таблиці 2.1.

Таблиця 2.1 – Модель для мобільних операційних систем

№	Галузь безпеки	Вразливість	Загроза, що використовує дану вразливість
1	2	3	4
1.	Безпека зберігання даних	1.1. Можливість отримати доступ до файлової системи	1.1. Несанкціонований доступ до важливих даних додатків
		1.2 Неосвіченість користувачів у питаннях безпеки	1.2 Можливість успішних фішингових атак та використання соціальної інженерії
		1.3 Зберігання чутливих даних користувачів у відкритому вигляді в коді додатків	1.3 Несанкціонований доступ до даних користувача

Продовження таблиці 2.1

2.	Безпека аутентифікації користувачів	2.1 Використання легко відтворюваного Pin–коду	2.1 Несанкціонований доступ до даних користувача
		2.2 Ймовірність співпадіння біометричних даних користувачів під час використання Touch ID або Face ID	2.2 Несанкціонований доступ до даних користувача
3.	Безпека передачі даних	3.1 Передача даних у відкритому вигляді	3.1 Несанкціонований доступ до даних користувача
		3.2 Використання незахищеного протоколу передачі даних	3.2 Можливість модифікації та витоку інформації
		3.3 Незахищене з’єднання з мережами загальногокористування	3.3 Можливість проведення зловмисником атаки WiFi Man in the Middle (MitM)

Продовження таблиці 2.1

4.	Безпека коду програм	4.1 Urlschema без валидації	4.2 Можливість реверс інжинірингу коду додатку і несанкціонований доступ до внутрішньої реалізації важливих функцій чи технологій
		4.2 Прихований функціонал додатку при певних дозволах користувача	4.2 Несанкціонований доступ до даних користувача
5.	Оновлення системи	5.1 Не своєчасне встановлення оновлень	5.1 Можливість проведення атак, що були можливі тільки для певних версій системи чи додатку
6.	Безпека архітектури системи чи додатку	6.1 Можливість зробити «Jailbreak» системи	6.1 Можливість встановлення на пристрій додатків невідомих розробників. З втратою системних засобів захисту та доступу до файлової системи, стаюсь можливими велика кількість атак

3 МЕТОДИ ТА ЗАСОБИ ЗАХИСТУ МОБІЛЬНИХ ДОДАТКІВ

3.1 Багаторівнева модель захисту додатків

Для захисту мобільних додатків пропонується комплексний системний підхід на основі багаторівневої моделі. На малюнку 3.1 наведено загальну графічну модель такого підходу.

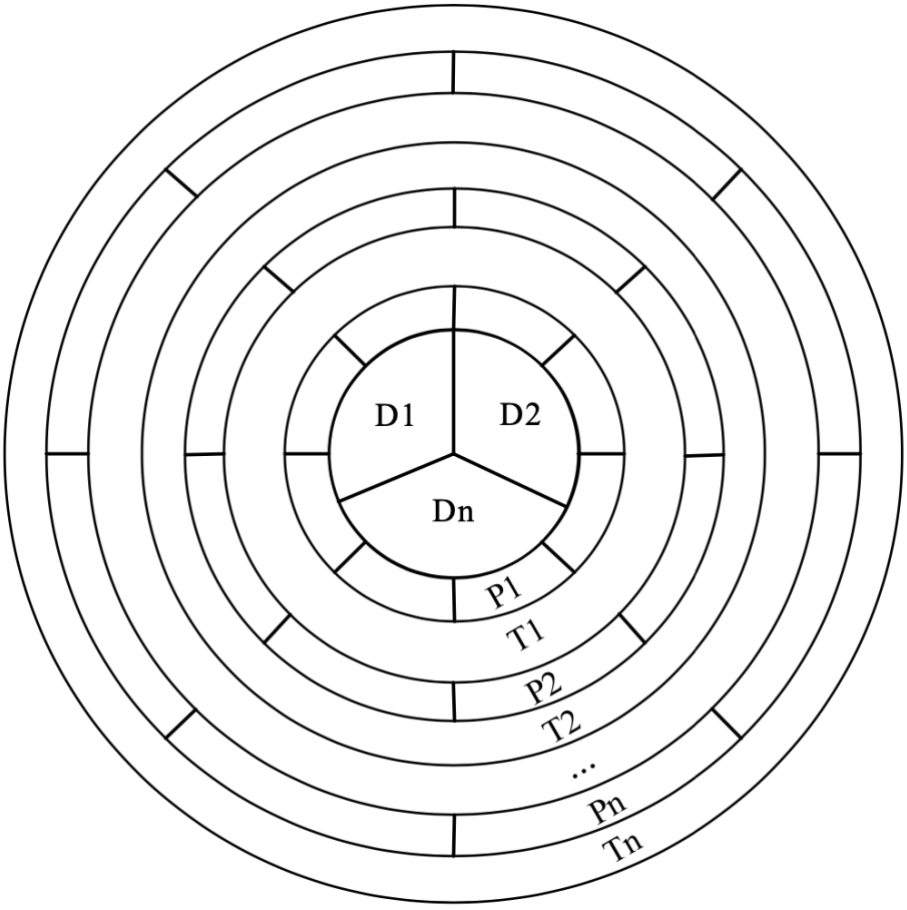

Рисунок 3.1 – Загальна графічна модель багаторівневого захисту

У центрі моделі розташовані об’єкти захисту додатка D, у кільцях навколо об’єктів – загрози P та засоби протидії T, які протидіють цим загрозам. Для підвищення безпеки мобільного додатка, зниження ймовірності атак і атак його несанкціонованого використання зловмисником запропоновано структуру системи захисту з певною надмірністю, а саме для покриття загрози низьких контрзаходів. На малюнку 3.2 представлена графічна модель багаторівневого

захисту мобільного додатка, яка містить відчутну інформацію про певні види загроз несанкціонованого використання.

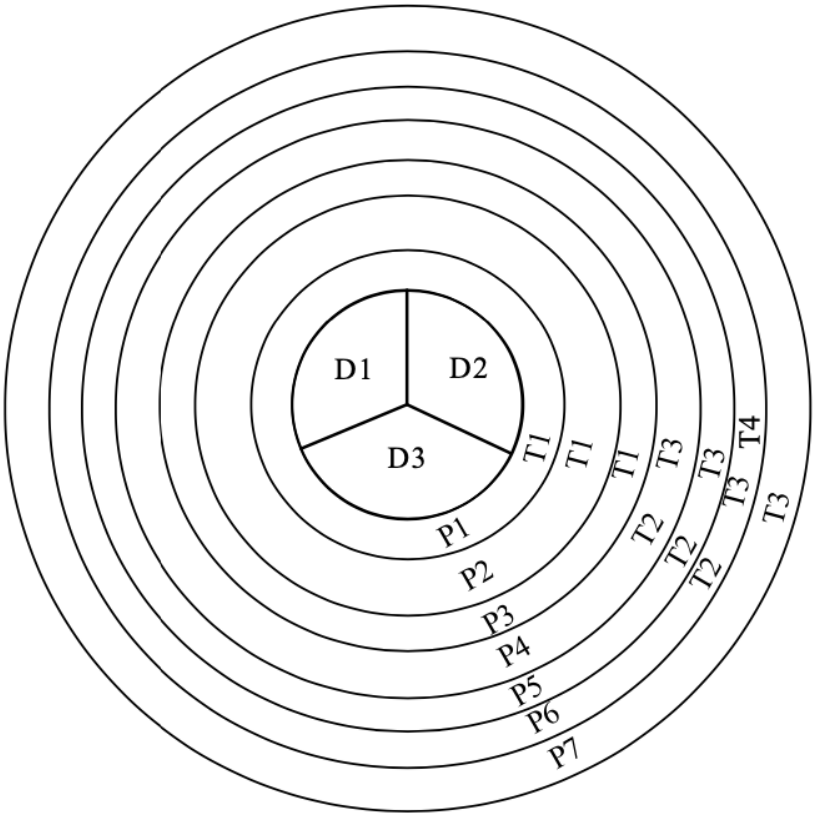

Рисунок 3.2 – Графічна модель багаторівневого захисту мобільних додатків від несанкціонованого використання

Об’єктами захисту є:

- код додатку D1;
- дані додатку D2;
- доступ до інфтерфейсу додатку D3.

Методи та засоби захисту iOS-додатку:

- обфускація коду T1;
- автентифікація T2;
- віддалений контроль та управління додатком T3;
- захист бази даних (БД) T4.

Зарози несанкціонованого використання додатку та відповідні контрзаходи:

- копіювання коду P1 захищається T1;

- аналіз коду P2 захищається T1;
- аналіз структури коду P3 захищається T1;
- заміна даних P4 захищається T2 і T3;
- крадіжка даних P5 захищається T2 і T3;
- перегляд даних P6 захищається T2, T3 та T4;
- перехоплення в каналах передачі P7 захищаєтьсяT3.

Використання такого ходу дозволяє значно підвищити надійність системи захисту, а також забезпечує гнучкість її налаштування в залежності від кращих помічників. Дана модель має надмірність методів захисту, але її перевагою є перекриття деяких загроз декількома методами захисту, що значно знижує ймовірність цієї загрози.

3.1.1 Архітектура системи захисту

На основі моделі багаторівневого захисту розроблено та реалізовано архітектуру системи захисту мобільного додатку. Представлено на рисунку 3.3.

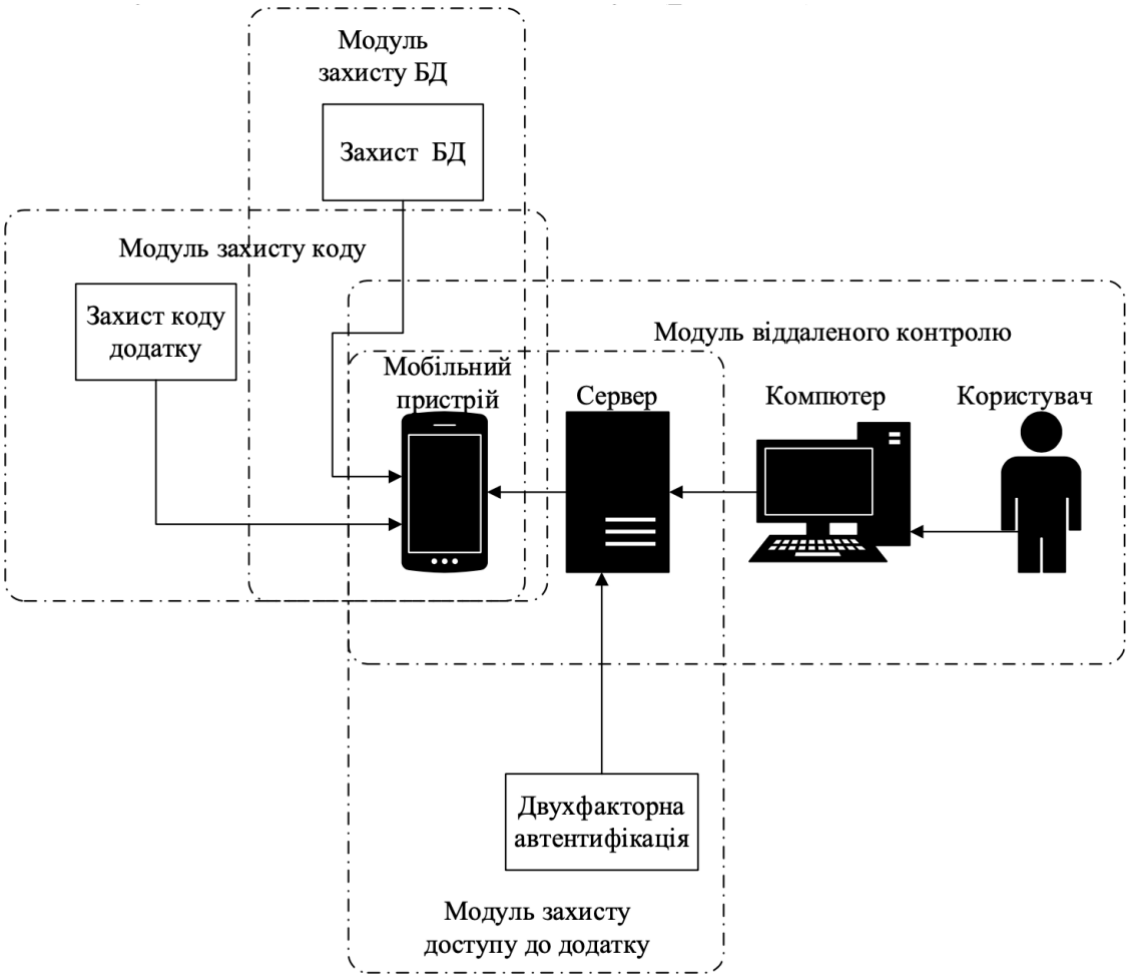

Рисунок 3.3 – Архітектура системи захисту

Архітектурні системи включають модуль обфускації програмного коду для захисту від зворотного інжинірингу, модуль багатофакторної аутентифікації для контролю доступу до програм, модуль віддаленого керування програмами та даними, а також модуль захисту бази даних для шифрування програм.

