# Android Application Attestation Library

## Use Cases

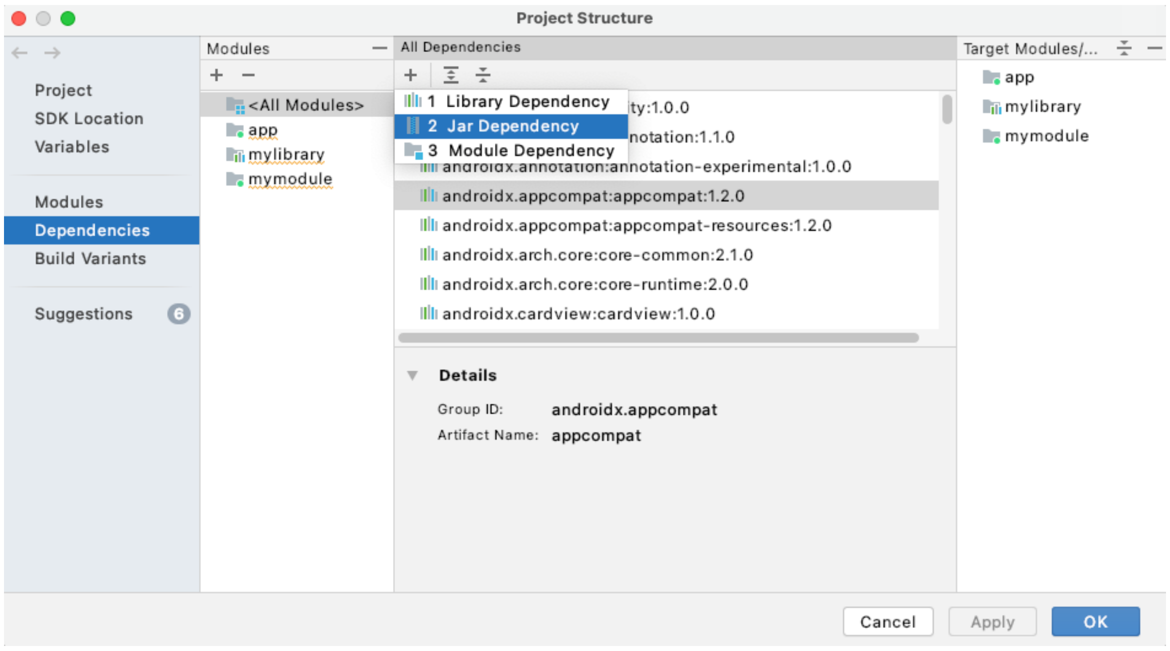
The library helps implement Android trustzone API without the developer having to worry about the intricacies of security protocols.
The library provides both a library package that can be imported and used within your android app. Also a python class that you can use from within your own python server.
This library helps with the implementation of the following

- Attestation hardware backed keypair attestation
  Read: https://developer.android.com/training/articles/security-key-attestation
- Implementing user confirmation
  Read: https://developer.android.com/training/articles/security-android-protected-confirmation

## Installation

### Android

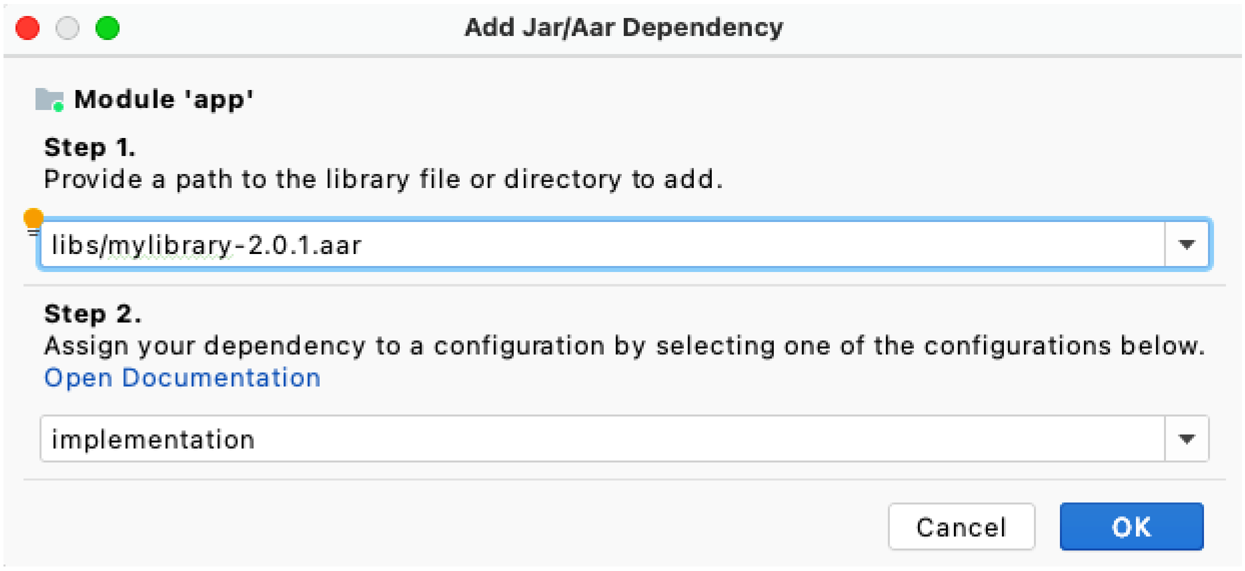
1. Add the library to your project in the following way
   - Navigate to **File > Project Structure > Dependencies**.
   - In the **Declared Dependencies** tab, click + and select **Jar Dependency** in the dropdown.



