SQLCipher：

SQLCipher是一个在SQLite基础之上进行扩展的开源数据库，SQLCipher具有占地面积小、性能因此它非常适合嵌入式应用的数据库保护，非常适合于移动开发。

优势：

加密性能高、开销小，只要5-15%的开销用于加密

完全做到数据库100%加密

采用良好的加密方式（CBC加密模式）

使用方便，做到应用级别加密

采用OpenSSL加密库提供的算法

用法：

**1.）在build.gradle文中添加如下代码,当前使用的是最新版本3.4.0**

dependencies {

compile 'net.zetetic:android-database-sqlcipher:3.4.0'

}

**2.）创建一个SQLiteOpenHelper 注意接下来所以有关Sqlite相关类全部引用net.sqlcipher.database的类**

import android.content.Context;

import android.util.Log;

import net.sqlcipher.SQLException;

import net.sqlcipher.database.SQLiteDatabase;

import net.sqlcipher.database.SQLiteOpenHelper;

public class DBCipherHelper extends SQLiteOpenHelper {

private static final String TAG = "DatabaseHelper";

private static final String DB\_NAME = "test\_cipher\_db";//数据库名字

public static final String DB\_PWD="whoislcj";//数据库密码

public static String TABLE\_NAME = "person";// 表名

public static String FIELD\_ID = "id";// 列名

public static String FIELD\_NAME= "name";// 列名

private static final int DB\_VERSION = 1; // 数据库版本

public DBCipherHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {

super(context, name, factory, version);

//不可忽略的 进行so库加载

SQLiteDatabase.loadLibs(context);

}

public DBCipherHelper(Context context) {

this(context, DB\_NAME, null, DB\_VERSION);

}

/\*\*

\* 创建数据库

\* @param db

\*/

@Override

public void onCreate(SQLiteDatabase db) {

//创建表

createTable(db);

}

private void createTable(SQLiteDatabase db){

String sql = "CREATE TABLE " + TABLE\_NAME + "(" + FIELD\_ID + " integer primary key autoincrement , " + FIELD\_NAME + " text not null);";

try {

db.execSQL(sql);

} catch (SQLException e) {

Log.e(TAG, "onCreate " + TABLE\_NAME + "Error" + e.toString());

return;

}

}

/\*\*

\* 数据库升级

\* @param db

\* @param oldVersion

\* @param newVersion

\*/

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

}

}

注意：SQLiteDatabase.loadLibs(context);这个千万别忘记调用

**3.)创建一个DBCipherManager数据库管理**

  具体实现传统的SQLiteOpenHelper都是完全相同的，不同的地方在获取数据库句柄的地方

  传统方式：

//获取可写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase();

//获取可读数据库

SQLiteDatabase db = dbHelper.getReadableDatabase();

现在的方式：需要传入一个password，这个password就是用于加密的秘钥

//获取写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase(DBCipherHelper.DB\_PWD);

//获取可读数据库

SQLiteDatabase db = dbHelper.getReadableDatabase(DBCipherHelper.DB\_PWD);

接下来就是具体实现：

import android.content.ContentValues;

import android.content.Context;

import android.util.Log;

import net.sqlcipher.Cursor;

import net.sqlcipher.SQLException;

import net.sqlcipher.database.SQLiteDatabase;

/\*\*

\* 数据库管理者 - 提供数据库封装

\*

\*/

public class DBCipherManager {

private static final String TAG = "DatabaseManager";

// 静态引用

private volatile static DBCipherManager mInstance;

// DatabaseHelper

private DBCipherHelper dbHelper;

private DBCipherManager(Context context) {

dbHelper = new DBCipherHelper(context.getApplicationContext());

}

/\*\*

\* 获取单例引用

\*

\* @param context

\* @return

\*/

public static DBCipherManager getInstance(Context context) {

DBCipherManager inst = mInstance;

if (inst == null) {

synchronized (DBCipherManager.class) {

inst = mInstance;

if (inst == null) {

inst = new DBCipherManager(context);

mInstance = inst;

}

}

}

return inst;

}

/\*\*

\* 插入数据

\*/

public void insertData(String name) {

//获取写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase(DBCipherHelper.DB\_PWD);

//生成要修改或者插入的键值

ContentValues cv = new ContentValues();

cv.put(DBCipherHelper.FIELD\_NAME, name);

// insert 操作

db.insert(DBCipherHelper.TABLE\_NAME, null, cv);

//关闭数据库

db.close();

}

/\*\*

\* 未开启事务批量插入

\* @param testCount

\*/

public void insertDatasByNomarl(int testCount){

//获取写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase(DBCipherHelper.DB\_PWD);

for(int i =0;i<testCount;i++ ){

//生成要修改或者插入的键值

ContentValues cv = new ContentValues();

cv.put(DBCipherHelper.FIELD\_NAME, String.valueOf(i));

// insert 操作

db.insert(DBCipherHelper.TABLE\_NAME, null, cv);

Log.e(TAG, "insertDatasByNomarl");

}

//关闭数据库

db.close();

}

/\*\*

\* 测试开启事务批量插入

\* @param testCount

\*/

public void insertDatasByTransaction(int testCount){

//获取写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase(DBCipherHelper.DB\_PWD);

db.beginTransaction(); //手动设置开始事务

try{

//批量处理操作

for(int i =0;i<testCount;i++ ){

//生成要修改或者插入的键值

ContentValues cv = new ContentValues();

cv.put(DBCipherHelper.FIELD\_NAME, String.valueOf(i));

