Android Encrypted DataStore Documentation

Overview

This document describes a **pure Android (non-KMP)** implementation of **secure token storage** using:

- AndroidKeyStore (AES encryption)
- AndroidX DataStore (Proto / OkioSerializer)
- Kotlin Serialization
- Koin for Dependency Injection

The design ensures: - Encryption at rest - Clear separation of concerns - Scalability and maintainability

---

Folder Structure

```
com.gurkha.hr
│
├── app
│   └── HrApplication.kt
│
├── crypto
│   ├── Cryptography.kt
│   └── AndroidCryptography.kt
│
├── datastore
│   └── token
│       ├── model
│       │   └── Token.kt
│       │
│       ├── local
│       │   ├── TokenJsonSerializer.kt
│       │   └── TokenDataStore.kt
│       │
│       └── di
│           └── TokenDataStoreModule.kt
│
├── di
│   ├── CryptoModule.kt
│   └── AppModule.kt
│
└── util
```

```
    └── dispatcher
        └── AppDispatchers.kt
```

Cryptography Layer

Cryptography.kt

```
package com.gurkha.hr.crypto

import kotlinx.serialization.KSerializer

interface Cryptography {
    suspend fun <T> encrypt(
        t: T,
        serializer: KSerializer<T>
    ): ByteArray?

    suspend fun <T> decrypt(
        bytes: ByteArray,
        deserializer: KSerializer<T>
    ): T?
}
```

AndroidCryptography.kt

```
package com.gurkha.hr.crypto

import android.security.keystore.KeyGenParameterSpec
import android.security.keystore.KeyProperties
import kotlinx.coroutines.sync.Mutex
import kotlinx.coroutines.sync.withLock
import kotlinx.serialization.KSerializer
import kotlinx.serialization.json.Json
import java.security.KeyStore
import javax.crypto.Cipher
import javax.crypto.KeyGenerator
import javax.crypto.SecretKey
import javax.crypto.spec.IvParameterSpec

class AndroidCryptography : Cryptography {

    private val keyStore = KeyStore.getInstance("AndroidKeyStore").apply {
        load(null)
    }

    private fun getKey(): SecretKey {
        val entry = keyStore.getEntry(KEY_ALIAS, null) as?
KeyStore.SecretKeyEntry
        return entry?.secretKey ?: createKey()
    }

    private fun createKey(): SecretKey {
        return KeyGenerator.getInstance(ALGORITHM).apply {
            init(
                KeyGenParameterSpec.Builder(
                    KEY_ALIAS,
                    KeyProperties.PURPOSE_ENCRYPT or
KeyProperties.PURPOSE_DECRYPT
                )
                    .setBlockModes(BLOCK_MODE)
                    .setEncryptionPaddings(PADDING)
                    .setRandomizedEncryptionRequired(true)
                    .setUserAuthenticationRequired(false)
                    .build()
            )
        }.generateKey()
    }

    private val cipherMutex = Mutex()
    private val cipherPool = mutableListOf<Cipher>()

    private suspend fun getCipher(): Cipher = cipherMutex.withLock {
        cipherPool.removeFirstOrNull() ?: Cipher.getInstance("AES/CBC/
PKCS7Padding")
    }

    private suspend fun releaseCipher(cipher: Cipher) = cipherMutex.withLock {
        if (cipherPool.size < MAX_POOL_SIZE) cipherPool.add(cipher)
    }

    private suspend fun encryptBytes(bytes: ByteArray): ByteArray {
        val cipher = getCipher()
        return try {
            cipher.init(Cipher.ENCRYPT_MODE, getKey())
            cipher.iv + cipher.doFinal(bytes)
        } finally {
            releaseCipher(cipher)
        }
    }

    private suspend fun decryptBytes(bytes: ByteArray): ByteArray {
        val cipher = getCipher()
        return try {
            val iv = bytes.copyOfRange(0, 16)
            val data = bytes.copyOfRange(16, bytes.size)
            cipher.init(Cipher.DECRYPT_MODE, getKey(), IvParameterSpec(iv))
            cipher.doFinal(data)
        } finally {
            releaseCipher(cipher)
        }
    }

    override suspend fun <T> encrypt(t: T, serializer: KSerializer<T>):
ByteArray? {
        return runCatching {
            val json = Json.encodeToString(serializer, t)
            encryptBytes(json.encodeToByteArray())
        }.getOrNull()
    }

    override suspend fun <T> decrypt(bytes: ByteArray, deserializer:
KSerializer<T>): T? {
        return runCatching {
            val json = decryptBytes(bytes).decodeToString()
            Json.decodeFromString(deserializer, json)
        }.getOrNull()
    }

    companion object {
        private const val KEY_ALIAS = "secret"
        private const val ALGORITHM = KeyProperties.KEY_ALGORITHM_AES
        private const val BLOCK_MODE = KeyProperties.BLOCK_MODE_CBC
        private const val PADDING = KeyProperties.ENCRYPTION_PADDING_PKCS7
        private const val MAX_POOL_SIZE = 10
    }
}
```

