CS 522—Spring 2016
Mobile Systems and Applications
Assignment Eleven—Data Security

This assignment builds on the previous assignments, by securing the data stored on the chat application. The idea is to encrypt the user and chat data stored on disk, and decrypt the data for retrieval and update. One way to do this is to encrypt the file system under the user password, but this only works if it is a single-user device. We will instead pursue an approach of encrypting the database using SQL Cipher, a version of SQLite that is available for iOS and Android. You will have to use some of the SQL Cipher versions of the SQLite APIs in your code, and you will need to include native code libraries for SQL Cipher in your chat app (the app that uses a database in the implementation of a content provider).

You can find more information about installing SQL Cipher here:

https://androidbycode.wordpress.com/2015/02/18/android-database-encryption-using-sqlcipher/

Rather than download the implementation of SQL Cipher and (carefully) copy it to your app's library folder (Section 2 of the above blog post), you can instead just add a dependency on the AAR for SQL Cipher, by adding this dependency to your project's Gradle build file:

```
compile 'net.zetetic:android-database-sqlcipher:3.3.1-1@aar'
```

Then follow Section 3 above, which describes the imports that you need to change in your content provider in order to use SQL Cipher rather than SQLite. In your content provider, you will need to load the libraries for SQL Cipher when you create the database:

```
public boolean onCreate() {
      Context context = getContext();
      SQLiteDatabase.loadLibs(context);
      openHelper = new DatabaseHelper(context);
      ...
}
```

The database will be encrypted with a 256-bit AES key. Rather than storing this key with the database, you should instead provide the key with every operation from a client of the content provider, providing the key as a query parameter in the URI specified for the operation. You should follow the following protocol for managing the key in your code.

First, define base classes for your activities, such as `BaseActivity` (extending `Activity`) and `BaseListActivity` (extending `ListActivity`). These classes should include the database key as a private field in the activity:

```
public class BaseActivity extends Activity {

    private char[] databaseKey;

    protected final char[] getDatabaseKey() {
        return databaseKey;
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        databaseKey =
            getIntent().getCharArrayExtra(SECURITY_DATABASE_KEY);
        if (databaseKey == null) {
            throw new IllegalArgumentException("..."));
        }
    }
}
```

Every intent that starts an activity or service should include the database key as an extra argument. An activity saves this key, and passes it to any manager objects it creates for accessing the content provider:

```
MessageManager messageMgr = new MessageManager(this, getDatabaseKey());
```

The manager object in turn caches the database key when it is created. When an operation is invoked, the manager then adds the database key as a query parameter to the URI used to identify the content being accessed:

```
char[] databaseKey;
Uri.Builder builder = uri.buildUpon();
Uri uriWithKey =
    builder.appendQueryParameter(SECURITY_PARAMETER_KEY,
                                 new String(databaseKey))
           .build();
```

The implementation of the content provider expects the database key to be provided in this way as a query parameter in the URI for each operation, for example:

```
@Override
public Cursor query(Uri uri, ...) {

    String k = uri.getQueryParameter(SECURITY_PARAMETER_KEY);
    char[] databaseKey = null;
    if (k == null) {
        throw new IllegalArgumentException("...");
    } else {
        databaseKey = k.toCharArray();
    ]
```

```
    try {
        SQLiteDatabase db =
            openHelper.getReadableDatabase(databaseKey);
        ...
    } finally {
        for (int i=0; i<databaseKey.length; i++) {
            databaseKey[i] = ' ';
        }
    }
}
```

Note that the operations for opening a database, for reading and writing, now require the private AES key used to encrypt the database.

The database is initialized with the encryption key the first time that a database operation is performed. It is important therefore that you perform some content provider operation, with the encryption key as a query parameter, before any other processing is performed in the content provider.

Where will the encryption key be stored? We will take the approach of storing it off the device, as a QR code that a user must provide when they start the app[1]. The device's camera is used to read this QR code and parse it for the database key. You are provided with a `Capture` app for doing this. Using an explicit intent, start the activity `edu.stevens.cs522.capture.client.CaptureActivity`[2]. This expects two arguments as intent extras:

1. A string message to be displayed.
2. An integer identifier for the camera to be used.

The activity is styled to execute as a dialog, hovering over your user interface. It will present a video of the viewfinder, with a red-lined square within the viewfinder image which you should make sure contains the image of the QR code. The camera never takes a picture, instead it reads the stream of viewfinder images until it successfully parses a QR code.

The capture activity should return to your application with a result intent. Check the result in `onActivityResult`:

```
@Override
public void onActivityResult(int requestCode,
                             int resultCode,
                             Intent data) {
    switch (resultCode) {
```

[1] You are provided with a QR code for a SQL Cipher key at the end of this document.

[2] Use a component name with package name `edu.stevens.cs522.capture` and fully qualified activity

[2] Use a component name with package name `edu.stevens.cs522.capture` and fully qualified activity name `edu.stevens.cs522.capture.client.CaptureActivity`.

```
        case CaptureClient.CAPTURE_OK:
            String databaseKey =
                      data.getStringExtra(CaptureClient.RESULT_KEY);
            // Launch chat activity with the key
            break;
        case CaptureClient.CAPTURE_CANCELED:
            // Client cancelled app startup
            break;
        case CaptureClient.CAMERA_UNAVAILABLE:
            // No camera available, cannot proceed
            break;
        default:
            throw new IllegalArgumentException("...");
    }
}
```

Permissions

You should also use permissions to protect the integrity of your app suite (you will have three apps at the conclusion of this assignment). Define a permission for using the capture app, require it for starting `CaptureActivity`, and define it as a signature permission. Your chat app will have to request this permission, in its manifest and at runtime, before it tries to start `CaptureActivity`. Similarly, since the mapping app you defined in Assignment 10 requires access to the content provider with sensitive information (chat messages and people's locations), define read and write permissions as signature permissions for the content provider. The mapping app will have to request read permission in order to access the content provider.

Submitting Your Assignment

Once you have your code working, please follow these instructions for submitting your assignment:

1. Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory Humphrey_Bogart.
2. In that directory you should provide the Android Studio projects for your Android apps.
3. Also include in the directory a report of your submission. This report should be in PDF format. Do not provide a Word document.

In addition, record short flash, mpeg, avi or Quicktime videos of a demonstration of your assignment working. Make sure that your name appears at the beginning of the video. For example, put your name in the title of the app. *Do not provide private information such as your email or cwid in the video*.

Your solution should be uploaded via the Canvas classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have a single Android Studio project, for the app you have built. You should also provide a report in the root folder, called README.pdf, that contains a report on your solution, as well as videos demonstrating the working of your assignment.

Here is a QR code that you can use for reading a secret key for your SQL Cipher database: