INKA ENTWORKS

appsealing

# iOS SDK 1.7.0.0

# for Cordova/Ionic

# User Guide

The software referenced herein, this User Guide, and any associated documentation is provided to you pursuant to the agreement between your company, governmental body or other entity (“you”) and INKA Entworks Corporation (“AppSealing”) under which you have received a copy of AppSealing Licensed Technology and this User Guide (such agreement, the “Agreement”). Defined terms not defined herein shall have the meanings ascribed to them in the Agreement. In the event of conflict between the terms of this User Guide and the terms of the Agreement, the terms of the Agreement shall prevail. Without limiting the generality of the remainder of this paragraph, (a) this User Guide is provided to you for informational purposes only, (b) your right to access, view, use, and copy this User Guide is limited to the rights and subject to the applicable requirements and limitations set forth in the Agreement, and (c) all of the content of this User Guide constitutes “Confidential Information” of AppSealing (or the equivalent term used in the Agreement) and is subject to all of the limitations and requirements pertinent to the use, disclosure and safeguarding of such information. Permitting anyone who is not directly involved in the authorized use of AppSealing Licensed Technology by your company or other entity to gain any access to this User Guide shall violate the Agreement and subject your company or other entity to liability therefore.

## Copyright Information



## Disclaimer

The remainder of this User Guide notwithstanding, this User Guide is provided “as is”, without any warranty whatsoever (including that it is error-free or complete). This User Guide contains no express or implied warranties, covenants or grants of rights or licenses, and does not modify or supplement any express warranty, covenant or grant of rights or licenses that is set forth in the Agreement. This User Guide is current as of the date set forth in the header that appears above on this page, but may be modified at any time without prior notice. Future revisions and updates of this User Guide shall be distributed as part of AppSealing SDK new releases. INKA Entworks shall under no circumstances bear any responsibility for your failure to operate AppSealing Licensed Technology in compliance with the then-current version of this User Guide. Your remedies with respect to your use of this User Guide, and INKA Entworks' liability for your use of this User Guide (including for any errors or inaccuracies that appear in this User Guide) are limited to those remedies expressly authorized by the Agreement (if any).

## Contact Information

Address: [06109] 1F 608, Nonhyeon-ro, Gangnam-gu, Seoul, South Korea
Technical support: https://helpcenter.appsealing.com
Website: www.appsealing.com

# Table of Contents



Part 7. Troubleshooting



# Prerequisite

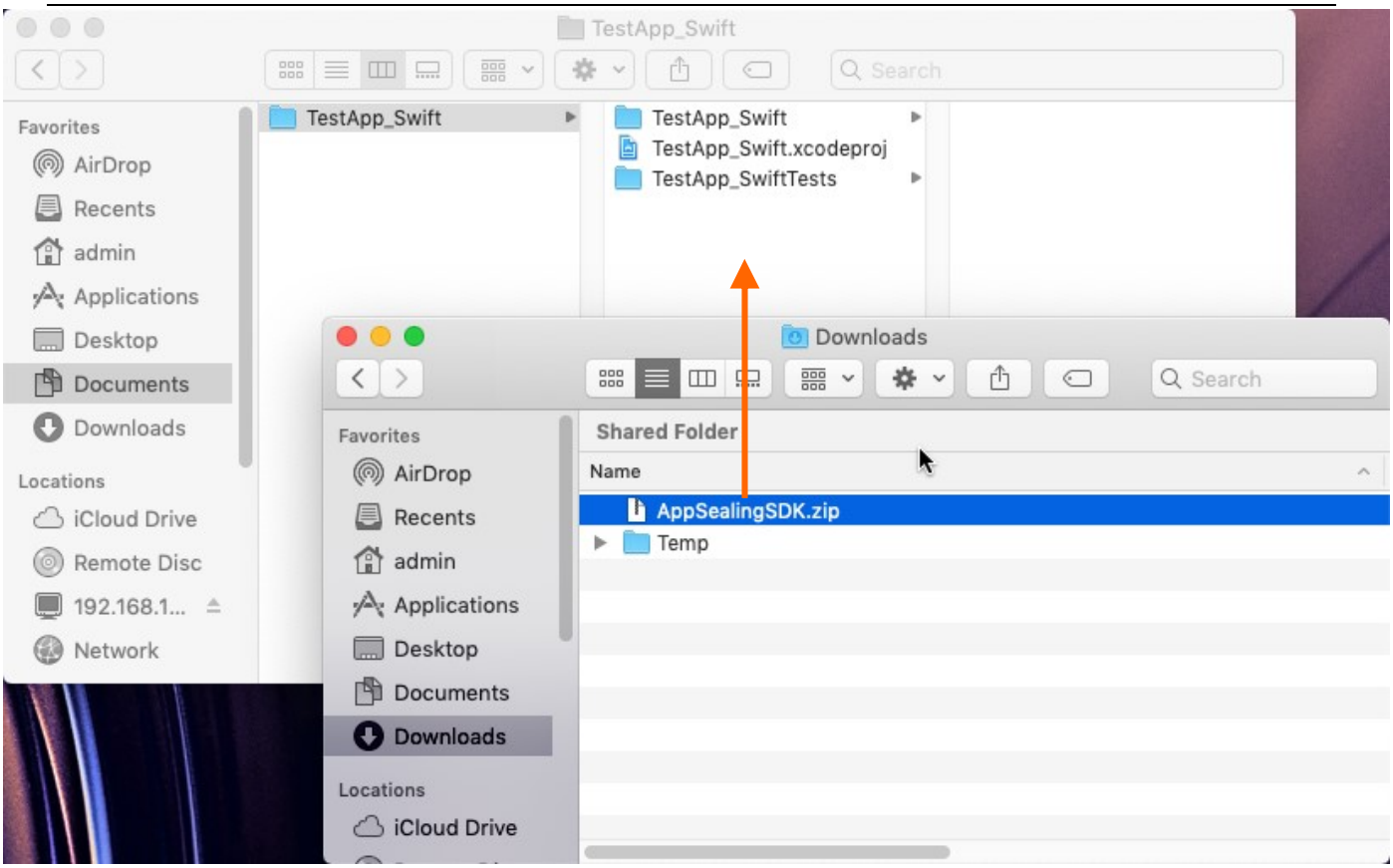
- Xcode 12.5.1 or newer (macOS 11 or newer)
- iOS Device with iOS 9.0 or newer

# Part 1. Put SDK in right place

After you downloaded AppSealingSDK zip file, you should un-compress it into your Xcode project folder. This document will use sample Xcode swift project named 'TestApp_Swift' for illustration, and its location is "~/Documents/TestApp_Swift".

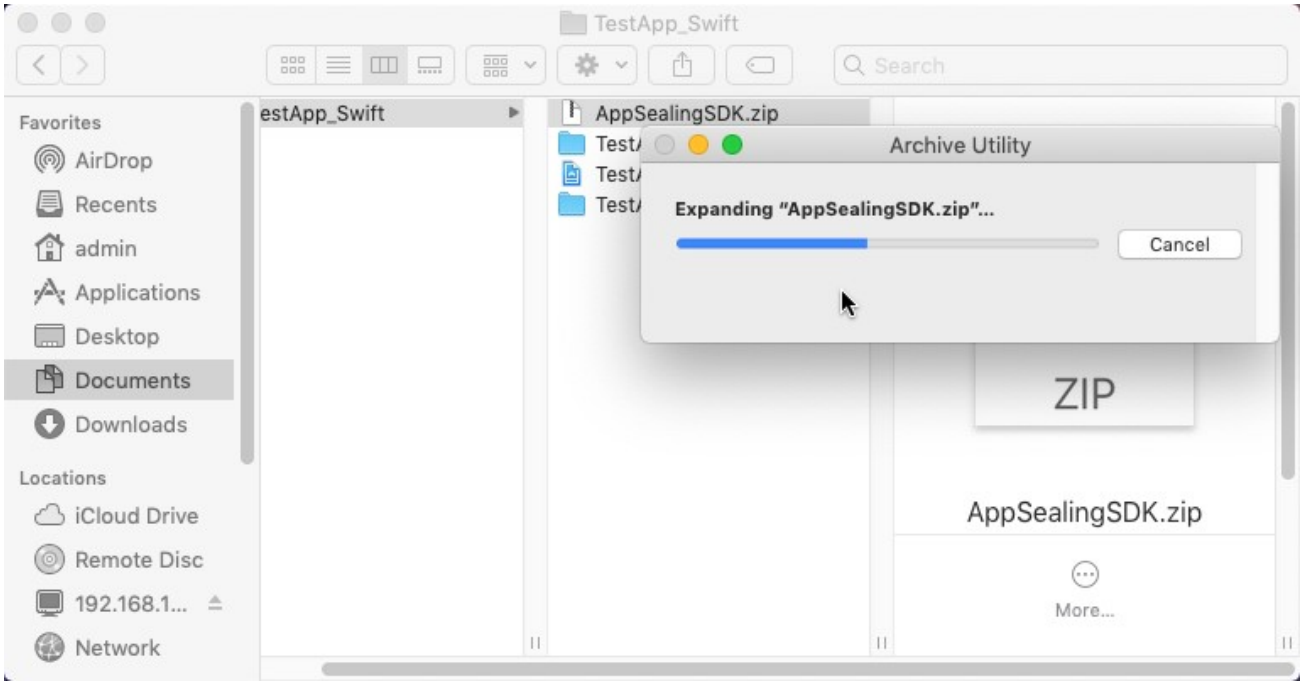
1-1 Move downloaded SDK file into your project folder



1-2 **Un-compress** moved SDK file by double clicking it




After uncompressing zip file, check whether all files are generated like following structure.

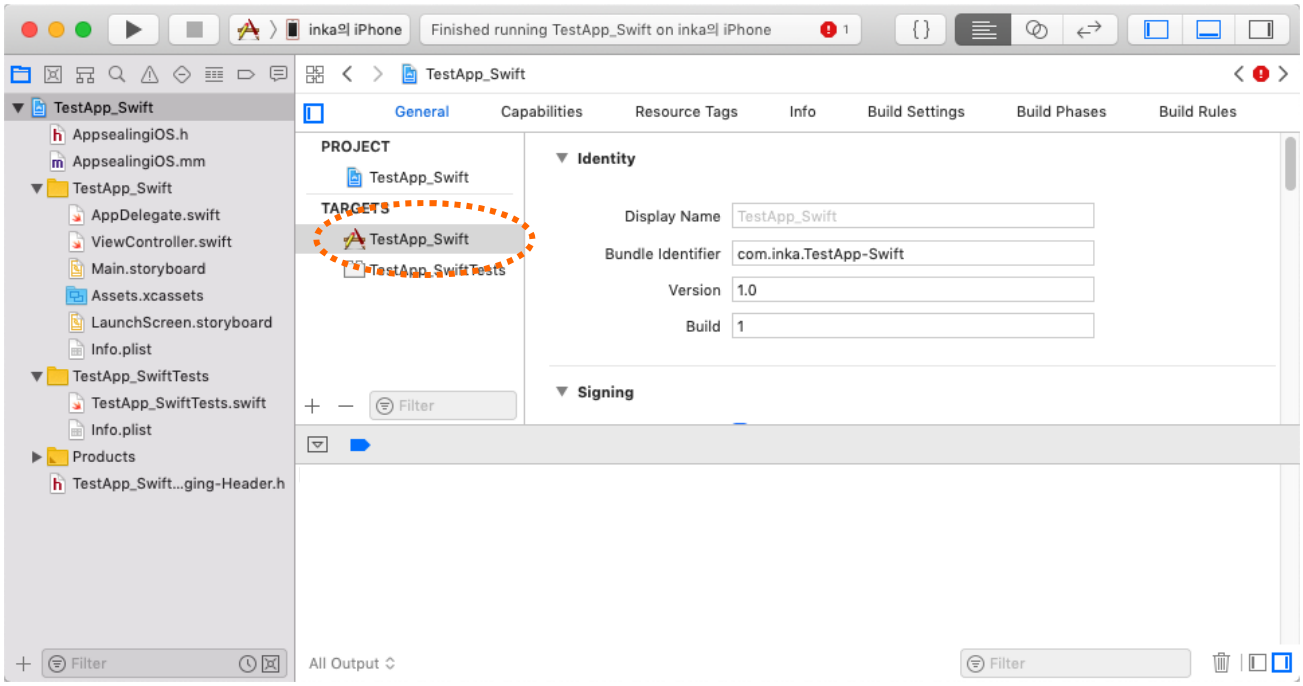
| Name | Size | Kind |
|---|---|---|
| Documents | -- | Folder |
| iOS_AppSealingSDK_UserGuide_for_XcodeProject_EN.pdf | 5.2 MB | PDF Document |
| iOS_AppSealingSDK_UserGuide_for_XcodeProject_KR.pdf | 5.3 MB | PDF Document |
| Libraries | -- | Folder |
| appsealing.lic | 464 bytes | Document |
| AppsealingiOS.h | 925 bytes | C Head...Source |
| AppsealingiOS.mm | 8 KB | Objective-C++ |
| libStaticAppSec_Debug.a | 203.6 MB | Document |
| libStaticAppSec.a | 177.8 MB | Document |
| Release-Notes.txt | 2 KB | Plain Text |
| Tools | -- | Folder |
| generate_genesis | 342 KB | Document |
| generate_hash | 25 KB | Document |

If any file is missing try re-download SDK or try expand zip file again.

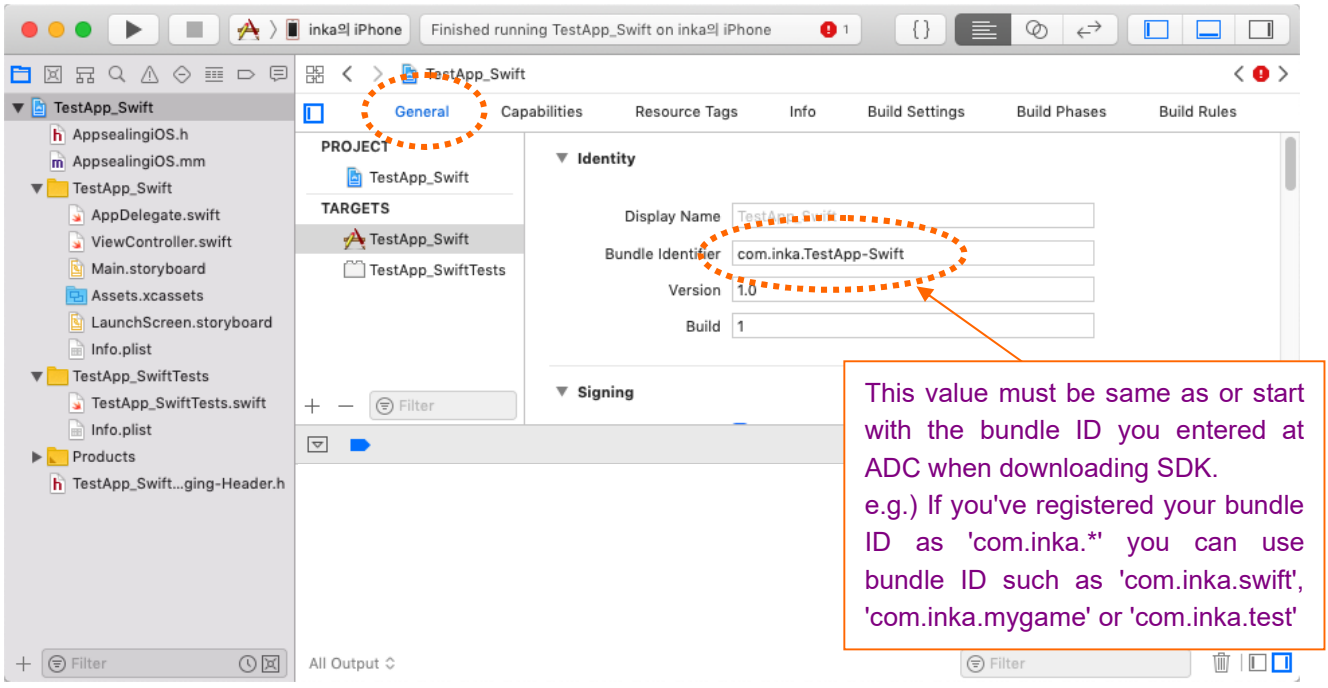
 

## 1-3 **Compare Bundle ID** of your project with the registered bundle ID of SDK

After you've uncompressed SDK zip file, you should check your bundle ID is valid. Open Xcode and select top project at left panel and select your project name (TestApp_Swift) in TARGETS list.