Для спрощення впровадження та використання захисту пропонується його розробка та програмна реалізація системи бібліотечного типу.

Модуль для захисту коду програми від зворотного інжинірингу з використанням обфускації коду під час збору програм у інсталяційний файл.

Модуль дистанційного керування має серверну та клієнтську частини. Користувач за допомогою веб–інтерфейсу може відправляти на мобільний пристрій команди керування додатком, а саме:

– очищення даних додатку;

– блокування/розблокування доступу до додатку; – резервне копіювання даних додатку;

– відновлення даних додатку.

Модуль багатофакторної аутентифікації також має архітектуру клієнт-сервер. Спочатку користувач керує логіном і паролем на сервері. Якщо вони підтверджені, програма шукає апаратний маркер через Bluetooth. Для апаратного токена використовується браслет з вбудованим Bluetooth-адаптером. Модуль захисту бази даних зосереджений на шифруванні інформації, включаючи дані аутентифікації. На рисунку 3.4 показано загальне функціонування системи захисту.

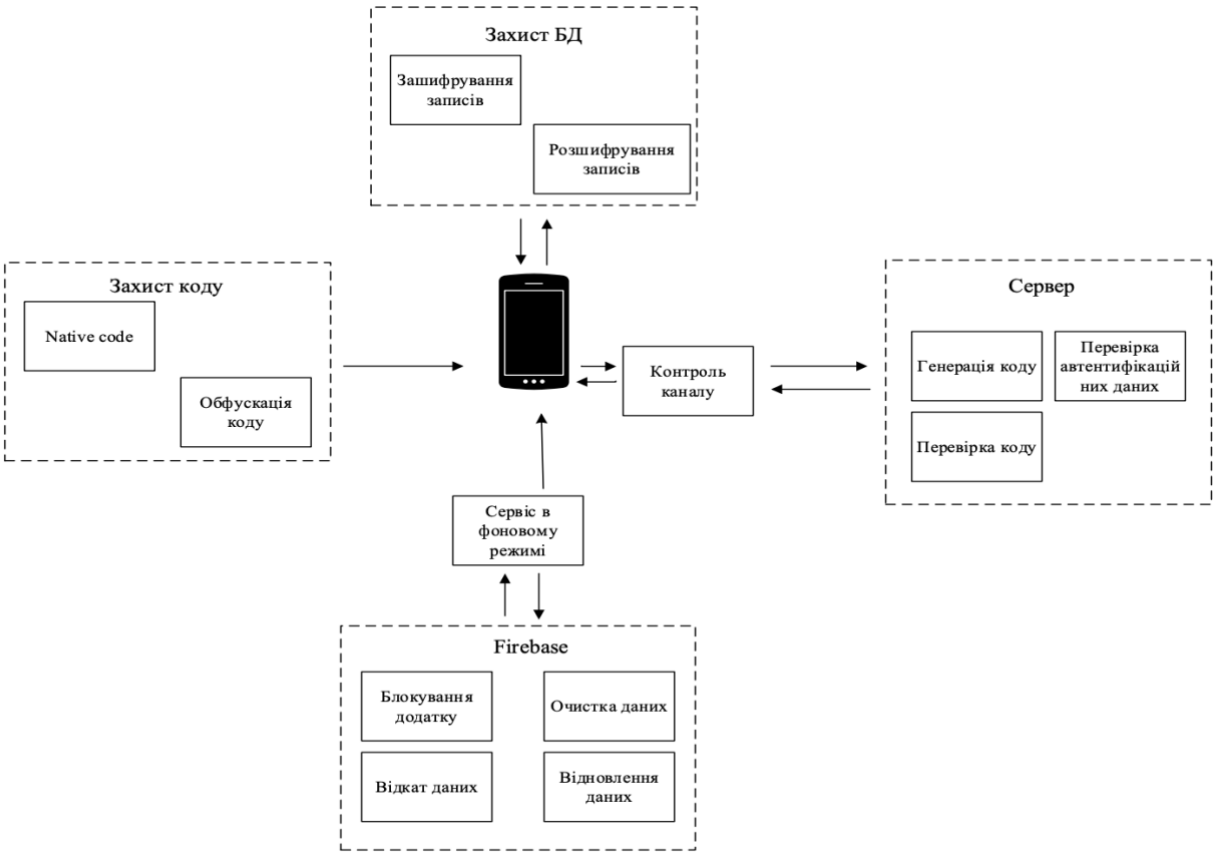

Рисунок 3.4 – Загальна модель функціонування системи захисту

Модульна аутентифікація використовується сервером для перевірки даних аутентифікації, код генерації підтверджується множинною і перевіркою цього коду. Виділений модуль управління включає частини сервера, які керують командами мобільного пристрою, команди отримує сервіс, який завжди працює у фоновому режимі. Модуль захисту бази даних шифрує всі записи в базі даних AES і розшифровує їх при зчитуванні з бази даних. Для захисту сервера програма запускає примусовий канал управління, тобто перевіряє, чи він не був відправлений/отриманий від сторонніх серверів.

3.1.2 Модуль захисту коду додатку

На основі аналізу вивчених варіантів захисту коду від зворотного інжинірингу було обрано ProGuard [9], код програми та всіх бібліотек, код яких стиснутий, захищений від «мертвого» коду та невикористаних змінних і обфуска. Щоб захистити код, було відтворено деякі функції в нативному коді. Обфускація коду виконується під час збору файлу apk. Додатки ресурсів, файли AIDL та додавання коду практикуються на компіляторі Java, де програма коду заплутується, а отриманий код перетворюється на байт-код. Потім береться код бібліотеки, доданої до проекту, і разом з основним кодом програми вони записуються у файл classes.dex. Потім завантажуються інші ресурси, які розміщуються в підключених бібліотеках або jar-файлах і додаються в apk-файл. Потім до файлу додається інформація про його налагодження підпису або ключ випуску. Виконується лексична обфускація коду програми. Під час збору додатків усі файли програми та коду з функцією обфускації збираються в файл apk. Значення, які замінюють імена класів, методів і змінних, знаходяться в таблиці. Таблиця букв складається з наступного списку англійського алфавіту a ... z. Якщо клас досить великий і недостатній, послідовність продовжується, включаючи 2 літери алфавіту aa ... zz. Перший параметр із класу, який потрібно замінити, вибирається з таблиці підстановки, а значення, які потрібно замінити, вибираються по порядку. Якщо значення з таблиці не відповідає імені класу

методу або змінної, відбувається заміна, якщо збігається, береться значення з таблиці. Після завершення обфускації процес збору файлів продовжується.

3.1.3 Модуль віддаленого контролю додатком

Для дистанційного керування та керування програмою використовуються такі функції: повне очищення даних програми; блокування входу в додаток; створення резервної копії даних програми на сервері; відновити дані програми з сервера. У розробці пропонується безкоштовний хмарний сервіс постачальником Firebase [10] від Google. Він дозволяє налаштувати базу даних для зберігання користувачів і керування push-повідомленнями за допомогою команд у програмі для дистанційного керування нею. Firebase також має вбудовану базу даних, яка містить маркери для надсилання команд. Під час першого входу в мобільний додаток створюється токен розміром 225 символів для ідентифікації мобільного пристрою. Програма віддаленого сервера шифрує повідомлення, яке потрібно додати, і керує push-сповіщеннями на мобільному пристрої. Програма отримує команду за допомогою служби, яка працює у фоновому режимі. Перед виконанням команди програма перевіряє справжність сервера; якщо сервер тестується, він виконується, якщо ні - ігнорується. Після цього отримане повідомлення розшифровується. Додаток із сервера може приймати такі команди: clear_app - повне очищення даних програми; block_app - блокування входу в програму; backup_data - створити резервну копію даних програми на сервері; recovery_data - відновлення даних програми з сервера.

Отримавши вищевказану командну програму у фоновому режимі, один модуль, який виконує дію з команд. Команда clear_app виконує очищення даних програми, видалення всієї бази даних, очищення кешу програми, остаточний захист. Команда block_app блокується під час роботи програми, тобто, коли запускається головний екран onCreate, який є першим у життєвому циклі Activity, це метод перевірки, який не блокує програму. Якщо вхід заблоковано, метод закриває програму. Команда backup_data використовується для резервного копіювання даних програми та надсилання їх на сервер у вигляді zip-архіву, якщо це файл, і у файлі розширення json, якщо це місце з

бази даних. Цьому користувачеві потрібен метод з бібліотеки і вкажіть, які дані і в якому вони будуть зберігатися. Команда recovery_data використовується для відновлення даних із сервера в програмі.

3.1.4 Модуль автентифікації

Схема автентифікації пароля має як суттєві переваги (простота впровадження, усвідомленість тощо), так і суттєві недоліки (шпигунство, вибір слабких паролів, повторний вхід тощо). Використання – стан без схеми пароля не дозволяє проводити активну аутентифікацію (постійну перевірку особистості користувача на основі аспектів його взаємодії з обчислювальним пристроєм) для контролю географічних параметрів або відносного розташування користувача/мобільного пристрою.

На основі аналізу варіантів активної автентифікації [1, 2] було обрано апаратні токени, які дозволяють виконувати багатофакторну автентифікацію за парольною схемою та на основі апаратного бездротового токену, що додатково включає в себе такі можливості:

– перевірку MAC–адреси бездротового пристрою;
– налаштування радіусу прийняття сигналу;
– налаштування часу повторної перевірки наявності токена.

Користувач відправляє дані аутентифікації на сервер, сервер перевіряє їх і відповідає мобільному додатку. В якості бази даних у додатку використовується Firebase, в ньому є таблиця з логінами та паролями, які порівнюються з даними, які підключені до мобільного пристрою. Якщо вони збігаються, мобільний додаток запускає службу, яка перевіряє, чи заряджений апаратний токен, наприклад, підключений через вбудований Buetooth, за допомогою якого мобільний пристрій підключається до маркера та зчитує його. Після підтвердження токена можна працювати з програмою. У додатку при вході в систему, незалежно від того, хто останній використовував програму, вікно аутентифікації фіксується. Дані аутентифікації (такі як логін і пароль) надсилаються на сервер для перевірки. Якщо користувача з такими даними не існує, сервер повертає програмі повідомлення про помилку. Після

підтвердження даних користувача мобільний пристрій надсилає повідомлення про зняття аутентифікації, в цей момент мобільний пристрій використовує Bluetooth і пошук служби. Він знаходить всі пристрої BLE і отримує їх MAC-адреси. Кожна MAC-адреса (MA) узгоджується з адресою, збереженою під час реєстрації на мобільному пристрої. Відповідність в базі даних забезпечує доступ до роботи з додатком. Після заданого інтервалу часу (Ti) відбувається розворот, поточний маркер знаходиться на відстані, яка не виконується (Ra). Якщо користувач із апаратним маркером перебуває за межами зони мобільного зв'язку Bluetooth, зв'язок між ними буде втрачено, і програма вимкнеться.

Для вирішення проблеми несанкціонованого копіювання інформації з бази даних, наприклад, для отримання доступу зловмисником до мобільного пристрою або зловмисним процесом, реалізовано модуль криптографічного захисту бази даних. Шифрування та вибір для окремих записів, а не для бази даних в цілому. Шифрування даних на основі блочного симетричного шифру AES із 128-бітним ключем. Ключ шифрування генерується на основі пакета програми та випадкового числа, що зберігається в програмі ресурсу. При цьому весь запис БД відповідає своєму. Ключі шифрування можна зберігати як локально, так і віддалено, а також додатково шифрувати. Модуль, реалізований на базі системи управління базами даних Realm, є однією з найшвидших мобільних систем [13].

3.2 Застосування захисту даних в додатках

Серед різноманіття методів захисту даних є кілька ключових методів контролю доступу до даних [9].

Перший – це ізоляція програмного забезпечення безпеки. Сучасні мобільні операційні системи чітко розрізняють доступ до програм і доступ користувачів. Наприклад, виявлені програми не можуть отримати доступ до даних, які не мають відповідних прав. Щоб отримати безкоштовний код на

пристрої, потрібно обійти це обмеження: «джейлбрейк» для iOS або отримати root-доступ для Android.

По-друге, це доступ через коди та біометричні дані. Контроль доступу до інтерфейсу ОС часто потрібен за допомогою безкоштовного примусового пароля, зазвичай цифрового. Ви можете активувати шаблон (графічний ключ - послідовність) на заблокованому екрані. Все частіше біометричний доступ: через розпізнавання обличчя або відбитки пальців.

Потім ви можете розділити шифрування файлів і дисків. Якщо зловмисник обходить методи захисту програмного забезпечення, він зіткнеться з зашифрованими даними. Це можуть бути як окремі файли, так і розділи диска. Шифрування даних зазвичай організовується асиметрично: одна частина ключа вбудована в апаратне забезпечення, а інша генерується на основі вибраного пароля користувача.

Безпечне оснащення приладів також незначне. Останнім часом виробники впроваджують захисні співпроцесори та їх аналоги.

Це захищає окремі біометричні шаблони від атак на програмне забезпечення та апаратних засобів, які посилюють шифрування конфіденційних даних.

