|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document History** | | | | |
| **Version** | **Date** | **Author** | **Section** | **Changes** |
| 1.0 | 24-01-2017 | Anurag, Adarsha, Kavya | All | Integration of ORMLite with Sqlcipher |

Secure Database Android Integration

|  |  |
| --- | --- |
| Author | Anurag, Adarsha, Kavya |
| Approved by |  |
| Email Id | [anurag.gautam@philips.com](mailto:anurag.gautam@philips.com) adarsha.shetty@philips.com  [kavya.g.kurpad@philips.com](mailto:kavya.g.kurpad@philips.com) |

1. INTRODUCTION 3

2. INTEGRATION 3

2.1 Maven repository Integration 3

2.2 Library Integration 3

Git source path: 3

2.3 Library versioning 3

2.4 Gradle dependencies 3

2.5 Prerequisites 4

3. How to execute a Secure Database 4

4. How to build new Secure database API 7

5. Frequently asked questions 7

a. What is ORMLite? 7

b. What is SQLCipher? 7

c. How SQLCipher integration differ from SQLite 7

d. SQLCipher Limitations 8

e. SQLCipher Security 8

6. Notes 8

# INTRODUCTION

This document provides an overview of integration procedure for secure database library in android mobile applications or any other components.

# INTEGRATION

## Maven repository Integration

The easiest and preferred way to use these components is using maven.

build.gradle changes

compile(**group**: **'com.philips.cdp'**, **name**: **'AppInfra'**, **version**: **'1.4.0'**, **ext**: **'aar'**)

## Library Integration

# Git source path:

<https://bitbucket.atlas.philips.com/projects/MAIL/repos/app-infra_secure_db_android/browse>

1. Check out the code from above path where in you can find sample demo app which depends on Secure Database library’s aar file. Secure database integrated with ORMLite and SqlCipher . It includes below library’s
2. compile **"com.j256.ormlite:ormlite-core:5.0"**
3. compile **“com.j256.ormlite:ormlite-android:5.0”**
4. compile **'net.zetetic:android-database-sqlcipher:3.5.4@aar'**
5. compile(group: 'com.philips.cdp', name: 'AppInfra', version: '1.4.0', ext: 'aar')

## Library versioning

Library version can be obtained by using below API

## Gradle dependencies

To support 64it mobile devices app exclude [arm64-v8a](https://developer.android.com/ndk/guides/abis.html#arm64-v8a) libraries from the APK in order to prevent runtime crashes due to library architecture conflicts (https://developer.android.com/studio/build/configure-apk-splits.html#configure-abi-split ?).

To achieve this using below line in build.gradle file , inside build config

ndk {  
 abiFilters **"armeabi"**, **"armeabi-v7a"**, **"x86"**, **"mips"** }

## Prerequisites

SecureStorageInterface.SecureStorageError ssError = new SecureStorageInterface.SecureStorageError();

1. Integrating application/component must pass password keyName to Secure Storage’s createKey(KeyType.Key, “keyName”, ssError). It will generate password, using AES Key Generator. This method comes under App Infra’s Secure storage library so we need Add below lines in build.gradle

compile(group: 'com.philips.cdp', name: 'AppInfra', version: '1.4.0', ext: 'aar')

1. After generate password via Secure Storage createKey(KeyType.Key, “keyName”, ssError), we have to use same key to retrieve generate password so we have to pass KeyName via constructor of helper class to call super class SecureDbOrmLiteSqliteOpenHelper constructor.

# How to execute a Secure Database

SecureDbOrmLiteSqliteOpenHelper is the class which is the helper class for app/component, it is extend from SQLiteOpenHelper. Note: SQLiteOpenHelper class is from net.sqlcipher.database, not android.database.

SecureDbOrmLiteSqliteOpenHelper which can be extended by your application to help manage when the application needs to create or upgrade its database.

1. Have to load Sqlcipher .so file via Calling “SQLiteDatabase.loadLibs(this) “. Note: To avoid multiple time loading this library call it from only first time run.

SQLiteDatabase.loadLibs(this)

1. Create Java Module Pojo class, and use proper annotation to create database fields.

Example: AddressBook

**public class** AddressBook **implements** Serializable {  
  
 */\*\*  
 \* Model class for address database table  
 \*/* **public static final** String ***ID\_FIELD*** = **"address\_id"**;  
  
 // Primary key defined as an auto generated integer  
 // If the database table column name differs than the Model class variable name, the way to map to use columnName

@DatabaseField(generatedId = **true**, columnName = ***ID\_FIELD***)  
 **public int addressId**;  
  
 @DatabaseField(columnName = **"first\_name"**)  
 **public** String **firstName**;  
  
  
 @DatabaseField(columnName = **"last\_name"**)  
 **public** String **lastName**;  
  
 @DatabaseField  
 **public** String **address**;  
  
 @DatabaseField(columnName = **"contact\_number"**)  
 **public** String **contactNumber**;  
  
 **public** AddressBook() {  
  
 }  
  
 **public** AddressBook(String firstName, String lastName, String address, String contactNumber) {  
 **this**.**firstName** = firstName;  
 **this**.**lastName** = lastName;  
 **this**.**address** = address;  
 **this**.**contactNumber** = contactNumber;  
 }  
  
}

1. Extend SecureDbOrmLiteSqliteOpenHelper to your application helper class. Import SecureDbOrmLiteSqliteOpenHelper from “com.philips.platform.securedblibrary.ormlite.sqlcipher.android.apptools.SecureDbOrmLiteSqliteOpenHelper”

Example: SecureDataBaseHelper **extends** SecureDbOrmLiteSqliteOpenHelper

1. And pass this values to super class constructor from your helper class

**super**(context, “dataBaseName”, **null**, databaseVersion, “keyName”);

context- your activity class context.

dataBaseName- Name of the database to create

null-Cursor factory or null if none.

databaseVersion-version number of database in int.

keyName-name of the key to retrieve generate password. Note: this key must be same as password generate keyName.