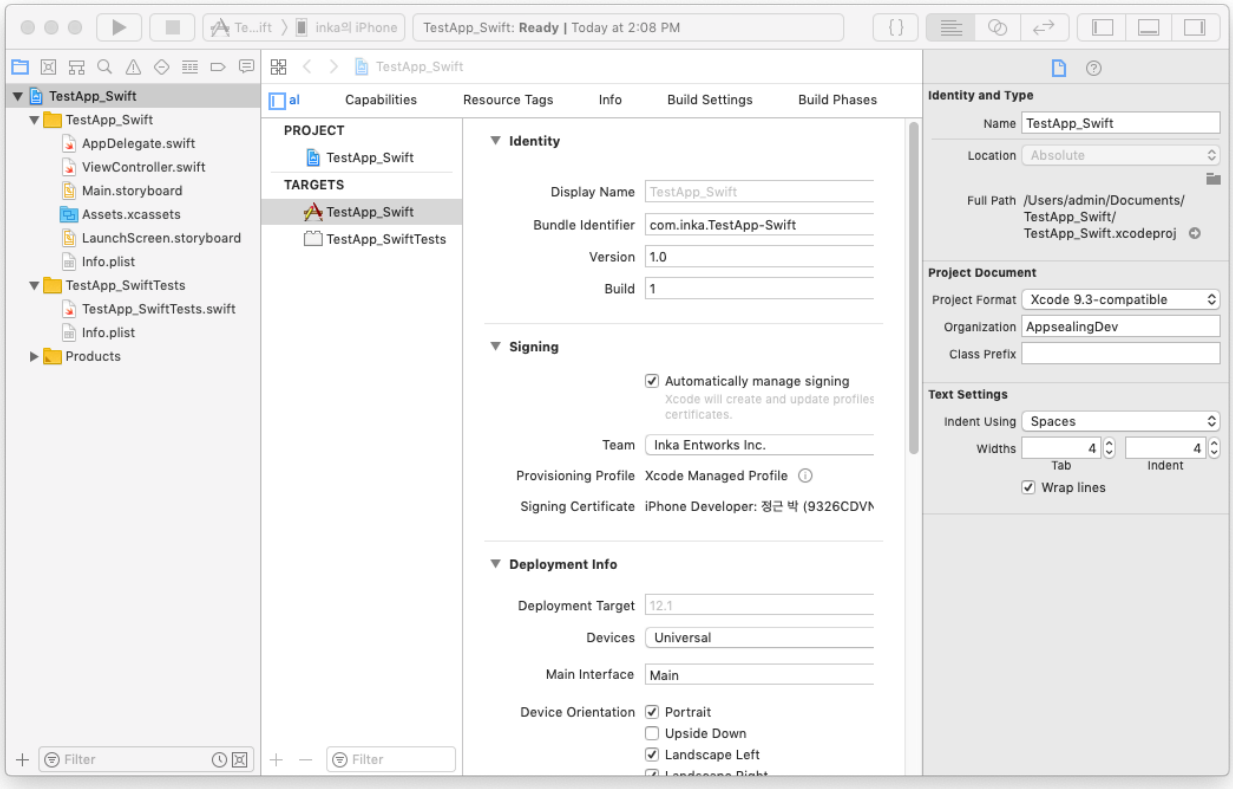

Then select "General" tab and check your bundle Id whether it is same as or starts with the value you entered when downloading SDK at AppSealing Developer Center (ADC).



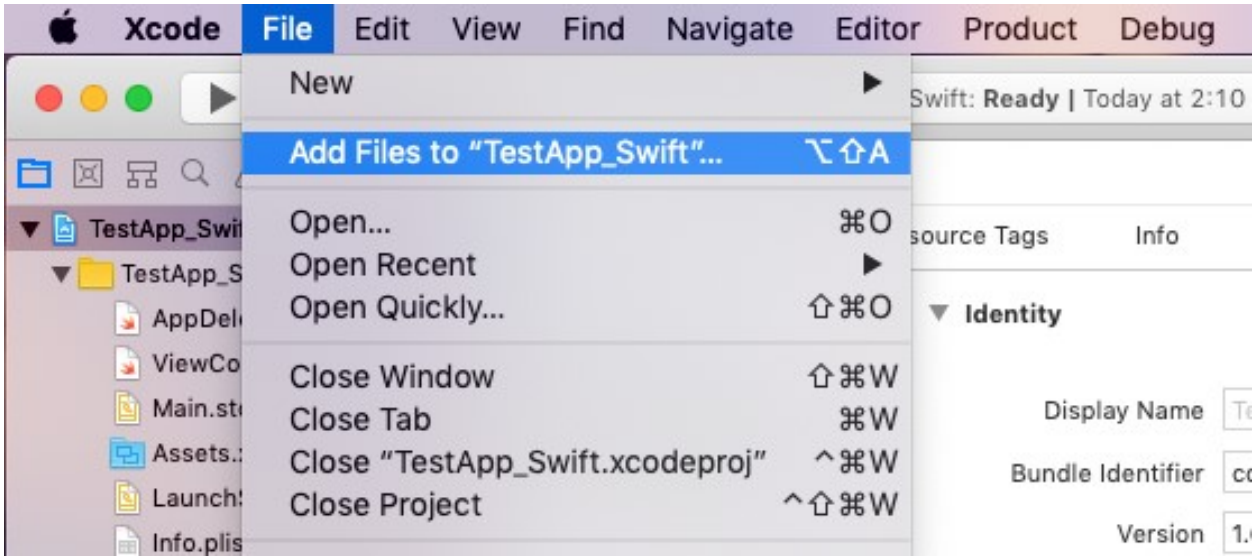

 

# Part 2. Add additional files to your project

## 2-1 Open your **Xcode project**

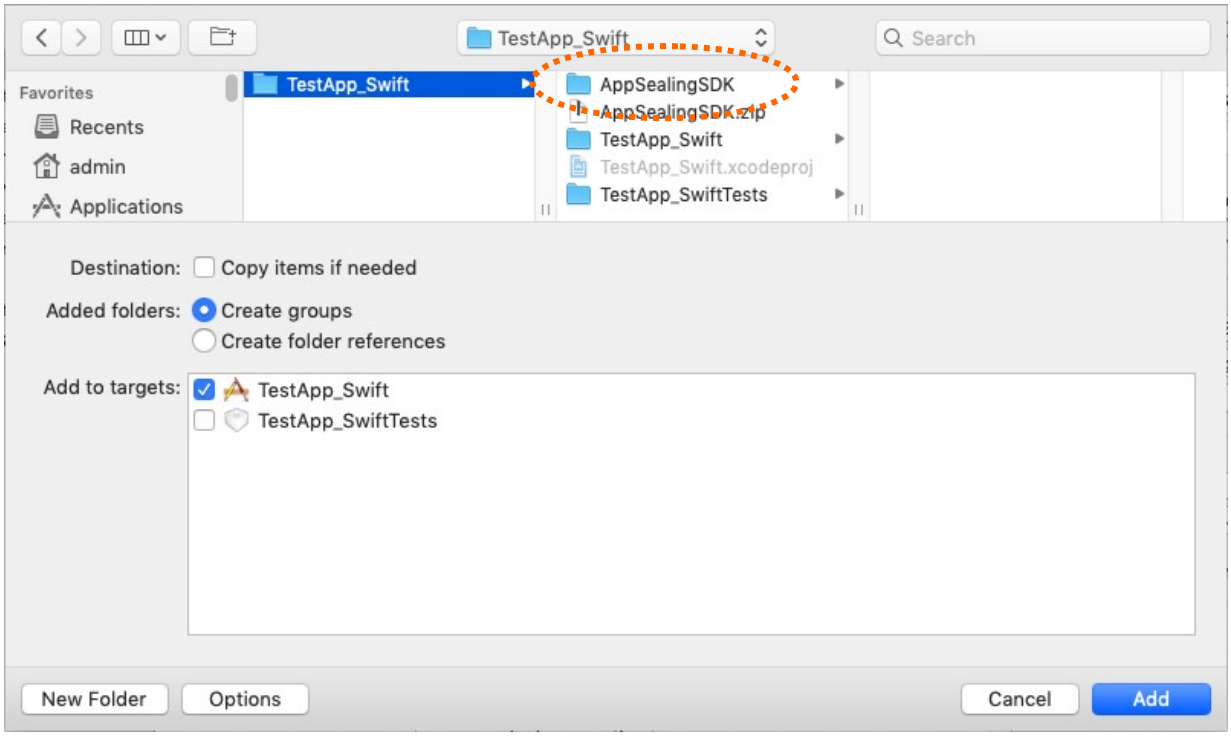


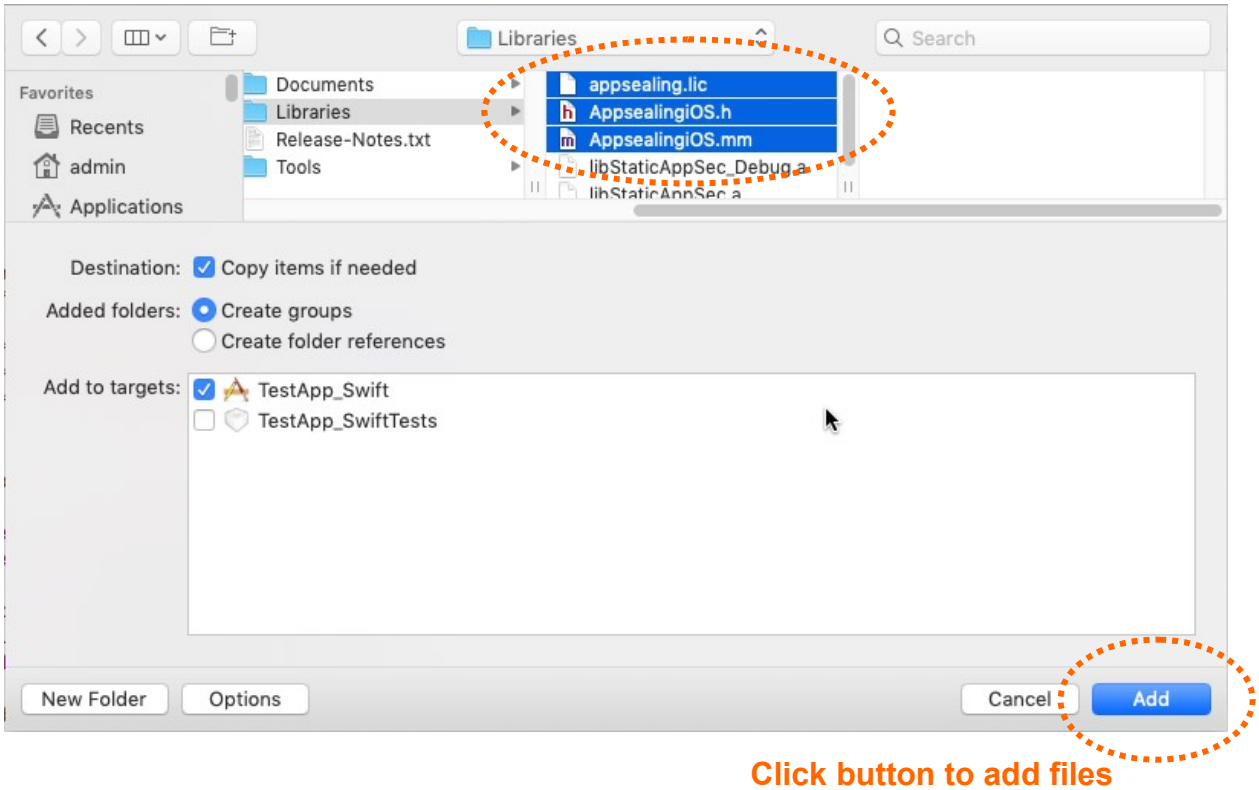

## 2-2 Perform **'File >> Add Files to "TestApp_Swift"...'** menu action



In dialog box, select "**appsealing.lic**", **"AppsealingiOS.h" and "AppsealingiOS.mm"** files in "AppSealingSDK/Libraries/" folder and click "Add" button.



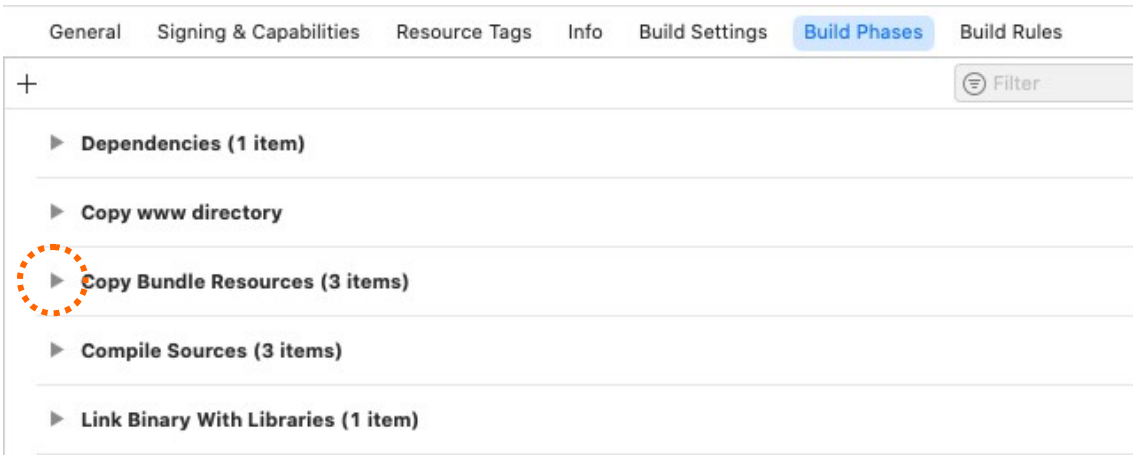

**Click button to add files**

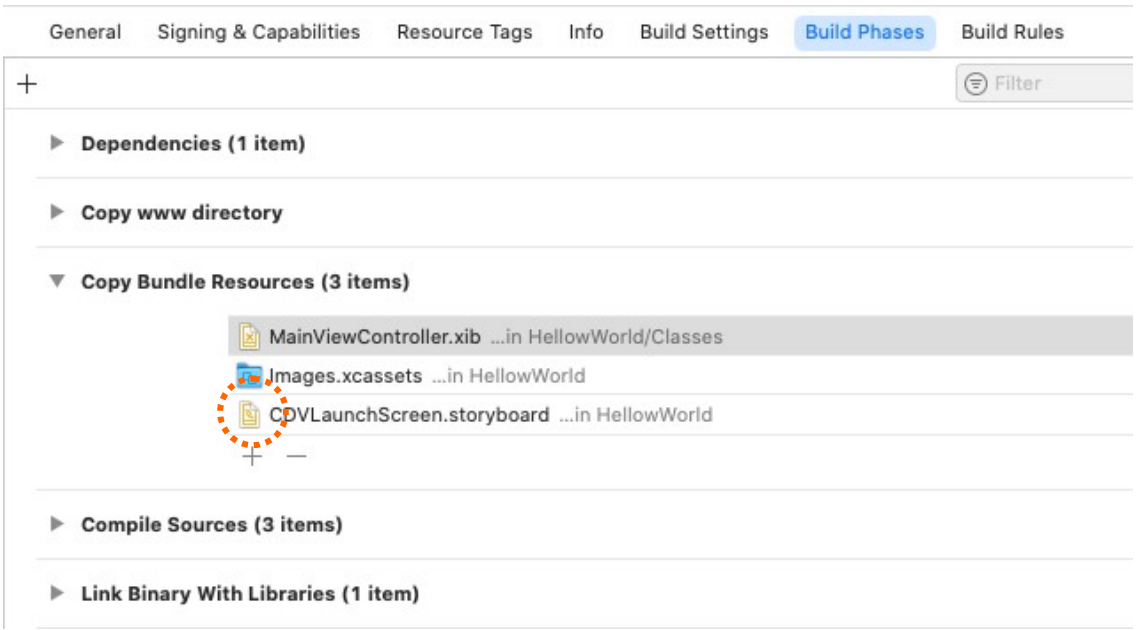
## Only if your project is Cordova based.

**(Do not refer this & next two pages if your project is not Cordova based)**

Select project target then move to "Build Phase" tab. Click the left arrow icon of the "Copy Bundle Resources" item at the bottom of the "Build Phase" tab to expand the area.

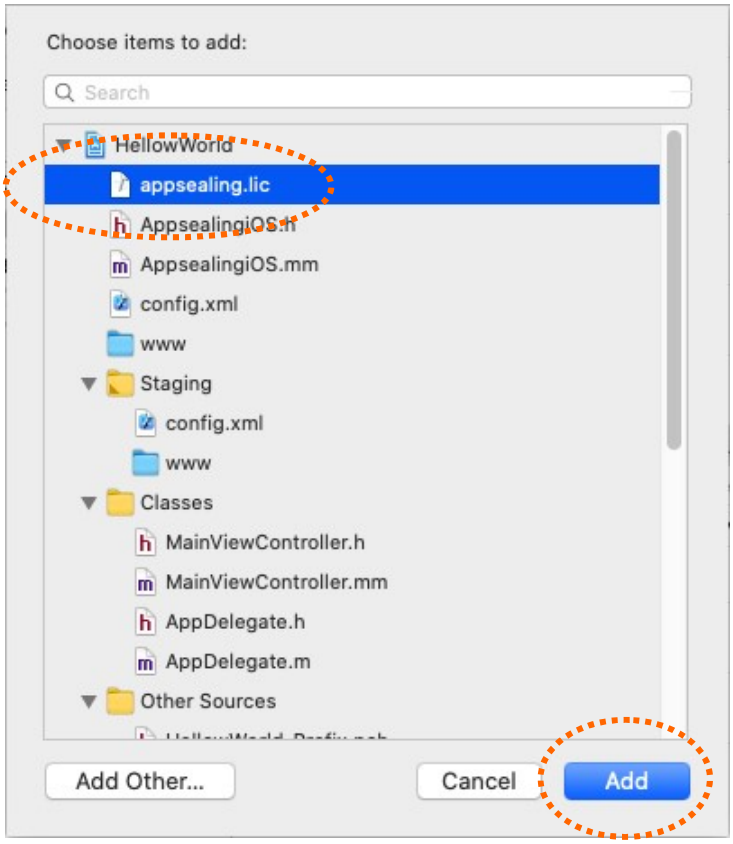



When it is expanded as shown below, click the "+" icon.

