

Mocana Lighthouse Getting Started Guide

6.11.2015

Overview

The Mocana Atlas **Lighthouse Software Development Kit** (Lighthouse SDK) is used within Mocana's Atlas enterprise mobile app security platform to provide a simple way for developers to add **X.509 certificate-based** authentication to their iOS and Android apps. The Lighthouse SDK and the Atlas platform allow enterprise administrators to present users with a **single login screen**, securely consolidating enterprise authentication and individual app logins.

Mocana Atlas Lighthouse SDK with the Atlas platform is a proven approach, that sets the new standard in enterprise mobile app security and end user experience. For more information, visit [Mocana Atlas Platform](#).

Prerequisites

It's easy to get started.

1. Join the [Mocana ON Developer Program](#) for free.

As a member of the **Mocana Developer Program** you will get everything you need to develop, wrap and deploy secure apps. At the bottom of the page, be sure to check the box labeled: **Integrate the Mocana Lighthouse SDK**.

2. Download

An email will be sent by Mocana in response to your request to join the Mocana ON Developer Program. When you receive your confirmation email, click the link to download the **Mocana Atlas Lighthouse SDK**.

These development environment-specific files enable the core features of the Lighthouse SDK:

Android

- `MAPCertificateProvider.java`

iOS

- `MAPSDK.h`
- `MAPSDK.m`

Included in the Lighthouse SDK are Android and iOS [Sample Applications](#) showing sample integrations. If you'd like to share your own example, just [let us know](#).

- **Android:** [`map-certificate-viewer`] (<https://github.com/MocanaCorp/Lighthouse-SDK/tree/master/samples/android>)
- **iOS:** [`X509 Embedded`] (<https://github.com/MocanaCorp/Lighthouse-SDK/tree/master/samples/ios/X509%20Embedded>)

3. Terminology

This document assumes a basic familiarity with **Mocana's Atlas Platform**. Enumerated below are key components involved in integrating the Lighthouse SDK. If you're already familiar with these technologies, and just want to look at the source code, feel free to skip ahead to the **API Reference** and integrate with your organization's back-end.

1. **App wrapping** injects enterprise security into mobile apps. You can *app-wrap* an app binary (.ipa or .apk) with Mocana's **MAP (Mobile Application Protection)** to apply policies, add further authentication or restrict an app or device from specific functions.
2. **MAP Wrapper** is Mocana's policy and security code that is injected into apps.
3. **MAP-Wrapped app** is your *successfully MAP-Wrapped* application.
4. **Atlas** is Mocana's mobile app security gateway.
5. **Active Directory (AD)** is Microsoft's directory services database that Atlas uses for authenticating app users.
6. **Certificate or Certification Authority (CA)** is the enterprise's certificate authority that Atlas uses to issue user certificates.
7. **Lighthouse SDK** allows app developers to obtain and use the user certificate managed by Atlas and the MAP Wrapper. Typically the certificate is used within the app for certificate-based authentication when connecting to enterprise services.

Test Scenarios

To better understand how to test and utilize the **Mocana Atlas Lighthouse SDK** we've outlined four short modules with increasing levels of complexity. These scenarios serve as a gentle introduction to concepts and best practices for mutual authentication over SSL using an X.509 certificate to provide secure Web communication.

The first module runs on a local client without the need for a test server. The second module is a Sample Application (**Android** or **iOS**) connected to Atlas. The third module is an example application, **LightHouseSampleClient**, connected to Atlas which in turn connects to a back-end test server,

MocanaLighthouseSampleServer. The last module provides resources for using Lighthouse SDK with the technologies and frameworks that are appropriate for your environment.

To begin, choose an environment, **Android** or **iOS** and a scenario:

1. **Introduction to Lighthouse SDK** In this scenario, you'll compile and run the Sample Application in debug mode on a local client without the need of a network connection.
 - **Android**
 - **iOS**
2. **Wrap the Sample App with MAP and connect to Atlas** In this scenario, you'll compile and run the Sample Application and then wrap it with MAP policies of your choosing. Run the MAP-wrapped Sample Application and test your connection to Mocana's hosted Atlas, using your **Mocana Developer Program** username and password, which is emailed upon MDP enrollment.
 - **Android**
 - **iOS** (See *code-signing script_note* in section two for iOS apps)
3. **Lighthouse Sample Client to Atlas-connected MocanaLighthouse Sample Server** In this **iOS only** scenario, you'll have access to the precompiled **MocanaSSO** application, the Lighthouse Sample Client and the Lighthouse Sample Server and to the source code. Having successfully connected to Atlas, you'll connect to the Mocana-hosted back-end test server: **gilbert.mocana.local**.
 - **API Reference**
 - **Authentication**
4. **Connect your Organization's Wrapped Application**, Integrate the Mocana Atlas Lighthouse SDK. Compile, wrap and connect to Atlas. Integrate your back-end production environment.