Token Model

Token.kt

```
package com.gurkha.hr.datastore.token.model

import kotlinx.serialization.Serializable
```

```
@Serializable
data class Token(
    val accessToken: String = "",
    val refreshToken: String = ""
)
```

DataStore Layer

TokenJsonSerializer.kt

```
package com.gurkha.hr.datastore.token.local

import androidx.datastore.core.okio.OkioSerializer
import com.gurkha.hr.crypto.Cryptography
import com.gurkha.hr.datastore.token.model.Token
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import okio.BufferedSink
import okio.BufferedSource
import okio.use
import org.koin.mp.KoinPlatform.getKoin

internal object TokenJsonSerializer : OkioSerializer<Token> {

    private val cryptography: Cryptography = getKoin().get()

    override val defaultValue: Token = Token()

    override suspend fun readFrom(source: BufferedSource): Token {
        val bytes = withContext(Dispatchers.IO) { source.readByteArray() }
        return cryptography.decrypt(bytes, Token.serializer()) ?: defaultValue
    }

    override suspend fun writeTo(t: Token, sink: BufferedSink) {
        sink.use {
            withContext(Dispatchers.IO) {
                cryptography.encrypt(t, Token.serializer())?.let { data ->
                    it.write(data)
                }
            }
        }
    }
}
```

TokenDataStore.kt

```kotlin
package com.gurkha.hr.datastore.token.local

import androidx.datastore.core.DataStore
import com.gurkha.hr.datastore.token.model.Token
import kotlinx.coroutines.flow.Flow

class TokenDataStore(private val dataStore: DataStore<Token>) {

    val token: Flow<Token> = dataStore.data

    suspend fun save(token: Token) {
        dataStore.updateData { token }
    }

    suspend fun clear() {
        dataStore.updateData { Token() }
    }
}
```

Dependency Injection (Koin)

CryptoModule.kt

```kotlin
package com.gurkha.hr.di

import com.gurkha.hr.crypto.AndroidCryptography
import com.gurkha.hr.crypto.Cryptography
import org.koin.dsl.module

val cryptoModule = module {
    single<Cryptography> { AndroidCryptography() }
}
```

TokenDataStoreModule.kt

```kotlin
package com.gurkha.hr.datastore.token.di

import androidx.datastore.core.DataStore
import androidx.datastore.core.DataStoreFactory
import com.gurkha.hr.datastore.token.local.TokenDataStore
import com.gurkha.hr.datastore.token.local.TokenJsonSerializer
import com.gurkha.hr.datastore.token.model.Token
import org.koin.android.ext.koin.androidContext
import org.koin.dsl.module
import java.io.File

val tokenDataStoreModule = module {

    single<DataStore<Token>> {
        DataStoreFactory.create(
            serializer = TokenJsonSerializer,
            produceFile = {
                File(androidContext().filesDir, "token.pb")
            }
        )
    }

    single { TokenDataStore(get()) }
}
```

AppModule.kt

```
package com.gurkha.hr.di

import com.gurkha.hr.datastore.token.di.tokenDataStoreModule

val appModules = listOf(
    cryptoModule,
    tokenDataStoreModule
)
```

Application Setup

HrApplication.kt

```
package com.gurkha.hr.app

import android.app.Application
import com.gurkha.hr.di.appModules
import org.koin.android.ext.koin.androidContext
import org.koin.core.context.startKoin

class HrApplication : Application() {
    override fun onCreate() {
        super.onCreate()

        startKoin {
            androidContext(this@HrApplication)
            modules(appModules)
        }
    }
}
```

Add to `AndroidManifest.xml`:

```
<application
    android:name=".app.HrApplication"
    ... />
```

Security Notes

- AES/CBC does **not provide integrity protection**
- For production-grade security, prefer **AES/GCM/NoPadding**
- AndroidKeyStore safely stores the encryption key

Summary

This setup provides: - Encrypted token storage - Clean Android architecture - Serializer-level security - Easy extensibility

This is suitable for production Android applications.