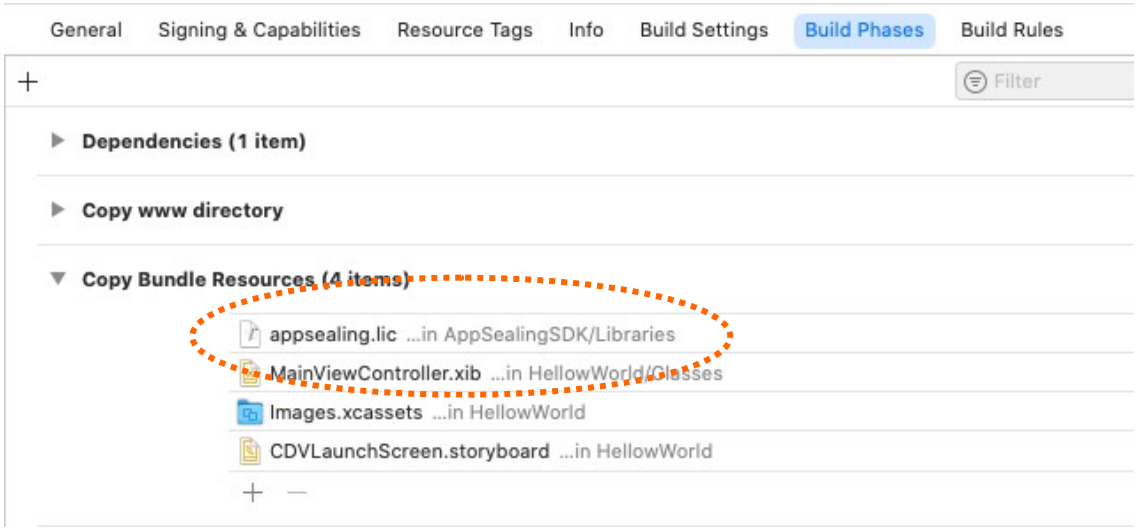



A file selection window will appear and the three files you added earlier are displayed at the top. Select "appsealing.lic" among them and click "Add" button.



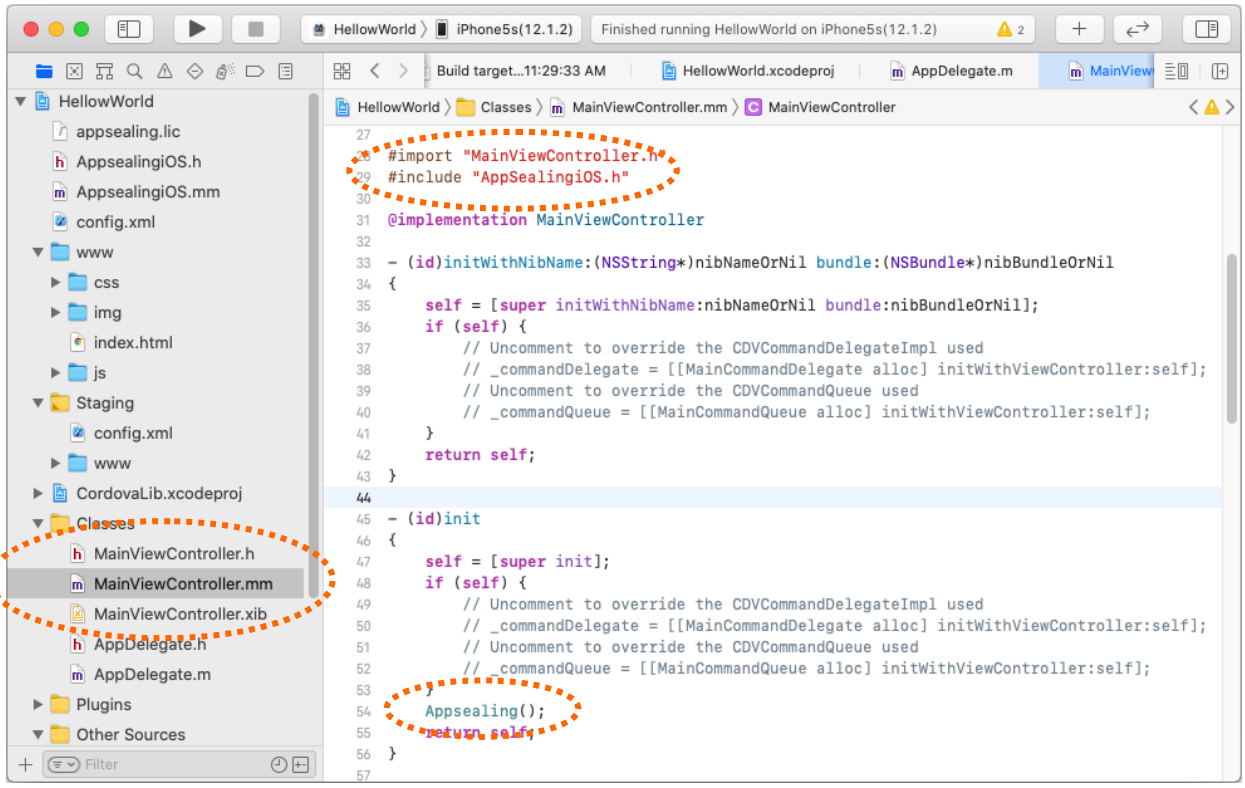

Make sure the "appsealing.lic" file is added as a bundle resource as shown below.



Now expand the "Classes" folder in the project navigation pane on the left and select the "MainViewController.m" file inside it then change the file extension to .mm. After making changes, open the file and add the line #include "AppSealingiOS.h" at the top. Then find the "init" method and put Appsealing(); line before return statement.

Finally, it should be in the form below..




For Cordova projects, If you do not add "Appsealing();" function call statement, the AppSealing static library is not automatically linked, **so this line must be added for the AppSealing function to work properly**.

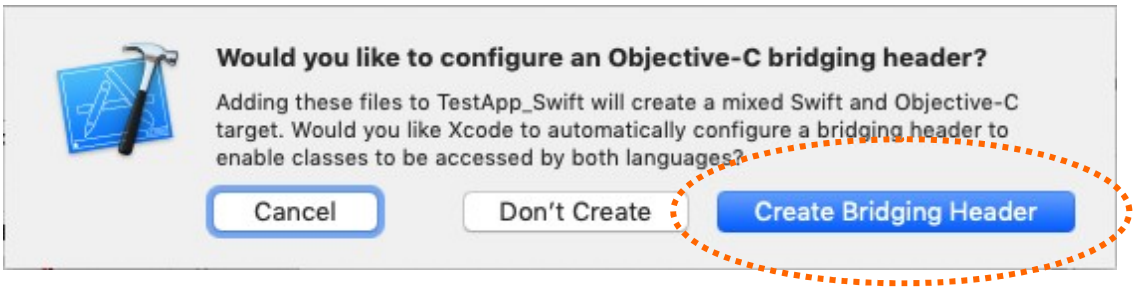
 

2-4 Add **Bridging Header** into Ionic project

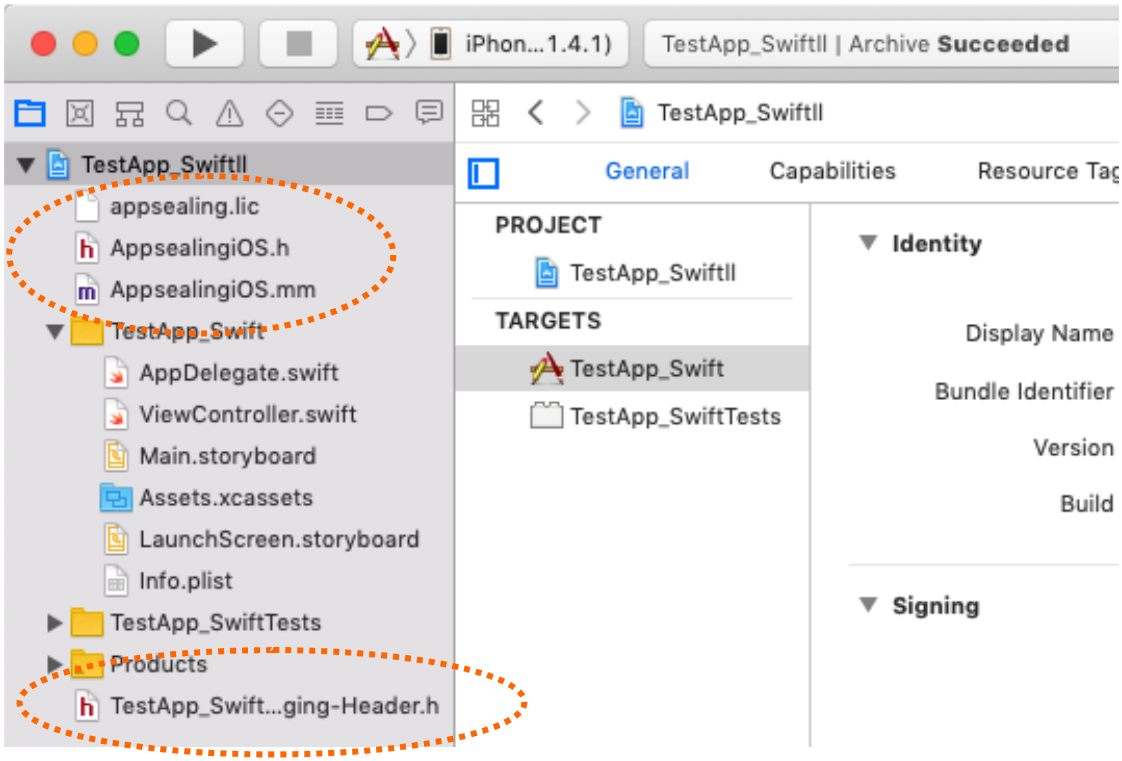
## Only if your project is Ionic based

**(Do not refer this & next pages if your project is Cordova based)**

Since your project is swift-based project, you would encounter a dialog box which asks you create a bridging header file or not only if there's no bridging header in your project yet. You should click "Create Bridging Header".




**Click this button to create Bridging Header**

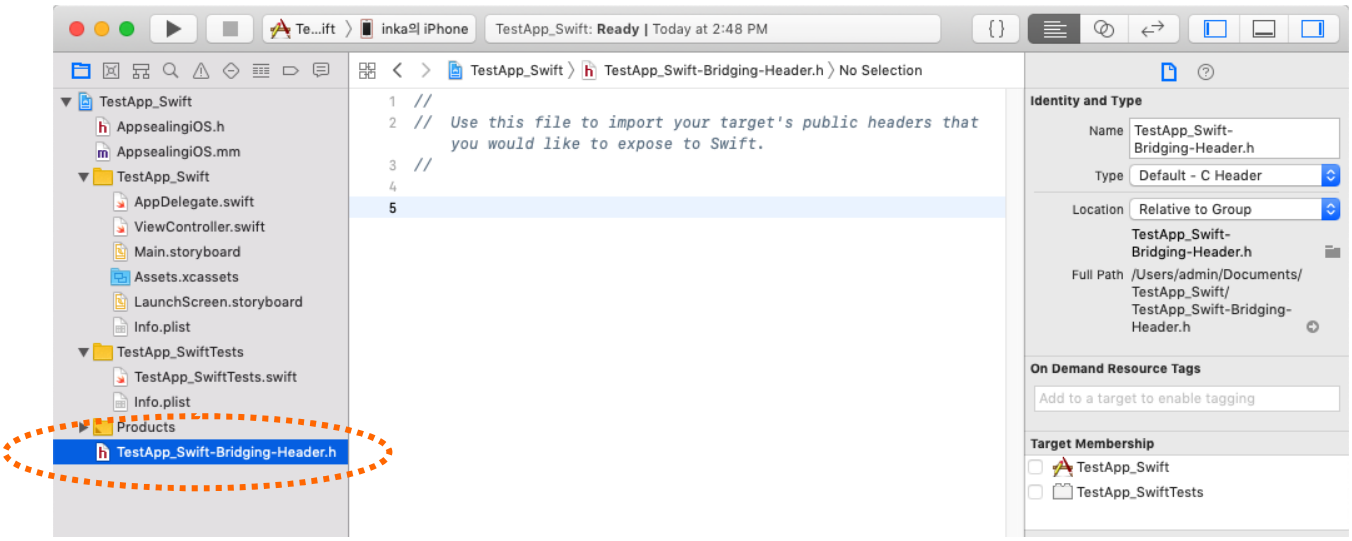



* Supposing your project already has bridging header file, the upper dialog box will not appear and you can use your existing bridging header file.

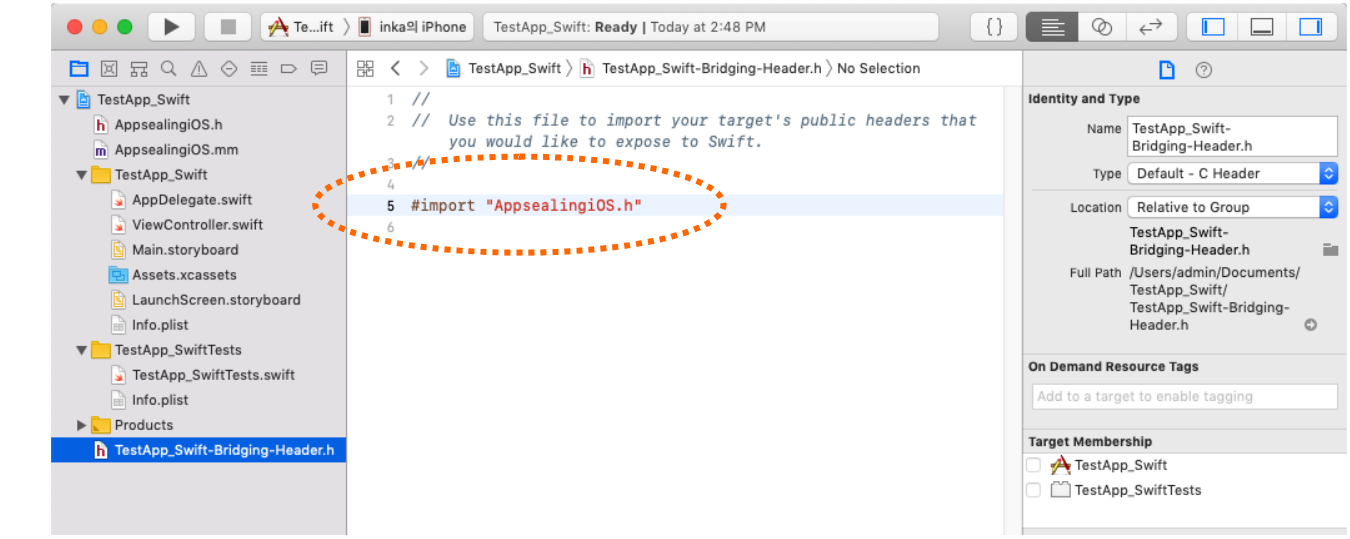
** If your project is Cordova based, the bridging header file is not used.

## 2-4 Append **AppSealing header** to **bridging header** file (Ionic project only)




Select newly created "TestApp_Swift-Bridging-Header.h" file (or existing bridging header file) and append *#import "AppsealingiOS.h"* to end of document like below.



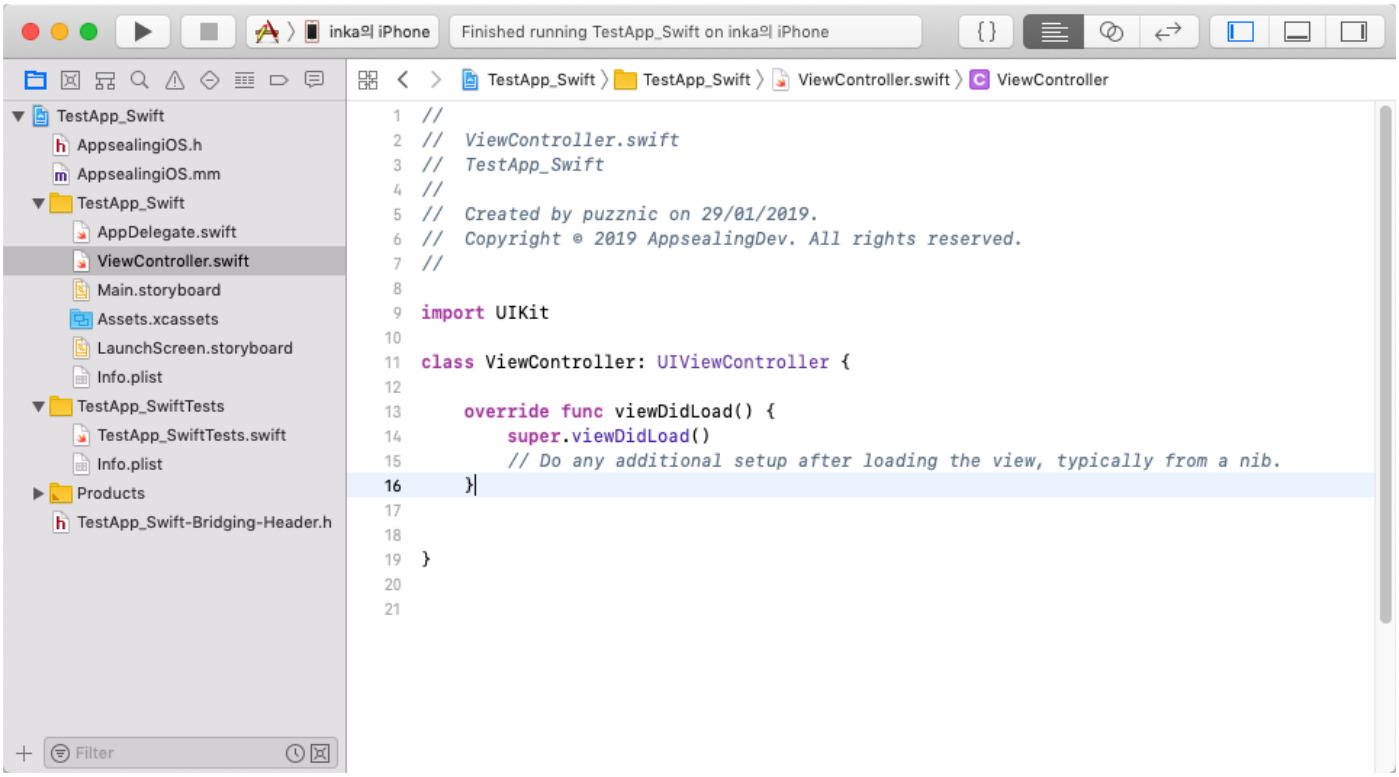
# Part 3. Add simple GUI for iOS security intrusion

The AppSealing library within your App is automatically activated when right after App has launched. If AppSealing library has detected any abnormal environment (such as jailbroken-device, executable has decrypted or debugger has attached) it will close the app after 20 seconds irrespectively of user action, so the app should notify the detection result to user and show some proper message box for user can recognize there's some invalid environment in his/her device.

If you want to show that dialog box in your app, you can easily do that inserting small chunk of code into "ViewController.swift" file. (ViewController.mm in case of Objective-C based project)

## 3-1 Show **UIAlertController window in your app**

First, open your Xcode project and open ViewController.swift" file.

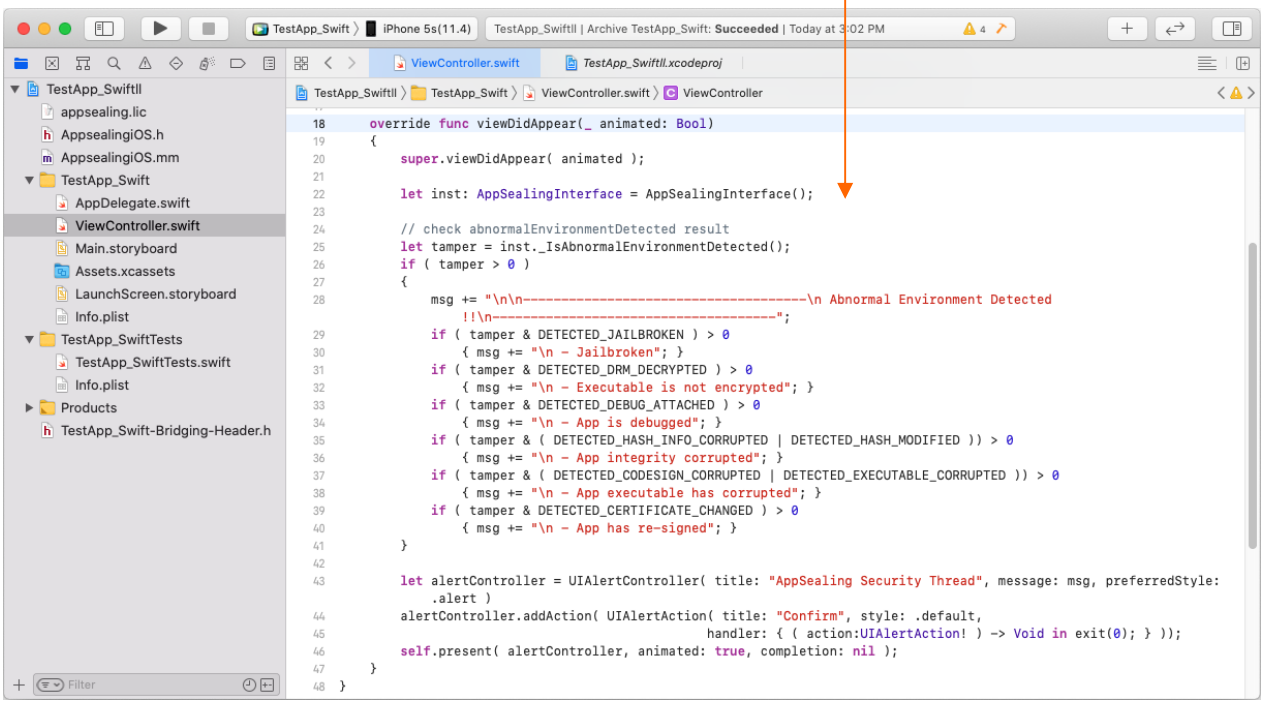



After you opened the swift file put following code into that (If ViewController.swift file has already included '**viewDidAppear**' method just insert the body of following code below 'super.viewDidAppear( animated );' line.)

Simple UI code into 'ViewController.swift' for **Swift** project

```
override func viewDidAppear(_ animated: Bool)
{
    super.viewDidAppear( animated );

    let inst: AppSealingInterface = AppSealingInterface();
    let tamper: Int32 = inst._IsAbnormalEnvironmentDetected();
    if ( tamper > 0 )
    {
        var msg = "Abnormal Environment Detected !!";
        if ( tamper & DETECTED_JAILBROKEN ) > 0
            { msg += "\n - Jailbroken"; }
        if ( tamper & DETECTED_DRM_DECRYPTED ) > 0
            { msg += "\n - Executable is not encrypted"; }
        if ( tamper & DETECTED_DEBUG_ATTACHED ) > 0
            { msg += "\n - App is debugged"; }
        if ( tamper & ( DETECTED_HASH_INFO_CORRUPTED | DETECTED_HASH_MODIFIED )) > 0
            { msg += "\n - App integrity corrupted"; }
        if ( tamper & ( DETECTED_CODESIGN_CORRUPTED | DETECTED_EXECUTABLE_CORRUPTED )) > 0
            { msg += "\n - App executable has corrupted"; }
        if ( tamper & DETECTED_CERTIFICATE_CHANGED ) > 0
            { msg += "\n - App has re-signed"; }
        let alertController = UIAlertController(title: "AppSealing",
                                message: msg, preferredStyle: .alert );
        alertController.addAction(UIAlertAction(title: "Confirm", style: .default,
                                handler: { (action:UIAlertAction!) -> Void in
#if !DEBUG   // Debug mode does not kill app even if security threat has found
                                    exit(0);
#endif
                                } ));
        self.present(alertController, animated: true, completion: nil);
    }
}
```




Sample UI code above is also included in "AppsealingiOS.mm" file so you can copy & paste int that file.

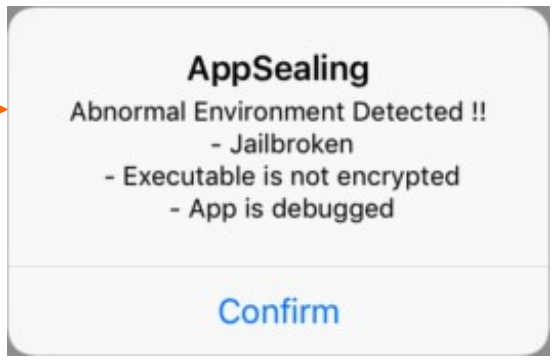
If your project is Objective-C based then you can use following code to show simple UI.

Simple UI code into 'ViewController.mm' for **Objective-C** project

```
#include "AppsealingiOS.h"
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];

    int tamper = ObjC_IsAbnormalEnvironmentDetected();
    if ( tamper > 0 )
    {
        NSString* msg = @"Abnormal Environment Detected !!";
        if (( tamper & DETECTED_JAILBROKEN ) > 0 )
            msg = [msg stringByAppendingString:@"\n - Jailbroken"];
        if (( tamper & DETECTED_DRM_DECRYPTED ) > 0 )
            msg = [msg stringByAppendingString:@"\n - Executable is not encrypted"];
        if (( tamper & DETECTED_DEBUG_ATTACHED ) > 0 )
            msg = [msg stringByAppendingString:@"\n - App is debugged"];
        if ( tamper & ( DETECTED_HASH_INFO_CORRUPTED | DETECTED_HASH_MODIFIED )) > 0
            msg = [msg stringByAppendingString:@"\n - App integrity corrupted"];
        if ( tamper & ( DETECTED_CODESIGN_CORRUPTED | DETECTED_EXECUTABLE_CORRUPTED )) > 0
            msg = [msg stringByAppendingString:@"\n - App executable has corrupted"];
        if ( tamper & DETECTED_CERTIFICATE_CHANGED ) > 0
            msg = [msg stringByAppendingString:@"\n - App has re-signed"];

        UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"AppSealing"
                                    message:msg
                                    preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction *confirm = [UIAlertAction actionWithTitle:@"Confirm"
                                  style:UIAlertActionStyleDefault
                                  handler:^(UIAlertAction * _Nonnull action) {
#if !DEBUG && !defined(DEBUG) // Debug mode does not kill app even if security threat has found
                                    exit(0);
#endif
                                }];
        [alert addAction:confirm];
        [self presentViewController:alert animated:YES completion:nil];
    }
}
```

Your app will show simple alert box like below when you run your app on abnormal device such as jailbroken or debug your app using Xcode or gdb. In such situation irrespective of user action the app will exit after 20 seconds automatically.

When security intrusion has detected in iOS device, simple alert view will appear and app will close after 20 seconds or when user touches "Confirm" button



If your project is Cordova based, you can put this code block into MainViewController.mm instead of ViewController.mm

(MainViewController.m file must have the extension changed to .mm)

If your project is Ionic based, the ViewController.swift file is not created, so you can create the GUI by adding the following modified code to AppDelegate.swift file

.

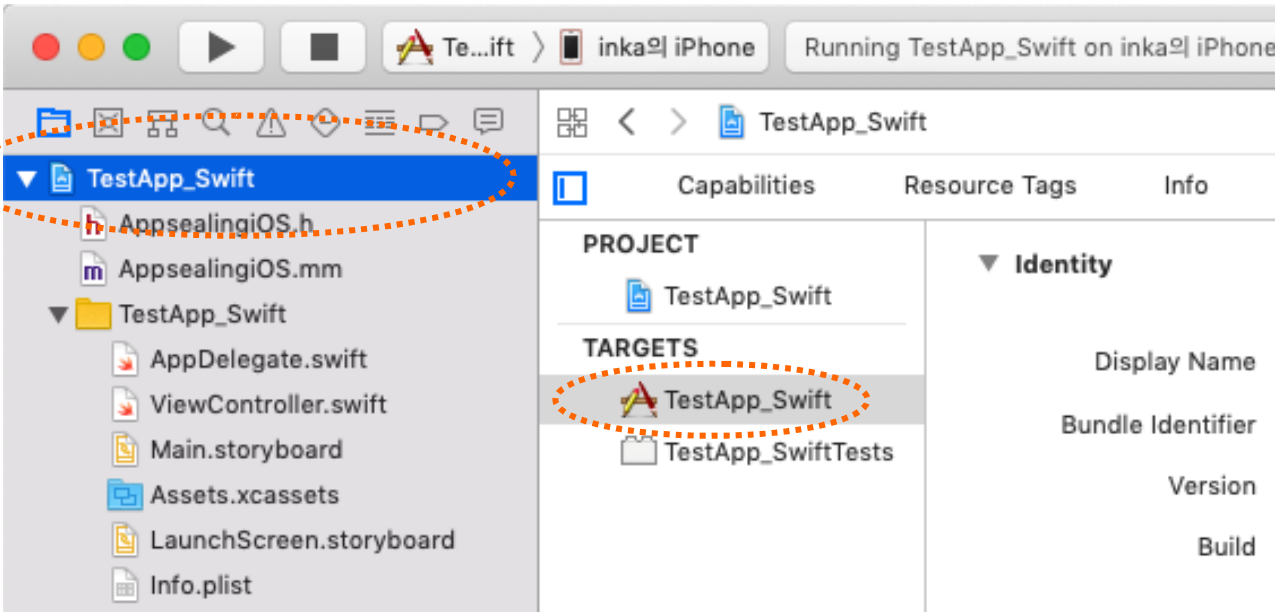
Simple UI code into 'AppDelegate.swift' for **Ionic** project

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
                launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool
{
    // Override point for customization after application launch.

    let inst: AppSealingInterface = AppSealingInterface();
    let tamper: Int32 = inst._IsAbnormalEnvironmentDetected();
    if ( tamper > 0 )
    {
        var msg = "Abnormal Environment Detected !!";
        if ( tamper & DETECTED_JAILBROKEN ) > 0
            { msg += "\n - Jailbroken"; }
        if ( tamper & DETECTED_DRM_DECRYPTED ) > 0
            { msg += "\n - Executable is not encrypted"; }
        if ( tamper & DETECTED_DEBUG_ATTACHED ) > 0
            { msg += "\n - App is debugged"; }
        if ( tamper & ( DETECTED_HASH_INFO_CORRUPTED | DETECTED_HASH_MODIFIED )) > 0
            { msg += "\n - App integrity corrupted"; }
        if ( tamper & ( DETECTED_CODESIGN_CORRUPTED | DETECTED_EXECUTABLE_CORRUPTED )) > 0
            { msg += "\n - App executable has corrupted"; }
        if ( tamper & DETECTED_CERTIFICATE_CHANGED ) > 0
            { msg += "\n - App has re-signed"; }
        let alertController = UIAlertController(title: "AppSealing",
                                message: msg, preferredStyle: .alert );
        alertController.addAction(UIAlertAction(title: "Confirm", style: .default,
                                handler: { (action:UIAlertAction!) -> Void in
#if !DEBUG   // Debug mode does not kill app even if security threat has found
                                            exit(0);
#endif
                                        } ));
         // show alert
        let alertWindow = UIWindow( frame: UIScreen.main.bounds )
        alertWindow.rootViewController = UIViewController()
        alertWindow.windowLevel = UIWindow.Level.alert + 1;
        alertWindow.makeKeyAndVisible()
        alertWindow.rootViewController?.present( alertController,
                                                 animated: true, completion: nil )
    }
    return true;
}
```

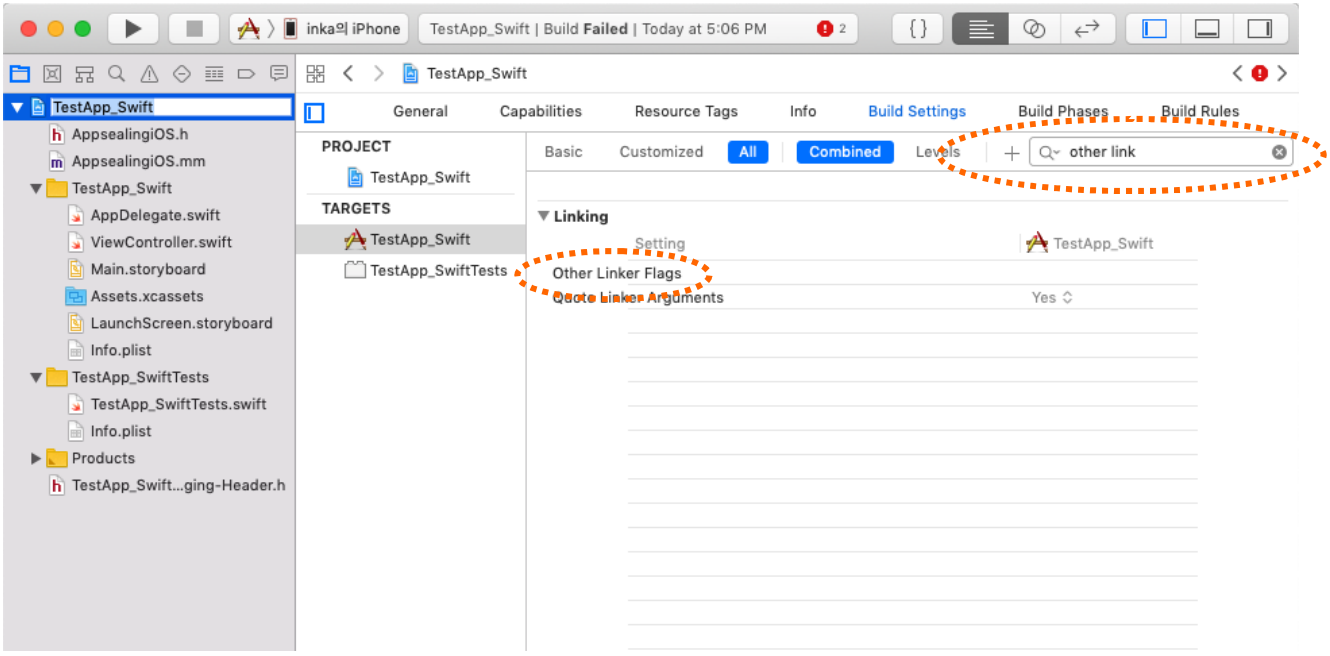
 

# Part 4. Modify "Build Settings" of your project

4-1 Select top project at Project Navigator and select "TestApp_Swift" target at TARGETS



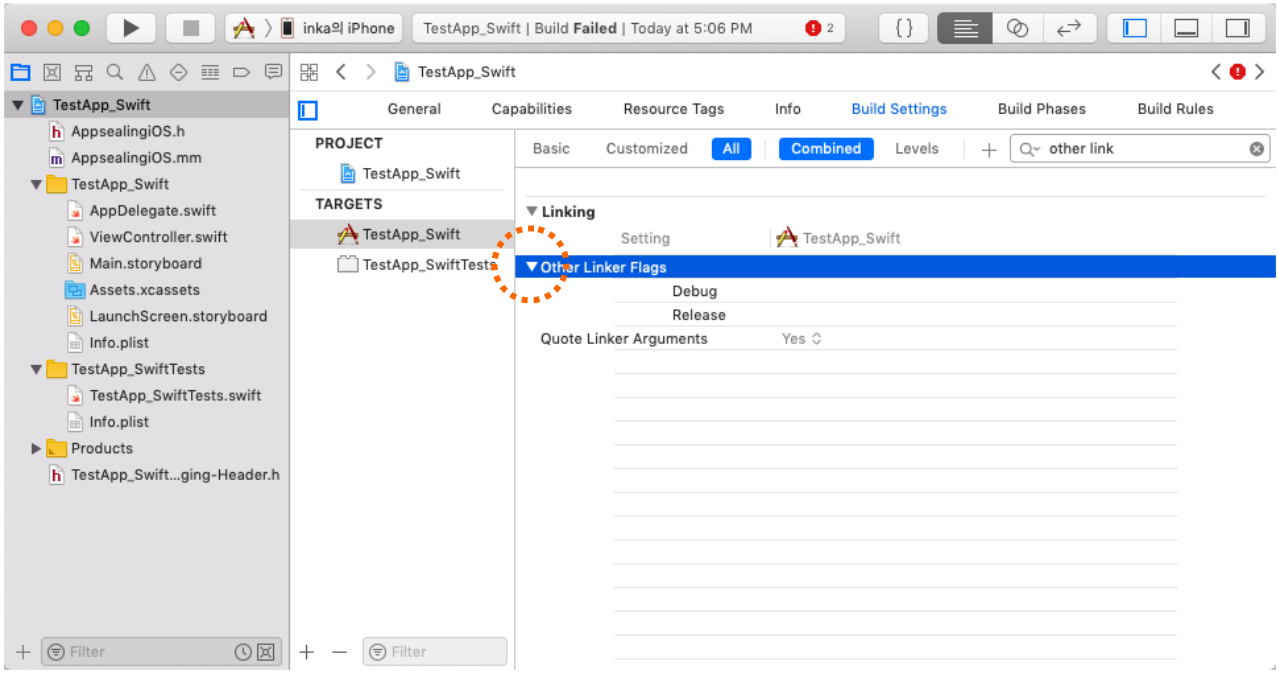

4-2 Select "**Build Settings**" tab and type "**Other link**" in search box



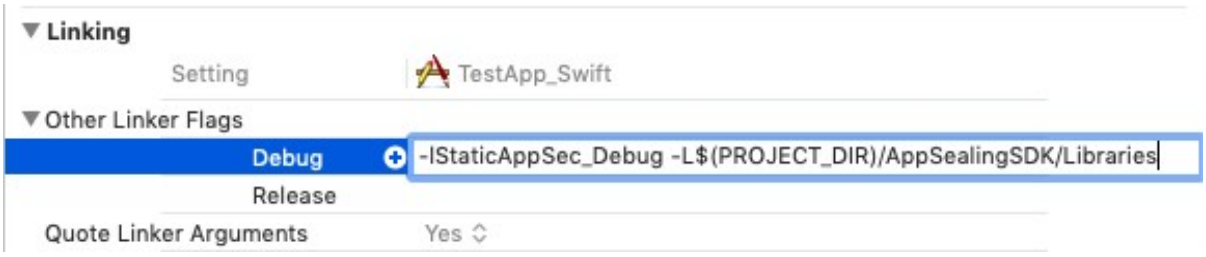
Then you will see **"Other Linker Flags"** field. Now, fill this settings by following steps.

1) Expand "Other Linker Flags" by clicking the triangle icon left to "Other Linker Flags"




2) Select "Debug" item and click to edit/insert value, then type in following text.

*-lStaticAppSec_Debug -L$(PROJECT_DIR)/AppSealingSDK/Libraries*




3) Select "Release" item and click to edit/insert value, then type in following text.

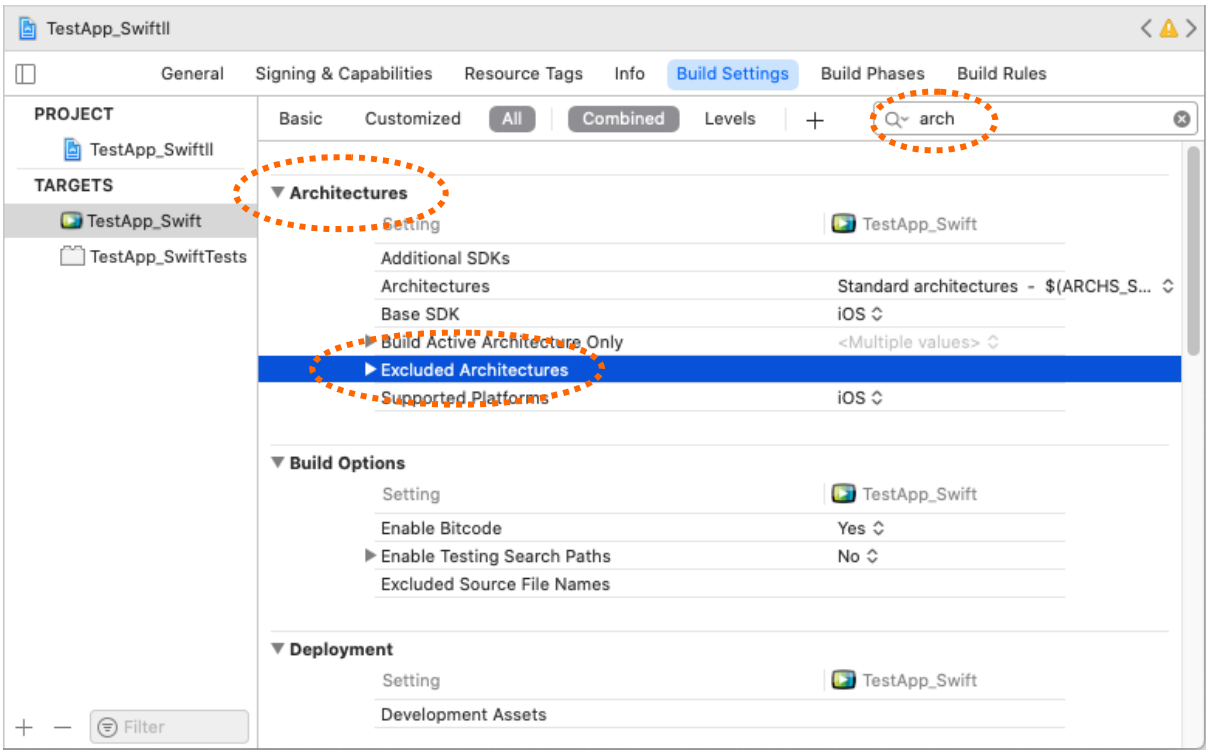
*-lStaticAppSec -L$(PROJECT_DIR)/AppSealingSDK/Libraries*

Below step is needed only when you launch app in Simulator with Release build. If you don't need to run your Release build in simulator you can skip this step.

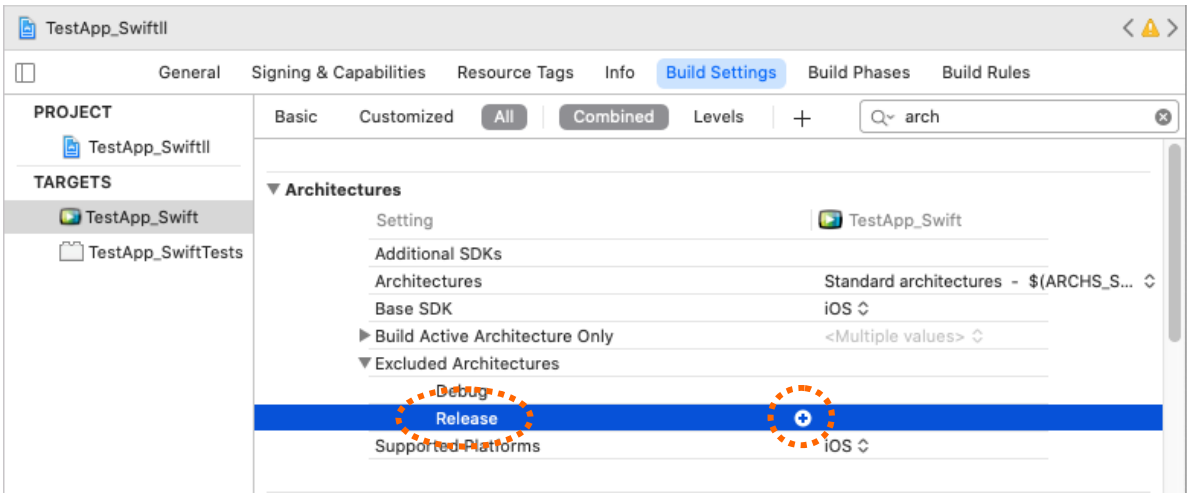
## 4-3 Configure **Build Settings** "**Architectures – Excluded Architectures**"

Search "arch" keyword at Build Settings panel.



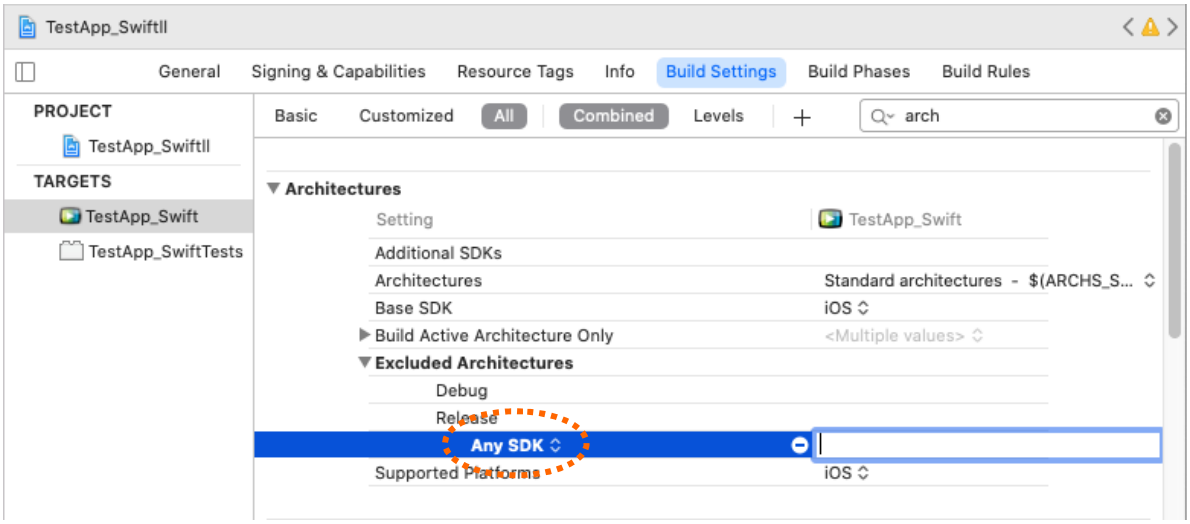

Follow next steps after **"Architectures"** field has shown.

1) Expand "Excluded Architectures" by clicking the left-side triangle icon.

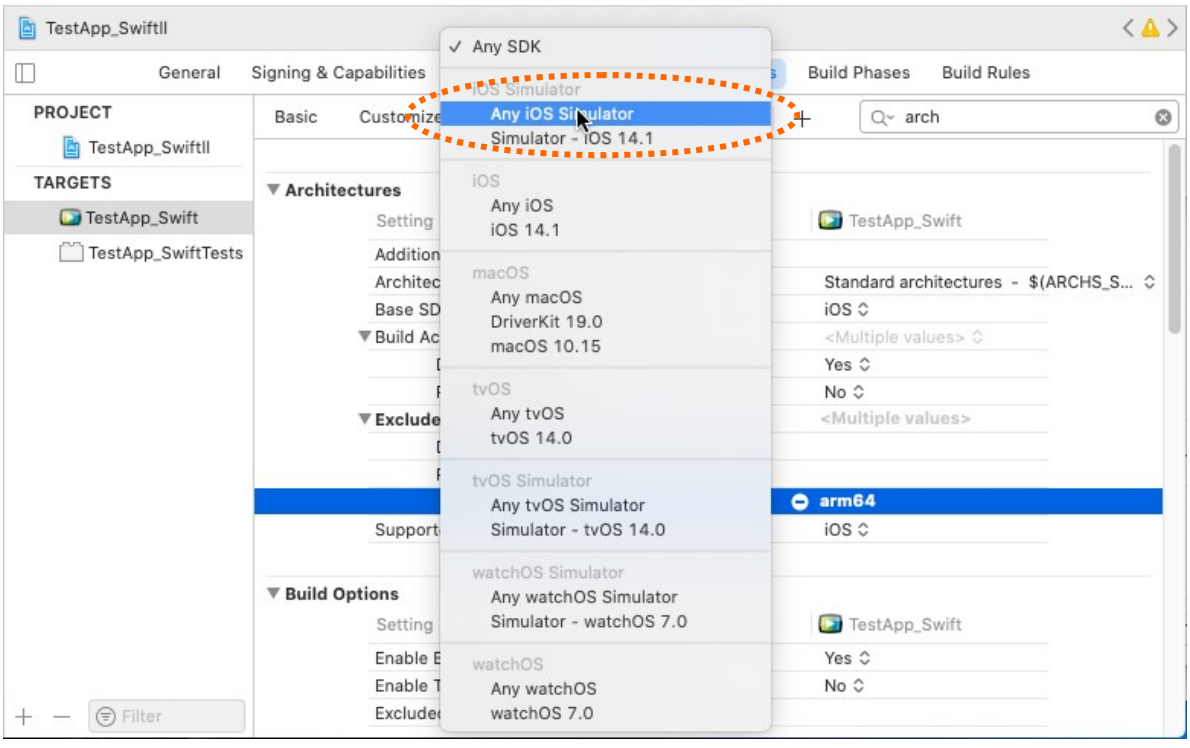


2) Select "Release" item and click right-side "+" icon.

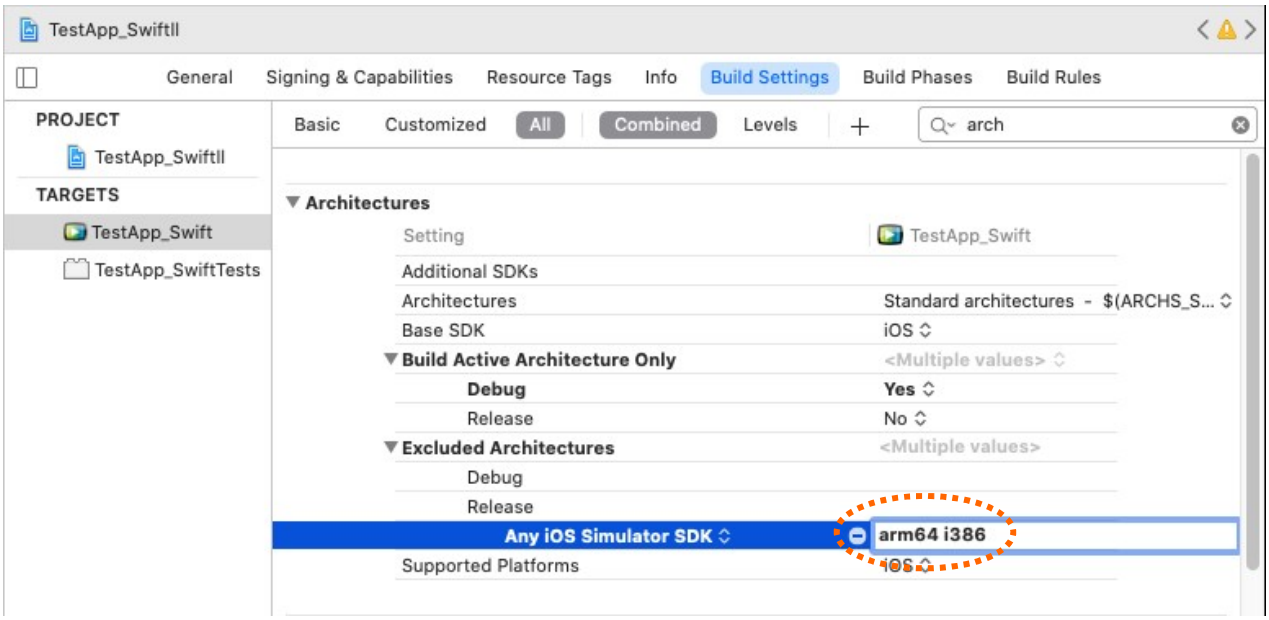



3) After "Any SDK" item has created click it to show popup menu.



4) Select "Any iOS Simulator" item in popup menus.