Development Environment

If you develop apps today there are no additional tools needed to use the Lighthouse SDK.

Android:

Compatible with Android 4.x or greater and [Android SDK Tools](#) version 1.6 or newer. Works with Integrated Development Environments (IDE) such as **Eclipse**, and **IntelliJ IDEA**. *NOTE: Requires SDK libraries for Google Android API v21.1.1 or higher*

The [system requirements](#) to develop with the Android SDK are:

- A computer capable of running Android Studio, Eclipse or IntelliJ.
- An Android mobile device.

iOS:

Compatible with iOS 7.x and greater. To develop with iOS and **Xcode**, Apple's Integrated Development Environment (IDE), you must:

- Be registered as an Apple's iOS Developer.
- Have a Mac running OS X version 10.9.4 or later.
- Xcode Command Line Tools.
- An iOS mobile device.

1. Introduction to Lighthouse SDK

Lighthouse SDK is used in combination with the Atlas platform. Developers of connected enterprise mobile apps can minimize the number of authentication prompts their users face by enabling certificate-based authentication to enterprise systems and services, typically with mutually-authenticated SSL. The Lighthouse SDK connects the app to the MAP Wrapper to obtain the user certificate that is provisioned and managed by Atlas and the MAP Wrapper.

In this scenario you'll compile and run the Android or iOS Sample Application in debug mode on a local client. Upon success, the application returns a certificate view with a green **thumbs-up** icon.

To begin, choose the **Android** or **iOS** Sample App:

Sample Applications

- **Android:** *map-certificate-viewer*
- **iOS:** *X509 Embedded*

Android Sample App: *map-certificate-viewer*

Download and test the **unwrapped** sample Android application on the Android device.

The Android sample project, **map-certificate-viewer**, is implemented as a *single view application* and does not have any major dependencies outside of the Lighthouse SDK (which is included in this Sample Application).

Note: The minimum requirements are the build tools and SDK libraries for Google Android API v21.1.1. It is also recommended to update your *build.gradle* file, located in the **map-certificate-viewer** directory. Once updated, your **build.gradle** file should resemble the following sample code:

```

apply plugin: 'android'

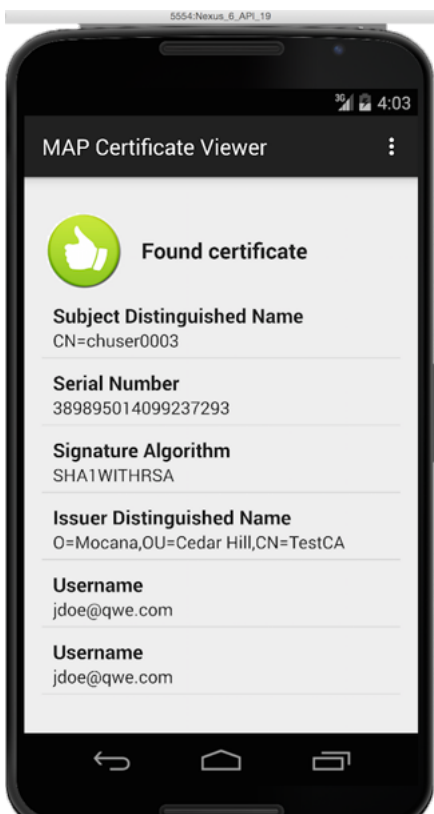
android {
    compileSdkVersion 21
    buildToolsVersion "21.1.1"

    defaultConfig {
        minSdkVersion 8
        targetSdkVersion 19
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
        }
    }
}

dependencies {
    compile 'com.android.support:appcompat-v7:+'
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
...

```

The Android Studio project has a file called **sample.p12** in the **assets** directory. Sample.p12 is a *dummy* certificate that is used to show how the **Lighthouse SDK works without requiring the application to be wrapped**. When launched the app will display the certificate subject name as stored in the sample certificate (sample.p12), and **Username** attribute as stored by the Mocana MAP Wrapper. Compile and run the app in the Android emulator and you should see a screen similar to the one below:



The file **MainActivity.java** is the main Android application file, where **integration with the Lighthouse SDK** occurs, specifically in **MAPCertificateProvider** imported at the top of the **MainActivity.java** file (line 19).

```
import com.mocana.map.android.sdk.MAPCertificateProvider;
```

Note that the **sample.p12** file is then imported (lines 35 - 38), so all calls to the **MAPCertificateProvider** methods will reference this embedded certificate.

```
if (DEBUG_SDK) {  
    MAPCertificateProvider.initCertificateForDebug(getApplicationContext(), "sample.p12", "secret")  
    MAPCertificateProvider.initUserForDebug("jdoe@qwe.com");  
}
```

The following line simply returns an **X509 Certificate object**, so all standard Android SDK methods can also be used to access the attributes of the certificate (line 49).