І останнє, але не менш важливе, це безпечне резервне копіювання та управління хмарою. Постачальники послуг вже давно надають безпечну модель для хмарного зберігання резервних копій пристроїв.

Сучасна зашифрована система резервного копіювання виключає прямий доступ до даних користувачів хмарних провайдерів, зловмисників або правоохоронних органів.

Безпека додатків включає [14] розробку, тестування та додавання механізмів захисту на рівнях доповнень, спрямованих на запобігання уразливостям, які можливі, зокрема, перед додаванням та внесенням змін.

Захист програми відбувається на рівнях аутентифікації, авторизації, шифрування, входу та перевірки безпеки програми. Крім того, розробники можуть зменшити кількість уразливостей при написанні коду програми [14].

Розробники програмного забезпечення можуть створювати процедури аутентифікації та авторизації в програмі, щоб користувачі, які мають відповідні повноваження, мали доступ до них. Самі процедури аутентифікації гарантують, що користувач є тим, за кого себе видає. Це можна перевірити, вимагаючи імені користувача та пароля під час входу в обліковий запис у програмі. При багатофакторній аутентифікації необхідно пройти певний тип перевірки. Вони можуть перевірити знання користувача (введення пароля), наявність певних інструментів (перевірка мобільного пристрою) або особисті характеристики (розпізнавання відбитків пальців або обличчя).

Після автентифікації користувач може отримати право доступу до програми та її використання. Система може підтвердити, що користувач має дозвіл на доступ до програми, порівнявши його особу зі списком авторизованих користувачів. Аутентифікація має відбутися перед авторизацією, щоб програма відповідала списку авторизованих користувачів, тільки облікові дані були перевірені користувачами.

Після автентифікації користувача та використання програми інші заходи можуть запобігти доступу кіберзлочинців до конфіденційних даних та їх використання. У цих програмах конфіденційні дані передаються від кінцевого користувача в хмарі. Цей трафік може бути зашифрований для забезпечення безпеки таких даних.

Адже за наявності порушення безпеки програм авторизація може підвищити якість користувача, який отримав доступ до даних, а також знати, як вони були використані. Файли журналу даних містять записи з мітками часу та інформацією про те, як і як використовувалася програма. Щоб переконатися, що всі ці засоби захисту працюють належним чином, необхідні перевірки безпеки програми.

Забезпечення безпеки додатків у хмарі створює додаткові проблеми. Хмарне середовище забезпечує спільний доступ до ресурсів, тому необхідні спеціальні заходи безпеки, щоб користувачі мали доступ лише до даних, які їм дозволено переглядати у своїх хмарних програмах. Конфіденційність стає ще

більш вразливою в хмарних програмах, оскільки вона передається користувачам до програм і назад через Інтернет.

Мобільні програми також використовуються через Інтернет, а не через приватну мережу, що робить їх уразливими для атак. Компанії можуть використовувати віртуальні приватні мережі (VPN), щоб створити інший рівень безпеки мобільних додатків для співробітників, які віддалено входять у свої облікові записи додатків. IT-фахівці можуть вирішити протестувати мобільні програми, щоб переконатися, що вони відповідають політикам безпеки компанії, особливо дозволяючи співробітникам використовувати ці програми на мобільних пристроях, які підключаються до корпоративної мережі.

Інструменти керування безпекою програм базуються на технологіях, які допомагають покращити захист програм на рівні коду, щоб зробити їх менш вразливими для порівняння. Багато з цих інструментів призначені для обробки реакції програм на неочікувані дані кіберзлочинцями, які використовують уразливості. При написанні коду програміст може надати додаткові можливості для контролю таких сюрпризів. Нечіткое тестування — це тип перевірки безпеки програми, під час якої розробники перевіряють результати несподіваного введення даних і значень, щоб визначити варіанти, які провокують непередбачувані роботи додатків і відкриті вразливості безпеки.

Розробники перевіряють бездоганну роботу в процесі її розробки, щоб переконатися в розробці місць у новій або оновленій версії програмного забезпечення. Перевірте безпеку, щоб переконатися, що програма працює за певним набором критеріїв. Якщо програма тестується, розробники повинні бути доступні лише для авторизованих користувачів. При тестуванні на вторгнення розробники використовують роль кіберзлочинців і шукають шляхи доступу до програми. Під час таких перевірок розробляються методи соціальної інженерії, спрямовані на ключових користувачів у разі несанкціонованого доступу. Експерти перевіряють безпеку як з авторизацією (увійшов), так і без неї, щоб виявити вразливості, невидимі в одному з цих станів.

3.3 Забезпечення безпеки iOS-додатків

У 2020 році Apple заявила, що кількість активних пристроїв у всьому світі перевищує 1 400 000 000. 48% смартфонів у США та західних країнах - контрольовані iPhone. Поступове збільшення кількості пристроїв Apple на світовому ринку приваблює не тільки злочинців, а й експертів з уразливостей, яким корпорація пропонує програми винагород до 2 000 000 доларів США.

Корпорація вкладає великі кошти, обмежуючи ОС і програми, які працюють на розширеннях Apple. Висока цінність подвигів і централізоване реагування на інциденти створюють класичну битву між щитом і мечем. Розглянемо поточні технічні аспекти захисту інформації користувачів Apple, але спочатку подивіться на минуле та згадайте історію функцій безпеки iOS.

Перша версія шифрування даних флеш-пам'яті з вимкненим живленням була реалізована в iOS 3 (2009). Ключ шифрування не залежав від коду доступного користувача і не потрібен для розшифровки. У iOS 8 (014) Apple значно збільшила кількість зашифрованих даних на пристроях за допомогою ключа, який генерується на основі пароля користувача. За допомогою функції захисту даних видалення даних відбувається разом із ключами без можливості відновлення. Однак у деяких випадках користувачі даних з’являлися шляхом міграції їх у відповідний розділ бази даних SQL на пристрої, що зберігає можливість подальшого відновлення.

Перехід від коду до біометрики пов’язаний з TouchID, який був створений у 2013 році, а в iOS 9 (2015) стандартну довжину цифрового пароля збільшено до шести символів. До цього чотиризначні паролі можна було поєднувати з програмними експлойтами. Ємнісний датчик відбитків пальців і наступний FaceID (2017), за словами Apple, підвищили і спростили безпеку, оскільки частота обробки біометричної системи обмежена SEP (Secure Enclave Processor). Ефективність цих нововведень викликає сумніви в наукових колах.

Далі йде перехід на архітектуру SEP і посилення апаратних компонентів пристроїв. З еволюцією iOS апаратні компоненти пристроїв Apple змінилися.

До впровадження чіпа A4 (власна одночіпова розробка SoC) Apple використовувала компоненти від Samsung, LG та інших виробників (2007-2009). Безпека була заснована на шифруванні завантажувальної пам'яті НІЗ за допомогою ключа AES (UID), який керував прискорювачем Crypto Engine.

Архітектура SEP (Secure Enclave Processor) була вперше реалізована в чіпі A7 (iPhone 5S, 2013 р.) і дозволяла використовувати функцію безпеки окремо від основного процесора, на якому запущена ОС і програми. Під час активації TouchID SEP використовує власний ключ UID для шифрування. Особливість в тому, що при включенні SEP навіть виробники не знають ключ UID. Починаючи з чіпа A7, відбувається послідовне підвищення безпеки та продуктивності, розширюючи функції SEP. Наприклад, у чіпі A12 (2020) Apple встановила захист від оновлення мікропрограми пристрою (режим DFU), як це реалізовано в режимі відновлення (режим відновлення).

Після анонсу iCloud Keychain у 2016 році та в iOS 11 (2017) CloudKit представив наступний API для сторонніх розробників. Перевага тут полягає в тому, що він особливим чином зберігає безкоштовний контейнер даних у хмарі: ніхто, крім користувача зі своїм брелоком, навіть сама Apple, не може його розшифрувати.

Ключі до сучасного захисту даних, які взаємодіють із користувачами iOS за допомогою пристрою та технології.

Аутентифікація відбувається шляхом фізичної взаємодії з пристроєм: цифрова / буквено-цифрова або біометрична аутентифікація. 6-значний пароль активовано за замовчуванням. Підтримується вибір експериментальних миючих буквено-цифрових фраз. Можливо, повністю відключити аутентифікацію, яка вкрай потрібна Apple. Спроби отримати цифрові паролі блокуються часовими інтервалами. TouchID (ємнісний датчик відбитків пальців) і FaceID (камера з розпізнаванням обличчя з чутливістю до глибини), які вимагають високоякісного Apple, щоб забезпечити більшу безпеку для користувачів.

Цифрові підписи коду, які доповнюють, допомагають серйозно обмежити виконуваний код на iOS. Це досягається за допомогою безпечного завантаження (перевірка підпису низького рівня (Boot ROM), що гарантує тривалий захист від ініціалізації виявленого програмного забезпечення) та підписання програми, яка є такою комбінацією: підпис, який контролює Apple, і сертифікати з відкрити ключ для масштабування системи. Для більш спеціалізованого запуску iOS-додатків всередині організації необхідно придбати так звану «засвідчення підпису компанії».

Пісочниця та код аналізу для захисту від виявлених програм надають доступ до даних користувача та API за допомогою ізольованого середовища (пісочниці). Додаток обмежує доступ до файлової системи, простору пам'яті. Підписаний маніфест вказує на дозвіл доступу до системних ресурсів і служб, таких як служби геолокації. Програми з App Store проходять автоматичну та ручну перевірку коду. Однак, навіть з такими суворими обмеженнями, деякі небажані програми можуть перевіряти та порушувати конфіденційність користувачів.

У разі, якщо зловмиснику вдається обійти механізми через логічні вразливості або вразливості безпеки, Apple розробила надійний пристрій системи шифрування даних - Data Protection. iOS використовує криптографічні стандарти AES, ECDH над Curve25519 та інші, затверджені Національним інститутом стандартів і технологій США (NIST). Доступ до даних прив’язаний до пристрою та контролюється користувачем. Ключ для шифрування даних формується на основі комбінацій, підібраних паролем користувача та UID (унікальний апаратний секретний криптоключ). Після перезавантаження необхідно відновити ключі шифрування, встановивши попередній пароль, тоді достатньо, щоб розблокувати ключі біометричні дані.

Обхід шифрування ускладнюється двома шляхами: функцією отримання ключа на основі пароля і методом обмеження припущень зі збільшенням тимчасових інтервалів, що представлено на рисунку 3.5. Apple використовує

кілька «класів захисту» (Class key) шифрування, які розробники вільні вибирати при створенні файлів або об'єктів даних:

1. Повний захист (CP) – через 10 секунд після блокування пристрою ключі шифрування видаляються.

2. Захищено поки не відкрито (PUO) – зберігається недовговічний відкритий ключ в пам'яті, передача зашифрованих файлів при заблокованому пристрої. Чи включається повний захист даних (CP) після того, як файл був створений і закритий.

3. Захищено до першої автентифікації користувача – першої розблокування (AFU) – при введенні першого пароля ключі шифрування розшифровуються в пам'яті і залишаються там при блокування пристрою.

4. Немає захисту (NP) – при вимкненому пристрої ключі шифрування зашифровані тільки апаратними ключами UID. Ці ключі постійно доступні в пам'яті, коли пристрій включено.

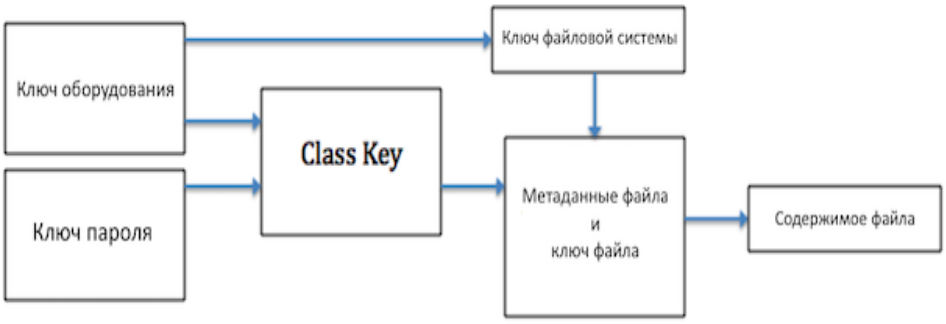

Рисунок 3.5 – Схема ключів iOS Data Protection

Keychain – це зашифроване сховище ключів та конфіденційної інформації (логіни/паролі), яке має відкритий API. Він захищений апаратними ключами та паролем користувача. Параметр Non-Migratory (NM) дозволяє розшифровувати дані лише на пристрої, де вони були зашифровані (використовується UID).

Резервне копіювання може виконуватися на персональному комп’ютері або в iCloud. Локальна резервна копія може бути зашифрована вибраним паролем, в результаті чого формується структура Keybag. Ненадійний пароль буде мало стійкий до грубої сили, оскільки він не пов’язаний з апаратними

клавішами пристрою. Тут iOS використовує 10 мільйонів ітерацій PBKDF2 для підвищення стійкості. Під час копіювання в Apple iCloud Curve25519 шифрується, що дозволяє робити резервну копію, коли пристрій заблоковано, не розкриваючи секретні ключі. Завдяки тому, що ключі зашифровані за допомогою iCloud Keychain, відомий виробник Apple або зловмисник може отримати доступ до вмісту резервної копії. Тому Keychain додатково шифрує обидва типи резервних ключів UID, щоб захистити копію на новому пристрої.

