Android SDK Quick Start Guide

Integrate the SensifyID SDK into your Android App

This guide describes the steps required to add the SensifyID SDK into your Android app. If you aren't ready to integrate the SDK into your Android app yet, and wish to learn more about the capabilities of the solution, visit the Zighra website.

Requirements

A minSdkVersion of 19 is required in order for the application to build and run properly. Modify your Android Studio project's build file (`build.gradle`) accordingly.

The SDK and sample application will need a mobile device to run, they will not work on emulators.

Download the SDK

The SDK is part of the downloaded package as a binary framework form (.aar). Once downloaded, add the AAR file as a dependency in your Android Studio's gradle (build.gradle(Project:<projname>) & build.gradle(Module: app) files.

NOTE:: Do not use the AAR on the Quick Start developer sample solution shared along with the package. You should be able to run ONLY the bundled sample with the library which comes along with it.

Obtain your App Key and Secret

Contact the Zighra Support team to get your app key and secret, which uniquely identify your app to the framework and the Zighra solution.

Other Dependencies

The SDK already embeds the library, com.madgag.spongycastle:prov:1.54.0.0, as a dependency. If the application also requires this library as a dependency, please contact support at Zighra solution.

Configure the SDK

Set the configuration values of the SDK as meta-data values in your Android Studio project's manifest file. The SDK is accessed as a shared instance (singleton) that should be configured only once during the lifecycle of the app.

Specify your app key, app secret and encryption setting in your project's AndroidManifest.xml file as follows. By default, encryption is disabled.

```
<meta-data android:name="zighra_app_key"        android:value="your_app_key"   />
<meta-data android:name="zighra_app_secret"     android:value="your_app_secret"/>

<!-- optional; by default false-->
<meta-data android:name="zighra_app_encryption" android:value="true/false"     />

<!-- optional; by default use SDK default URL (api.zighra.com)-->
<meta-data android:name="zighra_server_url" android:value="SensifyID WebAPI URL"     />

<!-- optional; by default use SDK default port (3001)-->
<meta-data android:name="zighra_server_port" android:value="SensifyID WebAPI port"     />
```

Access the SDK interface

Access the interface of the SDK with in your Activity, via `kineticFactory.getKinetic(Context cxt)`. It gives back the interface of the SensifyID SDK. You can access all SDK functionalities through this interface.

```
public class SetProfileActivity extends Activity {
 private Kinetic mKinetic;
 ...
 protected void onCreate(Bundle savedInstanceState) {
  ...
  // Get the  SDK interface
  mKinetic = KineticFactory.getKinetic(this);
  ...
  }
 ...
}
public class SwipeActivity extends Activity {
 private Kinetic mKinetic;
 ...
 protected void onCreate(Bundle savedInstanceState) {
  ...
  // access the singleton SDK instance
  mKinetic = KineticFactory.getKinetic(this);
  ...
  }
 ...
}
```

User Profile

The user of your app must be uniquely identified within the SensifyID system. Each user is identified by a `uCode` (user code) and `name` (user name) property. We recommend choosing a user code that relates to your app user's login identity, and passing this information to the SensifyID SDK via the `kinetic.setProfile(String name, String uCode ...)`. On the first call of this method, the SDK will communicate with the SensifyID web service to create a new profile for the user and respond with a unique profile code. On subsequent calls with the same user code, the user's profile code will be retrieved from the shared preferences on the device.

setProfile(String uName, String uCode) takes Kinetic.OnSetProfileSuccessListener and Kinetic.OnSetProfileFailureListener, to receive the results.

Create/set profile using the uName and uCode provided as follows

```
...
////Call this function to create/set current profile. Check the setProfile status passing anonymous l
mKinetic.setProfile(uName, uCode, new Kinetic.OnSetProfileSuccessListener() {
 @Override
 public void onSuccess(Kinetic.ProfileResponse profileResponse) {
  ...
 }},
 new Kinetic.OnSetProfileFailureListener() {
  @Override
   public void onFailure(Kinetic.ProfileResponse profileResponse) {
    ...
   }
})
...
```

Device Checks

In order to detect potential fraudulent access from another device, and ensure that any learned authentication models associated with a device cannot be adopted by a new device, your app must perform a device check with the SDK. A device check builds a unique fingerprint for the device and asks the SensifyID web service to validate the fingerprint against the previously stored fingerprint associated with the user's profile.

At a minimum, a device check should be performed in the following points in the runtime lifecycle of the app: At the beginning of your user's session with your app, such as after a login. When your user is logged in, a device check should be performed after app launch and when the app resumes from a suspended state.

Check device can be called via `mKinetic.checkDevice()` as follows

```
#!
...
//Call checkDevice and later in the activity, check the status by passing the anonymous listeners
mKinetic.checkDevice(new Kinetic.OnCheckDeviceSuccessListener() {
@Override
public void onSuccess(Kinetic.CheckDeviceResponse checkDeviceResponse) {
    ...
}
}, new Kinetic.OnCheckDeviceFailureListener() {
@Override
public void onFailure(Kinetic.CheckDeviceResponse checkDeviceResponse) {
    ...
}
}));
...
```

Integrate Gesture Authentication

The SDK provides several options for authenticating the user with a swipe gesture. Your app may use swipe-based authentication as either the primary way for the user to authorize the completion of a task, or as a secondary authentication presented to the user periodically or when a device check fails.

