Encrypted Databases Using SQLCipher

Rest and Motion

- Securing Data at Rest = Local Storage
  - Databases
  - SharedPreferences
  - Other Types of Files
- Securing Data in Motion = Internet (mostly)
  - SSL
  - OTR

The Droid Is Not Enough

- Lock Screen?
  - Mechanical brute forcing
- Internal Storage?
  - Rooting
- Full-Disk Crypto?
  - Digital brute forcing

Your Objectives (One Hopes)

- Cheap and Easy Security
  - Only have so much time to budget
  - Aiming for "low hanging fruit"
- Effective Security
  - "Using CryptoLint, we performed a study on cryptographic implementations in 11,748 Android applications. Overall we find that 10,327 programs – 88% in total – use cryptography inappropriately. The raw scale of misuse indicates a widespread misunderstanding of how to properly use cryptography in Android development."

You're Doing It Wrong

- Hardcoded Passphrases
- Manually Seeding SecureRandom
  - ...with a hardcoded seed
- Hardcoded Salts
- Insufficient Key Generation Iterations
- Non-Random Initialization Vectors

Introducing SQLCipher

- SQLCipher
  - Modified version of SQLite
  - AES-256 encryption by default, of all data
  - Relatively low overhead
  - Cross-platform
  - BSD license

Introducing SQLCipher

- SQLCipher Security
    - Customizable encryption algorithm
        - Based on OpenSSL libcrypto
    - Individual pages encrypted, with own initialization vector
    - Message authentication code (MAC) per page, to detect tampering
    - Hashed passphrase (PBKDF2) for key
        - 4,000 iterations, moving to 64,000 for 3.0

Introducing SQLCipher

- SQLCipher for Android
  - NDK-compiled binaries
  - Drop-in replacement classes for Android's SQLite classes
    - SQLiteDatabase
    - SQLiteOpenHelper
    - Etc.
  - Modify your code, third-party libraries also using SQLite

Integrating SQLCipher

- Step #1: Add to Project
  - Download ZIP file from: `http://sqlcipher.net/downloads/`
  - Copy ZIP's `assets/` into project's `assets/`
  - Copy ZIP's `libs/` into project's `libs/`
  - Call `SQLiteDatabase.loadLibs()` before use
    - Needs a `Context`

Integrating SQLCipher

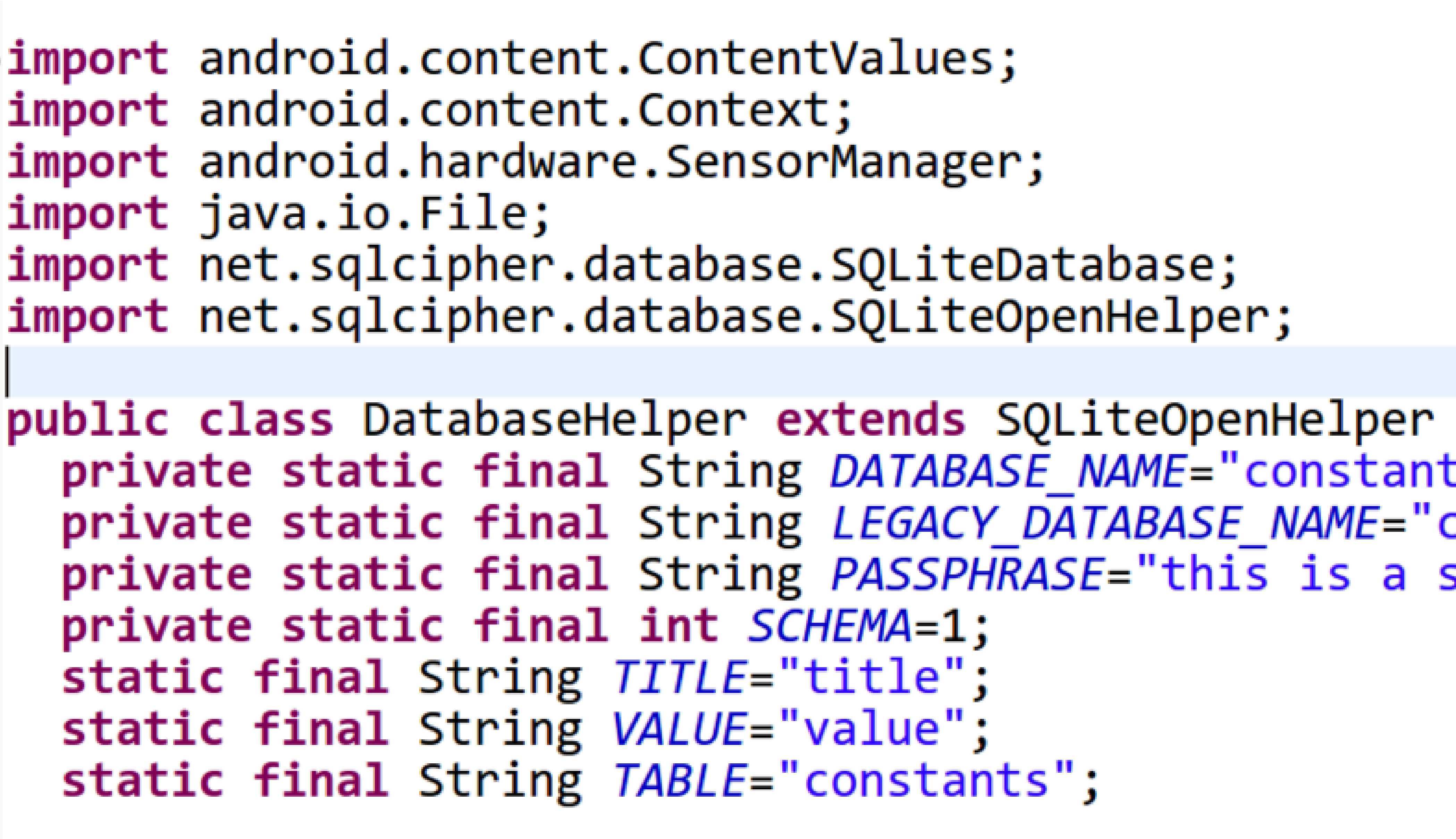
- Step #2: Replace Import Statements
  - Some `android.database.*` and all `android.database.sqlite.*` imports
    - Move to `net.sqlcipher` equivalents

Integrating SQLCipher

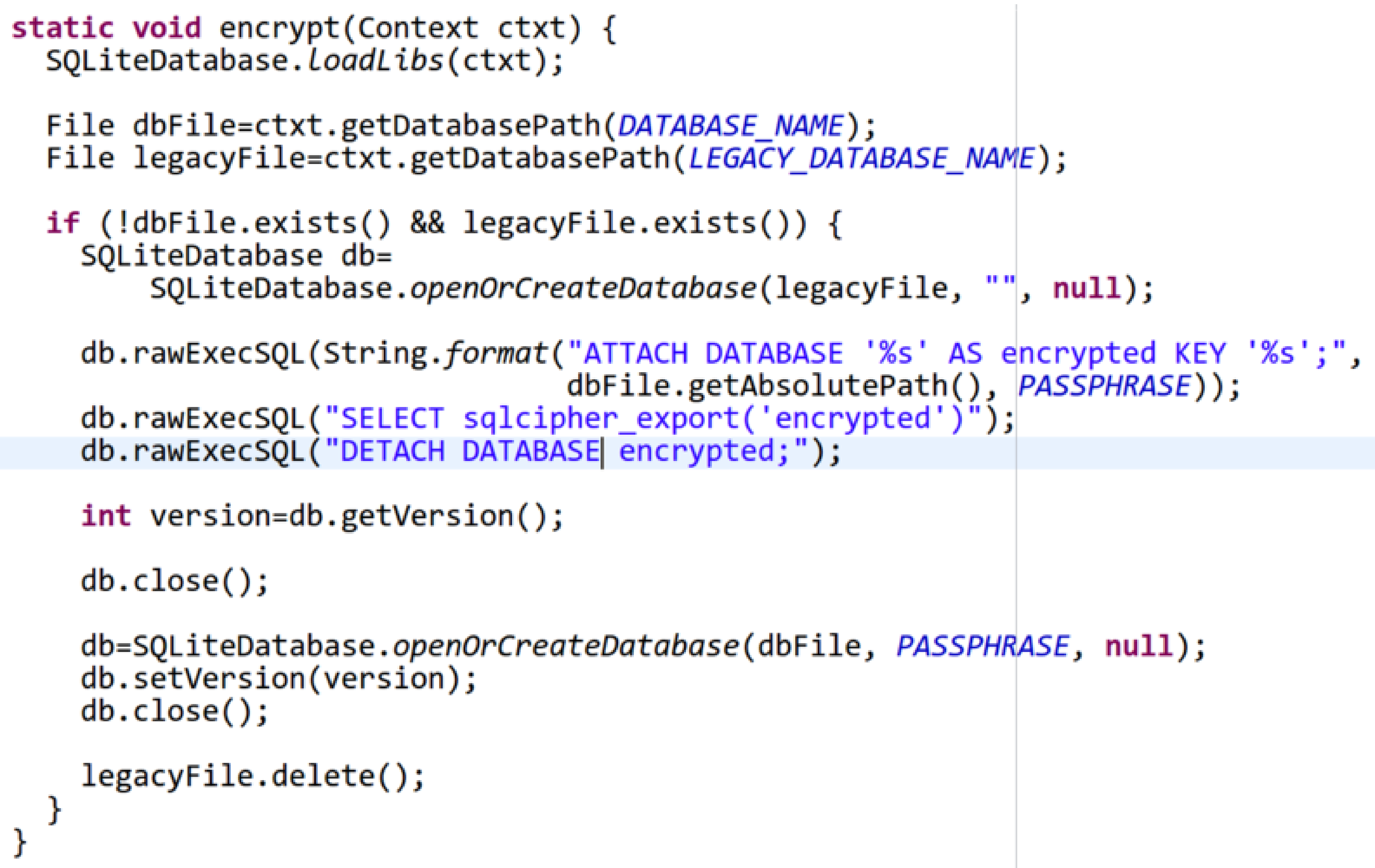
- Step #3: Supply Passphrases
  - `SQLiteDatabase` `openOrCreateDatabase()`, etc.
  - `SQLiteOpenHelper` `getReadableDatabase()` and `getWritableDatabase()`
  - Collect passphrase from user via your own UI and test cases