Резервне копіювання даних iCloud: програми, резервні копії Apple Watch, налаштування, головний екран і програми, iMessage, SMS та MMS, фотографії та відео, історія покупок у сервісах Apple, мелодії дзвінка, голосова пошта.

iCloud і iCloud Keychain iCloud дозволяє зберігати не тільки резервні копії, а й будь-яких інших користувачів, зазначених для синхронізації з хмарою. Дані передаються по мережі через TLS і зберігаються в зашифрованому вигляді. Криптографічне перетворення інструментів за допомогою 128-бітного ключа AES та ключа, отриманого з паролем користувача.

iCloud Keychain дозволяє синхронізувати (за допомогою асиметричного шифрування) Keychain між пристроями Apple, а в разі втрати пристрою - відновити Keychain за допомогою коду безпеки iCloud. Цей код генерується з паролем користувача для двофакторної аутентифікації або генерується автоматично, якщо його не ввімкнено. Використовуючи HSM (довірені апаратні модулі), Apple не має доступу до вмісту резервної копії iCloud Keychain, а код безпеки iCloud не передається з аутентифікацією HSM (захищений протокол віддаленого пароля SRP). Зашифрована копія iCloud Keychain керується користувачем пристрою лише після підтвердження того, що кількість останніх спроб була меншою за 10. Якщо було записано більше 10 спроб, цей запис буде видалено, і користувачам потрібно буде повторно зареєструватися на використовуйте брелок iCloud. Apple стверджує, що після розгортання HSM ключі підпису, необхідні для змін програмного забезпечення,

автоматично зменшуються, що обмежує доступ корпорацій до даних користувачів.

SEP–архітектура: TouchID і FaceID Secure Enclave Processor (SEP) – співпроцесор, який використовує зашифровану пам'ять і організуючий роботу по автентифікації, управління ключами, шифрування і генерації випадкових чисел. Якщо ядро iOS було зламано, то SEP забезпечує шифрування за рахунок своєї автономної архітектури, як представлено на рисунку 3.6.

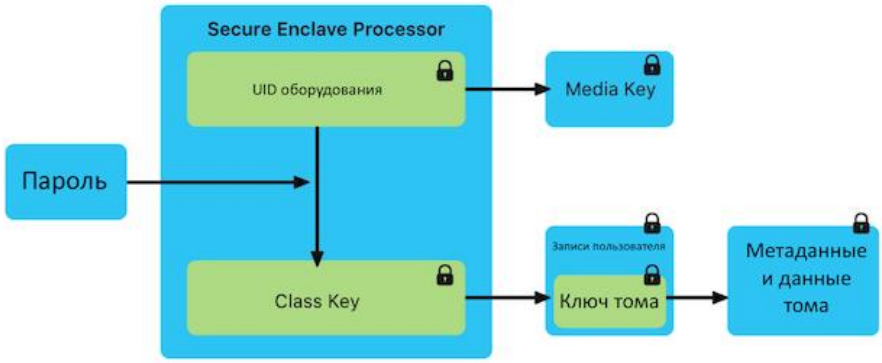

Рисунок 3.6 – Схема обробки ключа Secure Enclave Processor

Робота автентифікації TouchID / FaceID узгоджується за допомогою SEP представлена на рисунку 3.7.

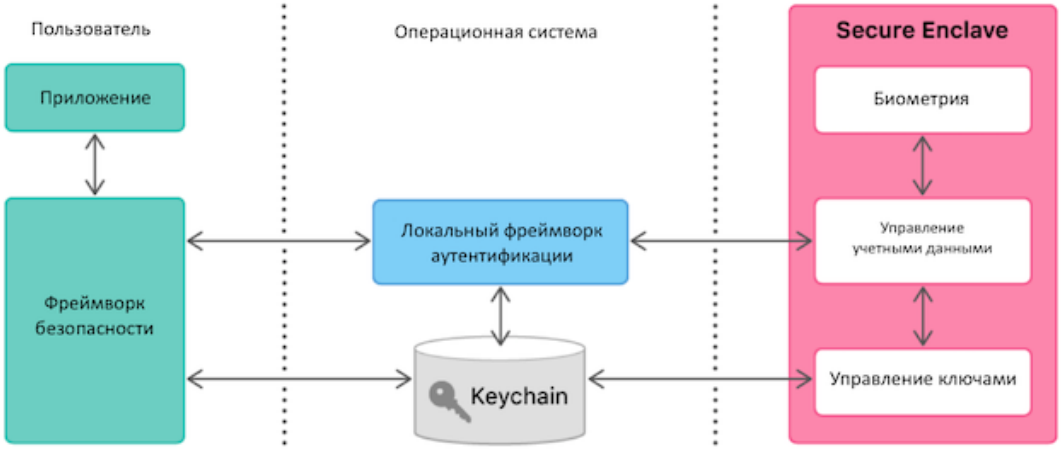

Рисунок 3.7 – Схема роботи Apple TouchID і FaceID

Зменшення фронту досягає обмежень шляхів вводу-виводу для заблокованого пристрою. Вам знадобиться один із цих способів доставки експлойту на ваш пристрій Apple: синхронізація фонових даних (електронна пошта, хмарні служби), push-повідомлення, деякі мережеві повідомлення (Wi-Fi, Bluetooth, стільниковий зв'язок) і порт Lightning на вашому пристрої . З введенням USB Restricted операційна система iOS обмежує доступ виявлених, підключених через USB / Lightning, таких як інструменти Forensic. iOS припиняє USB після першого підключення, пристрій не був прийнятий користувачем, як раніше. Рисунок правильний. Відомі надійні з'єднання через USB / Lightning зберігаються протягом 30 днів.

iMessage та FaceTime Apple підтримують наскрізне шифрування для відео/аудіо зв'язку між своїми пристроями за допомогою FaceTime. Він заснований на SRTP (безпечний транспортний протокол в режимі реального часу). iMessage використовує схему шифрування «signcryption». Apple Identity Service використовується для відкритих ключів - як довірений орган, що забезпечує ідентифікацію користувача.

Сучасний захист даних користувачів iOS Випуск нового пристрою Apple забезпечує покращення безпеки, які забезпечують зловмисників, спеціалістів з безпеки, які можуть отримувати шестизначні суми для виявлення вразливостей. Розглянемо основні сучасні методи обходу безпеки для iOS.

Джейлбрейк та програмне забезпечення часто використовуються для видалення обмеженої iOS і доступні для певного пристрою та версії iOS. Багато інструментів використовують його для доставки «корисного навантаження» різноманітним програмним компонентам, як правило, через USB / Lightning. Доставка може здійснюватися через повідомлення або через спеціально підготовлені веб-сторінки. Часто використовує ланцюжок експлойтів. Коли ви отримуєте контроль над ядром iOS, ви можете отримати дані, які не зашифровані. Подальша модифікація (патчування) програмного ядра дозволяє запускати будь-який код в системі.

Apple швидко реагує на уразливості, які призводять до джейлбрейку, що тримає ціну актуальних варіантів дуже високою. Jabbreak checkm8 / checkra1n є найефективнішим у 2020 році з точки зору досліджень і працює з пристроями до iPhone X та будь-якої версії iOS. Експлойти Cellebrite UFED Touch і 4PC можуть запускати резервне копіювання без авторизації користувача або активувати завантажувач за допомогою коду Cellebrite, щоб додати частини до файлової системи пристрою.

Вибір пароля Пароль інформація доступу до пристрою, необхідна для отримання ключів дешифрування класів CP і AFU.

До впровадження архітектури SEP обмеження щодо контролю вгадування та перевірки контролювалися процесором пристрою, що дозволяло здійснювати різні успішні атаки, які полягали у скиданні лічильників неправильно введених паролів.Попередня версія iOS могла відключати живлення для скидання лічильника, а в 2016 році була продемонстрована техніка, яка показувала відображення флеш-пам’яті NVRAM на iPhone 5C, щоб забезпечити необмежений введення пароля.

Обмеження вибору 80 мс робить безглуздими атаки для складних паролів із літерами та цифрами. Для 6-значного PIN-коду, який використовується для умов, обмеження SEP до вгадування є основною перешкодою. Оцінка часу перебору цифрових паролів наведено в iOS у таблиці 3.1.

Таблиця 3.1 – Оцінка часу перебору цифрових паролів в iOS

Довжина пароля	4 цифри	6 цифр	10 цифр
Всього варіантів	10 ^ 4 = 10 000	10 ^ 6 = 1 000 000	10 ^ 10
Всього використовується	9276	997090	
80 мс / спроба очікування	12,37 хвилини 6,19 хвилини	22,16 години 11,08 години	~25 років
10 хв. / спроба очікування	~70 днів ~35 днів	~20 років ~10 років	~200 000 років

Щоб взаємодіяти з потребою в привілеях ядра SEP. Джейлбрейк, ймовірно, потрібен у ланцюжку злому SEP. Після успішної атаки SEP зловмисник або спеціаліст з безпеки отримує необмежений доступ до шифрування та інших функцій безпеки пристрою, маючи можливість повністю розпакувати будь-які файли.

У 2018 році Grayshift представив інструмент GrayKey, який, за непідтвердженими даними, мігрує в обхід обмеження SEP. Існує ряд витоків, наприклад, у вигляді зламу пароля екрана блокування iPhone X за допомогою того ж інструменту. У січні 2020 року витік з ФБР повідомив, що GrayKey отримав доступ до заблокованого iPhone 11 Pro Max (iOS 13), який невіддільний для checkm8, залишаючи багато запитань: чи це робочий вибух чи компроміс SEP? Обхід блокування екрана Користувачі пристроїв можуть випадково виявити поведінку інтерфейсу iOS, що дозволяє обійти блокування екрана. Наприклад, це може бути поведінка під час доступу до камери, віджета погоди або годинника. Для відтворення успішного обходу потрібні точність і послідовність. Обхід блокування може відкрити несанкціонований доступ до фотографій, контактів, iTunes. Як правило, Apple швидко закриває такі обхідні шляхи.

Локальне отримання даних Стратегія отримання даних залежить від стану пристрою на момент захоплення: вимкнено чи увімкнено. При вилученні мобільних пристроїв правоохоронці Apple часто беруть різні пристрої для аналізу та супутні пристрої, такі як MacBook або Apple TV, використовують клітку Фарадея та автономний блок живлення, щоб не допустити підключення пристрою.

При компрометації iOS зловмисником останнім рівнем захисту даних буде шифрування, організоване Data Protection. Тому сьогоднішні методи отримання даних без згоди (або при недоступності) користувача можуть відбуватися по одному з трьох напрямків, перерахованих далі:

1. Доступ до пристрою за допомогою ключів. Якщо ключі шифрування завантажені в пам'ять, їх можна витягти. Наприклад, інструменти форензік Cellebrite UFED і XRY Logical дозволяють підключатися до пристроїв iOS через порт USB / Lightning або через Bluetooth, ініціювати резервне копіювання або переглянути файли.

2. Обхід захисних механізмів. У деяких випадках дані, недоступні для вилучення, можуть бути вилучені. Інструмент GrayKey, успішно отримав коди доступу користувачів, слугує прикладом. Якщо Ви знаходитесь в стані AFU (і навіть BFU), то відкрита можливість високочастотної атаки перебором пароля користувача. Успішна атака дозволить витягти всю файлову систему iOS, Keychain і вміст iCloud.

3. Альтернативні джерела даних. Тут застосовується атака на резервну копію iOS, яка зберігається на локальному комп'ютері і має порівняно слабкий захист від брутфорса. Також використовується можливість отримання даних користувача з хмарних сервісів.

Все більше і більше даних мобільних пристроїв синхронізується з хмарними сервісами, що зміщує акцент у бік отримання даних не з пристрою, а з хмари . Компрометація SEP може дозволити витягти Keychain, який відкриє доступ до служб iCloud, Slack, Twitter, Instagram, Facebook, Google, Uber, Dropbox. Ймовірно, дані маркери зберігаються в режимі AFU, щоб підтримувати функціонування при блокування пристрою, але це відкриває додаткові ризики, знижуючи захист SEP, так що тут доводиться покладатися тільки на безпеку ядра iOS.

Конфіскація суміжних пристроїв Apple (MacBook, Apple TV) може допомогти отримати дані для автентифікації в iCloud в поєднанні з іншими методами форензіки.

Виходячи з вищесказаного, сформулюємо декілька рекомендацій щодо підвищення захисту мобільних додатків iOS:

1) Посилення захисту даних. Apple розробила якісний фреймворк для захисту даних користувача, але поки що Data Protection не використовується в

максимально строгому режимі. Наприклад, токени аутентифікації для хмарних сервісів схильні до ризику вилучення з пам'яті. Клас захисту в стані AFU цілком надійний, але потрібно більш ретельна організація його роботи.