5) Enter value for "Any iOS Simulator SDK" as "arm64 i386". (Space between arm64 and i386)

Now AppSealing security features have been adopted to your project. Go on 'Build', 'Run', 'Archive' as usual.

## 4-4 Reminds about Xcode build mode

* AppSealing work differently in debug mode and release mode.

If you build an app in Debug mode, AppSealing`s security features are disabled for convenient development :

- Jailbreak detection
- Anti-debugging
- Not encrypted executable file detection
- App-Integrity corruption detection
- Re-signing detection

These features are enabled when you build app as Release mode.

You will build the app as Release mode when distributing to the App Store. If you test AppSealing with Release mode, your app should be distributed to App Store or 'TestFlight'. If not, the executable file will be detected as not encrypted, so the app will be closed.

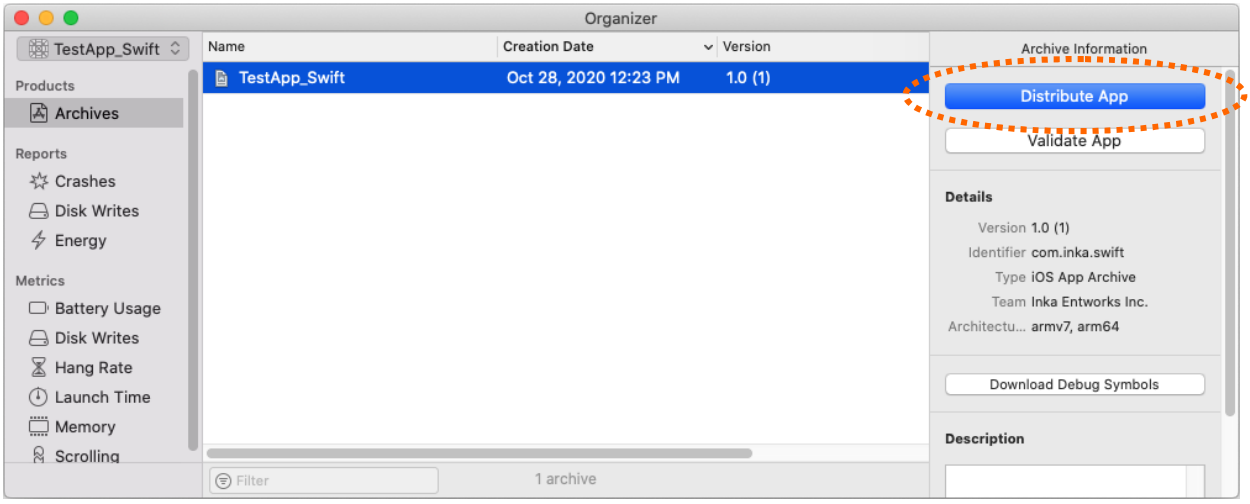
 

## 4-5 Generate App integrity & certificate verification snapshot

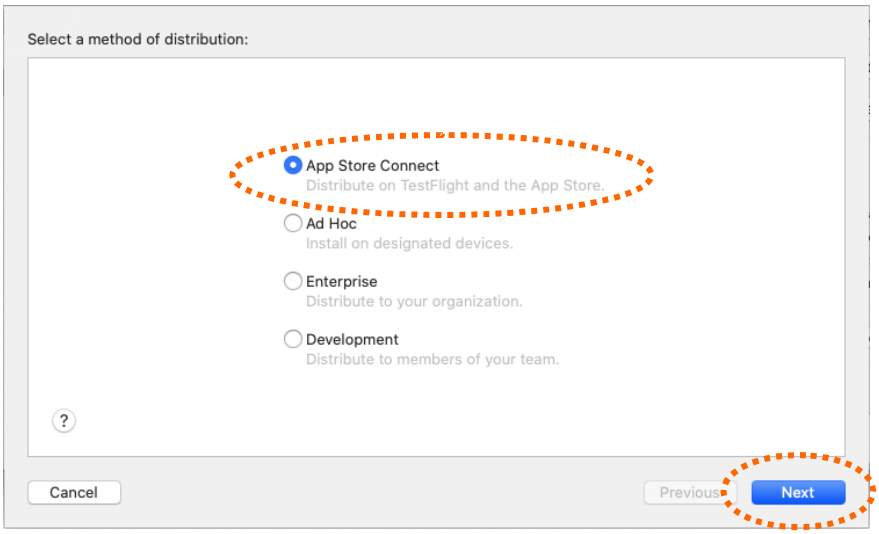
There is additional process to verify app integrity & certificate when you test your app or distribute app through app store. If you skip this step the app running on device will be terminated after few seconds for broken app integrity.

When distributing app built in Release mode through Development or Ad Hoc it will be terminated for security check which tells the executable has not encrypted by Fairplay DRM so skipping this step will have the same result, but **you should process this step when you distribute your app through TestFlight or App Store.**

Let's see the upload process to App Store or TestFlight step by step. Below is Organizer window after Archive from Xcode.

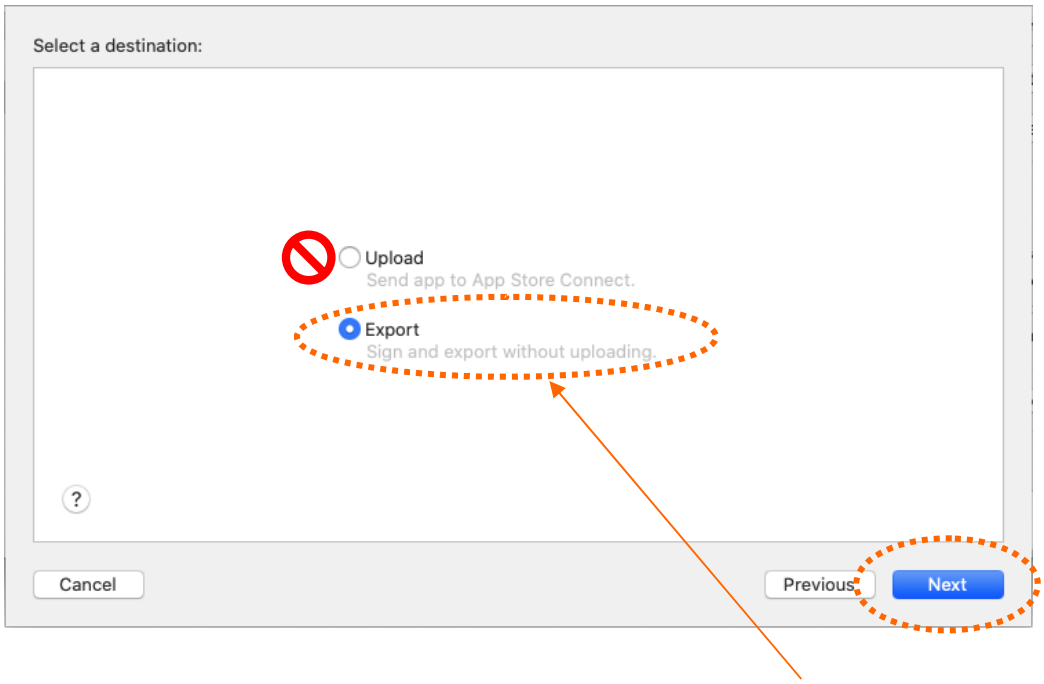



Click "Distribute App" button to generate IPA for uploading to App Store.

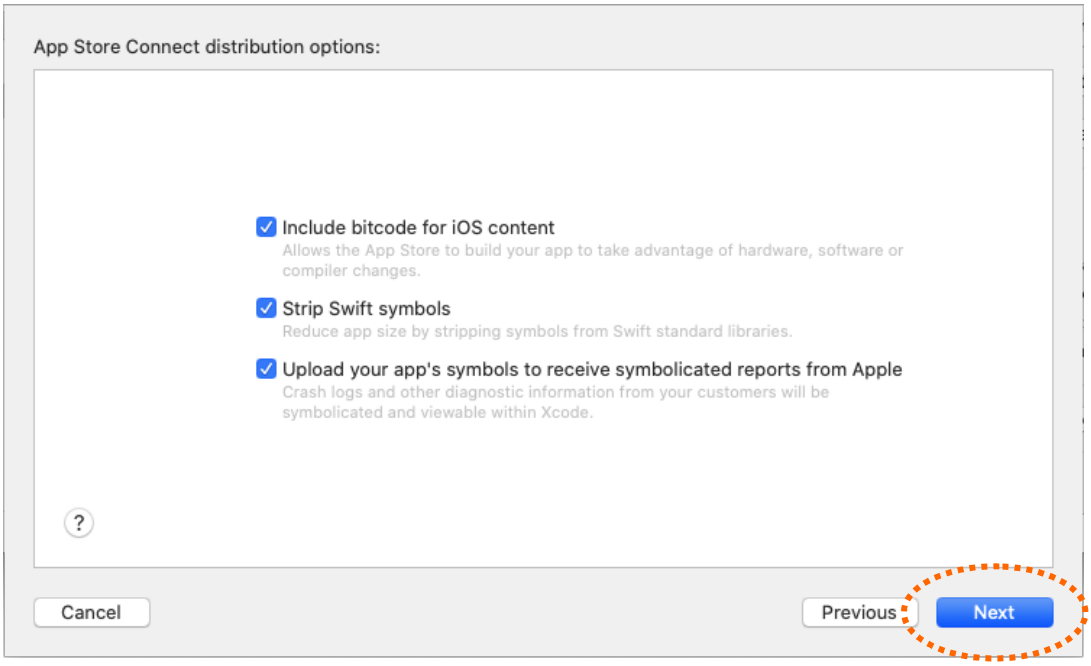


Click "Next" button with "App Store Connect" is selected.

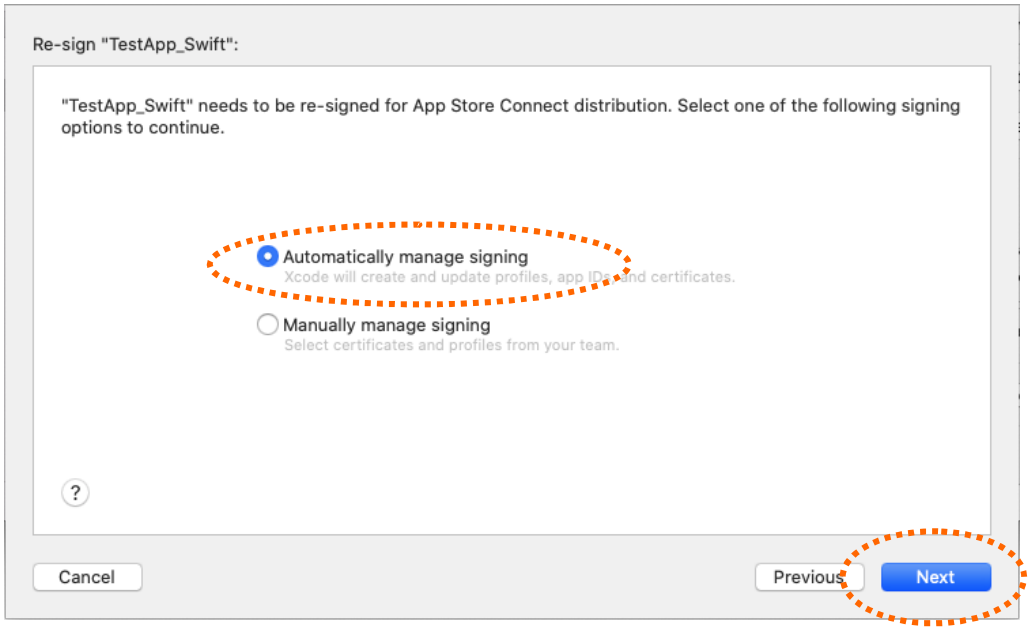



You usually selected "Upload" almost but you **must select "Export" to apply AppSealing**. This is because taking snapshot for app integrity and certificate is needed and your **app will not run normally on device without this process**. Click "Next" button with "Export" is selected.

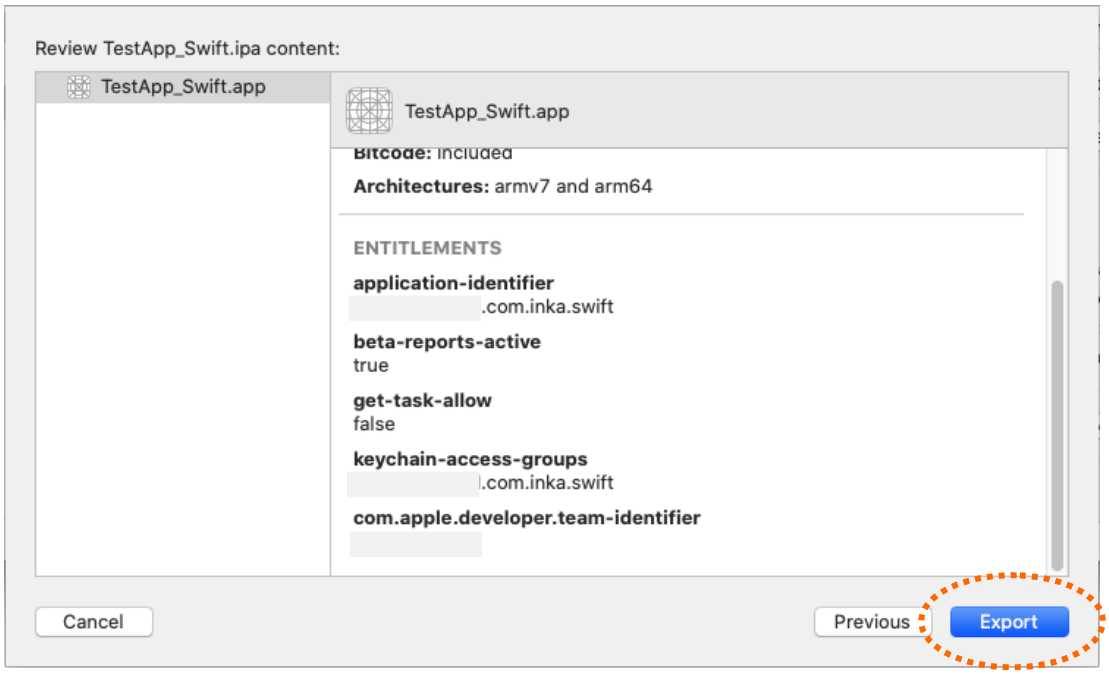



Click "Next" button with all options keep default.

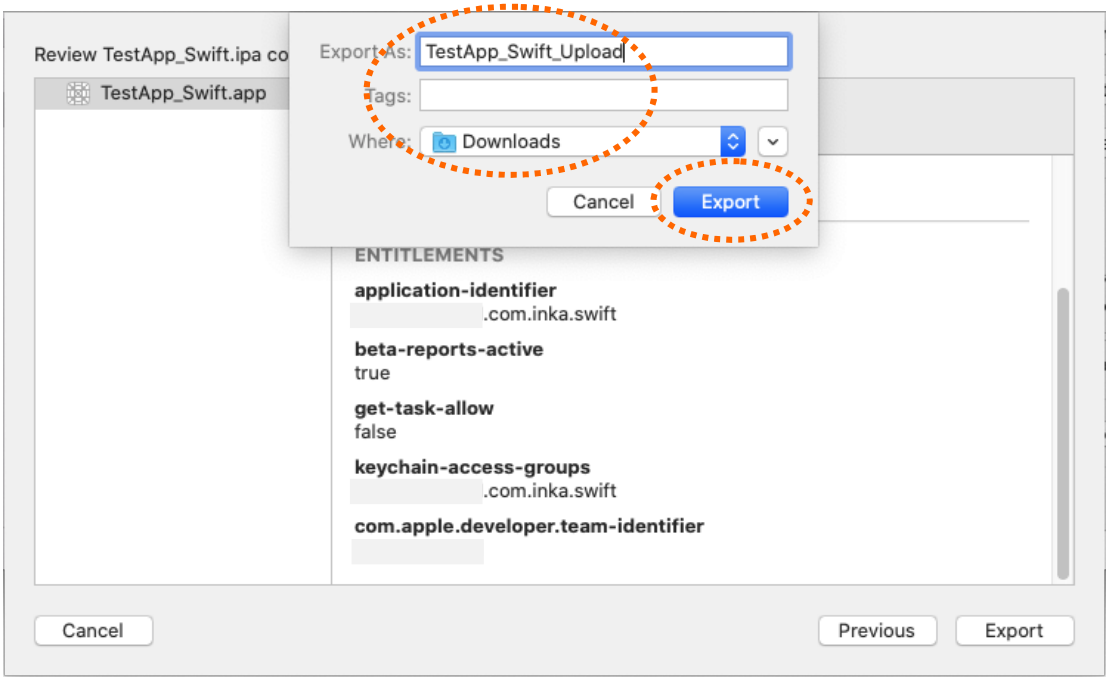



With default options retained, click "Next" button. Then you can see the window from which you can export as an IPA.



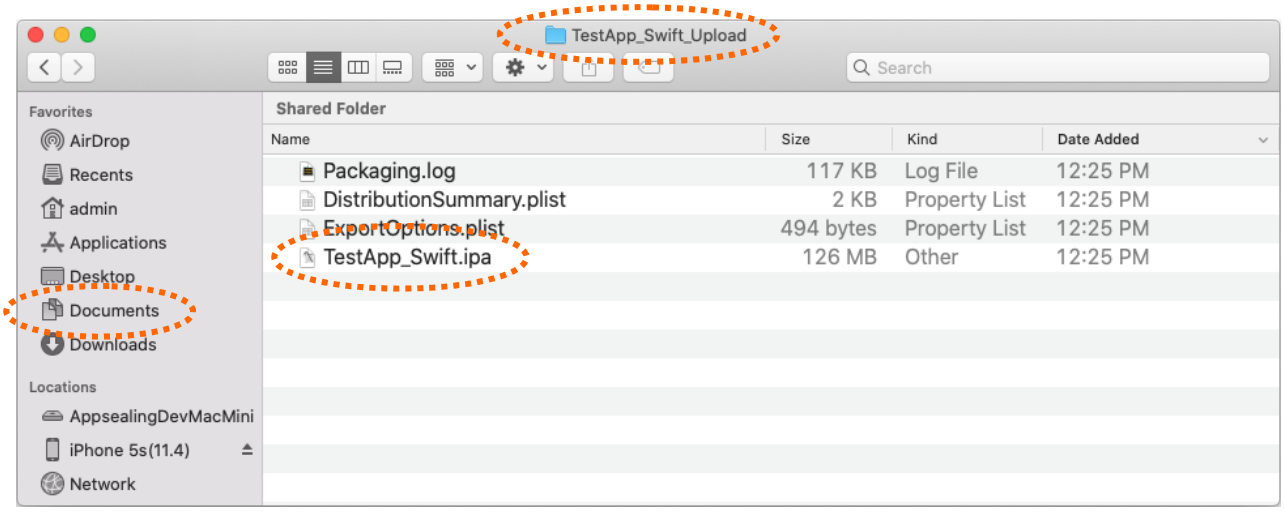

Verify the brief contents and click "Export" button.



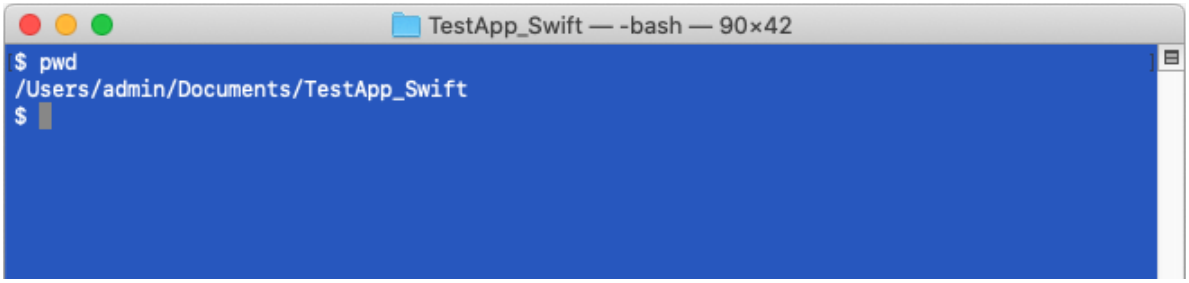

When destination dialog appear select store location and click "Export" button. This document used folder named "~/Downloads/TestApp_Swift_Upload"

After you've clicked "Export" button IPA file will be created at the designated folder. You can see the generated IPA file at finder like below. Now, you should keep in mind the location of IPA or remain finder widow opened.



Now you should process next step with exported IPA. Launch terminal app and move to Xcode project folder.




This document used project path for unity-exported Xcode project as "~/Documents/TestApp_Swift". You can verify the path name by pwd command like above picture.

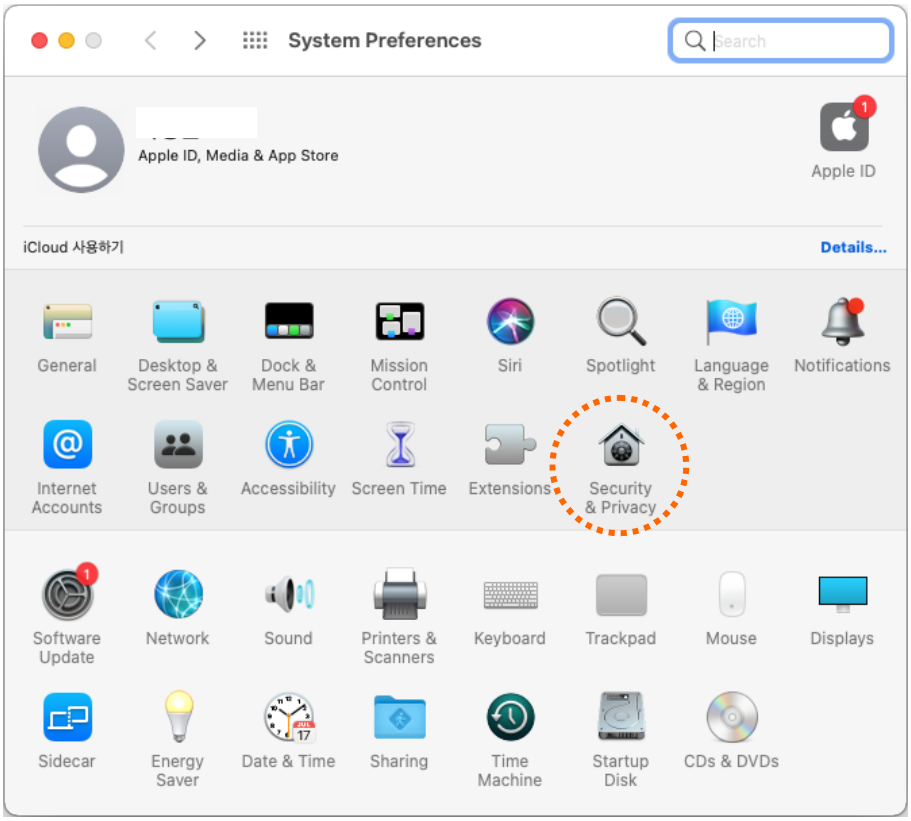
Run add permission command like below and open generate_genesis file. (You can open the file by double-clicking it in Finder)