// insert 操作

db.insert(DBCipherHelper.TABLE\_NAME, null, cv);

Log.e(TAG, "insertDatasByTransaction");

}

db.setTransactionSuccessful(); //设置事务处理成功，不设置会自动回滚不提交

}catch(Exception e){

}finally{

db.endTransaction(); //处理完成

//关闭数据库

db.close();

}

}

/\*\*

\* 删除数据

\*/

public void deleteData(String name) {

//生成条件语句

StringBuffer whereBuffer = new StringBuffer();

whereBuffer.append(DBCipherHelper.FIELD\_NAME).append(" = ").append("'").append(name).append("'");

//获取写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase(DBCipherHelper.DB\_PWD);

// delete 操作

db.delete(DBCipherHelper.TABLE\_NAME, whereBuffer.toString(), null);

//关闭数据库

db.close();

}

/\*\*

\* 删除所有数据

\*/

public void deleteDatas()

{

String sql="delete from "+ DBCipherHelper.TABLE\_NAME;

execSQL(sql);

}

/\*\*

\* 更新数据

\*/

public void updateData(String name) {

//生成条件语句

StringBuffer whereBuffer = new StringBuffer();

whereBuffer.append(DBCipherHelper.FIELD\_NAME).append(" = ").append("'").append(name).append("'");

//生成要修改或者插入的键值

ContentValues cv = new ContentValues();

cv.put(DBCipherHelper.FIELD\_NAME, name+name);

//获取写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase(DBCipherHelper.DB\_PWD);

// update 操作

db.update(DBCipherHelper.TABLE\_NAME, cv, whereBuffer.toString(), null);

//关闭数据库

db.close();

}

/\*\*

\* 指定条件查询数据

\*/

public void queryDatas(String name){

//生成条件语句

StringBuffer whereBuffer = new StringBuffer();

whereBuffer.append(DBCipherHelper.FIELD\_NAME).append(" = ").append("'").append(name).append("'");

//指定要查询的是哪几列数据

String[] columns = {DBCipherHelper.FIELD\_NAME};

//获取可读数据库

SQLiteDatabase db = dbHelper.getReadableDatabase(DBCipherHelper.DB\_PWD);

//查询数据库

Cursor cursor = null;

try {

cursor = db.query(DBCipherHelper.TABLE\_NAME, columns, whereBuffer.toString(), null, null, null, null);

while (cursor.moveToNext()) {

int count = cursor.getColumnCount();

String columName = cursor.getColumnName(0);

String tname = cursor.getString(0);

Log.e(TAG, "count = " + count + " columName = " + columName + " name = " +tname);

}

if (cursor != null) {

cursor.close();

}

} catch (SQLException e) {

Log.e(TAG, "queryDatas" + e.toString());

}

//关闭数据库

db.close();

}

/\*\*

\* 查询全部数据

\*/

public void queryDatas(){

//指定要查询的是哪几列数据

String[] columns = {DBCipherHelper.FIELD\_NAME};

//获取可读数据库

SQLiteDatabase db = dbHelper.getReadableDatabase(DBCipherHelper.DB\_PWD);

//查询数据库

Cursor cursor = null;

try {

cursor = db.query(DBCipherHelper.TABLE\_NAME, columns, null, null, null, null, null);//获取数据游标

while (cursor.moveToNext()) {

int count = cursor.getColumnCount();

String columeName = cursor.getColumnName(0);//获取表结构列名

String name = cursor.getString(0);//获取表结构列数据

Log.e(TAG, "count = " + count + " columName = " + columeName + " name = " +name);

}

//关闭游标防止内存泄漏

if (cursor != null) {

cursor.close();

}

} catch (SQLException e) {

Log.e(TAG, "queryDatas" + e.toString());

}

//关闭数据库

db.close();

}

/\*\*

\* 执行sql语句

\*/

private void execSQL(String sql){

//获取写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase(DBCipherHelper.DB\_PWD);

//直接执行sql语句

db.execSQL(sql);//或者

//关闭数据库

db.close();

}

}

**4.）具体怎么调用**

//清空数据

DBCipherManager.getInstance(MainActivity.this).deleteDatas();

//插入数据

for (int i = 0; i < 10; i++) {

DBCipherManager.getInstance(MainActivity.this).insertData(String.valueOf(i));

}

//删除数据

DBCipherManager.getInstance(MainActivity.this).deleteData(String.valueOf(5));

//更新数据

DBCipherManager.getInstance(MainActivity.this).updateData(String.valueOf(3));

//查询数据

DBCipherManager.getInstance(MainActivity.this).queryDatas();

**5.）事务支持和传统方式一样**

//获取写数据库

SQLiteDatabase db = dbHelper.getWritableDatabase();

db.beginTransaction(); //手动设置开始事务

try{

//在此处理批量操作

for(int i =0;i<testCount;i++ ){

//生成要修改或者插入的键值

ContentValues cv = new ContentValues();

cv.put(DBHelper.FIELD\_NAME, String.valueOf(i));

// insert 操作

db.insert(DBHelper.TABLE\_NAME, null, cv);

}

db.setTransactionSuccessful(); //设置事务处理成功，不设置会自动回滚不提交

}catch(Exception e){

}finally{

db.endTransaction(); //处理完成

//关闭数据库

db.close();

}