Relazione Tecnica: SecureNotes - App Android per Note e File Sensibili con Sicurezza Avanzata

1. Introduzione

Il presente documento descrive l'applicazione Android "SecureNotes", progettata per consentire agli utenti di creare e gestire note personali e archiviare file sensibili (PDF, immagini, documenti), garantendo elevati standard di sicurezza. L'obiettivo è fornire un ambiente protetto da accessi non autorizzati e manipolazioni. Sviluppata per Android API 26+ con Java e Android Studio, l'app assicura robustezza e compatibilità con l'ecosistema Android moderno.

2. Obiettivi dell'Applicazione

SecureNotes è concepita per raggiungere i seguenti obiettivi fondamentali:

- **Autenticazione Sicura:** Implementa sblocco tramite biometria (BiometricPrompt) o PIN di fallback. L'autenticazione è richiesta all'avvio, dopo un timeout di inattività e prima di accedere a contenuti sensibili, rafforzando la protezione.
- **Crittografia Dati Locali:** Tutti i dati (note e file) sono crittografati localmente end-to-end. Le API Android Keystore gestiscono le chiavi, utilizzando AES/GCM. Il database Room è protetto da **SQLCipher**, garantendo la cifratura dell'intero archivio dati. Nessun dato è salvato in chiaro.
- **Archivio Sicuro di File:** Permette di caricare, archiviare e visualizzare documenti (PDF, immagini, ecc.). I file sono criptati internamente con EncryptedFile di Jetpack Security. L'accesso è condizionato alla verifica dell'identità.
- **Timeout Automatico:** Un timeout di sessione configurabile (default 3 minuti) blocca l'app dopo l'inattività, richiedendo nuova autenticazione. Essenziale per proteggere i dati se il dispositivo è incustodito.
- **Backup Criptato:** Offre l'esportazione di un backup cifrato (formato .zip) di note e file. L'archivio è protetto da password utente e crittografato con AES. I backup automatici di Android sono disabilitati (android:allowBackup="false" nell'AndroidManifest) per massimizzare il controllo utente sulla privacy.

3. Architettura dell'Applicazione

SecureNotes adotta un'architettura modulare e scalabile basata sul pattern **MVVM (Model-View-ViewModel)**, integrato con il **Repository pattern**. Questa scelta promuove la separazione delle responsabilità, migliorando testabilità, scalabilità e manutenibilità.

- **Model:** Rappresenta dati e logica di business, includendo le entità **Note.java** (ID,

titolo, contenuto, timestamp) e **FileItem.java** (ID, nome, tipo MIME, percorso crittografato, dimensione, timestamp). Comprende anche i **DAO (NoteDao.java, FileDao.java)** per l'interazione con il database Room.

- **View:** L'interfaccia utente (file XML di layout e classi Activity/Fragment come MainActivity.java, NotesFragment.java) è responsabile della visualizzazione e della raccolta delle interazioni utente, delegando la logica al ViewModel.
- **ViewModel:** Agisce da intermediario tra Model e View. **NoteViewModel.java** e **FileViewModel.java** espongono dati osservabili tramite LiveData, gestiscono la logica di presentazione e interagiscono con i Repository. Persistono ai cambiamenti di configurazione del dispositivo.
- **Repository:** Implementato da **NoteRepository.java** e **FileRepository.java**, astrae le fonti dati (database Room). I Repository decidono la strategia di accesso ai dati e forniscono un'API pulita ai ViewModel. Eseguono operazioni su ExecutorService separati per evitare blocchi dell'UI.

Componenti Chiave e Moduli:

- **SecureNotesApplication.java**: Classe Application globale, inizializza SQLCipher (SQLiteDatabase.loadLibs(this);) e FileManager all'avvio, gestendo anche la pulizia dei file temporanei.
- **database/NoteDatabase.java**: Database Room astratto. L'integrazione di SQLCipher è gestita tramite net.sqlcipher.database.SupportFactory e una passphrase sicura, crittografando l'intero file secure_notes_database.db. Implementa il pattern Singleton e gestisce le migrazioni.
- **dao/**: Contiene le interfacce DAO (NoteDao.java, FileDao.java) per operazioni CRUD su Room. Offrono metodi LiveData per l'osservazione reattiva e metodi sincroni (getAllNotesSync(), getAllFileItemsSync()) per il backup.
- **model/**: Definisce le entità del database (Note, FileItem).
- **viewmodel/**: Contiene le implementazioni di ViewModel e Repository.
- **adapter/**: Ospita gli adapter per RecyclerView (FileAdapter.java, NoteAdapter.java), che gestiscono la visualizzazione degli elementi e supportano interazioni come click e swipe-to-delete.
- **fragments/**: Contiene i frammenti UI (NotesFragment.java, FileListFragment.java), che visualizzano liste di note/file, gestiscono l'aggiunta, modifica, apertura ed eliminazione, interagendo con i ViewModel e il FileManager.
- **utils/**: Raccoglie classi di utilità, inclusa **BackupManager.java** per operazioni di backup/ripristino ZIP criptate con AES/GCM e Gson per la serializzazione JSON.
- **KeyManager.java**: Centrale per la gestione delle chiavi e del PIN. Implementa hashing PIN robusto (PBKDF2WithHmacSHA256 con salt e iterazioni), gestisce la passphrase di SQLCipher (generata con SecureRandom e protetta in