Integrating SQLCipher

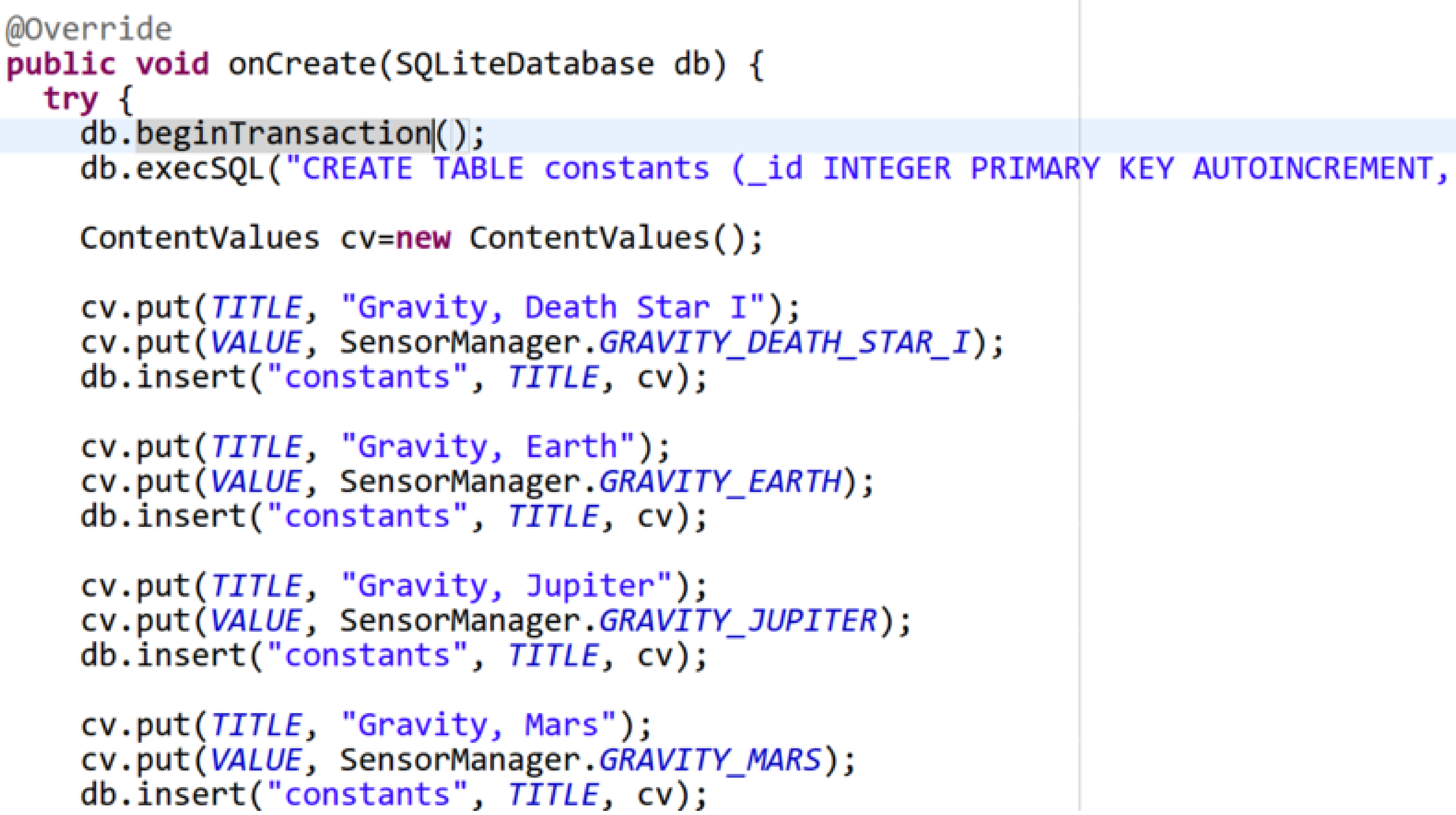
- Step #4: Testing
  - Tests should work when starting with a clean install
    - No existing unencrypted database
- Step #5: Beer!
  - Hooray, beer!