```
$ chmod +x AppSealingSDK/Tools/*
$ open AppSealingSDK/Tools/generate_genesis
```

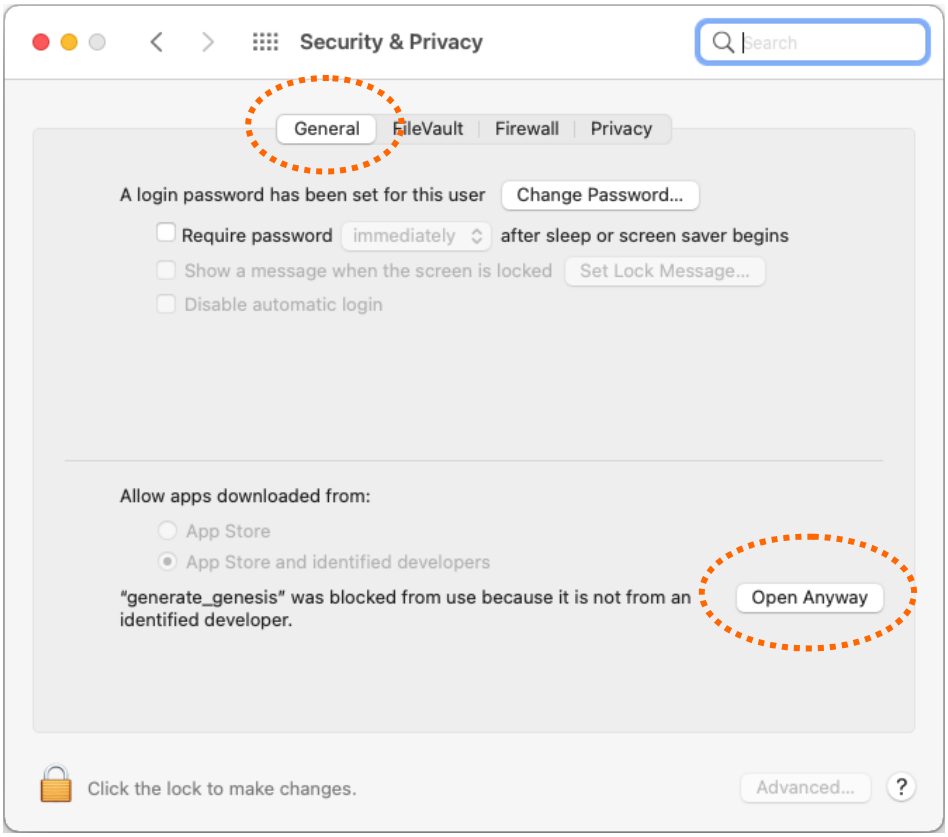
When you first install the SDK and open the generate_genesis file for the first time, the following warning window may appear

In this case, click the OK button to close the window, then go to the settings window and click the "Security and Privacy" item.

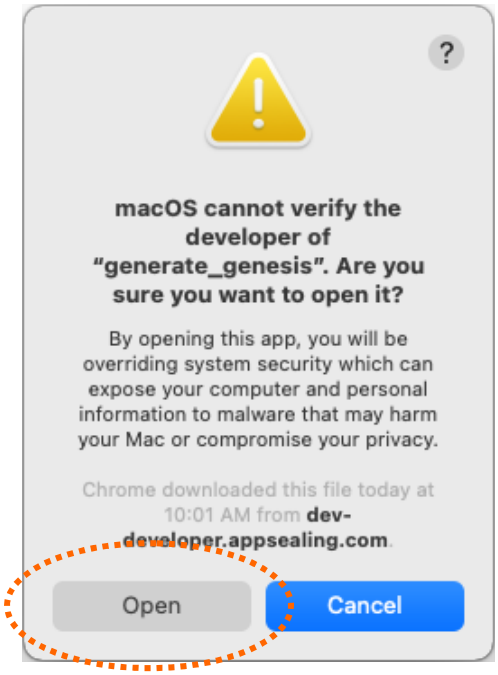



Select the "General" tab on the left and click the "Open Anyway" button at the bottom.

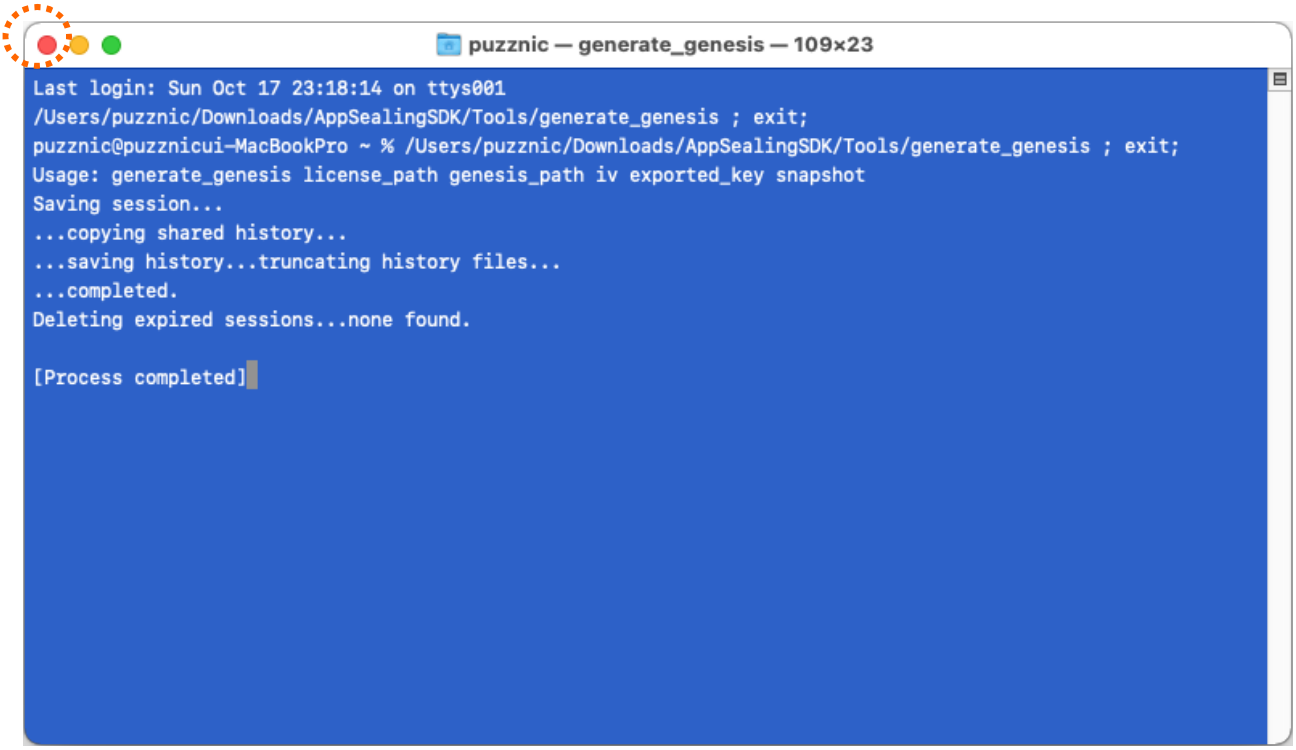


Click "Open" button when another confirmation window appears after clicking the "Open Anyway" button.



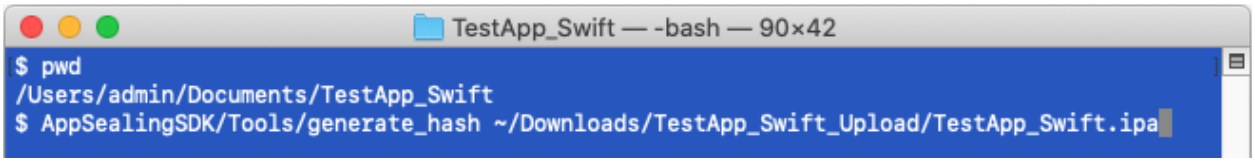

New terminal window will show the execution result of "Open" action, just close the window.



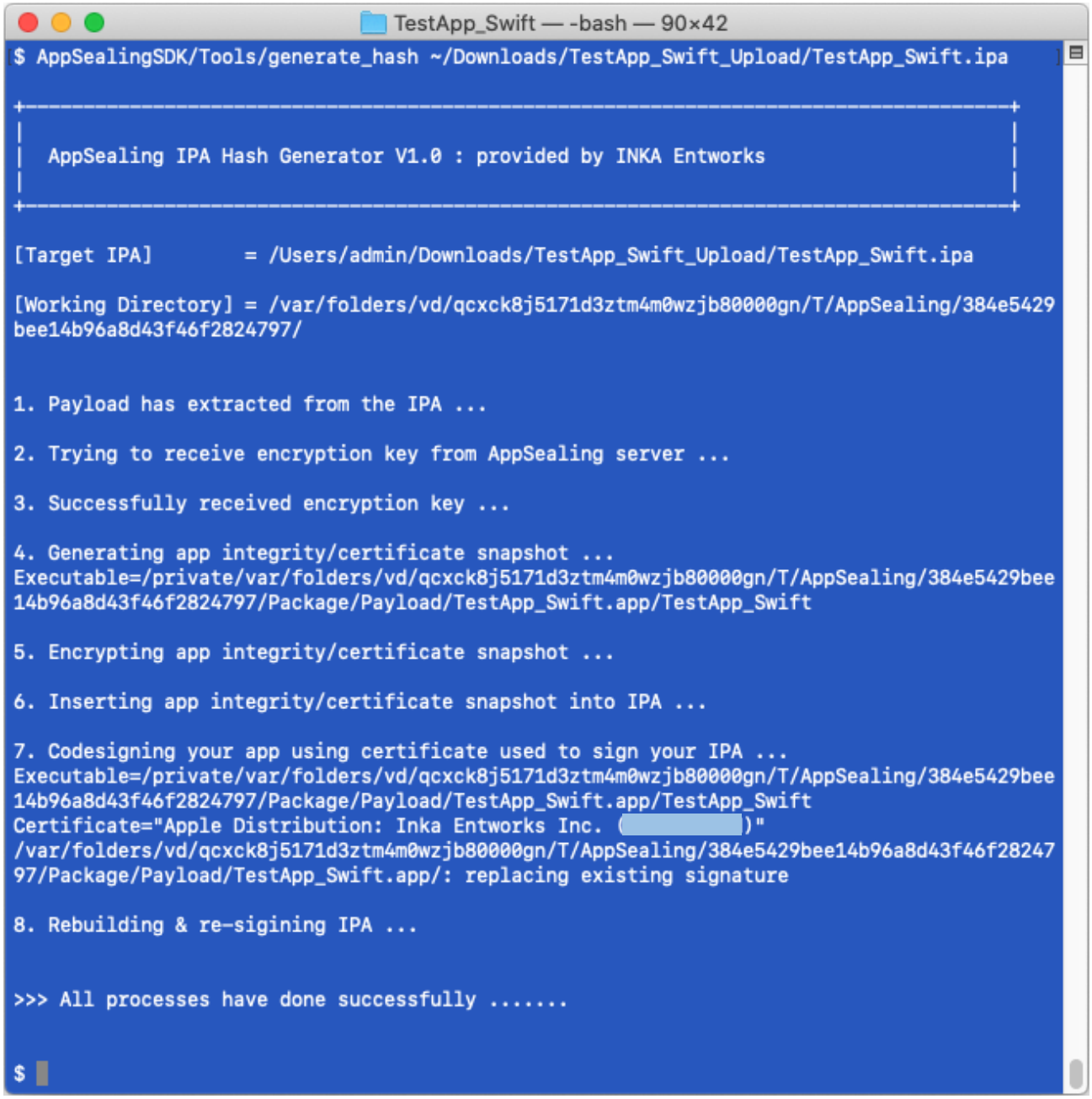

Now you run 'generate_hash' script like below. This script has only one parameter which is path to the exported IPA file in previous step. You can type the IPA path manually or drag & drop the IPA file from the opened Finder window in previous step.