2) Динамічний захист даних. Може бути реалізована при взаємодії з користувачем на основі алгоритмів навчання в поєднанні з посиленням захисту до класу CP. iOS прогнозує, які ключі повинні застосовуватися і коли, своєчасно видаляючи з пам'яті неактуальні ключі шифрування і завантажуючи актуальні в міру необхідності.

3) Наскрізне шифрування резервних копій і iCloud. Apple зберігає ключі, які можуть дешифрувати дані бекапов в iCloud. Використовуючи iCloud Keychain, можна якісно організувати наскрізне резервне копіювання, недоступне для Apple, але доступне для будь–якого довіреної пристрої користувача. Це ж можна застосувати і для контейнера CloudKit, що значно підвищить безпеку зберігання даних користувача в хмарі.

4) Ліквідація особливих випадків обходу шифрування. Ключ наскрізного шифрування iMessage зберігається в резервної копії iCloud, до якої Apple має доступ. Дану лазівку слід усунути шляхом перенесення ключа в контейнер CloudKit із застосуванням сценарію «end–to–end».

5) Паролі на локальні бекапи. Рекомендується підвищення складності паролів локальних резервних копій для захисту від перебору. Можна реалізувати це шляхом оптимізації інтерфейсу або навчання користувачів.

6) Посилення iCloud Keychain.Слабке місце – кластер HSM, тому як немає прозорості шифрування з боку сервера. Тут Apple може застосувати технологію Certificate Transparency, при якій вузли перевіряють адреси і коректність HSM, публічно транслюють інформацію або діляться нею між собою.

7) Обмеження на USB–інтерфейс. Більш строгі обмеження при управлінні USB / Lightning на рівні ядра iOS можуть посилити безпеку і запобігати роботу експлойтів, наприклад checkm8.

8) Обмеження на режими DFU і JTAG. Користувачі можуть в будь–який момент перевести свій пристрій в режим DFU, і експлойти потенційно здатні

використовувати цю лазівку. Тут можна організувати автентифікацію із застосуванням криптографії. Аналогічні рекомендації можна застосувати до режиму JTAG, який, як правило, більшості користувачів немає необхідності використовувати.

9) Прозорість і функціональні протиріччя. Протиріччя в підході Apple до питання захисту даних користувача слід усунути. Наприклад, ставлячи / знімаючи опцію синхронізації даних з iCloud, користувач стикається з заплутаною політикою, з ризиком витоку даних. Це саме можна сказати і до інтерфейсу, до налаштувань і управління iCloud за замовчуванням.

10) Вплив iOS–спільнот. Величезна кількість користувачів і популярність iOS створюють базу любителів і дослідників різного рівня. Взаємодія з цими спільнотами (не тільки на рівні пошуку вразливостей), наприклад за допомогою відкритого вихідного коду, допоможе не тільки підсилити безпеку iOS, а й почути потреби користувачів, привести інновації.

4 АРХІТЕКТУРА ZKA ДЛЯ ЗАХИСТУ МОБІЛЬНИХ ДОДАТКІВ

4.1 Визначення Zero Knowledge Proofs

Термін «нульове знання» (ZK) часто використовується дуже вільно, але концепція реалізації ZK використовує шифрування даних, щоб гарантувати, що тільки постачальник даних може використовувати саму систему. Приклади включають Sync.com, pCloud, SpiderOak, Tresorit і MEGA, де шифрування з відкритим ключем використовується для хмарного зберігання та служб резервного копіювання, тому власник даних має приватний ключ, який можна використовувати для читання або розшифровки. місце, зашифроване його відкритим ключем, яким керує сервер. SpiderOak навіть намагається встановити набір стандартів нульової конфіденційності.

Альтернативні підходи до асиметричного хмарного сховища впроваджуються такими компаніями, як Storj і SAFE Network, які справді децентралізують і розподіляють персональне сховище за допомогою технології блокчейн і заохочують натовп зберігати і піклуватися про ваші дані.

Можливості для нових знань також включені в платформу бази даних. Відоме як гомоморфне шифрування, можна обчислювати та функціонувати бази даних за зашифрованими даними, не розшифровуючи їх. Microsoft, Google і SAP вже включили подібну функціональність у свою базу даних за допомогою відкритої бібліотеки продуктів, розробленої MIT, під назвою CryptDB.

Джерело терміну Zero-Knowledge походить з роботи трьох криптографів з Массачусетського технологічного інституту та Університету Торонто, які працювали над документом у 1985 році під назвою «Складність знань про інтерактивні доказові системи». Шафі Голдвассер, Сільвіо Мікалі та Чарльз Ракофф показали, що за допомогою ряду інтерактивних запитів постачальник знань, що перевіряє, може продемонструвати екзаменатору, що доказ знання є дійсним. Вони також переконалися, що під час цього процесу запитання чи

виклики не виявляють жодної інформації про інформацію, крім її правдивості. Результатом стала концепція доказів нульового знання.

Щоб переконатися, що дані були повними та точними, верифікатор грає в гру, подібну до 20 запитань, з доказувачем (постачальником знань), використовуючи математично замасковані фрагменти самих даних. Кожного разу, коли ставиться запитання (він же виклик), дані (знання) злегка перекручуються, щоб користувач не міг зібрати весь набір даних. Чим більше запитань дає правильний відповідь, тим більша ймовірність, що дані, які він має, повні й точні.

Доказ з нульовим знанням має задовольняти трьом властивостям:

Повнота: якщо твердження вірне, чесний верифікатор (тобто той, хто належним чином дотримується протоколу) буде переконаний у цьому факті чесним доповідачем.

Обґрунтованість: якщо твердження хибне, жоден доказувач шахрайства не зможе переконати чесного верифікатора, що воно істинне, за винятком певної невеликої ймовірності.

Нульове знання: якщо твердження істинне, жоден верифікатор не дізнається нічого, крім того факту, що твердження істинне. Іншими словами, достатньо знати твердження (а не секрет), щоб уявити сценарій, який показує, що доказувач знає секрет. Це формалізується, показуючи, що кожен верифікатор має певний симулятор, який, враховуючи лише твердження, що підлягає доказу (і не має доступу до доказувача), може створити стенограму, яка «виглядає як» взаємодія між чесним довідником і відповідним верифікатором.

Перші два з них є властивостями більш загальних інтерактивних систем доведення. Третє – це те, що робить доказ нульовим знанням. Докази з нульовим знанням не є доказами в математичному розумінні цього терміна, оскільки існує деяка невелика ймовірність, помилка обґрунтованості, що шахрайський довідник зможе переконати перевіряючого у хибному твердження. Іншими словами, докази з нульовим знанням є імовірнісними

«доказами», а не детермінованими доказами. Однак існують методи зниження похибки надійності до мізерно малих значень.

4.2 Використання ZKA для розробки мобільних додатків

Все, що відбувається в системі Zero Knowledge, шифрується перед сервером, а ключ шифрування ніколи не відкривається для відправки. Перший принцип ZKA — наскрізне шифрування для клієнтів, які використовують криптовалюту. Сервер нічого не знає про джерело даних.

По-друге, всі операції виконуються із зашифрованими даними. Це означає, що якщо користувач збирається додати новий запис до бази даних, його необхідно додати в зашифрованому вигляді. Якщо ви хочете поділитися деякими даними, погодьтеся надіслати їх у зашифрованому вигляді. Пошук вмісту зашифрованих даних.

Ці принципи не додають додаткового в загальному розумінні цього терміна, а лише забезпечують використання стороннього шифрування відповідно до безпеки. Алгоритми та протоколи з нульовим знанням гарантують, що ключі, паролі, файли чи будь-який інший конфіденційний матеріал ніколи не передаються в незашифрованому або оборотному вигляді. Немає часу, коли ключі шифрування або незашифровані файли будуть видимі для серверів або адміністраторів служб.

ZKA використовує криптографію і вимагає довіри до пристрою, який запускає криптокод. Ваш мобільний пристрій має безпечне середовище завдяки веб-переглядачу або більшості настільних комп’ютерів. Однак існують особливі ризики, описані в перших частинах.

У обміні повідомленнями використовується миттєвий обмін повідомленнями. Крім клієнтів, ніхто не може прочитати секретне повідомлення. Клієнти мають можливість довіряти один одному і серверу, щоб забезпечити криптографічний захист. Основою ZKA є наскрізна довіра без

витоку нічого на рівні зберігання чи передачі. Існує багато прикладів чатів E2EE.

Під час аутентифікації ви можете використовувати інтерактивні криптопротоколи, інформацію як протоколи з нульовим підтвердженням знань. ZKP дозволяє двом сторонам порівнювати секрет, не розкриваючи його, ефективно усуваючи витік секретів під час передачі.

Наприклад, якщо клієнт передає дані користувачам, вам потрібно зашифрувати ці ключі та передати ключі кожному користувачеві. Такий підхід призводить до дублювання даних: тепер клієнту потрібно заощадити більше, а якщо потрібно оновити дані або змінити політику доступу, потрібно повторно розшифрувати та зашифрувати деякі або всі записи. Але є кращі підходи до обміну даними: клієнт може додати доступ до певних блоків зашифрованих даних для певних користувачів. Отримувати доступ, шифрувати або повторно шифрувати лише те, що потрібно; і коли вам потрібно змінити політику доступу (до якої ви маєте доступ), повторно зашифруйте лише один невеликий блок доступу.

Кредитна історія є чудовим прикладом поширення конфіденційних даних між кількома організаціями. Бюро кредитних історій ведуть кредитну історію та за їх бажанням додають її в банк. Деталі кредитної історії містять багато конфіденційної інформації про життя користувачів і ваші інтереси, тому її поширення краще звести до мінімуму.

Зазвичай кредитна історія зберігається в зашифрованому вигляді, як єдиний блок даних, тож при витоку користувач втрачає все. Розбиття файлу на окремі зашифровані блоки та дозвіл банкам маніпулювати лише певними блоками мінімізує ризики.

4.2.1 Використання наскрізного шифрування в ZKA

Основа безпечної архітектури починається з шифрування, спеціального наскрізного шифрування. Система шифрує конфіденційні дані користувача, щойно він входить до будь-якого системного клієнта. Перш ніж вони зберігаються на пристрої, вони шифруються. Немає такої речі, як

незашифровані дані сховища, навіть URL-адрес для облікових записів клієнтів, за винятком випадків, коли ви контролюєте, переглядаючи інформацію в системному клієнті, де ви ввели свою адресу електронної пошти та головний пароль.

Звідти всі репозиторії залишаються зашифрованими, коли вони надсилаються в системну хмару або на серверну систему, розміщену на власному сервері. Синхронізуйте дані з іншими клієнтами після того, як вони залишаться зашифрованими, якщо у вас є адреса електронної пошти та головний пароль.

Це означає, що системи як компанії не можуть бачити ваші паролі, вони залишаються зашифрованими за допомогою спеціальної електронної пошти та головного пароля. Система не зберігає і не може отримати доступ до основного користувача користувача.

Дані використовують 256-бітне шифрування AES, галузевий стандарт, який неможливо зламати. Для головного PBKDF2 SHA-256 використовується для отримання ключа, який шифрує.

Важливою частиною наскрізного шифрування є ключ дешифрування. Поки воно залишається лише у кінцевого користувача, рішення може перейти до архітектури без знань. Іноді ряд постачальників програмного забезпечення та послуг просувають шифрування, але зберігають ключ. Ці люди не набувають нульової кваліфікації знань, як постачальники програмного забезпечення та послуг, технічних можливостей розшифровки.

4.2.2 Управління ключами в ZKA

Якщо користувачі мають контроль над ключем шифрування, вони контролюють доступ до даних і можуть надавати зашифровані сервери менеджера паролів, при цьому компанія з управління паролями не має доступу до даних або не знає про них.

Це фундаментальна передумова для добре розроблених менеджерів паролів. Вони надають надійні та унікальні паролі, доступ до яких маєте лише ви. Користувачам не потрібно знати секретні дані, тому користувачі повинні

контролювати ключ шифрування. Це називається шифруванням з нульовим знанням.

Але є інша інформація, крім секретних даних репозитарію, якою можна поділитися з програмним забезпеченням або постачальником послуг. Наприклад, адреса електронної пошти може служити унікальним ідентифікатором клієнта. Можна стверджувати, що це не нульове знання, і це було б правильно. Нульові знання повинні застосовувати секретні дані. У випадку з менеджером паролів це означає всю інформацію в сховищах паролів. При цьому важливо розуміти реалії програмного забезпечення, сервісів і користувачів, а також те, що існування комерційних відносин між сторонами вимагає обміну знаннями. У світі менеджерів паролів ця лінія може бути розмита. Як згадувалося раніше, це деякі менеджери паролів, які зберігають незашифровані URL-адреси, і веб-сайти, які зберігають паролі. Хоча це перевіряється користувачем, воно в кінцевому підсумку надає детальну інформацію про те, які веб-сайти відвідують користувачі, хто вони, і кожен вхід.

4.2.3 Реалізація роботи ZKA над спільними даними

Модель нульової довіри спочатку виникла як спосіб для організацій вийти за межі традиційного уявлення про внутрішні та зовнішні загрози їхнім IT–операціям. Сьогодні компаніям необхідно захищатися від загроз, що надходять як зсередини, так і ззовні. У моделях з нульовою довірою часто використовуються такі технології, як керування ідентифікацією та доступом, шифрування, багатофакторна аутентифікація та дозволи для роботи. Модель роботи такої системи представлено на рисунку 4.3.