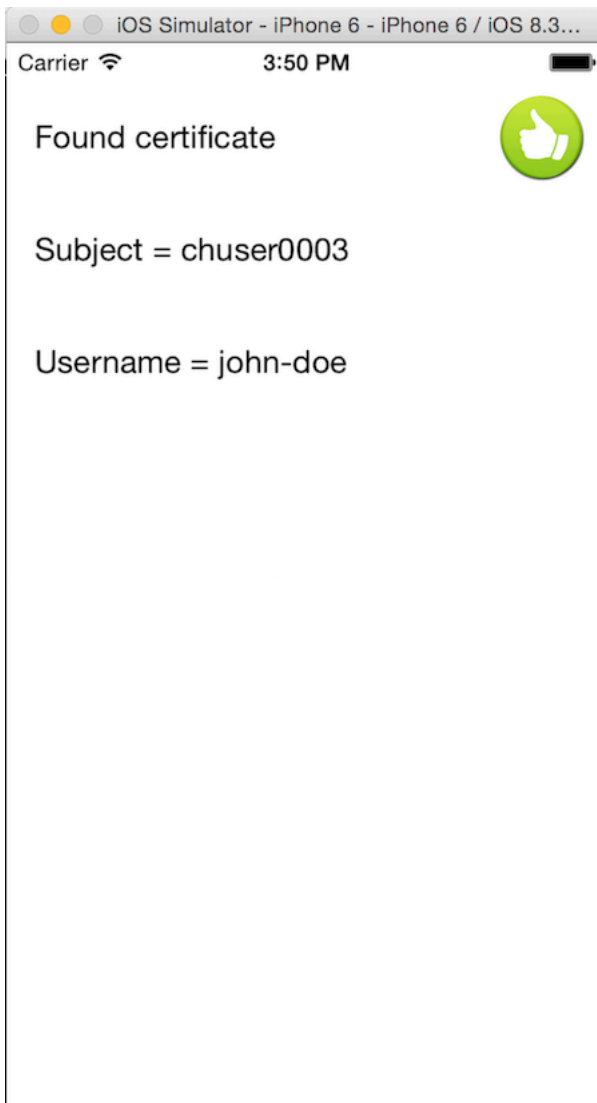
```
X509Certificate certificate = MAPCertificateProvider.getUserIdentityCertificate();
```

The entire set of methods provided by **MAPCertificateProvider** are listed in the API Reference later on in this document. Feel free to skip ahead to that section for more detailed information.

iOS Sample App: *X509 Embedded*

Download and test the **unwrapped** iOS Sample Application on the iOS Device

The iOS sample project, **X509 Embedded**, is implemented as a single view app. The Xcode project has a file called **sample.p12**. This is a *dummy* certificate used to show how the **Lighthouse SDK works without requiring the application to be wrapped**. When launched the app will display the *certificate Subject name* as stored in the sample certificate (sample.p12), and the **Username** attribute as stored by the Mocana MAP Wrapper. Compile and run the app in the iOS simulator and you should see a screen similar to the one below:



The following method is where the main integration occurs between the Lighthouse SDK and the Sample Application. Please note that to access any Lighthouse SDK methods you *must* include the `MAPSDK.h` and `MAPSDK.m` files in your project.

```
(void)showCertificate
```

Specifically the certificate's **subject information** is retrieved by calling the following Lighthouse iOS method. Note that the Lighthouse SDK returns a `SecIdentityRef` object. A `SecIdentityRef` object contains a `SecKeyRef` object and an associated `SecCertificateRef` object. These objects can be manipulated using standard iOS methods for both `SecCertificateRef` and `SecKeyRef`.

```
SecIdentityRef identity = MAP_getUserIdentityCertificate();
```

The entire set of methods provided by `MAPSDK.m` are listed in detail later on in this document. Feel free to skip ahead to the **API Reference** for more information.

2. Wrap the Sample App with MAP and connect to ATLAS

In this scenario, you'll compile and run the Sample Application and then wrap it with MAP using the Lighthouse VPN policy. Upon successful completion, compile and run the MAP-wrapped Sample Application and test the connection to Mocana's hosted Atlas, using your Mocana provided username and

password.

- **Android:**
- **iOS:** [code-signing script](#) required)

Note: Following the wrapping process the app must be signed with your iOS developer's credentials. Mocana provides a code-signing script that appropriately handles the format of the MAP wrapped app. You will be prompted to login to the support portal with your MDP credentials to access and download the script.

Start by securing the Sample Application using Mocana **MAP** (Mobile App Protection), the *client* component of the **Atlas** platform. This section explains how to use the MAP Console to wrap your applications and ensure they are enveloped with airtight data security policies.

A. Login to the Mocana MAP console

Connect your Android or iOS mobile device to your host system and verify connectivity. Your **confirmation email** contains a link to your new hosted MAP console, where you'll be able to wrap and test your apps.

1. **Open a web browser** on your system, and click the url (example below) to open your MAP console.