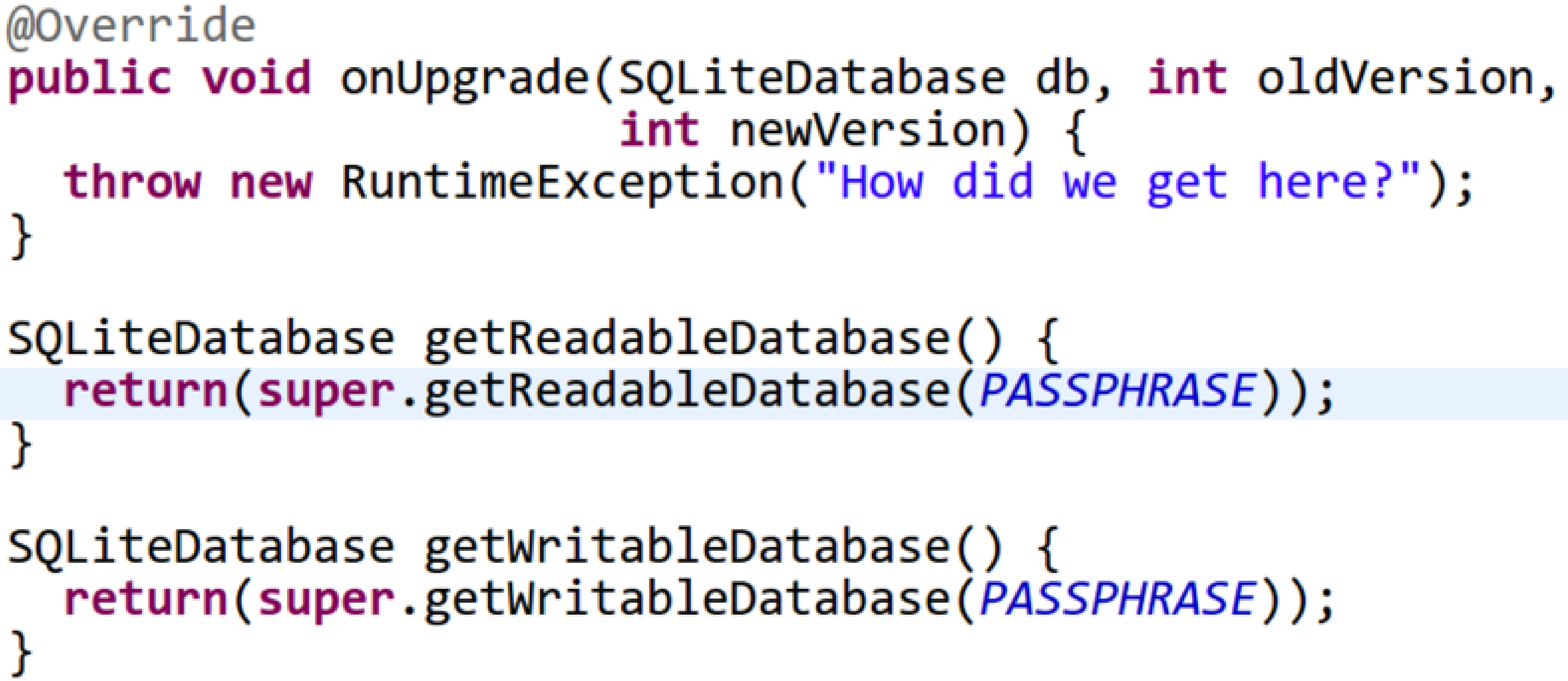
```
import android.content.ContentValues;
import android.content.Context;
import android.hardware.SensorManager;
import java.io.File;
import net.sqlcipher.database.SQLiteDatabase;
import net.sqlcipher.database.SQLiteOpenHelper;

public class DatabaseHelper extends SQLiteOpenHelper
  private static final String DATABASE_NAME="constant
  private static final String LEGACY_DATABASE_NAME="c
  private static final String PASSPHRASE="this is a s
  private static final int SCHEMA=1;
  static final String TITLE="title";
  static final String VALUE="value";
  static final String TABLE="constants";
```



```
static void encrypt(Context ctxt) {
  SQLiteDatabase.loadLibs(ctxt);

  File dbFile=ctxt.getDatabasePath(DATABASE_NAME);
  File legacyFile=ctxt.getDatabasePath(LEGACY_DATABASE_NAME);

  if (!dbFile.exists() && legacyFile.exists()) {
    SQLiteDatabase db=
        SQLiteDatabase.openOrCreateDatabase(legacyFile, "", null);

    db.rawExecSQL(String.format("ATTACH DATABASE '%s' AS encrypted KEY '%s';",
                                dbFile.getAbsolutePath(), PASSPHRASE));
    db.rawExecSQL("SELECT sqlcipher_export('encrypted')");
    db.rawExecSQL("DETACH DATABASE encrypted;");

    int version=db.getVersion();

    db.close();

    db=SQLiteDatabase.openOrCreateDatabase(dbFile, PASSPHRASE, null);
    db.setVersion(version);
    db.close();

    legacyFile.delete();
  }
}
```



```
@Override
public void onCreate(SQLiteDatabase db) {
  try {
    db.beginTransaction();
    db.execSQL("CREATE TABLE constants (_id INTEGER PRIMARY KEY AUTOINCREMENT,

    ContentValues cv=new ContentValues();

    cv.put(TITLE, "Gravity, Death Star I");
    cv.put(VALUE, SensorManager.GRAVITY_DEATH_STAR_I);
    db.insert("constants", TITLE, cv);

    cv.put(TITLE, "Gravity, Earth");
    cv.put(VALUE, SensorManager.GRAVITY_EARTH);
    db.insert("constants", TITLE, cv);

    cv.put(TITLE, "Gravity, Jupiter");
    cv.put(VALUE, SensorManager.GRAVITY_JUPITER);
    db.insert("constants", TITLE, cv);

    cv.put(TITLE, "Gravity, Mars");
    cv.put(VALUE, SensorManager.GRAVITY_MARS);
    db.insert("constants", TITLE, cv);
```



```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion,
                      int newVersion) {
  throw new RuntimeException("How did we get here?");
}

SQLiteDatabase getReadableDatabase() {
  return(super.getReadableDatabase(PASSPHRASE));
}

SQLiteDatabase getWritableDatabase() {
  return(super.getWritableDatabase(PASSPHRASE));
}
```

About the Bloat

- 4MB base
- Additional ~5MB for x86
- Additional ~3MB for ARM
- Why?
  - Complete independent copy of SQLite
  - Static library implementation of OpenSSL
  - Independent copy of ICU collation ruleset

Mitigating the Bloat

- Option #1: Multiple APKs
  - Use Gradle for Android to create CPU-specific builds of your APK
  - Upload the builds to the Play Store, which will distribute right build to right device
- Option #2: `libhoudini`
  - Available on some x86 devices, allows running ARM native binaries, so no need to package x86
  - Downside: speed