Authentication is performed by attaching an existing `View`, such as a `TextView` in your user interface to swipe authenticator. The authenticator listens to the user's direct interactions with the view (swipe movements) and the user's indirect interactions (the position and movement of the device), and submits an authentication request to the SensifyID web service.

Attach the view once, such as during your Activity's `onCreate(Bundle savedInstanceState)` method. The SensifyID web service returns a response indicating how closely the user's gesture matches the learned model constructed from previous interactions.

You must also pass authentication listeners, Kinetic.OnAuthenticationSuccessListener and Kinetic.OnAuthenticationFailureListener at Attaching function call, on the SDK instance that examines the response and returns a value that expresses how the app may handle the gesture authentication response. The response includes a numeric Score that reflects how closely the gesture matches the model. Based on this, the app should call reportAction(String action...), as described in the next section called Action Reporting:

Attach the view to swipe authenticator as follows

```
mKinetic.attachSwipeAuthenticator(view, new Kinetic.OnAuthenticationSuccessListener() {
@Override
public void onSuccess(Kinetic.AuthenticationResponse authenticationResponse) {
    ...
}
}, new Kinetic.OnAuthenticationFailureListener() {
@Override
public void onFailure(Kinetic.AuthenticationResponse authenticationResponse) {
```

```
    ...
}
});
});
```

The attaching function takes optional listeners to handling pre- and post-authentication events :

- `Kinetic.OnWillAuthenticationListener` listens to the event dispatched before SDK inform SensifyID web service starting authentication.
- `Kinetic.OnDidAuthenticationListener` listens to the event dispatched after the authentication completed.
- `KineticTouchListener` which capture and do the action for touch event.

The response includes a `status` property (enum 'modelNotReady', 'success' ... etc) that indicates whether the request was processed successfully and state of the authentication model. If the status is modelNotReady, it indicates the authentication model is still learning the user's gesture patterns. The learning phase should end after approximately 15 gesture authentication attempts.

The following code segment show how to use the 'status' and 'score' property to evaluate the model state and gesture matches in order to call reportAction(String action...)

```
...
float swipeThreshold = 70 f;
Kinetic.AuthStatus status = authenticationResponse.getStatus();
switch (status) {
    case modelNotReady:
        Toast.makeText(getApplicationContext(), "Training in progress", Toast.LENGTH_SHORT).show();
        mKinetic.reportActionForAuth("allow", new Kinetic.OnReportActionSuccessListener() {
            @Override
            public void onSuccess(Kinetic.ReportResponse reportResponse) {
                ...
            }
        }, new Kinetic.OnReportActionFailureListener() {
            @Override
            public void onFailure(Kinetic.ReportResponse reportResponse) {
                ...
            }
        });
        break;
    case success:
        // Examine the gesture score
        if (authenticationResponse.getScore() > swipeThreshold) {
            //Action Reporting
            mKinetic.reportActionForAuth("allow", new Kinetic.OnReportActionSuccessListener() {
                @Override
                public void onSuccess(Kinetic.ReportResponse reportResponse) {
                    ...
                }
            }, new Kinetic.OnReportActionFailureListener() {
                @Override
                public void onFailure(Kinetic.ReportResponse reportResponse) {
                    ...
                }
            });
        } else {
            // Authentication failed. Present authentication UI.
        }
        break;
    default:
        // Authentication failed. Present authentication UI.
}
...
```

Action Reporting

When a gesture fails to authenticate due to a low score, the app should take additional steps to authenticate the user and then report the subsequent action as `allow` or `deny` back to the SensifyID SDK. Invoke the `mKinetic.reportAction(String action)` method, passing the action value

- Pass the action value `allow` to inform the SensifyID web service that your app considers the gesture value, and the gesture should be incorporated into the learning authentication model.
- Pass the action value `deny` value to inform the SensifyID web service that the gesture should **not** be incorporated into the learning authentication model.

The response includes a `status` property that indicates whether the request was processed successfully, and whether the authentication model is still learning the user's gesture patterns. The learning phase should end after approximately 15 gesture authentication attempts.

```
...
 //call reportAction with `allow` to incorporate the gesture into the learning authentication model
 mKinetic.reportActionForAuth("allow", new Kinetic.OnReportActionSuccessListener() {
             @Override
             public void onSuccess(final Kinetic.ReportResponse reportResponse) {
                 ...
             }
         }, new Kinetic.OnReportActionFailureListener() {
             @Override
             public void onFailure(final Kinetic.ReportResponse reportResponse) {
                 ....
             });

//or call reportAction with `deny` to not incorporate the gesture into the learning authentication mo
 mKinetic.reportAction("deny", null, null);
...
```

The reportAction also has two optional parameters :

- `TaskAction` specify the particular action for gesture check/task.
- `ExtraInfo` additional optional information pass to server

Lifecycle handling (Android Only)

The SDK is running certain background tasks to collect the data during the activity lifetime. It is best practice to notify the SDK when the activity is onPause() or onResume() so that the SDK can stop and restart the background tasks respectively.

```
class Activity extends AppCompatActivity {
    ...

    @Override
    protected void onResume() {
        super.onResume();

        //Integrate Gesture Authentication
        mKinetic.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();

        //Integrate Gesture Authentication
```

```
        mKinetic.onPause();
    }
    ...
}
```