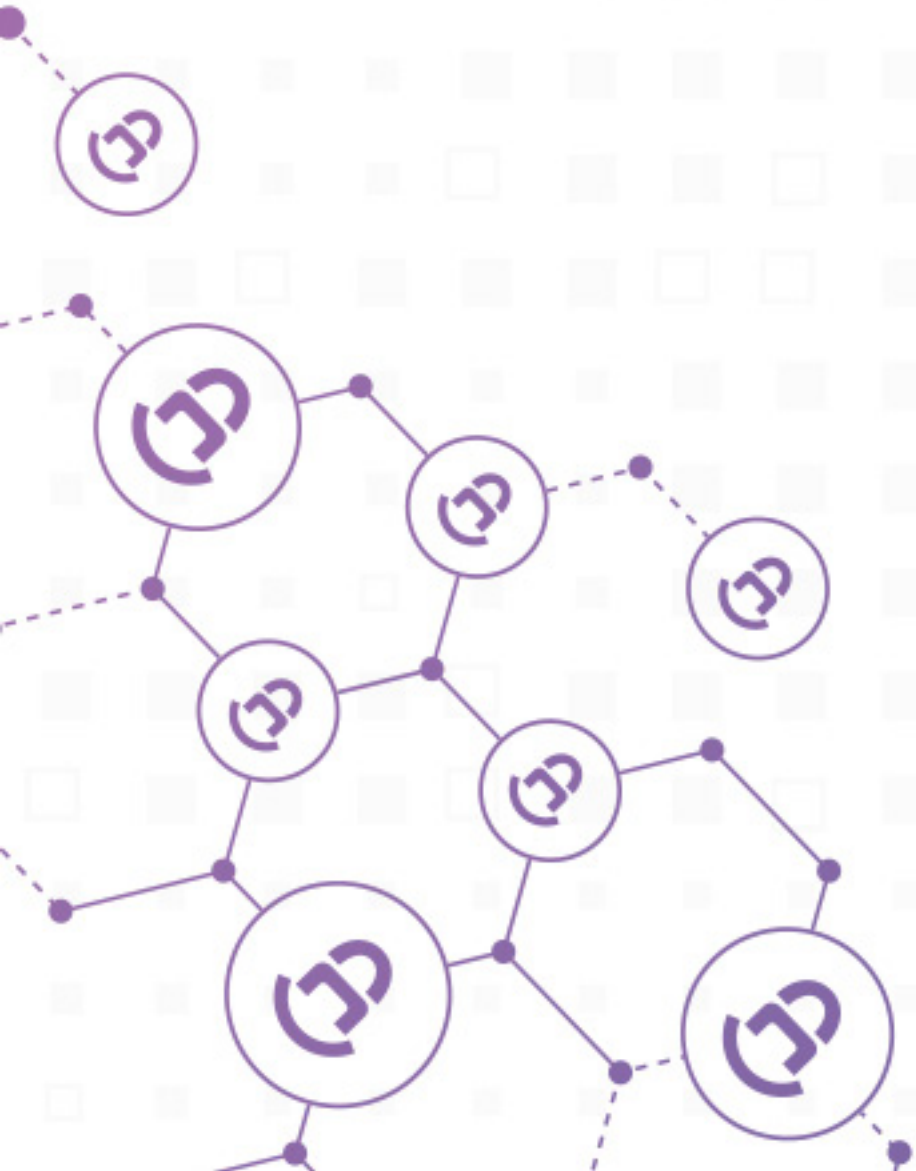
Version 1.0

APPSEALING SECURE

WEBVIEW USER GUIDE



appsealing

TABLE OF CONTENTS

INTRODUCTION

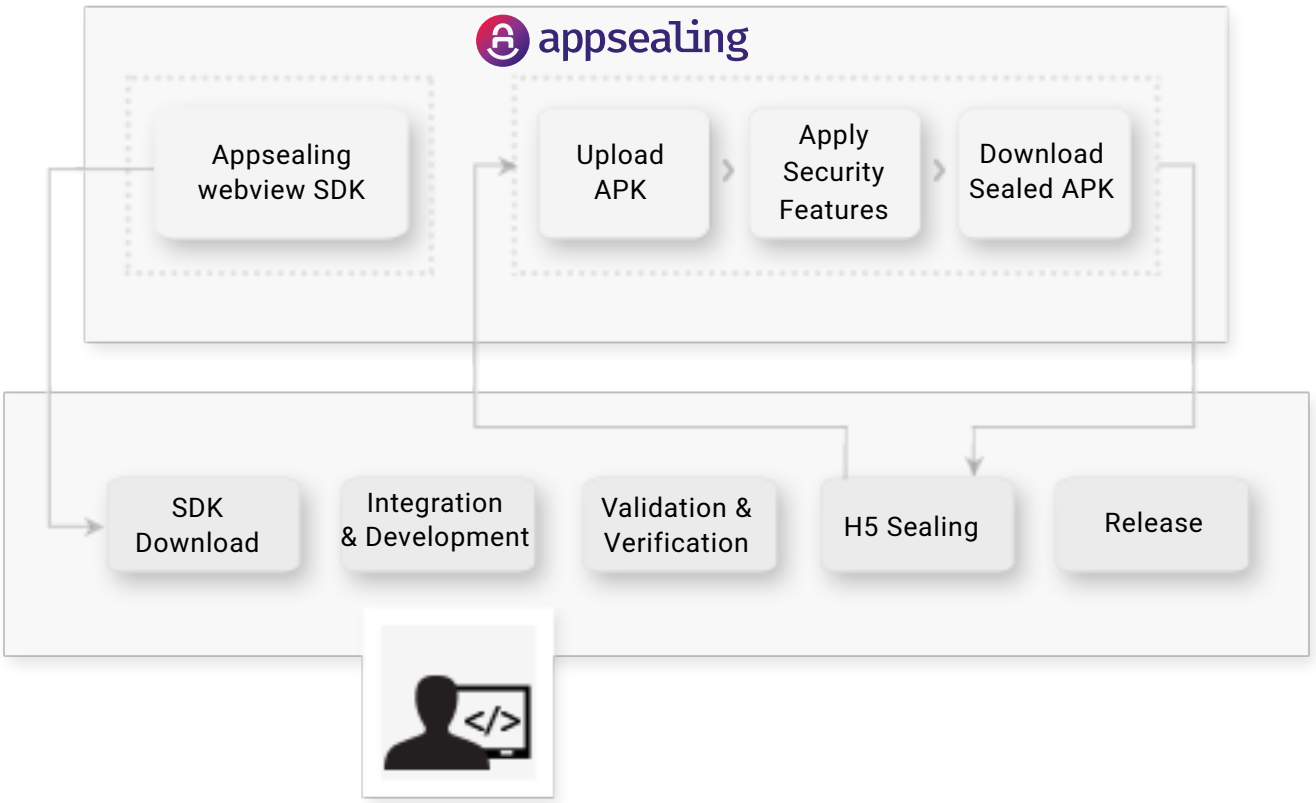
Android WebView provided by Android is used to develop various types of hybrid applications, such as connecting the native application to the web or creating applications based on local web content.

Despite the advantages of these hybrid applications, each of the security threats of the native application and the web application exists due to the characteristics of the hybrid application. In particular, hybrid applications expose the original source in JavaScript. This makes them vulnerable to attacks on key algorithms, key information and sensitive data in the business Hybrid applications developed using AppSealing Secure WebView secure key business logic code and intellectual property. AppSealing Secure WebView is based on Chromium, and provides API similar to Android WebView to help developers easily apply AppSealing Secure WebView.

For details on the HTML5 content encryption process, please refer to the AppSealing website or this user guide. For additional technical inquiries related to using the SDK, please use the AppSealing Help Center (https://helpcenter.appsealing.com).

NOTE: AppSealing Secure Webview SDK can be downloaded after joining ADC. After signing up, you will go through an approval process and you can download it at the ADC site later.

HYBRID APP SECURITY APPLICATION SCENARIO



1. Download AppSealing Secure WebView SDK.
2. Develop Hybrid App to which AppSealing Secure WebView SDK is applied by referring to the documents included in AppSealing Secure WebView SDK (this document and AppSealing Secure WebView API document and sample app).
3. Verify Hybrid App which AppSealing Secure WebView SDK is applied to through the customer's internal development process.
4. Apply AppSealing Hybrid App security service (hereinafter referred to as Hybrid App security service) to the app that has been internally verified to apply security for internal content and the entire developed app.
5. After confirming the operability of the App to which the Hybrid App security service is applied, release it according to the normal development procedure of the customer.

INTEGRATION & DEVELOPMENT

3.1 Development support environment

Android 5.0 or higher

This SDK has been tested with Android SDK 3.6.2.

3.2 SDK Package Contents

Item	Description
AppSealingSecureWebviewLibrary.aar	Android library file containing various resources, assets, and native libraries that make up AppSealing Secure WebView
AppSealingSecureWebViewSample.zip	Android studio project files containing sample app
AppSealingSecureWebViewAPIDoc.zip	APIs list and descriptions provided by AppSealing Secure WebView

3.3 Android Studio project settings

You can add the AppSealing Secure WebView SDK to your project with the following process.