ty:1.0.0
notation:1.1.0
androidx.annotation:annotation-experimental:1.0.0
androidx.appcompat:appcompat:1.2.0
androidx.appcompat:appcompat-resources:1.2.0
androidx.arch.core:core-common:2.1.0
androidx.arch.core:core-runtime:2.0.0
androidx.cardview:cardview:1.0.0

Group ID: androidx.appcompat
Artifact Name: appcompat

  - In the **Add Jar/Aar Dependency** dialog, first enter the path to your .aar or .jar file, then select the configuration to which the dependency applies. If the library should be available to all configurations, select the "implementation" configuration.



Add Jar/Aar Dependency

Module 'app'

**Step 1.**
Provide a path to the library file or directory to add.

libs/mylibrary-2.0.1.aar

**Step 2.**
Assign your dependency to a configuration by selecting one of the configurations below.
Open Documentation

implementation

  - Check your app's build.gradle file to confirm a declaration similar to the following (depending on the build configuration you've selected):
  - implementation files('my_path/my_lib.aar')
2. Open the app module's `build.gradle` file and add a new line to the `dependencies` block as shown in the following snippet:
  - `dependencies {`
  - `    implementation "androidx.biometric:biometric:1.1.0"`
  - `}`

3. Click **Sync Project with Gradle Files**.

## Python Server

This server runs on python 3.x
Put the **attestationlibrary.py** file into your server directory.
Ensure you have the following python packages by running:

- pip install cryptography
- pip install pyOpenSSL
- pip install pycryptodome

## Usage

### Android

1. Import the library controller
    - Import com.AppAttestLib.AttestationController;
2. To generate the keypair and return the certificate chain associated with it
    - AttestationController myAttestationController = new AttestationController();
    - myAttestationController.registerBioConfirm(Context,attestationChallenge);
    - Where attestationChallenge is the challenge to set in the keypair returned by getInitialRegistrationParameters on the python server
    - myAttestationController.registerBioConfirm returns the CertificateChain associated with the created keypair in a single string that should be passed to parseCertificateChain on the server side.
3. To generate the Authorization Prompts and sign the message
    - AttestationController myAttestationController = new AttestationController();
    - myAttestationController.callBioConfirm(Context,promptDetails);
    - Where promptDetails contain the nonce+prompt returned by getInitialauthorizationParameters in the python server
    - myAttestationController.callBioConfirm returns a single string with both the message and its signature to be used by verifyauthorizationSignature on the server.

### Python Server

1. Import the server
    - At the top of your server code write the following import statement
        - from attestationlibrary import AttestationServer
2. Create server object
    - myAttestationServer = AttestationServer()
    - Make this accessible to all function call (make global for ease)
3. Create attestationChallenge
    - myAttestationServer.getInitialRegistrationParameters(uid)
    - Where uid is userid
    - myAttestationServer.getInitialRegistrationParameters returns attestationChallenge as string
    - Give this as input to registerBioConfirm on client
4. Verify CertificateChain
    - myAttestationServer.parseCertificateChain(uid,certificateChainString)
    - Where uid is userid
    - Where certificateChainString is concatanatedString of certificates generated by registerBioConfirm on client
    - myAttestationServer.parseCertificateChain returns true on successful verification of certificate chain
5. Get key properties from attestation certificate
    - myAttestationServer.getKeyProperties(userid)

- Userid is the unique id with which the AttestationServer keeps track of users
- myAttestationServer.getKeyProperties returns list of tuples of keyProperties i.e (trusted confirmation required,true)
- WARNING: call this after parseCertificateChain

6. Prepare for authorizationPrompt
   - myAttestationServer.getInitialAuthorizationParameters(userid, messageToAuthorize)
   - messageToAuthorize is the message you want authorized
   - Userid is the id of the user that is going to authorize the message
   - myAttestationServer.getInitialAuthorizationParameters returns the parameters i.e nonce+prompt needed by callBioConfirm on the client
7. Verify Signature
   - myAttestationServer.verifyAuthorizationSignatures(userid,messageWithSignatures)
   - Where messageWithSignatures is the concatenated string with both message and its signature
   - myAttestationServer.verifyAuthorizationSignatures returns true if signature valid