About the Performance

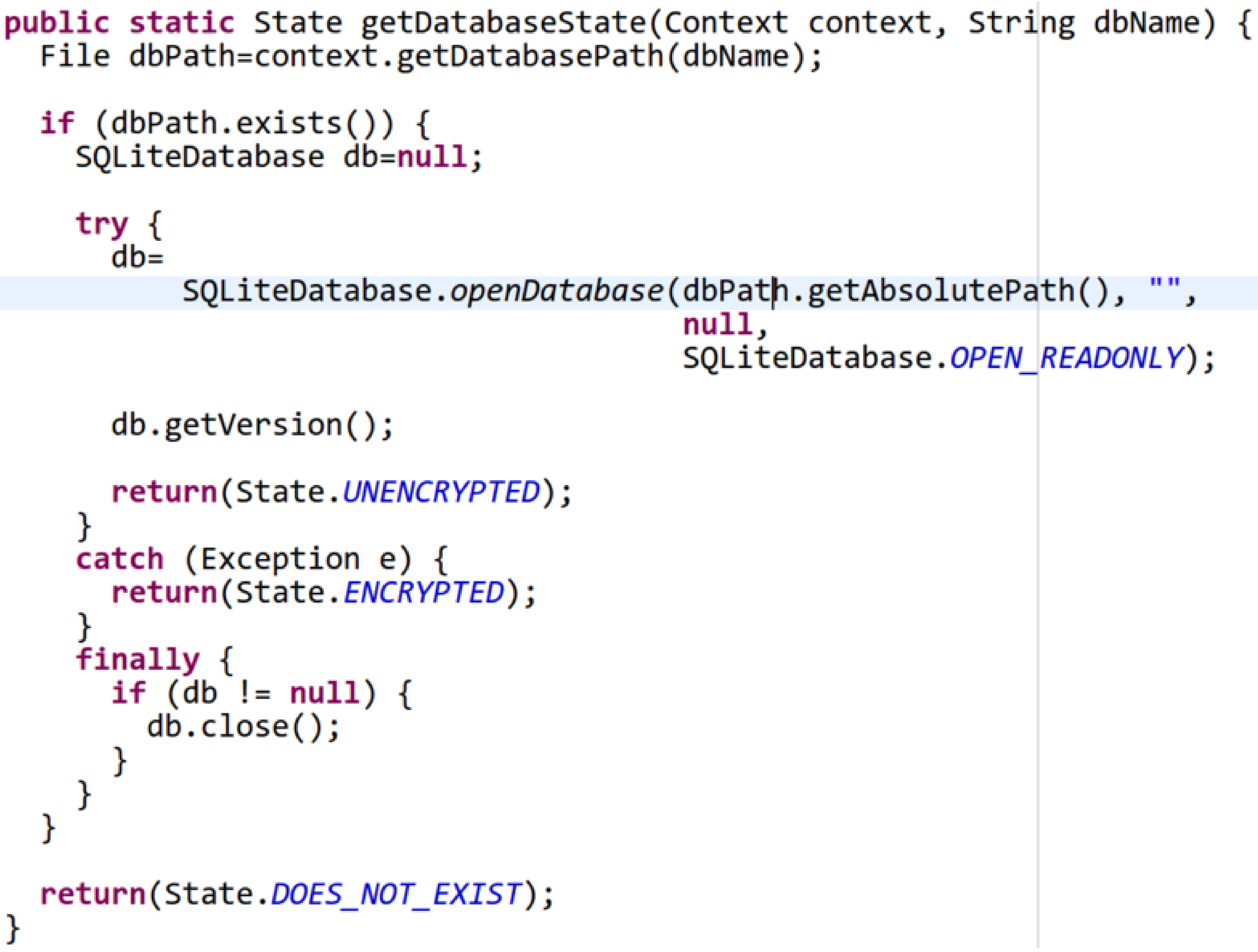
- Overall: Not Bad
  - Depending upon benchmark, may not notice any speed changes of significance
  - CPU time for crypto is dwarfed by I/O time
- Makes Bad Things Worse
  - Avoid table scans!

