Progetto Android: SecureNotes

Titolo: SecureNotes – App Android per note e file sensibili con sicurezza avanzata
Target: Android API 26+ (Android 8.0 Oreo e successivi)
Linguaggio: Java (preferito) o Kotlin
IDE: Android Studio

Obiettivo dell'app

Creare un'applicazione Android che permetta agli utenti di scrivere note personali e archiviare file sensibili (es. PDF, immagini, documenti), garantendo la **massima sicurezza** dei dati grazie a:

- **Autenticazione biometrica o a PIN**
- **Criptazione locale end-to-end** (note e file)
- **Timeout di sessione automatica**
- **Backup locale criptato**
- **Accesso ai file solo dopo verifica d'identità**

Funzionalità principali

1. Autenticazione Sicura

- Sblocco tramite **biometria (impronta, volto)** oppure **PIN fallback**
- Autenticazione richiesta:
    - All'avvio dell'app
    - Dopo timeout di inattività
    - Prima di visualizzare contenuti sensibili

2. Crittografia Dati Locali

- Tutti i dati (note e file) vengono criptati con **Android Keystore API**
- Utilizzo di **AES/GCM** per la cifratura simmetrica
- Nessun dato in chiaro salvato nel filesystem/app

3. Archivio Sicuro di File

- Possibilità di caricare e visualizzare documenti (PDF, immagini, ecc.)
- I file sono memorizzati criptati internamente (es. encryptedFile API)

4. Timeout automatico

- Timeout configurabile (default: 3 minuti)
- Alla scadenza, l’app si blocca e richiede nuova autenticazione

5. Backup Criptato

- Backup cifrato esportabile in formato .zip

- Protezione del backup con password e criptazione AES
- Nessun salvataggio automatico su cloud (salvataggio locale o manuale)

Architettura suggerita

- **MVVM + Repository pattern**
- **Jetpack Security (EncryptedSharedPreferences, EncryptedFile)**
- **Jetpack Biometric API**
- **Room Database con crittografia (SQLCipher se necessario)**
- **WorkManager per backup pianificati**

Interfaccia Utente (UI)

1. **Login screen**
   - Autenticazione biometrica o PIN
2. **Dashboard**
   - Lista note criptate con anteprima
   - Accesso a archivio file
3. **Editor Note**
   - Editor testo
   - Salvataggio automatico criptato
4. **Archivio File**
   - Caricamento file
   - Accesso solo dopo autenticazione
5. **Impostazioni**
   - Timeout sessione
   - Esporta backup criptato
   - Cambia PIN

Test e Validazione

- Test sicurezza:
  - Nessun dato visibile in SQLite o file system
  - App bloccata dopo timeout
  - Reverse engineering con apktool → codice offuscato (ProGuard o R8)

Obiettivo offuscamento del codice

Rendere illeggibile e più difficile da analizzare il codice Kotlin/Java dell'app, riducendo il rischio di:

- Estrazione delle chiavi (se non ben protette)
- Manipolazione del codice
- Accesso alle funzioni sensibili via decompilazione (es. apktool + jadx)

Step-by-step: Abilitare e configurare l'offuscamento

1. Assicurati che la build sia in modalità `release`

L'offuscamento avviene solo in **build release** (non in debug):

Nel `build.gradle (Module: app di groovy)`:

```
buildTypes {
    release {
        minifyEnabled true
        shrinkResources true
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
    }
}
```

- `minifyEnabled true` → abilita R8 (offuscamento)
- `shrinkResources true` → rimuove le risorse inutilizzate
- `proguardFiles` → include i file di regole (default + custom)

2. Configura le regole in `proguard-rules.pro`

Esempio di base per *SecureNotes*:

```
# Mantieni classi di AndroidX
-keep class androidx.** { *; }

# Mantieni classi usate dalla Biometric API
-keep class androidx.biometric.** { *; }

# Mantieni entità Room
-keep class com.yourpackage.model.** { *; }
-keepclassmembers class * {
    @androidx.room.* <methods>;
}

# Mantieni classi per encrypted preferences/file
-keep class androidx.security.crypto.** { *; }

# Offusca tutto il resto
# Non è necessario aggiungere -dontobfuscate

# Mantieni entry point (Application)
-keep class com.yourpackage.SecureNotesApplication { *; }

# Per evitare errori con riflessione, mantieni classi usate dinamicamente
-keepnames class * {
    @com.google.gson.annotations.SerializedName <fields>;
}
```

Evita di usare `-dontobfuscate` e `-dontoptimize` a meno che tu non abbia problemi specifici da debug.

3. Genera la release offuscata

Nel terminale Android Studio:

```
./gradlew clean assembleRelease
```

Troverai l'APK offuscato in:

```
app/build/outputs/apk/release/app-release.apk
```

4. Verifica l'offuscamento

1. Apri l'APK con strumenti come:
   - jadx (decompilatore Java)
   - apktool (disassemblaggio Smali)
2. Controlla che:
   - I nomi delle classi/metodi siano criptici (`a()`, `b()`)
   - I nomi delle variabili non siano comprensibili
   - Le stringhe sensibili non siano in chiaro

Suggerimenti di sicurezza avanzati

- Usa il **Keystore** per salvare le chiavi (non hardcoded!)
- Firma l'APK con una **chiave privata** sicura (non debug key)
- Aggiungi protezioni runtime:
  - **Root detection**
  - **App tamper detection**
  - **Obfuscazione di stringhe (via native code o XOR runtime)**

Output richiesti (per studenti o team)

- Codice sorgente su GitHub/GitLab
- APK firmato
- Documento tecnico (5-6 pagine):
  - Architettura
  - Sicurezza implementata
  - Scelte tecniche
  - Limiti e rischi futuri

Tecnologie e librerie consigliate

- `androidx.security.crypto`
- `androidx.biometric`
- `androidx.room` + `SQLCipher`
- `WorkManager` per backup

- `EncryptedFile`, `EncryptedSharedPreferences`

Suggerimenti extra

- Supporta modalità scura
- Usa backup opzionale solo dopo conferma utente
- Aggiungi funzione di autodistruzione note (note a tempo)
- Consenti tagging/filtri sulle note per migliorarne l'organizzazione