EncryptedSharedPreferences), e deriva chiavi AES per i backup. Inizializza EncryptedSharedPreferences con Android Keystore.
- **SecurityManager.java**: Presenta una parziale sovrapposizione di ruoli con KeyManager per l'inizializzazione di EncryptedSharedPreferences e la gestione della passphrase del database. Fornisce anche un metodo per ottenere istanze di EncryptedFile.
- **FileManager.java**: Gestisce le operazioni sui file sensibili. Utilizza EncryptedFile per crittografare/decrittografare file, salvandoli in encrypted_files. Gestisce file temporanei decifrati in una directory di cache, esposti tramite FileProvider per una condivisione sicura, e puliti alla terminazione dell'app.
- **LoginActivity.java**: Schermata di autenticazione principale. Gestisce BiometricPrompt e il fallback PIN (verificato con KeyManager). Inizializza il database crittografato dopo l'autenticazione e impedisce l'uscita non autorizzata.
- **MainActivity.java**: Schermata principale. Gestisce la navigazione tra i fragment e implementa un robusto timeout di sessione: un Handler e Runnable monitorano l'inattività, riportando l'app alla LoginActivity e chiudendo il database.
- **SettingsActivity.java**: Gestisce le impostazioni. Permette la configurazione del timeout, l'impostazione/rimozione del PIN (con KeyManager), e l'esecuzione di backup/ripristino criptati tramite BackupManager con UI di progresso.

4. Sicurezza Implementata

La sicurezza è il pilastro di SecureNotes, con misure di protezione multilivello.

4.1. Autenticazione Sicura

- **BiometricPrompt API (androidx.biometric):** Utilizzata per un'autenticazione biometrica standardizzata, gestita robustamente da LoginActivity con feedback utente e fallback.
- **PIN Fallback con Hashing Robusto:** KeyManager.java implementa hashing sicuro del PIN con **PBKDF2WithHmacSHA256**, un salt casuale e 10000 iterazioni. Hash e salt sono in EncryptedSharedPreferences, rendendo la forza bruta computazionalmente proibitiva.
- **Autenticazione Condizionale:** Richiesta all'avvio dell'app e dopo un timeout di inattività (gestito da MainActivity con Handler e Runnable che monitorano dispatchTouchEvent).

4.2. Crittografia Dati Locali

- **Android Keystore API:** Fondamento per la gestione sicura delle chiavi crittografiche. Genera e archivia chiavi "non estraibili" in un contenitore sicuro, spesso hardware.

- **AES/GCM:** Algoritmo di crittografia simmetrica standard, usato per note e file. Offre confidenzialità, integrità e autenticazione.
- **EncryptedSharedPreferences (androidx.security.crypto):** Utilizzata da KeyManager e SecurityManager per memorizzare preferenze sensibili (hash PIN, passphrase DB) in modo crittografato, protette da Android Keystore.
- **EncryptedFile (androidx.security.crypto):** FileManager.java usa EncryptedFile per crittografare/decrittografare i file sensibili. I file crittografati sono salvati internamente, e quelli decifrati temporaneamente sono gestiti con FileProvider per una condivisione sicura e pulizia automatica.
- **SQLCipher per la Crittografia del Database Room:** SecureNotesApplication.java carica SQLCipher all'avvio. NoteDatabase.java crea il database Room con net.sqlcipher.database.SupportFactory e una passphrase di 256 bit (gestita da KeyManager e protetta in EncryptedSharedPreferences). L'intero file del database SQLite è crittografato a riposo.

4.3. Gestione del Backup Criptato e Permessi di Storage

- **Backup Locale Criptato:** L'app permette l'esportazione di un backup .zip criptato di note e file. La password utente deriva una chiave AES tramite PBKDF2 per la crittografia dell'archivio. SettingsActivity gestisce la richiesta password e l'esecuzione asincrona del backup con feedback di progresso.
- **Nessun Cloud Automatico e Controllo Utente:** La configurazione android:allowBackup="false" nell'AndroidManifest disabilita i backup automatici, garantendo che i dati non lascino il dispositivo senza esplicito consenso.
- **Gestione dei Permessi di Storage:** L'app richiede READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE e MANAGE_EXTERNAL_STORAGE per gestire l'esportazione/importazione dei backup. FileProvider e androidx.documentfile assicurano una gestione sicura e conforme dei file esterni.

4.4. Offuscamento del Codice (ProGuard/R8)

Per resistere al reverse engineering, l'app è configurata per l'offuscamento tramite R8.

- **minifyEnabled true e shrinkResources true:** Abilitano l'ottimizzazione del codice e la rimozione delle risorse inutilizzate, riducendo le dimensioni dell'APK e offuscando nomi di classi, metodi e campi.
- **proguardFiles:** proguard-rules.pro definisce regole dettagliate per mantenere le classi principali dell'app, i componenti Android standard e le classi critiche di librerie come androidx.biometric, androidx.security.crypto, androidx.room, **net.sqlcipher**, androidx.work e com.google.code.gson.
- **Verifica dell'offuscamento:** Test con jadx hanno confermato l'efficacia dell'offuscamento, rendendo i nomi criptici e le stringhe sensibili non in chiaro,

aumentando la difficoltà di analisi.