`https://<nameofcompany>.partner.mocana.com>`

2. **Log in to the MAP administrative console**, using the email address that you signed up with and the randomly generated password included in your welcome email.

Email: user@emailaddress.com
Password: Rando0mlyGener8ted

3. Click **Sign In**. You are now ready to upload your app.

Logged in as John Lennon. Log out


MOCANA MAP™


Applications App Federations Policies Users Support Settings

Welcome to the Mocana Mobile App Protection Administrative Console.

Applications

10

 Android apps: 5

 iOS apps: 5

Policies

15

Users

12

[What's new in this version?](#)

Copyright © 2014 Mocana Corporation
Pat. 8,549,656
Version 3.3.0.8461 (more info) | [Documentation](#) | [Quick wrap](#)

1. Choose the **Applications** tab and select your application's platform: **Android** or **iOS**
2. **Upload** your Android or iOS app by clicking the **Upload new App** button.
3. Attach your application file, click **Upload file to App Catalog**.

B. Wrap Sample App with the Lighthouse VPN Policy

To wrap your app choose the **Per Application VPN**. By selecting this option, the **Encrypted Data at Rest** policy will also be automatically enabled.

1. Under **Per-Application VPN**: select the **Lighthouse VPN** profile. In later tests you may wish to select and apply additional policies to uniformly enforce your organization's security requirements.
2. Once the Lighthouse VPN policy is selected, click the **Apply policies** button. A small **M** image (Mocana icon) will appear overlaid on your application's logo to indicate that this is a MAP-wrapped application with one or more policies applied to it.