Upgrading to Encryption

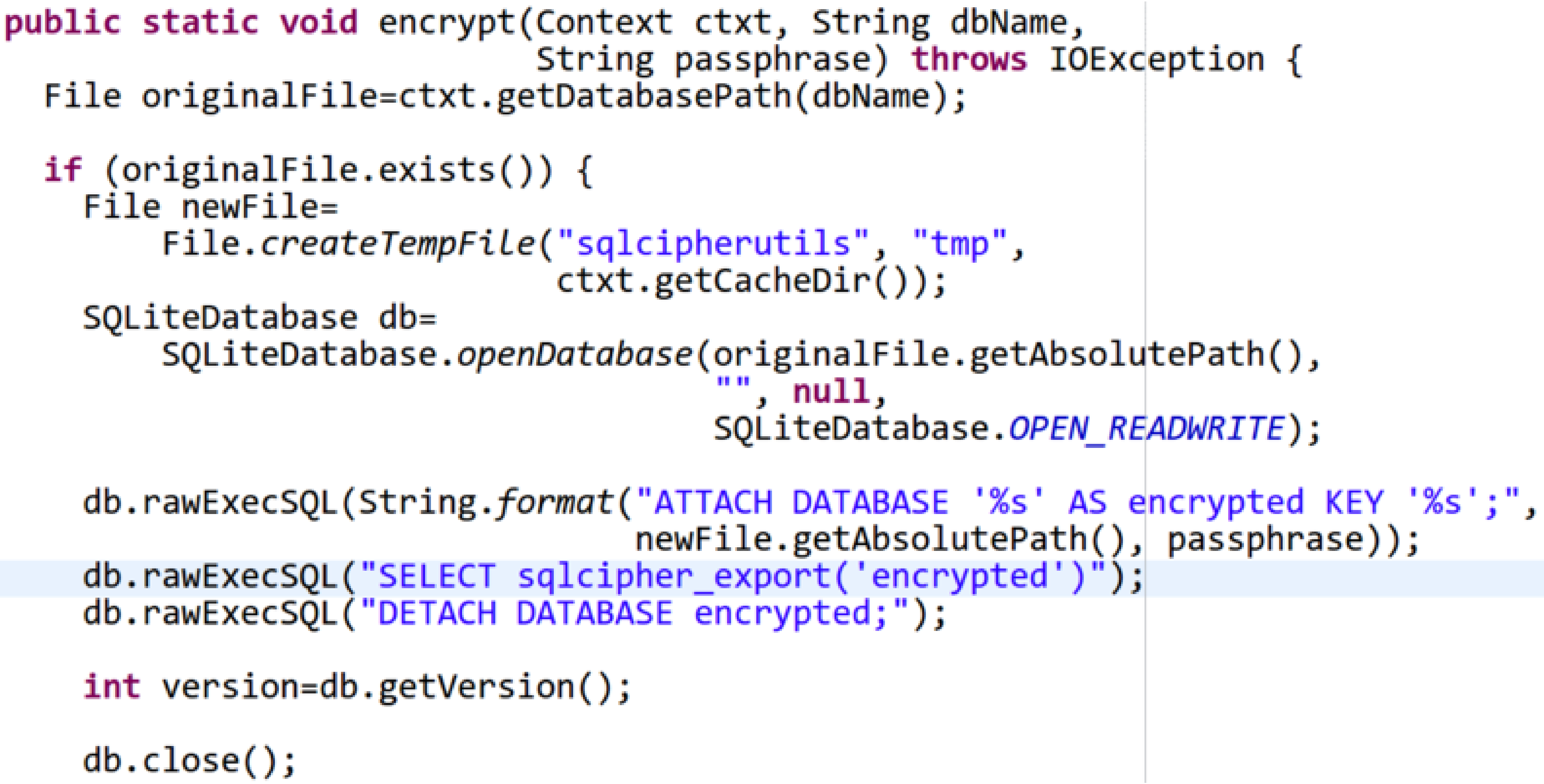
- Option #1: Encrypt It Immediately
  - Recipe for replacing a regular database with an encrypted one
  - Requires getting the passphrase from the user
- Option #2: Encrypt It Someday
  - User option to encrypt the database, triggered from your UI
- Option #3: Static Passphrase Immediately
  - With user option to re-encrypt later

Upgrading to Encryption

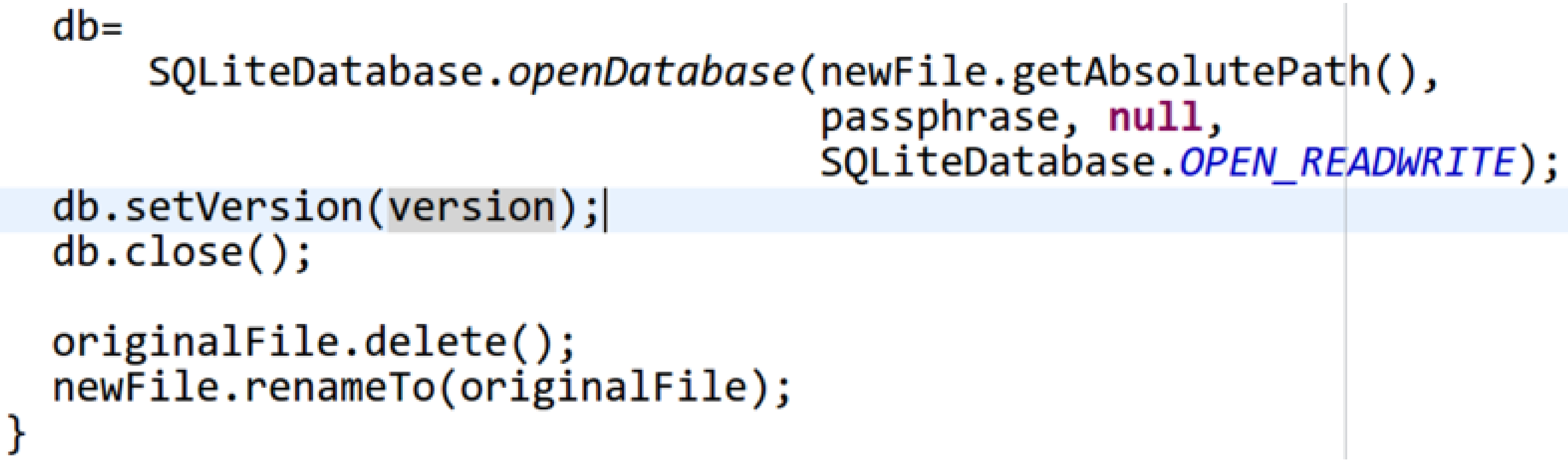
- SQLCipherUtils
  - Found in CWAC-LoaderEx
  - Helper Methods
    - `getDatabaseState()`: some indication if the database is encrypted or not
    - `encrypt()`: replace unencrypted database with an encrypted one