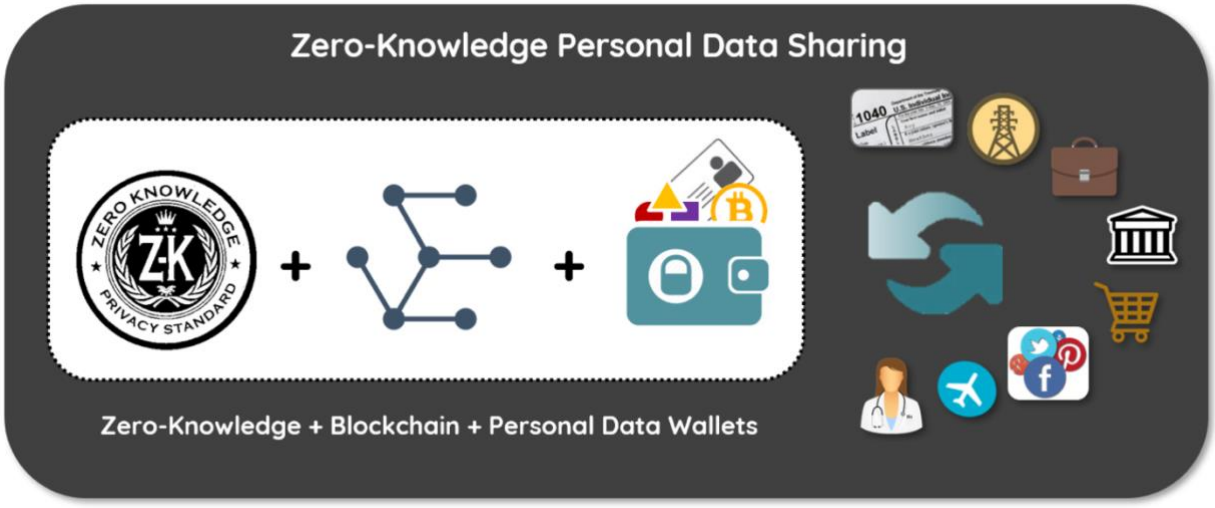

Рисунок 4.2 – Демострація роботи ZKA з персональними даними

Звичайно, між менеджерами паролів і користувачами, які використовують програмне забезпечення або послуги, імовірно, буде хоча б якийсь елемент довіри між двома сторонами. Постачальник керування паролями вірить, що користувач не порушуватиме умови обслуговування, а користувач вірить, що постачальник керування паролями виконає заявлену пропозицію. Проте всім краще, якщо межі необхідної довіри обмежені, так що навіть можливість скомпрометації конфіденційних даних повністю виключена, звідси модель нульової довіри.

Незважаючи на те, що ми підтримуємо наших клієнтів у довірливих відносинах, ми можемо зменшити залежність від неявної довіри через пропозицію системи. Таке розгортання дає підприємствам більшу гнучкість і контроль над своєю інфраструктурою. Запуск власного екземпляра системі може бути в мережі повітряного проміжку, що ще більше знижує ризики, якщо користувач від’єднається від Інтернету.

Розглянемо кроки реалізації ZKA для мобільних додатків:

1) обгортання ключів: поділ одного блоку даних на менші зашифровані блоки. Відокремлення ключів зберігання від ключів користувача, шифрування даних за допомогою ключів зберігання;

2) керування привілеями: зберігати ключі доступу в контейнерах; криптографічно контролювати те, що користувачі можуть змінювати збережені ключі, порівнюючи фактичні теги власності (автентифікований MAC), щоб довести, що вони можуть виконувати операцію;

3) контрольні запити: оскільки дані доступні лише через певний крипторівень, користувачі повинні переконатися, що архітектура перекладає кожну операцію над набором даних у команди крипто API; щоб ніхто не мав змоги розмістити дані у відкритому тексті, минаючи ядро системи;

4) захист від інших атак: криптографія лише звужує набір варіантів для атаки. Системі все одно потрібні резервні копії; потрібно використовувати ZKP для захисту від атак повторного відтворення, потрібно забезпечити довіру коду, безпечний життєвий цикл ключів і використовувати традиційні речі, такі як виявлення вторгнень, моніторинг та перевірка трафіку;

Можливо використовувати підхід ZKA скрізь, де потрібно надати безпечний доступ до невеликих блоків даних, які спільно використовують різні клієнти. Складні документи з коментарями або детальними таблицями (наприклад, Google Docs, Dropbox Paper тощо). У багатьох випадках користувачі не повинні мати доступ до всього документа. Файлові системи є прекрасним прикладом невеликих блоків даних, структурованих і спільних з різними правами контролю доступу. Якщо кожен BLOB у базі даних є захищеним, а права доступу захищені, то створюється наскрізне сховище документів, де права на кожен документ або поле можуть бути надані кожному.

5 РОЗРОБКА ДОДАТКУ ДЛЯ ОБМІНУ ЗАПИСАМИ

5.1 Мова програмування

Протягом багатьох років основна мова програмування для розробки мобільних додатків на iOS була Objective-C [20], своєрідна мова побудована на основі мови C з використанням можливостей ООП і динамічною типізацією. Але в липні 2014 року все змінилося. Apple була абсолютно новою об’єктно-орієнтованою мовою програмування Swift[22], яка завдяки зручному синтаксису та новим функціям значно покращила життя iOS–розробників та допомогла збільшити їх кількість у всьому світі.

Основні переваги Swift над Objective–C:

- класи більше не розбиваються на інтерфейс і його реалізацію;
- спрощення синтаксису створення полів і властивостей класів;
- Optional і Generics типи;
- підтримка функціонального програмування;
- збільшена безпека управління пам’яттю;
- підвищена читабельність коду і зменшення його кількості;
- повноцінна взаємодія з кодом написаним на Objective–C;
- підтримка динамічних бібліотек.

Наразі Swift стабільно входить в 15 найпопулярніших мов програмування у світі за індексом TIOBE, який оцінює популярність мови програмування на основі обчислення результатів пошукових запитів, що містять згадку про мову. Аналізуючи дані за квітень 2020 року Swift займає 11 сходинку, що свідчить про високу популярність для такої, відносно молодої по сучасним міркам, мови.

5.2 Середовище розробки

З 2003 року Xcode був єдиним повноцінним інтегрованим середовищем розробки програмного забезпечення для iOS, watchOS, tvOS і MacOS. Розширюється безкоштовно через онлайн-магазин App Store. Це середовище розробки містить всі необхідні інструменти і надає інструменти для успішної розробки, проектування, тестування, налаштування інтерфейсу різних додатків для різних продуктів Apple. Очевидною перевагою Xcode є наявність Interface Builder - інтегрованого в Xcode додатка, що містить інструменти для створення графічних інтерфейсів. Завдяки простоті та зручності використання весь графічний інтерфейс можна зробити в Interface Builder, а потім зв’язати всі візуальні елементи з файлами реалізації, так званими контролерами, що описує логіку взаємодії з ними. Це середовище також включає інструменти для налагодження програм, які можуть виявляти помилки як у коді, так і в інтерфейсі та працювати з гілкою пам'яті; емулятор віртуального пристрою та напрямок запуску та тестування програм на власному смартфоні; вбудована система контролю версій Git; функції моделювання геолокації, так необхідні при розробці додатків, що взаємодіють з картами; Swift Playgrounds — це середовище розробки Swift, яке дозволяє швидко закрити фрагмент коду, щоб перевірити його, або перевірити роботу певного алгоритму без компіляції всього проекту.

Звичайно, це не єдине інтегроване середовище розробки для програмування iOS, але єдине, яке може повністю створити додаток будь-якого рівня складності, без використання сторонніх інструментів і бібліотек. Xcode регулярно оновлюється, офіційно підтримується Apple і має велику документацію, тому я вибрав його як середовище розробки для свого мобільного додатка.

5.3 Опис шаблону проектування

Сьогодні існує багато шаблонів проектування, але одним з найпопулярніших і найпоширеніших з них Model–View–Controller (MVC) [16].

MVC – схема проектування архітектури програмної системи, де модель, користувацький інтерфейс та взаємодія з користувачем розділені на три окремих рівні: рівень моделі, представлення і контролеру, кожен з яких виконує свою функцію, як показано на рисунку 5.1.

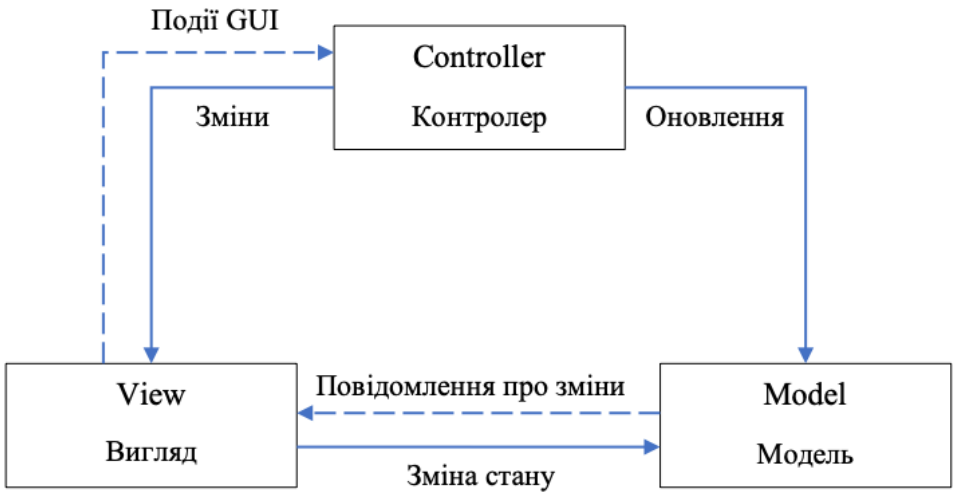

Рисунок 5.1 – Схема взаємодії в MVC

Модель – це частина архітектури додатку, яка відповідає за керування даними. Контролер реагує на дії користувача й оновлює вигляд, використовуючи зміни в моделі. Вигляд відповідальний за правильне представлення даних з моделі за допомогою графічного користувацького інтерфейсу або іншими словами – GUI.

Традиційна архітектура MVC, відрізняється від архітектури, яка використовується при сучасній розробці iOS–додатків. У традиційній архітектурі MVC всі три сутності тісно пов’язані між собою, що суттєво знижує можливість повторного використання кожного з цих компонентів. Через цей

чинник реалізовувати її у середовищі операційної системи iOS немає ніякого сенсу.

В офіційній документації Apple зазначений шаблон MVC, який був названий розробниками як Cocoa MVC[17]. Його схема проектування зображена на рисунку 5.2. В цьому шаблоні контролер являється посередником між рівнем представлення і моделлю, які нічого не знають про існування один одного і не можуть взаємодіяти між собою. Також особливістю цього шаблона проектування архітектури є те, що контролер UIViewController настільки залучений у життєвий цикл представлення UIView, що іноді важко сказати, що ці два рівні дійсно розділені між собою.

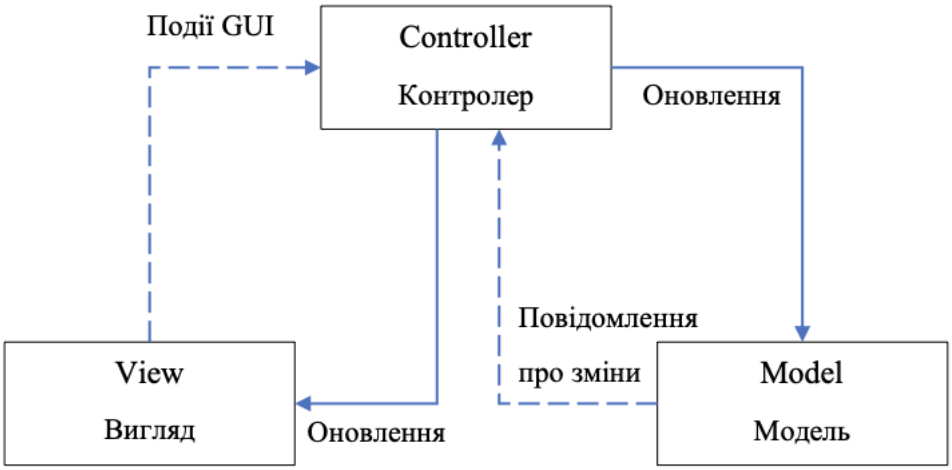

Рисунок 5.2 – Схема взаємодії в Cocoa MVC

Cocoa MVC є найкращим шаблоном проектування з точки зору швидкості реалізації і простоти використання. Він незамінний в більшості мобільних додатків невеликої складності. Тому даний шаблон був використаний під час розробки архітектури для мого мобільного застосунку.