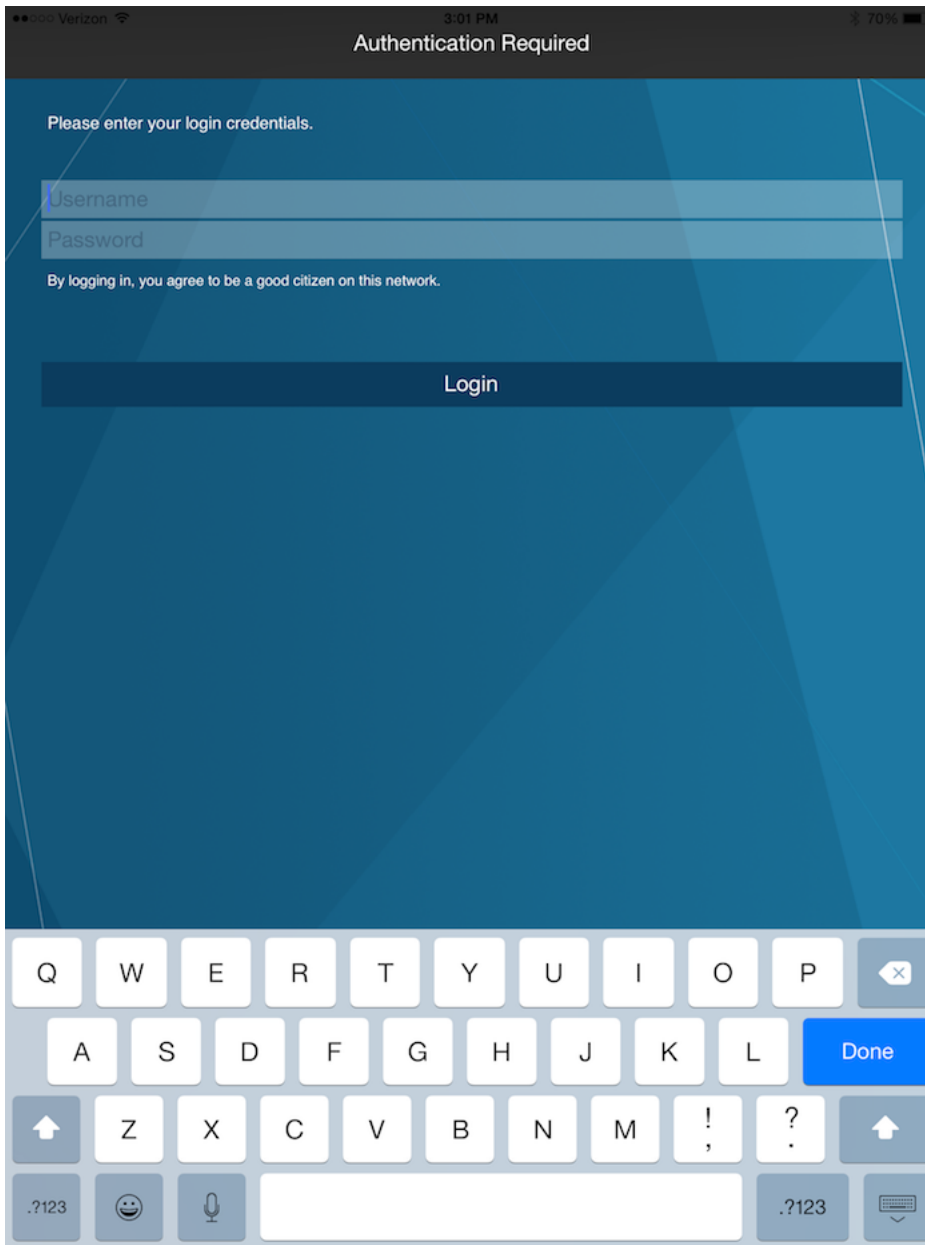


3. **Download** your newly wrapped app.
4. **Sign your Application: iOS only** Following the wrapping process iOS apps must be signed with your iOS developer's credentials. Mocana provides a code-signing script, `map_sign.sh`, that appropriately handles the format of the MAP wrapped app (download [map_sign.sh](#) here). Please use it for signing all MAP wrapped iOS apps. Note that unsigned or improperly signed apps will not install or may not run correctly. For more information, visit the Mocana support portal: [https://success.mocana.com/hc/en-us/articles/201220436-If-I-am-using-an-MDP-portal-how-do-I-sign-an-iOS-app-after-it-has-been-wrapped- "map_sign.sh"](https://success.mocana.com/hc/en-us/articles/201220436-If-I-am-using-an-MDP-portal-how-do-I-sign-an-iOS-app-after-it-has-been-wrapped-?map_sign.sh)).
5. **Install** the app on your mobile device and run it.

You have now successfully wrapped and signed the Sample Application, and installed it on your device. The *first time* you launch the wrapped version of the sample app you will be walked through screens which enroll the app with Atlas. This enrollment includes being issued a user certificate. The enrollment process happens only once. Subsequent usage may prompt for you to enter only your password. The following subsection details how to launch and enroll your app.

C. Enroll with Atlas

Tap to launch and run your app. Your app connects with Atlas automatically and you will see a Mocana Atlas login screen.



Login to Mocana-hosted Atlas with the following information:

- **Username** This will be the prefix of your Mocana Developer Program signup email, **for example:** `ewarren@whitehouse.gov`, the username is **ewarren**
- **Password** Initial password will always be: `abcd1234!`

ATLAS username: ewarren
ATLAS password: abcd1234!

Upon successful authentication, a login screen appears asking for an **Enrollment PIN**. The Enrollment PIN was auto-generated upon your initial Atlas sign-in and has been emailed to you. Obtain the PIN from this email.