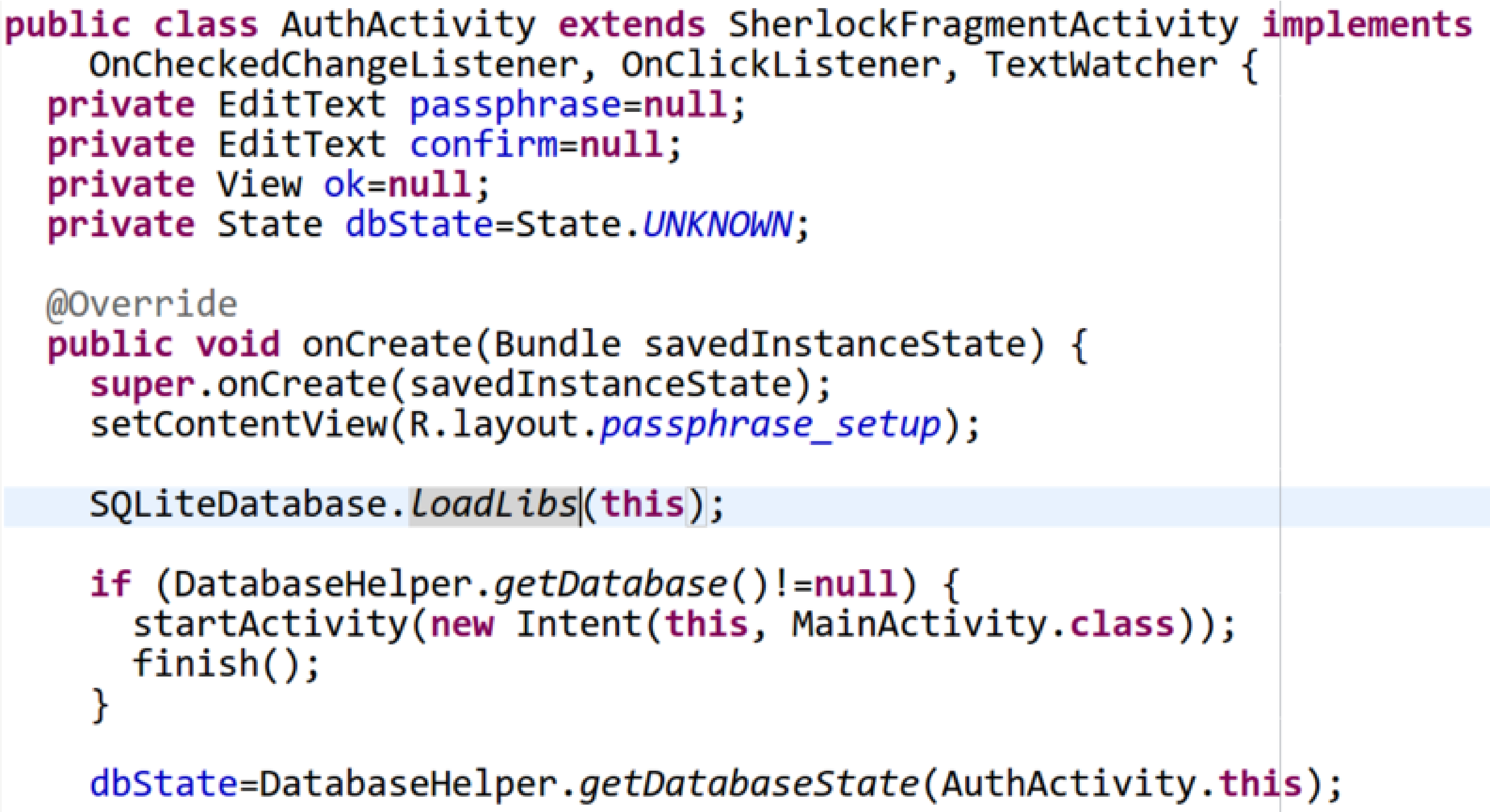
```java
public static State getDatabaseState(Context context, String dbName) {
  File dbPath=context.getDatabasePath(dbName);

  if (dbPath.exists()) {
    SQLiteDatabase db=null;

    try {
      db=
          SQLiteDatabase.openDatabase(dbPath.getAbsolutePath(), "",
                                      null,
                                      SQLiteDatabase.OPEN_READONLY);

      db.getVersion();

      return(State.UNENCRYPTED);
    }
    catch (Exception e) {
      return(State.ENCRYPTED);
    }
    finally {
      if (db != null) {
        db.close();
      }
    }
  }

  return(State.DOES_NOT_EXIST);
}
```