3.3.1 Add AppSealing Secure Webview Library

Copy the <AppSealingSecureWebviewLibrary.aar> file to the library file storage space in the root folder `($ Project_Dir)` of the project you are using. If there is no folder for saving the library in the project folder you are using, create a "libs" folder in the `$Project_Dir\app\` folder and copy the <AppSealingSecureWebviewLibrary.aar> file. And add below contents to build.gradle file

```
dependencies {
...
    implementation fileTree(dir: 'libs', include: ['*.aar'])
...
}
```

3.3.2 Configuration for Java 1.8 support

For JAVA 1.8 support, the following is reflected in build.gradle.

```
android {
...
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
...
}
```

3.3.3 Setting to support uncompressed assets

Modify the compression uncompressed settings for Assets files. Uncompressed option must be set for data used inside AppSealing Secure WebView to read files normally. Modify the build.gradle as below.

```
android {
  ...
    aaptOptions { noCompress 'dat', 'pak' }
  ...
}
```

Currently, AppSealing Secure WebView requires uncompressed settings for the extensions “dat” and “pak”

3.3.4 Build Tool setting

Set the below SDK in module `build.gradle`. AppSealing Secure WebView SDK version 16 or higher is supported.

```
defaultConfig {

    minSdkVersion 16
    targetSdkVersion 28

}
```

Please refer to the above described settings as it is applied to the sample project distributed in the SDK.

3.4 Application development guide

3.4.1 Initialize AppSealing Secure WebView

To use AppSealing Secure WebView in the activity you use, you need to initialize the process to use `<AppSealing Secure WebView>.In onCreate()`, the following procedure is performed to start AppSealing Secure WebView.

```
@Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
.
.
.
        setContentView(R.layout.inka_webview_activity);
        mPreferences = new ASPreferences();

        mASInitializer = new ASInitializer(new ASInitializer.
        ASInitListener() {
            @Override
            public void onASInitStarted() {
                Log.d(TAG, "onASInitStarted");
            }
```

```
@Override
            public void onASInitCancelled() {
                Log.d(TAG, "onASInitCancelled");
            }

            @Override
            public void onASInitFailed() {
                Log.d(TAG, "onASInitFailed");
            }

            @Override
            public void onASInitCompleted() {
                Log.d(TAG, "onASInitCompleted");
                initASetting();
            }

            @Override
            public void onASFirstRenderProcessReady() {

            }
        }, this);

        if (!mASInitializer.initSync(disableSandbox)) {
          // Something is wrong.
        }
```

3.4.2 Web contents loading

After the initialization process, create an instance of AppSealing Secure WebView and load the desired page. To load the page, code as below.

```
boolean incognito = false;
 mASWebView = new ASWebView(MainActivity.this, incognito);
 mASWebView.loadUrl("http://google.com");
```

Local page access example

```
mASWebView.loadUrl("file:///android_asset/index.html");
```

In this case, [Applicaiontion_Root] /app/src/main/asset/index.html must exist. See sample application codes.

Remote webpage access example

3.4.3 How to use setUIClient

```
mASWebView.loadUrl ("http://www.naver.com");
```

AppSealing Secure WebView, like Android WebView, provides a way to override the behavior of the default web browser. AppSealing Secure WebView provides a method called `setUIClient()` that matches the `setWebChromeClient()` method of Android WebView. Please refer to the example below for specific usage.

```
mASWebView.setUIClient(new ASUIClient(mASWebView) {
            @Override
            public boolean onJsAlert(ASWebView view, String url, String
message, ASJavascriptResult result) {
                return super.onJsAlert(view, url, message, result);
            }

            @Override
            public void onGeolocationPermissionsShowPrompt(ASWebView view,
java.lang.String origin,
                    ASGeoLocationPermissionsCallback callback) {
                callback.invoke(origin, true, false);
            }

            @Override
            public void onPageLoadStarted(ASWebView view, java.lang.String
url) {
                Log.e(TAG, "onPageLoadStarted =" + url);
                mIsLoading = true;
                if (mUrlTextView != null)
                    mUrlTextView.setText(url);


mStopReloadButton.setImageResource(android.R.drawable.ic_menu_close_clear_
cancel);
            }

            @Override
            public void onPageLoadStopped(ASWebView view, java.lang.String
url, ASUIClient.LoadStatus status) {
                Log.e(TAG, "onPageLoadStopped =" + url + " status=" +
status);
                mIsLoading = false;
                mStopReloadButton.setImageResource(R.drawable.ic_refresh);

           }
```

```
@Override
            public void onFullscreenToggled(ASWebView view, boolean
enterFullscreen) {
                super.onFullscreenToggled(view, enterFullscreen);
                Log.d(TAG, "onFullscreenToggled : " +
enterFullscreen);

                 LinearLayout toolBar = (LinearLayout)
findViewById(R.id.toolbar);
                toolBar.setVisibility(enterFullscreen ? GONE :
VISIBLE);

            }
              @Override
            public boolean onConsoleMessage(ASWebView view, String
message, int lineNumber, String sourceId,
                    ASUIClient.ConsoleMessageType messageType) {
                Log.d(TAG, "onConsoleMessage:" + message + "
source=" + sourceId + " line#=" + lineNumber);
                super.onConsoleMessage(view, message, lineNumber,
sourceId, messageType);
                return true;
            }
            // ...
        });
```

