DOI 10.58351/2949-2041.2026.35.6.008
УДК 004.056.5

Дедюхин Никита Андреевич, магистрант
Волгоградский государственный университет

РАЗРАБОТКА ЛОКАЛЬНОГО ANDROID-ПРИЛОЖЕНИЯ ДЛЯ ЗАЩИЩЕННОГО ХРАНЕНИЯ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ

Аннотация. В статье рассматривается разработка локального Android-приложения SecureHub для защищенного хранения пользовательских данных. Описаны архитектура приложения, использование SQLCipher и Android Keystore, локальная аутентификация, уровни защиты секретов и журнал жизненного цикла. Предложенный подход ориентирован на хранение данных без серверной части и облачной синхронизации.

Ключевые слова: Android; защищенное хранение; SQLCipher; Android Keystore; движок правил доступа; журнал жизненного цикла.

ВВЕДЕНИЕ

Мобильные устройства используются для хранения паролей, токенов доступа, личных заметок, документов и других конфиденциальных данных. При этом Android-приложения могут подвергаться угрозам, связанным с неправильным выбором места хранения, ошибками в работе с разрешениями, доступом к общим каталогам и небезопасным использованием криптографических механизмов [1, 3]. Поэтому задача локального защищенного хранения данных остается актуальной для мобильной разработки.

Исследования безопасности Android показывают, что утечки данных часто возникают не только из-за уязвимостей операционной системы, но и из-за архитектурных решений самих приложений. К таким решениям относятся сохранение чувствительной информации в открытом виде, использование внешнего или общего хранилища, отсутствие защиты ключевого материала и недостаточный контроль действий пользователя [4, 5].

Целью работы является разработка Android-приложения для локального защищенного хранения пользовательских данных. В отличие от облачных менеджеров паролей, рассматриваемое приложение не требует сервера и учетной записи. Основной акцент сделан на шифровании локального хранилища, локальной аутентификации и контроле использования секретов.

АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Приложение SecureHub разработано для операционной системы Android с использованием языка Kotlin. Архитектура разделена на несколько слоев: пользовательский интерфейс, доменная логика, слой данных и низкоуровневые механизмы безопасности. Такое разделение позволяет изолировать работу с базой данных, криптографией и правилами доступа от экранов приложения.

Пользовательский интерфейс реализуется на Jetpack Compose. Состояние экранов передается через ViewModel и StateFlow, поэтому бизнес-логика не размещается внутри Composable-функций. Доменный слой содержит модели секретов, интерфейсы репозиториев и policy engine (движок правил доступа), который принимает решение о доступности действий с учетом уровня защиты объекта.

Основные типы данных приложения включают пароли, токены, учетные данные для автозаполнения, приватные заметки и файлы. Для текстовых секретов в базе данных хранятся заголовок, тип, содержимое, временные метки и поля, связанные с автозаполнением. Для файлов в базе сохраняются только метаданные, а само содержимое размещается в защищенном файловом контейнере.

ЗАЩИЩЕННОЕ ХРАНЕНИЕ ДАННЫХ

В качестве основного хранилища используется Room поверх SQLCipher. Room выступает как удобный слой доступа к SQLite, а SQLCipher обеспечивает шифрование базы данных на уровне страниц. Такой подход позволяет сохранить преимущества реляционного хранилища и одновременно защитить содержимое базы от прямого чтения файла [7].

Ключевой материал не должен храниться в открытом виде. Для его защиты применяется Android Keystore, предназначенный для генерации и хранения криптографических ключей внутри защищенной среды устройства [2, 6]. Настройки приложения, такие как тема, автоблокировка и включение биометрии, могут храниться отдельно, но секретные данные не размещаются в DataStore или обычных настройках.

При первом запуске пользователь создает PIN-код (Personal Identification Number, персональный идентификационный номер). PIN-код не сохраняется в открытом виде: для проверки используется хэш и соль. После нескольких неверных попыток ввода добавляется задержка. Также поддерживается биометрическая аутентификация через системный механизм Android BiometricPrompt.