5.4 Використані бібліотеки

У процесі розробки я використав менеджер залежностей CocoaPods[10]. Він допомагає iOS–розробникам легко додавати сторонні бібліотеки, код яких

знаходиться у відкритому доступі. CocoaPods створює єдину централізовану систему керування залежностями в проекті. Після додавання до нього нової бібліотеки, назва якої вказується в єдиному текстовому файлі «Podfile» на рисунку 5.3, менеджер автоматично слідкує за її подальшою підтримкою і оновленнями.

```
target "SecurePostsExample" do
  pod 'themis', "0.10.0"
  pod 'Firebase/Core'
  pod 'Firebase/Auth'
  pod 'Firebase/Database'
  pod 'FirebaseUI/Database'
end
```

Рисунок 5.3 – Podfile мого проекту

Google Firebase – це програмне забезпечення для розробки додатків, яке підтримується Google, яке дає змогу розробникам розробляти iOS, iOS та веб–програми. Firebase надає інструменти для відстеження аналітики, звітування та виправлення збоїв додатків, створення маркетингу та експерименту з продуктом. Firebase пропонує ряд послуг, зокрема:

1) Аутентифікація. Аутентифікація Firebase полегшує розробникам створення безпечних систем аутентифікації та покращує вхід та підключення для користувачів. Ця функція пропонує повне рішення для ідентифікації з підтримкою облікових записів електронної пошти та паролів, аутентифікації телефону, а також входу в Google, Facebook, GitHub, Twitter тощо.

2) Realtime database – база даних Firebase Realtime Database –це хмарна база даних NoSQL, яка дозволяє зберігати та синхронізувати дані між користувачами в режимі реального часу. Дані синхронізуються між усіма клієнтами в режимі реального часу і залишаються доступними, коли програма переходить в автономний режим.

Themis –це бібліотека високорівневих криптографічних послуг з відкритим кодом для захисту даних під час аутентифікації, зберігання, обміну повідомленнями, мережевим обміном тощо. Themis вирішує 90% типових

випадків використання захисту даних, які є загальними для більшості програм. Themis допомагає створювати як прості, так і складні криптографічні функції легко, швидко та безпечно. Themis дозволяє розробникам зосередитися на головному: розробці своїх додатків.

5.5 Налаштування Firebase

В роботі було використано Firebase в якості віддаленої бази даних. Користувачі повинні зареєструватися, після чого вони зможуть публікувати нотатки. Власні нотатки відображаються на сторінці "Мої дописи", нотатки всіх користувачів – на сторінці "Усі дописи". Щоб налаштувати Firebase було викано наступні кроки:

1. Змінити назву bundle.
2. Використано офіційний посібник з налаштування Firebase.
3. Зареєстровано додаток на консолі додатків Firebase.
4. Отримано GoogleService–Info.plist і додано його до проекту.
5. У налаштуваннях програми на консолі Firebase увімкнено автентифікацію електронною поштою, як представлено на рисунку 5.4.
6. Увімкнути базу даних Relatime у тестовому режимі.

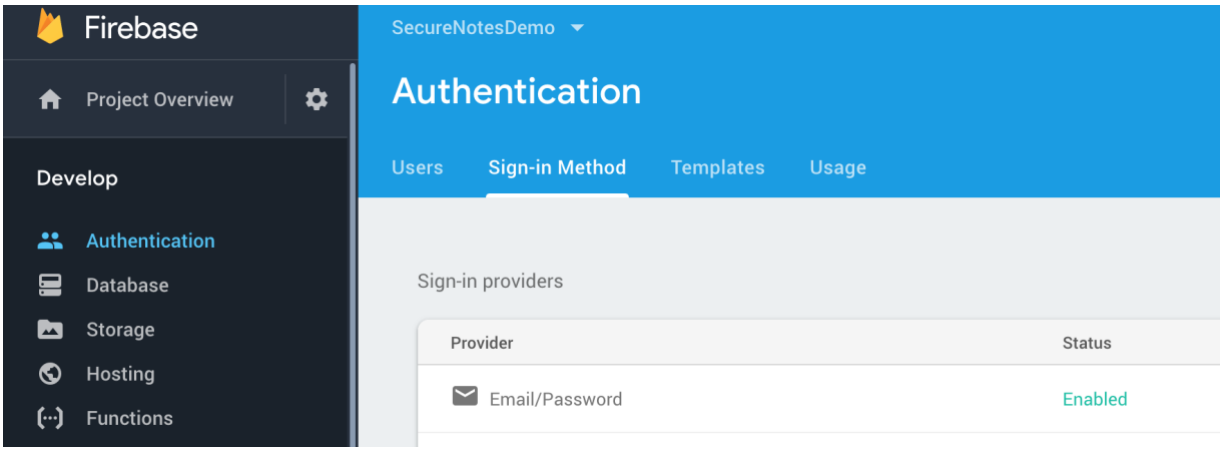

Рисунок 5.4 – Демонстрація налаштування FireBase

5.6 Шифрування повідомлень

Нотатки можуть бути містити конфіденційну інформацію, тому система шифрує їх, щоб зробити видимими лише для авторів. Для цього було обране симетричне шифрування: використовується один ключ для шифрування та дешифрування нотатків. Кожен користувач має свій унікальний ключ.

Шифрування повідомлення:

1) Створення секретного ключа користувача (SK). Зберігання його в надійному місці. Використання KDF, щоб зробити секретний ключ міцнішим.

2) Шифрування тексту повідомлення за допомогою SK та AES.

3) Кодування тіла повідомлення до base64 із процентним кодуванням (для безпечної передачі через мережу).

4) Зберігання допису у серверній частині.

Розшифрування повідомлення:

1) Отримання секретного ключу користувача (SK).

2) Декодування тіла повідомлення з base64 та видалення відсоткового кодування.

3) Розшифрування тексту повідомлення за допомогою SK та AES.

4) Показ розшифрованої нотатки.

Було використано Themis Secure Cell в режимі Seal для постшифрування. Він має попередньо вбудований KDF і використовує AES256 GCM. Дані будуть зашифровані та додані тегом auth, що захища їх від підробки.

Створення SecureCell:

```
guard let cellSeal = TSCellSeal(key: secretKey.data) else {
 print("Failed to encrypt post: error occurred while initializing object cellSeal")
 throw EncryptionError.cantCreateSecureCell
}
```

Шифрування тіла повідомлення:

```
let encryptedMessage: Data
do {
```

```
        encryptedMessage = try cellSeal.wrap(postBody.data(using: .utf8)!,
                                             context: nil)
      } catch let error as NSError {
        print("Failed to encrypt post: error occurred while encrypting body \(error)")
        throw EncryptionError.cantEncryptPostBody
      }
      return EncryptedData(data: encryptedMessage)
```

Розшифрування зашифрованого тіла повідомлення:

```
    var decryptedMessage: Data = Data()
    do {
      decryptedMessage = try cellSeal.unwrapData(encryptedPost.data,
                                                 context: nil)
    } catch let error as NSError {
      print("Failed to decrypt post: error occurred while decrypting: \(error)")
      throw EncryptionError.cantDecryptPostBody
    }
```

Розшифрування зашифрованого повідомлення :

```
    guard let decryptedBody = String(data: decryptedMessage, encoding: .utf8) else {
        print("Failed to decrypt post: error occurred while encoding decrypted post body")
        throw EncryptionError.cantEncodeDecryptedPostBody
      }
      return decryptedBody
```

З метою демонстрації було розшифровано повідомлення лише на сторінці PostDetails, залишаючи їх зашифрованими у загальному списку, як представленно на рисунку 5.5.

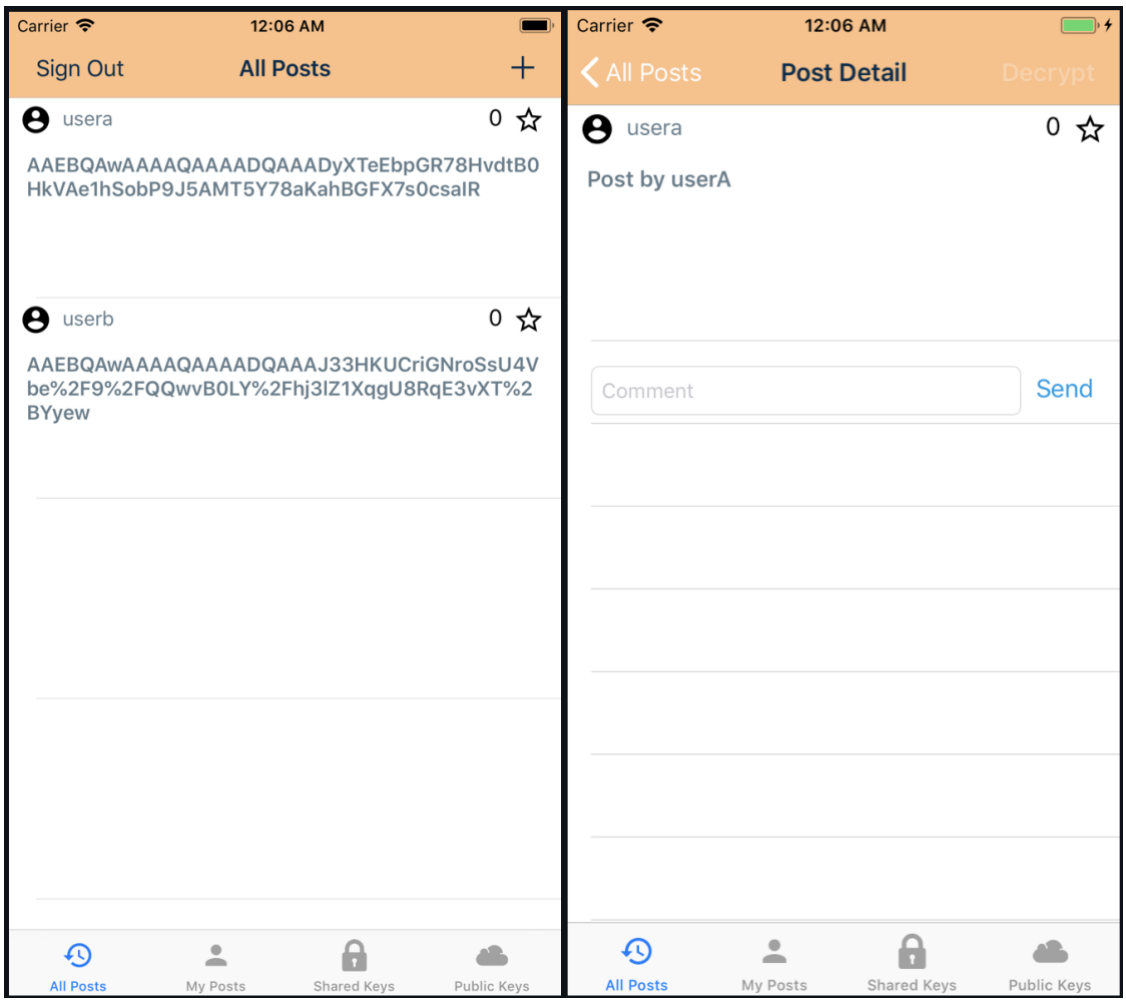

Рисунок 5.5 – Демонстрація роботи додатку

На рисунку 5.6 приведено стан роботи бази даних. Як зазначалось, система створена таким чином, що нотатки користувачів не зберігаються у відкритому вигляді.

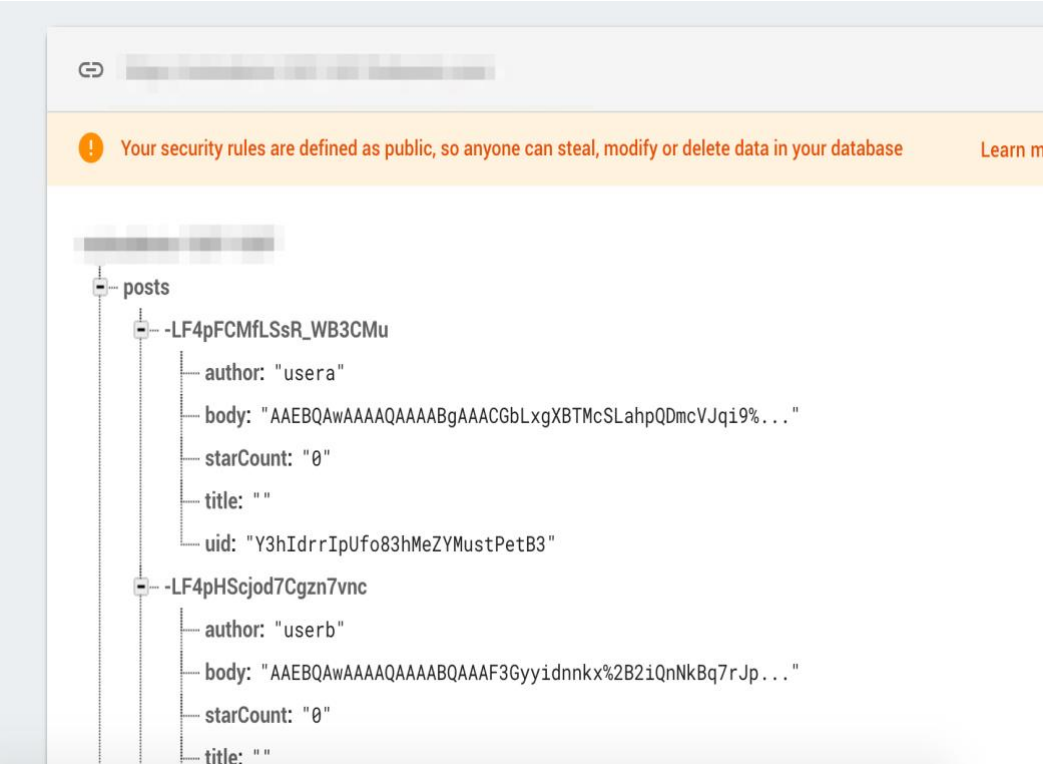

Рисунок 5.6 – Демонстрація роботи FireBase

5.7 Обмін зашифрованими повідомленнями між користувачами

Користувачам важлива функцію обміну повідомленнями. Кожна нотатка зашифрована, тому користувачі повинні ділитися секретними ключами. Але ключі дуже чутливі, додаток не можемо поділитися ними відкритим текстом. Тому система використовує асиметричне шифрування, щоб обернути SK, використовуючи власний закритий ключ і відкритий ключ іншого користувача.