3.4.4 How to use setResourceClient

Like Android WebView, it provides a way to override the behavior of the web browser associated with each network connection that the web browser sends and receives. AppSealing Secure WebView provides `setResourceClient()` method. This method is mapped to the `setResourceClient()>`method of Android WebView. You can use it as below

```
mASWebView.setResourceClient(new ASResourceClient(mASWebView) {
            @Override
            public void onLoadStarted(ASWebView view, String url) {
                Log.d(TAG, "onLoadStarted url=" + url);
                super.onLoadStarted(view, url);
            }
```

```
 @Override
            public void onLoadFinished(ASWebView view, String url)
{
                Log.d(TAG, "onLoadFinished url=" + url);
                super.onLoadFinished(view, url);
            }
@Override
            public ASWebResourceResponse
shouldInterceptLoadRequest(ASWebView view, ASWebResourceRequest
request) {
                Log.d(TAG, "shouldInterceptLoadRequest");
                return super.shouldInterceptLoadRequest(view,
request);

            }

            @Override
            public void onProgressChanged(ASWebView view, int
progressInPercent) {
                Log.d(TAG, "onProgressChanged Progress=" +
progressInPercent);
                super.onProgressChanged(view, progressInPercent);

mUrlTextView.removeCallbacks(mClearProgressRunnable);
                mProgressDrawable.setLevel((int) (progressInPercent
* 100));
                if (progressInPercent == 100)

mUrlTextView.postDelayed(mClearProgressRunnable,
COMPLETED_PROGRESS_TIMEOUT_MS);
            }

            @Override
            public void onReceivedLoadError(ASWebView view, int
errorCode, String description, String failingUrl) {
                Log.d(TAG,
                        "onReceivedLoadError errorCode=" +
errorCode + " url=" + failingUrl + " Desc=" + description);
                super.onReceivedLoadError(view, errorCode,
description, failingUrl);
            }

        });
```

By using the sample application source provided separately and various AppSealing APIs, you can write the application in the form you want. For more information on the AppSealing API, see the AppSealing API Document.

APPLYING APPSEALING HYBRID APP SECURITY SERVICE

In order to operate the local HTML5 content security and the application's own security function of the application that is equipped with the AppSealing Secure WebView SDK, Hybrid App security service provided by Inca Networks must be applied.

Hybrid App security service can be applied in the following order.

1.Go to the AppSealing Developer Console page (https://developer.appsealing.com/).
2. Create account or log in to use Hybrid App security service.
3. Upload the Hybrid App with AppSealing Secure Webview and set the Hybrid App security service desired by the customer. And apply the set Hybrid App security service. (It takes a few minutes)
4. When the Hybrid App security service is finished, you can download the APK with the security service applied.
5. Download the APK and apply the signature again.
6. Verify the APK with security applied.

Please refer to the AppSealing Help Center for details on setting up the Hybrid App security service.

CHECK HYBRID APP SECURITY SERVICE IS APPLIED

In order to operate the local HTML5 content security and the application's own security function of the application that is equipped with the AppSealing Secure WebView SDK, Hybrid App security service provided by Inca Networks must be applied.

Local contents test

1. Save the HTML content containing JavaScript in the android asset folder.
2. Make sure that the local HTML content is properly loaded in AppSealing Secure WebView before applying Hybrid App security service.
3. After checking the step #2, build the APK and apply the Hybrid App security service.
4. After applying the Hybrid App security service, download the generated APK.
5. Open the downloaded APK using the zip tool, and check if the encrypted JavaScript exists in the same name in the android asset folder.
6. After checking item 5, check if the locally saved HTML page is properly loaded from the downloaded APK.

Remote contents test

1. Stores HTML content containing JavaScript on a remote web server.
2. Make sure that the HTML content of the remote web server is properly loaded in AppSealing Secure WebView before applying Hybrid App security service,
3. After checking the step #2, zip the HTML content to be applied to the encryption and security service among the HTML content of the remote web server and apply the Hybrid App security service.
4. After applying the security service, download the created zip file.
5. Open the downloaded zip file and check if there is JavaScript encrypted with the same path and name as the original zip file.
6. After checking the step #5, unzip the downloaded zip file and save it on the remote web server.
7. Check if the HTML page with Hybrid App security service is loaded properly in the AppSealing Secure WebView with Hybrid App security service.

CHECK HYBRID APP SECURITY SERVICE IS APPLIED

Please contact us through the AppSealing Help Center (https://helpcenter.appsealing.com/) for inquiries or technical support during the test.

Until the Hybrid App security service is applied to the APK, the encrypted Javascript page cannot be loaded normally.