Example: **public** SecureDataBaseHelper(Context context, String dataBaseName, **int** databaseVersion, String keyName) {  
 **super**(context, dataBaseName, **null**, databaseVersion, keyName);  
   
}

1. And Inside onCreate() of helper class create table.

@Override  
 **public void** onCreate(SQLiteDatabase database, ConnectionSource source) {  
 **try** {  
 TableUtils.*createTable*(source, **tableName**);  
 } **catch** (SQLException e) {  
 }  
  
 }

Example:

@Override  
 **public void** onCreate(SQLiteDatabase database, ConnectionSource source) {  
 **try** {  
 TableUtils.*createTable*(source, **AddressBook.class**);  
 } **catch** (SQLException e) {  
 }  
  
 }

Helper class sample code:

**public class** SecureDataBaseHelper<T> **extends** SecureDbOrmLiteSqliteOpenHelper {  
   
  
 **public** SecureDataBaseHelper(Context context, String dataBaseName, **int** databaseVersion, String databaseKey) {  
 **super**(context, dataBaseName, **null**, databaseVersion, databaseKey);  
   
  
 }  
  
  
 @Override  
 **public void** onCreate(SQLiteDatabase database, ConnectionSource source) {  
 **try** {  
 TableUtils.*createTable*(source, **tableName**);  
 } **catch** (SQLException e) {  
 }  
 }  
  
 @Override  
 **public void** onUpgrade(SQLiteDatabase database, ConnectionSource source, **int** oldVersion, **int** newVersion) {  
 **try** {

TableUtils.*createTable*(source, **tableName**);  
 TableUtils.*dropTable*(source, **tableName**, **true**);  
 } **catch** (SQLException e) {  
 }  
 }  
}

1. Have to create objet instance of your application helper class.

Example: SecureDataBaseHelper secureDataBaseHelper = new SecureDataBaseHelper(this, DATABASE\_NAME, DATABASE\_VERSION, DATABASE\_PASSWORD\_KEY);

1. Call getPassword() from helper class instance to get password from SecureDbOrmLiteSqliteOpenHelper class. In SecureDbOrmLiteSqliteOpenHelper

Class retrieve password from Secure storage getKey(“keyName”). Here keyName same as keyName of generate password.

Example: secureDataBaseHelper. getPassword()

1. Call getWritableDatabase(password) from helper class instance to get database writable permission.

Example:

String password=: secureDataBaseHelper. getPassword();

secureDataBaseHelper. getWritableDatabase(password);

# How to build new Secure database API

Please refer sample demo application for more details

# Frequently asked questions

# What is ORMLite?

ORMLite is lightweight Java ORM supports for android Sqlite Database.The full form of ORMLite is Object Relational Mapping Lite(ORMLite).and it provides some light weight functionality to store and retrieve Java Objects.And it avoiding the complexity and more standard Object Relational Mapping.

ORMLite simply add objects of java using annotations.and its have powerful abstract Database Access Object classes.also provides simple and  flexible query using QueryBuilder.Auto generates SQL to create and drop database tables and its have basic supports for database transactions.

More Guide Line about ORMLite Read below link:

http://ormlite.com/javadoc/ormlite-core/doc-files/ormlite\_1.html#Getting-Started

# What is SQLCipher?

SQLCipher is used for a full database encryption. What *full* means is that the DB file is entirely encrypted, not only separate rows or tables. It is a Modified version of SQLite and AES encryption by default.

More Guide Line about SQLCipher Read below link:

<https://www.zetetic.net/sqlcipher/sqlcipher-for-android/>

<http://lomza.totem-soft.com/tutorial-add-sqlcipher-to-your-android-app/>

# How SQLCipher integration different from SQLite

There is small changes in import, we have to import:  
  
   import net.sqlcipher.database.SQLiteDatabase;  
   import net.sqlcipher.database.SQLiteOpenHelper;

import net.sqlcipher.database.SQLiteDatabase;  
  import net.sqlcipher.database.SQLiteQueryBuilder  
  
instead of  
  
  import android.database.sqlite.SQLiteDatabase;  
  import android.database.sqlite.SQLiteOpenHelper;

  import android.database.sqlite.SQLiteDatabase;  
  import android.database.sqlite.SQLiteQueryBuilder;  
  
  
  
  
Method signatures of both libraries  are same so no modification is required in function calls.  
  
There is a small difference between SQLite and SQLCipher's db.getReadableDatabase and db.getWriteableDatabase functions though  
android's SQLite  functions receives no argument while SQLCipher's functions take (String) password as an argument. This password is used to encrypt when writing and decrypt when reading from database.

# SQLCipher Advantages

SQLCipher is often the go to choice for Android apps for the following reasons:

* Strong encryption (256-bit AES)
* Mature technology
  + Maintained and supported by its developers and the open source community
* Supports virtually the same API as standard Android database functions

# SQLCipher Limitations

It Adds 3MB to APK Size per CPU architecture.

# SQLCipher Security

* Customizable encryption algorithm based on OpenSSL libcrypto
* Individual pages encrypted with own initialization vector.
* Message authentication code (MAC) per page to detect tempering.
* Hashed passphrase (PBKDF2) for key.

# Notes

1. Please refer sample application for more details.