5. Scelte Tecniche

Le scelte tecniche bilanciano sicurezza, prestazioni, manutenibilità e best practice Android.

- **Jetpack Components:** Adozione strategica per modernizzare lo sviluppo:
  - **Architecture Components (ViewModel, LiveData, Room):** Fondamentali per MVVM, garantiscono persistenza dei dati UI e gestione reattiva.
  - **Jetpack Security (androidx.security.crypto):** Cruciale per crittografia semplificata di preferenze e file con Android Keystore.
  - **Jetpack Biometric API (androidx.biometric):** Per un'autenticazione biometrica standardizzata e compatibile.
  - **Room Persistence Library con SQLCipher:** Per la gestione del database SQLite locale con crittografia completa a livello di file.
  - **WorkManager:** Consigliato per operazioni in background affidabili e differibili (es. futuri backup automatici).
- **Gson (com.google.code.gson):** Per serializzazione/deserializzazione efficiente di oggetti Java in JSON.
- **DocumentFile (androidx.documentfile):** Per interazione sicura e conforme con il Storage Access Framework (SAF) per i file esterni.

6. Limiti e Rischi Futuri

Nessun sistema è immune da vulnerabilità; è cruciale identificare limiti e rischi per il miglioramento continuo.

6.1. Limiti Attuali:

- **Dipendenza dall'Hardware Secure Element:** La sicurezza delle chiavi Android Keystore dipende dalla robustezza dell'implementazione hardware del Secure Element sul dispositivo.
- **Backup Manuale:** La natura manuale del backup, pur garantendo privacy, introduce il rischio di perdita dati se non eseguito regolarmente.
- **Complessità della Gestione Chiavi:** Errori nella gestione delle chiavi (anche con Android Keystore) possono portare a perdita irreversibile dei dati.
- **Mancanza di Sincronizzazione Cloud Criptata:** Limita la comodità di accesso da più dispositivi, pur essendo un vantaggio per la privacy.
- **Gestione del Permesso MANAGE_EXTERNAL_STORAGE:** Permesso molto ampio che richiede giustificazione chiara e attenta gestione per non compromettere privacy e fiducia.

- **Sovrapposizione di Ruoli tra KeyManager e SecurityManager:** Una certa sovrapposizione nelle responsabilità potrebbe essere consolidata per migliorare manutenibilità e chiarezza architetturale.

6.2. Rischi Futuri:

- **Attacchi di Reverse Engineering Avanzati:** Nonostante l'offuscamento R8, attaccanti determinati potrebbero tentare di decifrare il codice tramite analisi dinamica.
- **Vulnerabilità del Sistema Operativo:** L'app è esposta a vulnerabilità del sistema Android o delle librerie sottostanti; gli aggiornamenti di sicurezza sono cruciali.
- **Attacchi di "Root Detection" e "App Tampering":** Dispositivi rootati o app manomesse possono compromettere la sicurezza; l'implementazione di meccanismi di rilevamento è cruciale.
- **Attacchi di Forza Bruta sul PIN/Password:** PIN o password deboli, senza blocchi dopo tentativi falliti, sono vulnerabili.
- **Fuga di Stringhe Sensibili:** Stringhe hardcoded potrebbero essere esposte; l'offuscamento a runtime (XOR, JNI) sarebbe un miglioramento.

7. Possibili Sviluppi Futuri

Per migliorare ulteriormente SecureNotes:

- **Funzione di Autodistruzione Note (Note a Tempo):** Permettere la cancellazione automatica di note sensibili dopo un timer.
- **Tagging/Filtri per Note:** Migliorare l'organizzazione e la ricerca delle note.
- **Integrazione di Root Detection e App Tamper Detection:** Controlli a runtime per bloccare l'app su dispositivi compromessi.
- **Obfuscazione di Stringhe a Runtime:** Per proteggere ulteriormente le stringhe sensibili.

Conclusione

SecureNotes rappresenta un esempio robusto e ben progettato di applicazione mobile che pone la sicurezza dei dati al centro del suo design e della sua implementazione. L'adozione di pattern architetturali moderni come MVVM e Repository, l'uso delle API di sicurezza native di Android (come Android Keystore, BiometricPrompt e Jetpack Security), l'integrazione di SQLCipher per la crittografia del database, e l'attenzione all'offuscamento del codice tramite R8 contribuiscono a creare un ambiente affidabile e protetto per la gestione di informazioni sensibili. I limiti attuali e i rischi futuri identificati in questa relazione tecnica non solo offrono una comprensione approfondita delle sfide rimanenti, ma delineano anche una chiara roadmap per futuri miglioramenti e per rafforzare ulteriormente la postura di sicurezza dell'applicazione,

garantendo che SecureNotes possa evolvere per affrontare nuove minacce e soddisfare le esigenze degli utenti in un panorama digitale in continua evoluzione.