КОНТРОЛЬ ИСПОЛЬЗОВАНИЯ СЕКРЕТОВ

Особенностью SecureHub является то, что приложение не только хранит секреты, но и определяет правила их использования. Для каждого объекта формируется паспорт безопасности. В нем указывается уровень защиты, статус жизненного цикла и признак разрешения переноса на другое устройство. Основные уровни защиты приведены в таблице 1.

Таблица 1

Уровни защиты секрета в приложении SecureHub

Уровень защиты	Правило использования секрета
Обычный	Просмотр, копирование, редактирование и перенос разрешены без повторного подтверждения.
Строгий	Чувствительные действия выполняются после PIN-кода или биометрии.
Только автозаполнение	Прямой просмотр и копирование запрещены; секрет используется через автозаполнение.
Запечатанный	Ручное раскрытие помечает секрет как раскрытый и требует замены.

Перед выполнением действия экран или сервис приложения обращается к движку правил доступа. На вход передается действие пользователя, тип объекта и паспорт безопасности. В ответ возвращается одно из решений: разрешить действие, потребовать повторную аутентификацию, показать предупреждение или заблокировать действие. Такой подход позволяет использовать единые правила для просмотра, копирования, автозаполнения, редактирования, переноса и удаления секретов.

Алгоритм проверки действия с секретом представлен на рисунке 1. Схема показывает, что пользовательское действие сначала проходит через логику приложения и движок правил, а затем результат фиксируется в зашифрованном хранилище.

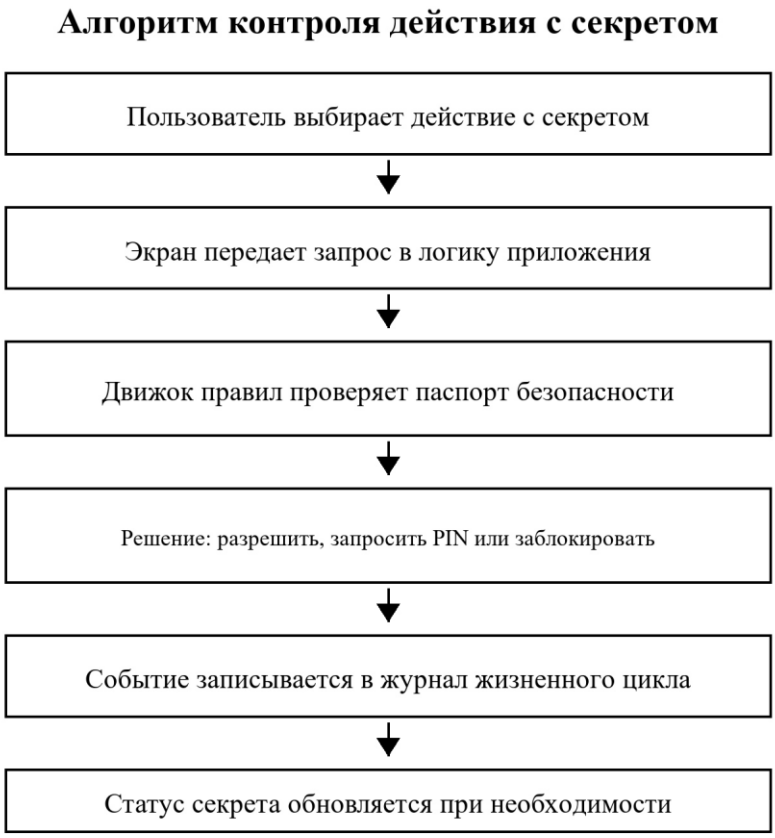

Рис. 1. Последовательность проверки действия с секретом в SecureHub

Для учетных данных используется Android Autofill Framework (системный механизм автозаполнения Android). При запросе автозаполнения приложение проверяет контекст: имя пакета, домен и хэш подписи стороннего приложения. Это снижает риск передачи пароля неподходящему приложению. Похожие исследования показывают, что механизмы автозаполнения требуют отдельного анализа безопасности, поскольку ошибка сопоставления контекста может привести к раскрытию учетных данных [8].