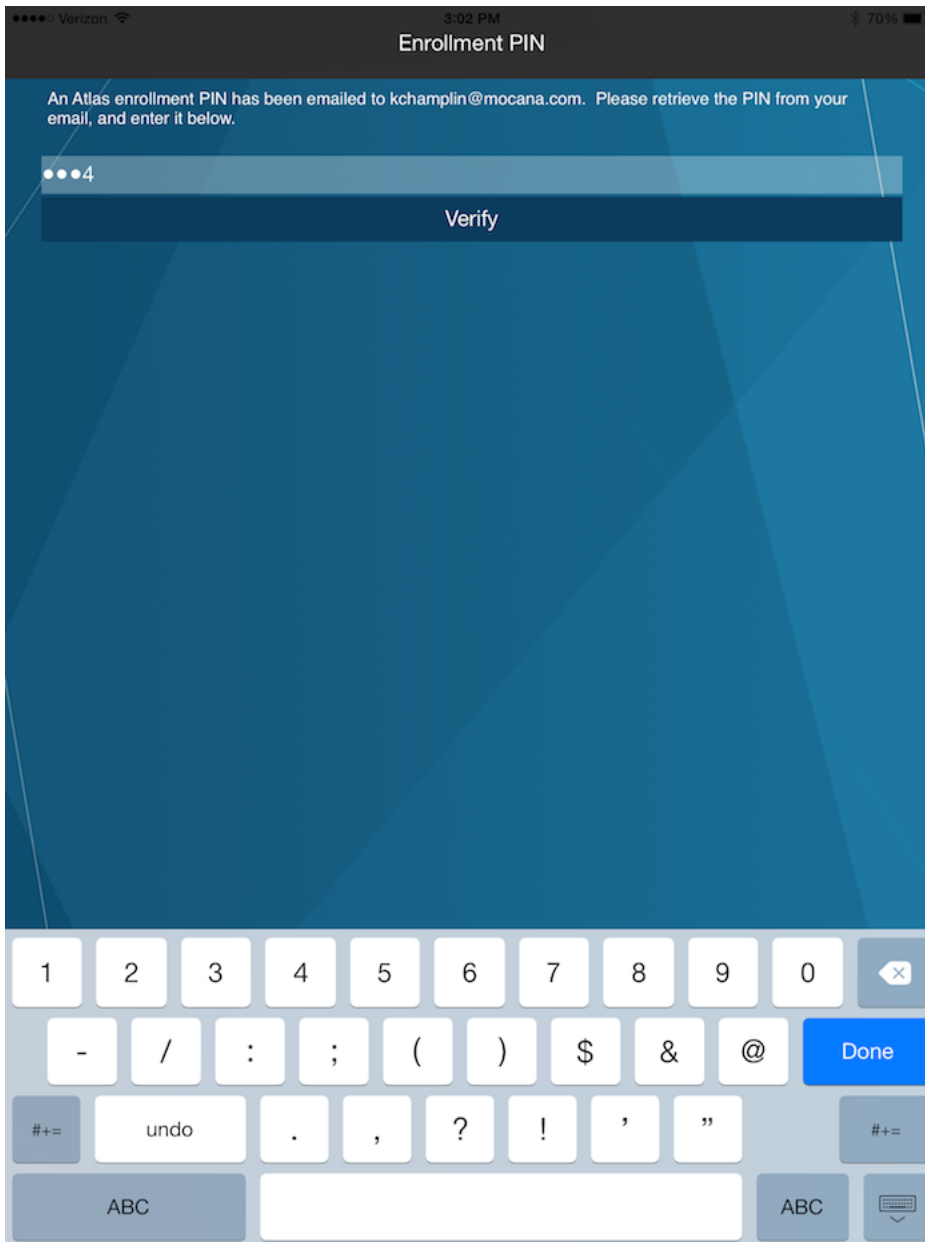
Enrollment Invitation

You have initiated an enrollment of your mobile device with the Mocana Atlas™ Application. Please enter the following Atlas Enrollment PIN into the "Mocana Intranet" application:

6384

Please email your support contact if you have any questions.

Return to the Atlas login screen, enter your four-digit PIN and tap **Verify**



Upon successful Atlas enrollment, you will see the certificate that was issued to you through Atlas during the Atlas enrollment process.

3. ATLAS connected Lighthouse Sample Client to Test Server

There is an additional example application, **MocanaSSO**, located on your MAP server. Apply the Lighthouse VPN Policy to it, download the wrapped version of **MocanaSSO** and code sign with your iOS signing certificate. Once **MocanaSSO** is installed please enroll with Atlas using the same credentials you used for the earlier Android or iOS Sample Application.

You should then see a screen similar to the following:

Welcome to Mocana Lighthouse Sample App

You're Authenticated As:

kchamplin

To review the source and understand how this application was integrated with the Lighthouse SDK, please see the following github repositories: [LightHouseSampleClient](#) which in turn connects to [MocanaLighthouseSampleServer](#).

The next section contains both **API Reference** and **Authentication** context for the Lighthouse SDK.

API Reference

In addition to callbacks defined specifically for X.509 certificate management, these function calls defined for Mocana Atlas Lighthouse SDK will expose the following properties and methods:

Android Lighthouse SDK methods

- `hasUsername`
- `getUsername`
- `hasCertificate`
- `getKeyStoreForUserCertificate`
- `getUserIdentifyCertificate`

Android Debug

- `initCertificateForDebug`
- `initUserForDebug`

iOS Lighthouse SDK methods

- `MAP_hasUsername`
- `MAP_getUsername`
- `MAP_hasCertificate`
- `MAP_hasUserIdentityCertificate`
- `MAP_getUserIdentityCertificate`

iOS Debug

- `MAP_initCertificateForDebug`
- `MAP_initUserForDebug`

Android (Java)

Name	Type	Description	Return Value
<i>hasUsername</i>	Boolean	If the application been wrapped with a VPN profile and the wrapped application has established a tunnel to this VPN	true
<i>getUsername</i>	String	If <i>hasUsername</i> is true, returns the <i>username</i> typed by the user to connect to the VPN, null if <i>hasUsername</i> is false	username
<i>hasCertificate</i>	Boolean	If the application been wrapped with an ATLAS profile that has been configured to perform certificate enrollment and the wrapped application has established a tunnel to this ATLAS	true
<i>getKeystoreForUserCertificate</i>	String	If <i>hasCertificate</i> is true, returns the java keystore that contains the public key and private key of the X509Certificate created during certificate enrollment with ATLAS, null if <i>hasCertificate</i> is false	username
<i>getUserIdentifyCertificate</i>	X509Certificate	If <i>hasCertificate</i> is true, returns the X509Certificate that contains the public key of the X509Certificate created during certificate enrollment with ATLAS, null if <i>hasCertificate</i> is false	X509Certificate

Android Debug

Name	Type	Description	Return Value
<i>initCertificateForDebug</i>	Boolean	<i>initCertificateForDebug</i> , returns the X509Certificate that contains the public key of the X509Certificate created during certificate enrollment with ATLAS or null if <i>initCertificateForDebug</i> is false	X509Certificate
<i>initUserForDebug</i>	String	If <i>initUserForDebug</i> is true, returns the debug <i>username</i> to connect to the VPN, null if <i>initUserforDebug</i> is false	username

iOS (Objective-C)