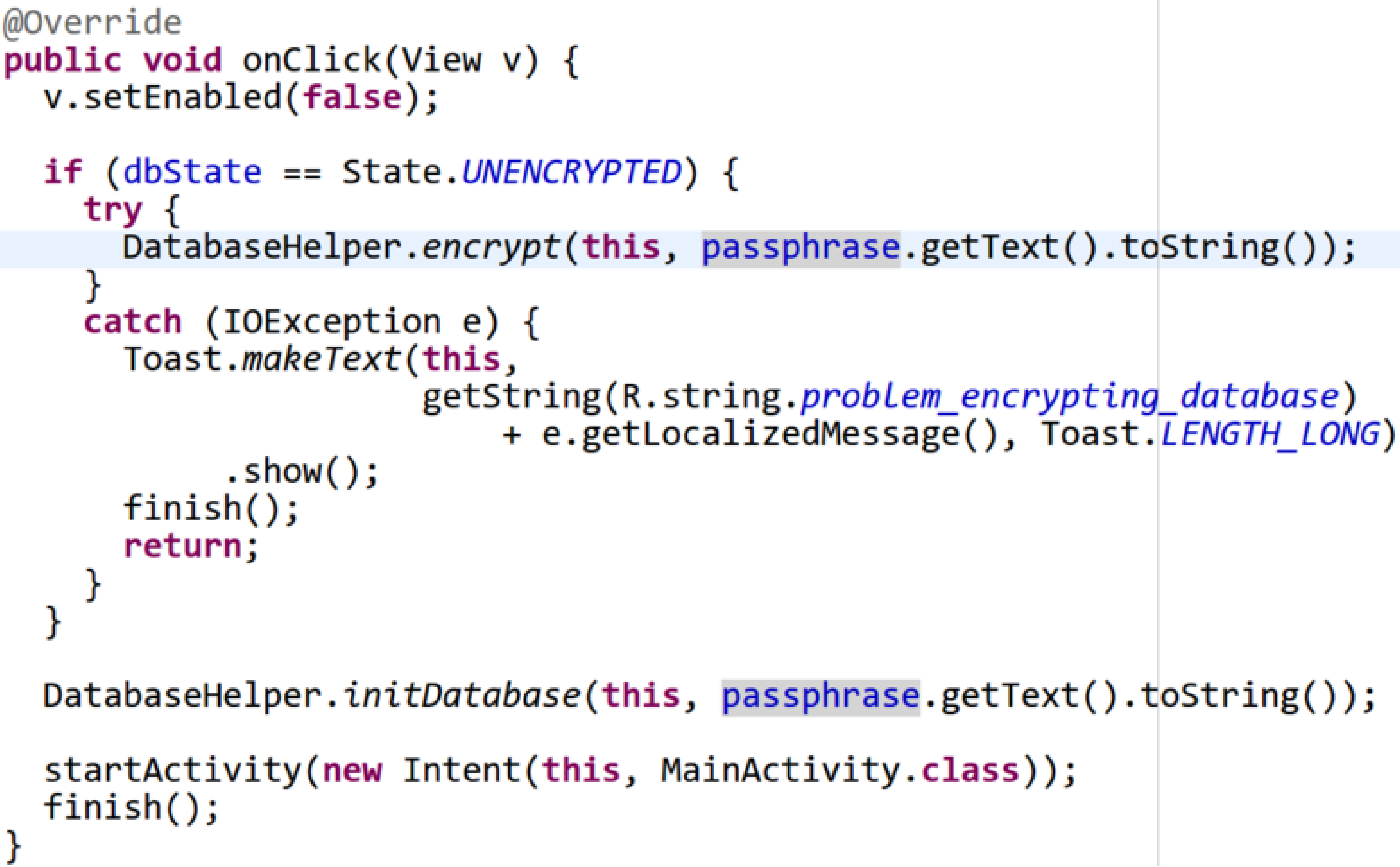
```
public static void encrypt(Context ctxt, String dbName,
                           String passphrase) throws IOException {
  File originalFile=ctxt.getDatabasePath(dbName);

  if (originalFile.exists()) {
    File newFile=
        File.createTempFile("sqlcipherutils", "tmp",
                            ctxt.getCacheDir());
    SQLiteDatabase db=
        SQLiteDatabase.openDatabase(originalFile.getAbsolutePath(),
                                    "", null,
                                    SQLiteDatabase.OPEN_READWRITE);

    db.rawExecSQL(String.format("ATTACH DATABASE '%s' AS encrypted KEY '%s';",
                                newFile.getAbsolutePath(), passphrase));
    db.rawExecSQL("SELECT sqlcipher_export('encrypted')");
    db.rawExecSQL("DETACH DATABASE encrypted;");

    int version=db.getVersion();

    db.close();
```



```
  db=
      SQLiteDatabase.openDatabase(newFile.getAbsolutePath(),
                                  passphrase, null,
                                  SQLiteDatabase.OPEN_READWRITE);
  db.setVersion(version);
  db.close();

  originalFile.delete();
  newFile.renameTo(originalFile);
}
```



```
public class AuthActivity extends SherlockFragmentActivity implements
    OnCheckedChangeListener, OnClickListener, TextWatcher {
  private EditText passphrase=null;
  private EditText confirm=null;
  private View ok=null;
  private State dbState=State.UNKNOWN;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.passphrase_setup);

    SQLiteDatabase.loadLibs(this);

    if (DatabaseHelper.getDatabase()!=null) {
      startActivity(new Intent(this, MainActivity.class));
      finish();
    }

    dbState=DatabaseHelper.getDatabaseState(AuthActivity.this);
```



```java
@Override
public void onClick(View v) {
  v.setEnabled(false);

  if (dbState == State.UNENCRYPTED) {
    try {
      DatabaseHelper.encrypt(this, passphrase.getText().toString());
    }
    catch (IOException e) {
      Toast.makeText(this,
                     getString(R.string.problem_encrypting_database)
                         + e.getLocalizedMessage(), Toast.LENGTH_LONG)
           .show();
      finish();
      return;
    }
  }

  DatabaseHelper.initDatabase(this, passphrase.getText().toString());

  startActivity(new Intent(this, MainActivity.class));
  finish();
}
```

Integrating SQLCipher

- ContentProvider
  - Can work, but need to get passphrase to it before using the database (e.g., `call()`)
  - Typically means that it does not work well for providers used totally independently from app
    - Still usable for activities launched from an app, where the data should be retrieved immediately, before the process gets terminated

Integrating SQLCipher

- Loaders
  - CWAC-LoaderEx and SQLCipherCursorLoader
    - Works
    - Problem: must route all database modifications through Loader to automatically get the updated Cursor
  - ContentProvider
  - Just Say No
    - Use your own AsyncTasks, event bus, etc.

IOCipher

- Virtual Filesystem
  - Use replacement classes for java.io.File and kin
- Encrypted using SQLCipher for Android
  - No actual files stored directly
  - Shares encryption key with your own tables
- Still Early Days
- https://guardianproject.info/code/iocipher/

Slides!