Все значимые действия фиксируются в журнале жизненного цикла. В журнал попадают события создания, изменения, просмотра, копирования, автозаполнения, переноса, удаления и блокировки запрещенного действия. Для проверки целостности события связываются хэш-цепочкой. В качестве алгоритма хэширования может применяться ГОСТ Р 34.11-2012 [9]. Если цепочка нарушена, приложение может пометить секрет как проблемный и показать пользователю предупреждение.

ПРОВЕРКА РЕАЛИЗОВАННЫХ МЕХАНИЗМОВ ЗАЩИТЫ

Для проверки реализованных механизмов применялась ручная проверка основных пользовательских сценариев и анализ выполнения базовых требований к локальному защищенному хранению данных. Проверялось отсутствие хранения секретов в открытом виде, использование SQLCipher и Android Keystore, наличие локальной аутентификации, автоблокировка приложения и корректное разделение настроек и секретных данных.

Дополнительно была выполнена ручная проверка пользовательских сценариев: создание секрета, просмотр, редактирование, удаление, поиск по заголовку, включение биометрии, попытка выполнить запрещенное действие и проверка записи события в журнал. Для оценки производительности применялся простой контрольный тест, измеряющий время записи, чтения и поиска разного количества записей в локальном хранилище.

Полученные результаты показывают, что выбранная архитектура подходит для приложения, ориентированного на локальное хранение данных. При этом для коммерческого использования требуется дополнительный внешний аудит безопасности, тестирование на разных устройствах и проверка устойчивости к анализу приложения.

ЗАКЛЮЧЕНИЕ

В результате работы было разработано Android-приложение SecureHub для локального защищенного хранения пользовательских данных. В приложении реализованы локальная аутентификация по PIN-коду и биометрии, зашифрованное хранение записей через Room и SQLCipher, защита ключевого материала через Android Keystore, хранение файлов, автозаполнение учетных данных и локальный перенос данных между устройствами.

Научно-практическая особенность решения заключается в объединении нескольких механизмов: уровней защиты секретов, policy engine, паспорта безопасности и журнала жизненного цикла с проверкой целостности. Благодаря этому приложение не только сохраняет данные в зашифрованном виде, но и контролирует способы их использования. Предложенный подход может быть использован как основа для дальнейшего развития локальных защищенных хранилищ на платформе Android

Список литературы:

1. A Study of Android Application Security / W. Enck [et al.] // Proceedings of the 20th USENIX Security Symposium. 2011. P. 1–16. URL: https://www.usenix.org/legacy/event/sec11/tech/full_papers/Enck.pdf

2. Elenkov N. Android Security Internals: An In-Depth Guide to Android’s Security Architecture. San Francisco: No Starch Press, 2015. 434 p. URL: https://nostarch.com/androidsecurity

3. Android Data Storage Locations and What App Developers Do with It from a Security and Privacy Perspective / K. Heid [et al.] // Proceedings of the 8th International Conference on Information Systems Security and Privacy. 2022. P. 380–387. URL: https://doi.org/10.5220/0010895800003120

4. Understanding the Security Risks of Android Storage / L. Lu [et al.] // Proceedings of the 21st USENIX Security Symposium. 2012. P. 1–16. URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/lu

5. An Empirical Study of Cryptographic Misuse in Android Applications / M. Egele [et al.] // Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security. 2013. P. 73–84. URL: https://doi.org/10.1145/2508859.2516693

6. Android Developers. Android Keystore system. URL: https://developer.android.com/privacy-and-security/keystore

7. SQLCipher for Android. URL: https://github.com/sqlcipher/android-database-sqlcipher

8. The Emperor’s New Autofill Framework: A Security Analysis of Autofill on Android / S. Oesch [et al.] // USEC 2021. P. 1–12. URL: https://doi.org/10.14722/usec.2021.23010

9. RFC 6986. GOST R 34.11-2012: Hash Function. URL: https://www.rfc-editor.org/rfc/rfc6986