Name	Type	Description	Return Value
<i>MAP_hasUsername</i>	Boolean	If the application been wrapped with a VPN profile and the wrapped application has established a tunnel to this VPN	true
<i>MAP_getUsername</i>	String	If <i>MAP_hasUsername</i> is true, returns the <i>username</i> typed by the user to connect to the VPN, null if <i>hasUsername</i> is false	username
<i>MAP_hasCertificate</i>	Boolean	If the application been wrapped with an ATLAS profile that has been configured to perform certificate enrollment and the wrapped application has established a tunnel to this ATLAS	true
<i>MAP_hasUserIdentityCertificate</i>	Boolean	If <i>MAP_hasCertificate</i> is true, returns true else returns false	true
<i>MAP_getUserIdentityCertificate</i>	String	If <i>MAP_hasUserIdentityCertificate</i> is true, returns the <i>SecIdentityRef</i> identity that contains the public key of the certificate created during certificate enrollment with ATLAS or null if <i>MAP_hasUserIdentityCertificate</i> is false	SecIdentityRef

iOS Debug

Name	Type	Description	Return Value
<i>MAP_initCertificateForDebug</i>	Boolean	<i>MAP_initCertificateforDebug</i> , returns the X509Certificate that contains the public key of the certificate created during Certificate enrollment with ATLAS or null if <i>initUserforDebug</i> is false	X509Certificate
<i>MAP_initUserForDebug</i>	String	If <i>MAP_initCertificateforDebug</i> is true, returns the debug <i>secIdentityRef</i> identity to connect to the VPN, null if <i>MAP_initCertificateForDebug</i> is false	secIdentityRef

Authentication

When correctly implemented, Lighthouse SDK is called as a final step in establishing a secure communication channel within your existing enterprise system. To clarify the process of **authentication** and **authorization** via mutual authentication (username/X.509 certificate) we'll examine these four stages of connection:

1. Authentication
2. Certificate Enrollment
3. VPN Tunnel
4. Application Startup

Steps 1, 2, and 3 happen when any wrapped application connects to Atlas. The added functionality of Mocana's Atlas Lighthouse SDK begins is initiated in the final step 4, upon the application startup request.

1. Authentication

- MAP Wrapper -> Atlas : User authentication
- Atlas -> AD : Validate credentials
- Atlas <-- AD : Validated

2. Certificate Enrollment

- MAP Wrapper <-- Atlas : Begin certificate enrollment
- MAP Wrapper -> Wrapper : Generate CSR (Certificate Signing Request)
- MAP Wrapper -> Atlas : Send CSR
- Atlas -> Certificate Authority : Sign request
- Atlas <-- Certificate Authority : Signed
- Wrapper <-- Atlas : Install certificate