Користувач Аліса ділиться своєю зашифрованою публікацією та зашифрованим секретним ключем для користувача Боба:

1. Має власний секретний ключ SK–a.
2. Шифрує власні повідомлення за допомогою SK–a:
3. EncrPost–a = SecureCell(Post–a, SK–a)
4. Ділиться своїм відкритим ключем з Бобом (Pub–a).
5. Шифрує SK–a для свого друга Боба SharedKey–(a–>b), використовуючи власний закритий ключ (Priv–a) і відкритий ключ Боба (Pub–b):

6. SharedKey–(a–>b) = SecureMessage(SK–a, Priv–a, Pub–b)

Користувач Боб хоче прочитати повідомлення від користувача Аліси:

1. Ділиться своїм відкритим ключем з Алісою (Pubk–b).
2. Розшифровує секретний ключ Аліси, використовуючи власний закритий ключ (Priv–b) та її відкритий ключ (Pub–a):
3. SK–a = SecureMessage(SharedKey–(a–>b), Priv–b, Pub–a)
4. Розшифровує допис Аліси за допомогою її секретного ключа:
5. Post–a = SecureCell(EncrPost–a, SK–a)

Зазвичай відкриті ключі зберігаються в PKI (інфраструктура відкритих ключів), де вони зберігаються від несанкціонованого доступу. Користувач Аліса повинен довіряти службі PKI, щоб бути впевненим, що Pub–b дійсно належить Бобу. Існують різні способи налаштування PKI, наприклад, за допомогою служби посередника, наприклад keybase.io. У нашому прикладі ми збережемо відкритий ключ на тому самому сервері.

Будемо використано Themis Secure Message для шифрування асиметричних ключів. Кожен ключ буде зашифрований і підписаний користувачем за допомогою пари ключів EC.

Шифрування власного SK:

1. Створення власної пари ключів. Підготуйте власний закритий ключ як дані. Підготуйте власний секретний ключ як дані.

2. Отримайте відкритий ключ іншого користувача як String, декодуйте його з base64 і видаліть екранування відсотків.

3. Створіть контейнер SecureMessage з власним приватним ключем та іншим відкритим ключем користувача.

4. Шифруйте власний секретний ключ за допомогою Secure Message для іншого користувача.

5. Закодуйте SK в рядок (додайте кодування base64 і відсоткове кодування).

6. Надати доступ до бекенду.

Створення пару ключів:

```
func generateMyKeyPair() throws –> KeyPair {
 guard let keyGeneratorEC: TSKeyGen = TSKeyGen(algorithm: .EC) else
{
  print("Error occurred while initializing object keyGeneratorEC")
  throw EncryptionError.cantCreateKeyGenerator
 }

 let privKey = Key(data: keyGeneratorEC.privateKey as Data)
 let pubKey = Key(data: keyGeneratorEC.publicKey as Data)

 return KeyPair(privateKey: privKey, publicKey: pubKey)
}
```

Створення контейнеру SecureMessage з власним приватним ключем та іншим відкритим ключем користувача:

```
guard let encrypter = TSMessage.init(inEncryptModeWithPrivateKey:
myPrivateKey.data.
peerPublicKey: userPublicKey.data) else {
                print("Error occurred while creating TSMessage
Encryptor")
                throw EncryptionError.cantCreateSecureMessage
 }
```

Шифрування власного секретного ключа для іншого користувача:

```
do {
   return EncryptedData(data: try encrypter.wrap(mySecretKey().data))
 } catch let error as NSError {
  print("Failed to encrypt own SK: error occurred while encrypting:
\(error)")
  throw EncryptionError.cantEncryptOwnSecretKey
 }
```

Розшифрування іншого повідомлення користувача:

1. Отримати власну пару ключів. Підготуйте власний закритий ключ як дані.
2. Підготуйте інший зашифрований SK як рядок
3. Створіть контейнер SecureMessage з власним приватним ключем та іншим відкритим ключем користувача.
4. Декодувати зашифрований SK до даних (вилучити відсоткове кодування, створити об’єкт даних із рядка, закодованого base64).
5. Розшифрування SK іншого користувача за допомогою Secure Message.

6. Кодування SK (кодування користувача base64 і кодування відсотка).

7. Розшифрування повідомлення іншого користувача, використовуючи його секретний ключ.

8. Показати розшифрований пост.

ВИСНОВОК

При розробці мобільного додатку слід враховувати, що дані, якими оперує цей додаток, можуть представляти певний інтерес для третіх осіб. Ступінь цінності цих даних варіюється в широких межах, проте, навіть найбільш проста приватна інформація, наприклад, пароль входу в додаток, вимагає опрацювання її захисту. Особливо це важливо в світлі поширення мобільних додатків на всі сфери електронних послуг, включаючи фінансові, банківські операції, зберігання і передачу особистих даних і так далі.

Мобільні пристрої – приваблива мішень для хакерських атак, адже в них зберігаються і обробляються великі обсяги особистої інформації і дані банківських карт. Як показують результати нашого дослідження, при розробці мобільних додатків питань безпеки приділяється недостатньо уваги, і основна проблема пов'язана з небезпечним зберіганням даних. Зберігання інформації користувачів у відкритому вигляді, незахищені дані на знімках стану екрану, ключі і паролі в вихідному коді – це лише деякі з тих недоліків, що відкривають простори для проведення кібератак.

Користувачі можуть самі сприяти компрометації своїх пристроїв: розширювати стандартні можливості смартфона, позбавляючи його захисту, переходити за підозрілими посиланнями в SMS повідомленнях, завантажувати програми з неофіційних джерел, тому безпека призначених для користувача даних – відповідальність не тільки розробників додатків, але і самих власників мобільних пристроїв.

В роботі була запропонована модель захисту iOS-додатку від несанкціонованого використання, яка відрізняється своєю багаторівневою структурою з перекриттям загроз, що дозволяє організувати ефективний захист доступу до додатку на основі багатофакторної автентифікації, захисту програмного коду на основі обфускації та захисту даних додатку на основі

віддаленого контролю та управління ним, а також збереження даних в захищеній базі даних.

На основі запропонованої моделі розроблено архітектуру системи захисту мобільних додатків. Система містить модуль обфускації коду додатку для захисту від реверс–інженірингу, модуль автентифікації для контролю доступу до додатку, модуль віддаленого контролю додатку та управління даними, модуль захисту БД для шифрування даних додатку.

На основі запропонованої архітектури системи розроблено модуль для захисту від несанкціонованого використання iOS-додатку, який дозволяє забезпечити захист від загроз конфіденційності, цілісності та доступності. Розроблений модуль дозволяє зменшити тривалість розробки проектування при реалізації вимог щодо безпеки. При цьому система захисту є гнучкою, тобто при розробці додатку можна до нього інтегрувати той чи інший модуль захисту залежно від встановлених вимог.

Ризики пов'язані не тільки з вразливостями в захисті клієнта або сервера, але і з уразливими, які виникають в процесі клієнт–серверного взаємодії. Обмін даними з відкритого протоколу загрожує повною компрометацією переданого трафіку. Але навіть захищені з'єднання не завжди надійні, що говорить про відсутність у розробників глибокого розуміння важливості питань безпеки. ZKA – це підхід до проектування, який вирішує проблеми довіри до сервера та середовища передачі. Це досить легко вирішити, коли взаємодії прості і стають досить складними, коли ми говоримо про бази даних або співпрацю. В цій роботі показано кілька прикладів вирішення типових проблем за допомогою ZKA. Також було розроблено мобільний додаток для обміну записами, що демонструє переваги використання ZKA для передачі даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Voitovych O.P ., Hurskyi M.V ., Snigovyy D.S., Kupershtein L.M. “Monitoring tool for Android operating system”, in Scientific journal Herald of Khmelnytskyi national university 2017. Issue 3, Volume 249. 236–241 p. 2. Tabassum, Gulista, Shikha Pandit, and Nupur Ghosh. "Android Application Security", in Journal of Emerging Technologies and Innovative Research. Vol. 1. No. 7. 2014.

2. Kupershtein L.M., Voitovych O.P., Kaplun V. A., Pro– kopchuk S.O. “The database–oriented approach to data protection in Android operation system”, in Scientific journal Herald of Khmelnytskyi national university 2018, Issue 1, 18–22 p.

3. Zhang, N., Yuan, K., Naveed, M., Zhou, X., & Wang, X. “Leave me alone: App–level protection against runtime information gathering on android” In Security and Priva– cy (SP), 2015 IEEE Symposium on (pp. 915–930). IEEE. 5. Hassanshahi, B., & Yap, R.H. “Android Database At– tacks Revisited”. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. P. 625–639.

4. Baryshev, Y., Kaplun, V. and Neiyumina, K. “Discre– tional model and method of distributed information re– sources access control”. In Scientific Works of Vinnytsia National Technical University. 2 (Jun. 2017).

5. Kim, N. Y., Shim, J., Cho, S. J., Park, M., & Han, S. “Android Application Protection against Static Reverse Engineering based on Multidexing”. J. Internet Serv. Inf. Secur., 6(4), 54–64 (2016).

6. Dong, S., Li, M., Diao, W., Liu, X., Liu, J., Li, Z., & Zhang, K. “Understanding Android Obfuscation Tech– niques: A Large–Scale Investigation in the Wild”. arXiv preprint arXiv:1801.01633 (2018).

7. Baryshev Yu., Kaplun V. “Remote user authentication method for network services” in Information Technology and Computer Engineering. 2014 Vol.

8. Введение в информационную безопасность автоматизированных сис–тем:учебное пособие / В. В. Бондарев. –Москва : Издательство МГТУ им. Н. Э. Баумана, 2016. –250, [2] с. : ил.

9. Official Certified Ethical Hacker Review Guide: Kimberly Graves, Copyright © 2007 by Wiley Publishing, Inc., Indianapolis, Indiana, ISBN–13: 978–0–7821–4437–6.

10. Identity and Data Security for веб Development: Best Practices – USA: "O'Reilly Media, Inc.", 2016. – 204 с. – ("O'Reilly Media, Inc."). – (ISBN 1491936967, 9781491936962).

11. Emerging Technologies for Authorization and Authentication: First International Workshop – Barcelona: Springer, 2018. – 138 с. – (Springer). – (ISBN–13: 978–3030043711).

12. Client–Side Attacks and Defense, 2012. – 296 с. – (Sean–Philip Oriyano, Robert Shimonski). – (ISBN 1597495905, 9781597495905).

13. Mastering Modern веб Penetration Testing, 2016. – 298 с. – (Packt Publishing Ltd). – (ISBN 1785289144, 9781785289149).

14. The Basics of веб Hacking: Tools and Techniques to Attack the веб, 2013. – (Elsevier). – (ISBN 0124166598, 9780124166592).

15. SQL Injection Attacks and Defense, 2009. – 496 с. – (Syngress). – (ISBN 0080958575).

16. The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities, 2006. – 1200 с. – (Pearson Education). – (ISBN 0132701936).

17. DDoS Attacks: Evolution, Detection, Prevention, Reaction, and Tolerance, 2016. – 312 с. – (CRC Press). – (ISBN 1498729657).

18. Inside the multitouch FingerWorks tech in Apple's tablet [Електронний ресурс]. – 2010. – Режим доступу до ресурсу: https://appleinsider.com/articles/10/01/23/in–side_the_multitouch_fingerworks_tech_in_apples_tablet/page/3.

19. Neuburg M. iOS 13 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics / Matt Neuburg. – O'Reilly Media, 2019. – 680 с.

20. TIOBE Index [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe–index/.

21. Xcode Overview [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/index.html#//apple_ref/doc/uid/TP40010215–CH24–SW1.

22. Model–View–Controller [Електронний ресурс] – Режим доступу до ресурсу: https://patterns.arc42.org/patterns/mvc/#mvc–triads.

23. Model–View–Controller (Cocoa MVC) [Електронний ресурс] – Режим доступу до ресурсу:

https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia–CocoaCore/MVC.html#//apple_ref/doc/uid/TP40008195–CH32–SW1.