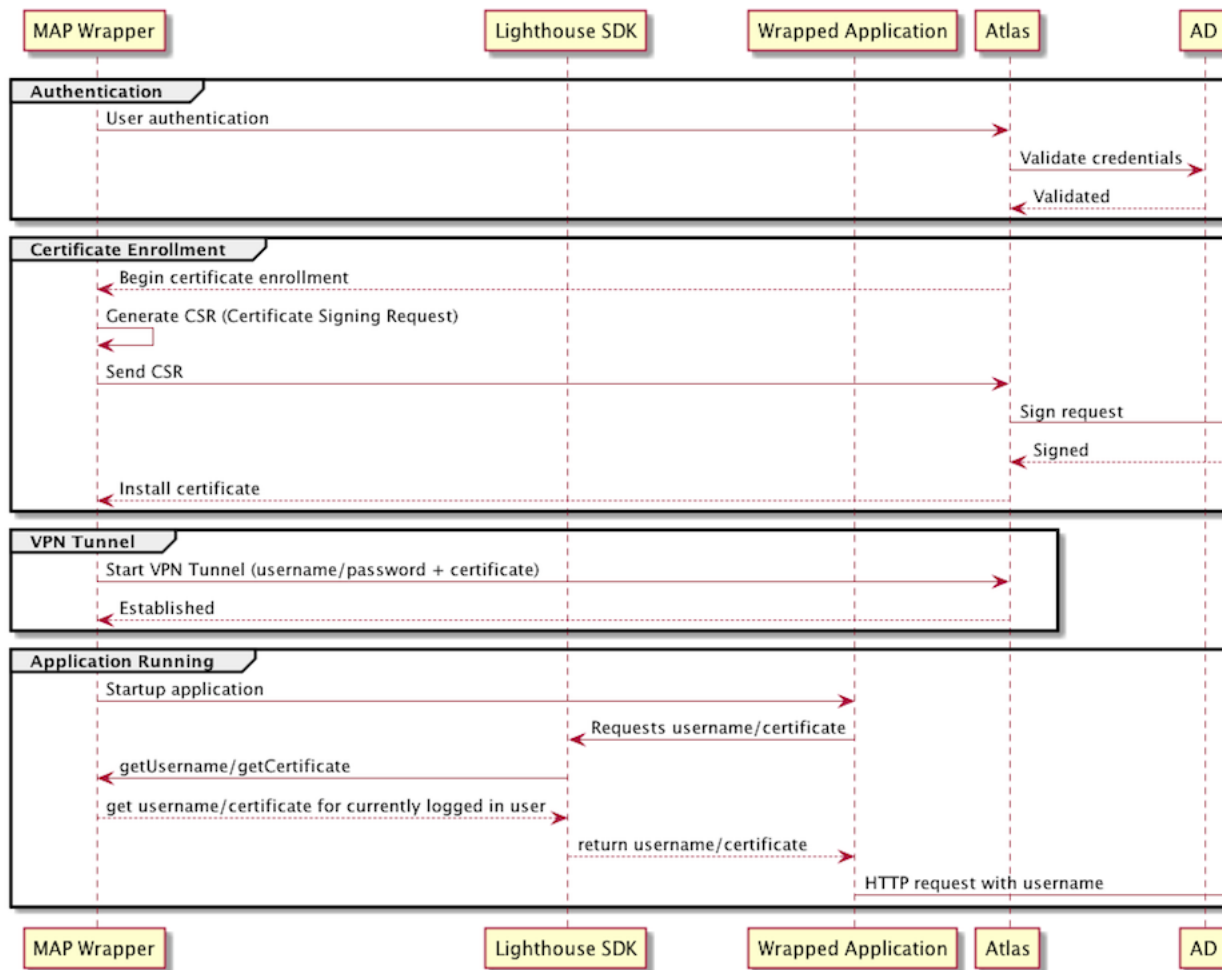
3. VPN Tunnel

- Wrapper -> Atlas : Start VPN Tunnel (username/password + certificate)
- Wrapper <-- Atlas : Established

4. Application Startup --> Lighthouse SDK implementation

- Wrapper -> Application: Startup application
- **Application -> Lighthouse SDK : *Requests username/certificate***
- **Lighthouse SDK -> Wrapper : *getUsername/getCertificate***
- **Wrapper --> Lighthouse SDK : *get username/certificate* for currently logged in user**
- **Lighthouse SDK --> Application : *return username/certificate***
- Application -> Your organization's stack: *HTTP request with username.*

Authentication Flow with ATLAS platform and Lighthouse SDK



4. Connect and integrate ATLAS with your organization's back-end production environment

There are many common enterprise systems and services that support X.509 certificate-based client authentication, including most SAP systems, Domino, Apache, IIS and more. Applications your organization has built to connect to those systems can use Atlas and the Lighthouse SDK to provide prompt-less user authentication and a highly secure, yet streamlined user experience.

For assistance with the Atlas platform and the Lighthouse SDK please use [Mocana's success portal](#).

Resources

Below you'll find resources and tutorials that teach how to integrate **mutual authentication over SSL** and/or **Certificate Enrollment** for different development environments. In addition there are code examples in the [Sample Applications](#) section which demonstrate the use and context of the Mocana Atlas Lighthouse SDK APIs.

Android and iOS

[Mutual authentication over SSL for Android and iOS](#)

Android

[Android - TLS/SSL Mutual Authentication](#)

[Creating self-signed certificates for use on Android](#)

iOS

[iOS Certificate Enrollment](#)

[Use HTTPS certificate handling to protect your iOS app](#)

Need help?

We're always happy to help out with code or any other questions you might have. Contact our [success team](#). This web-based resource center includes access to software updates, technical support news, FAQs, and documentation. You can also submit a request for technical support.

FAQs

Question: Can the Lighthouse SDK be used to obtain user information other than what is contained in the certificate from the Atlas platform?

Answer: Today's Lighthouse SDK does not currently support fetching other user credentials (such as AD username or password) from the Atlas platform or MAP Wrapper.

Question:* Do I need any special build settings or libraries?

Answer: No.

Question: Where and how do I insert the code?

Answer: The API Reference and Sample Applications provide methods and context for where and how to implement the Lighthouse SDK.

Certificate Errors

Certificate errors occur when there's a problem with a certificate or a web server's use of the certificate. Here are some common scenarios:

No Certificate is returned by the Lighthouse SDK If `nil` is returned by the API calls it may indicate your app is not wrapped, or that the sample .p12 certificate was not properly included in your project. If you're working with the Mocana Sample Applications, make certain that the **sample.p12** file is present and in the root directory of your project.

Certificate is missing or red, thumbs down in sample app If you're working with the Mocana Sample Applications, and you get a red, thumbs down, it means the certificate is missing from project or wasn't able to be read from the filesystem. Make certain that the **sample.p12** file is present and in the root directory of your project. If you're supplying your own certificate, make sure the certificate is present and has the correct file path.