```
$ AppSealingSDK/Tools/generate_hash ~/Downloads/TestApp_Swift_Upload/TestApp_Swift.ipa
```




After you execute the script you will see the progress like below and snapshot for app integrity and certificate will be added to the IPA file.

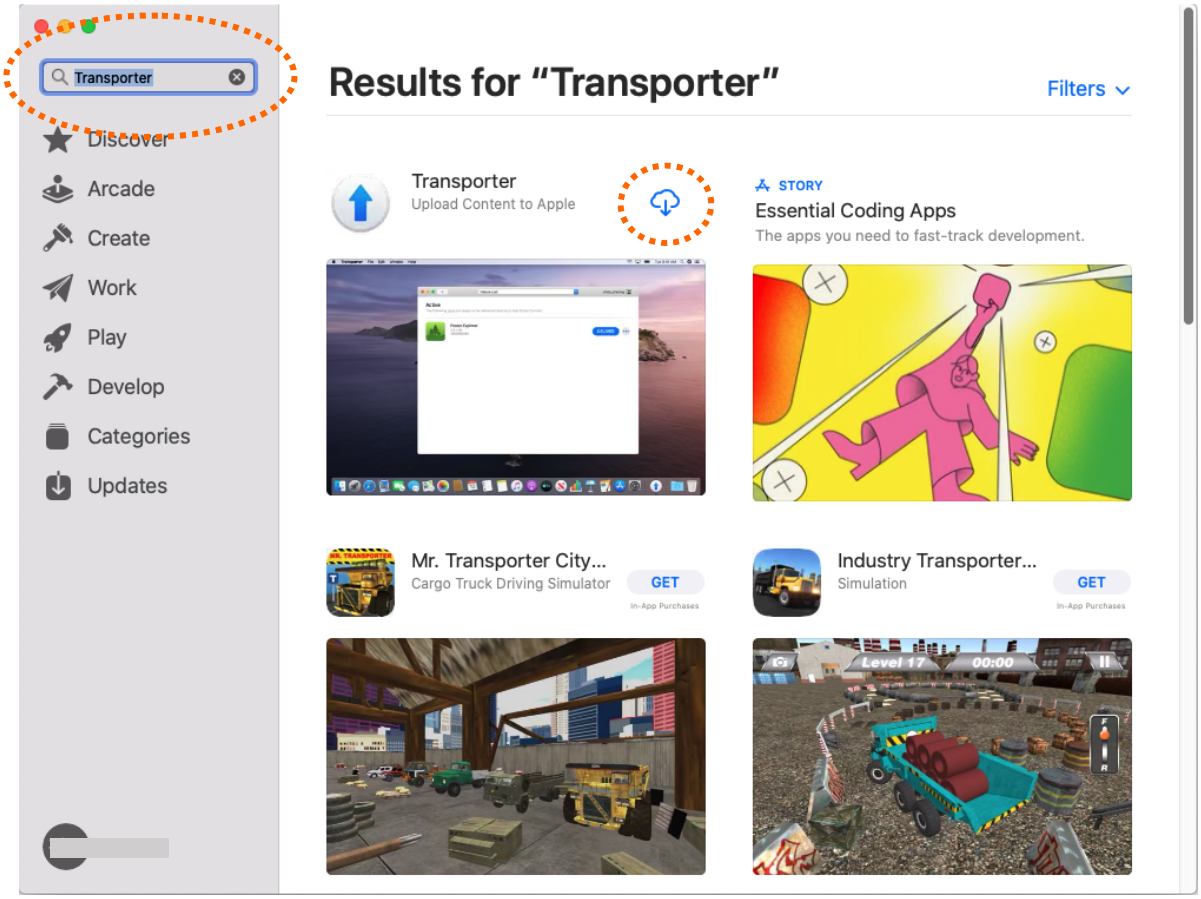



This process has to be applied to distribution step as "Ad Hoc", "Enterprise", "Development" identically.

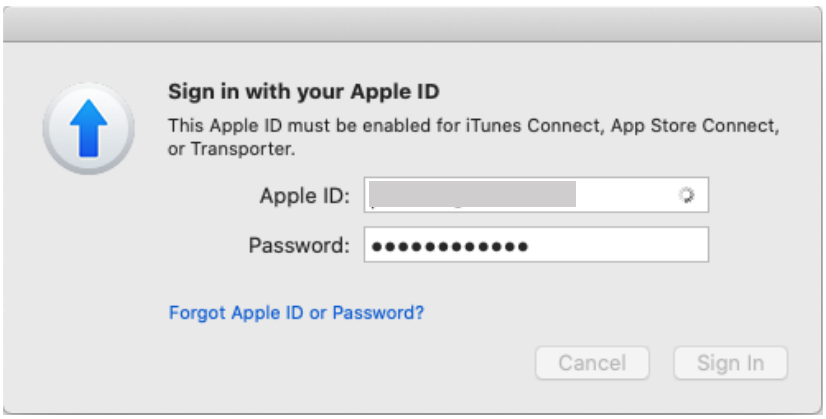
 

## 4-6 Upload re-signed IPA to App Store Connect

Now you can upload re-signed IPA to App Store Connect. This document uses Transporter app (MAC) for convenient uploading. If the Transporter app has not installed in your MAC you can open Mac AppStore, search "Transporter" and install.

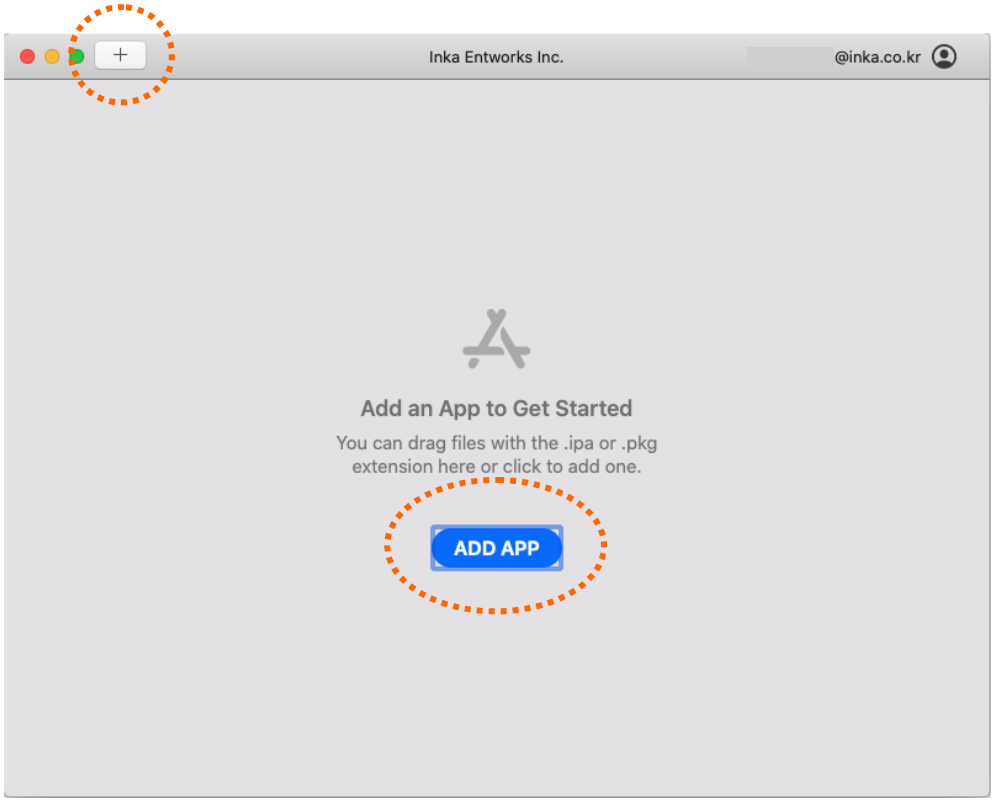


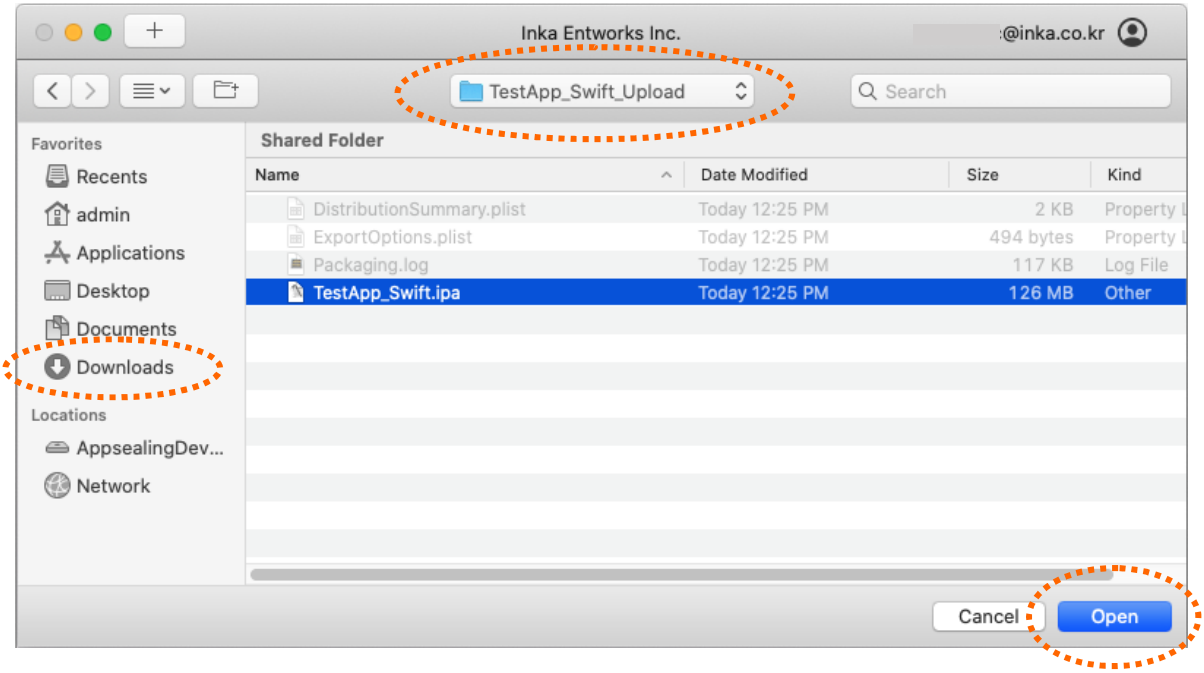

Launch Transporter after installation you are requested for Apple ID like below. Enter your Apple ID and password. (This step is required only once for the first time)



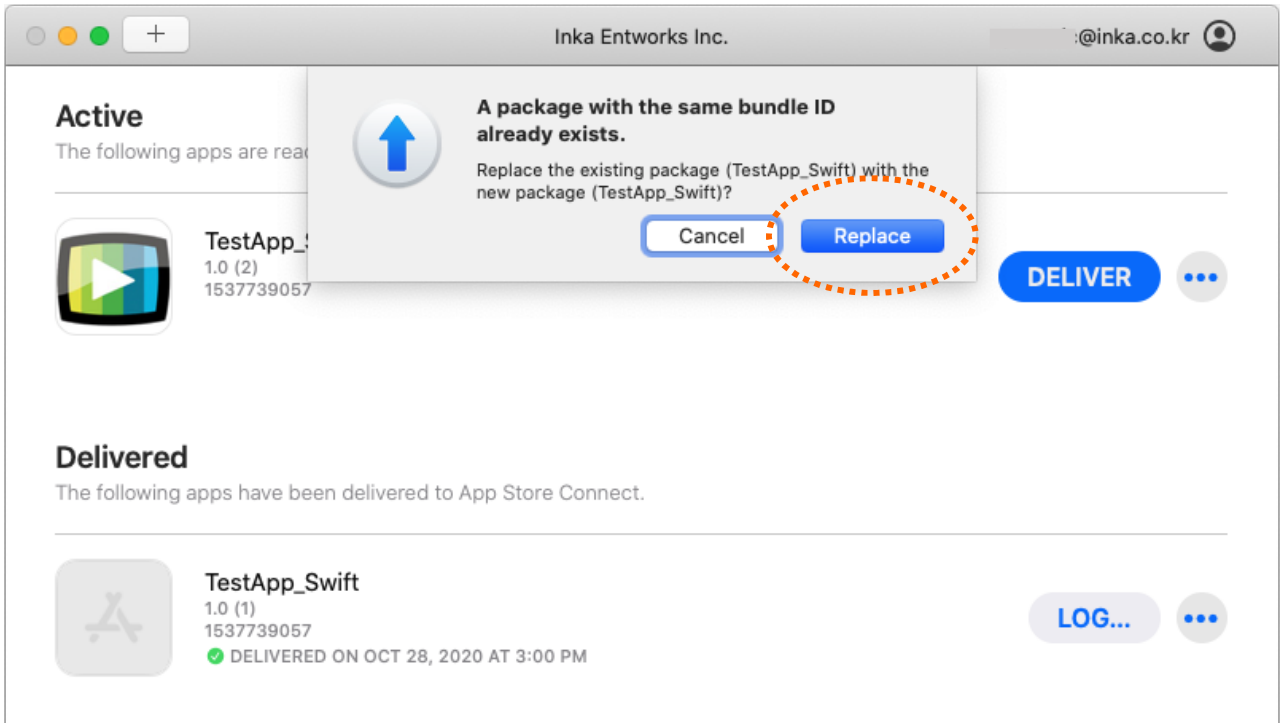
After you login with your ID and password you can see the Transporter window like below. Click the "+" button upper-left or "ADD APP" button in the middle to select IPA to be uploaded and select the re-signed IPA in previous step.



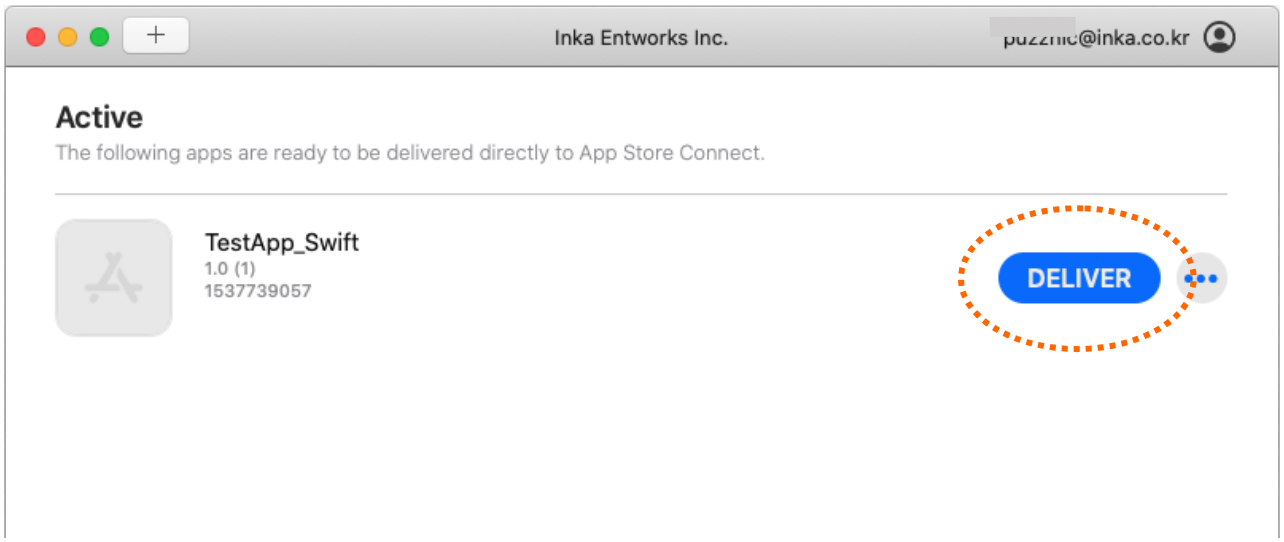



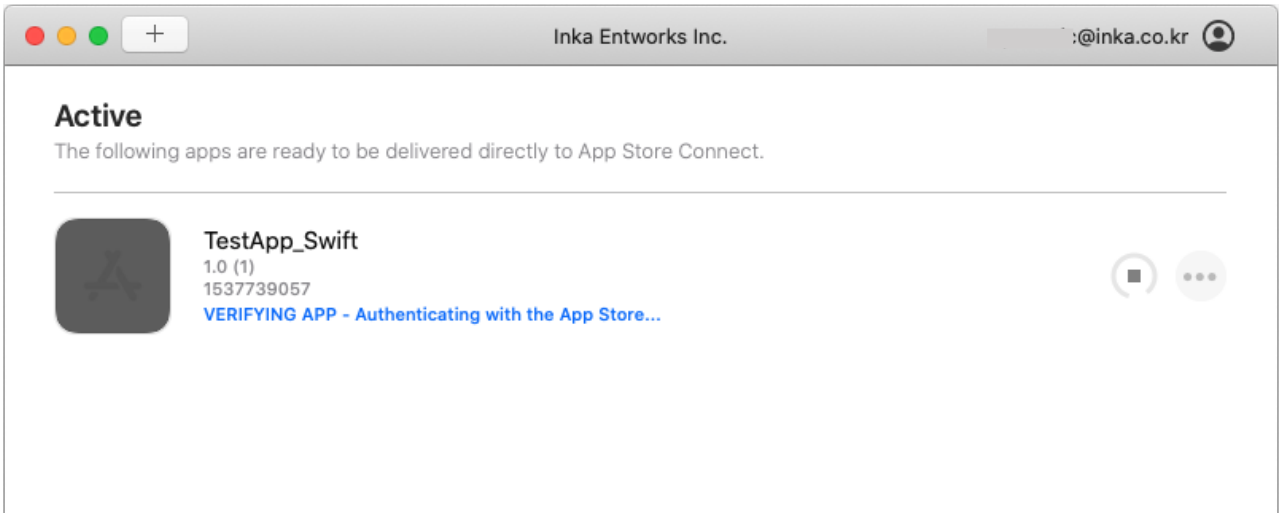
When you update your app by adding IPA file with new version or higher build number a warning dialog can appear like below because of same bundle ID. In this case just click "Replace" button to upload new IPA.

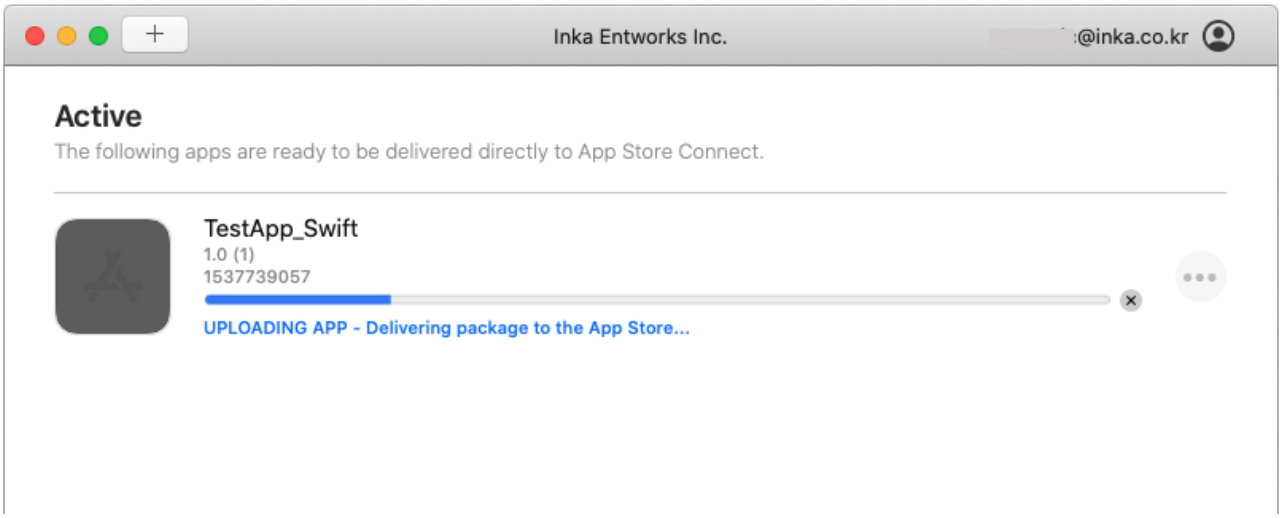


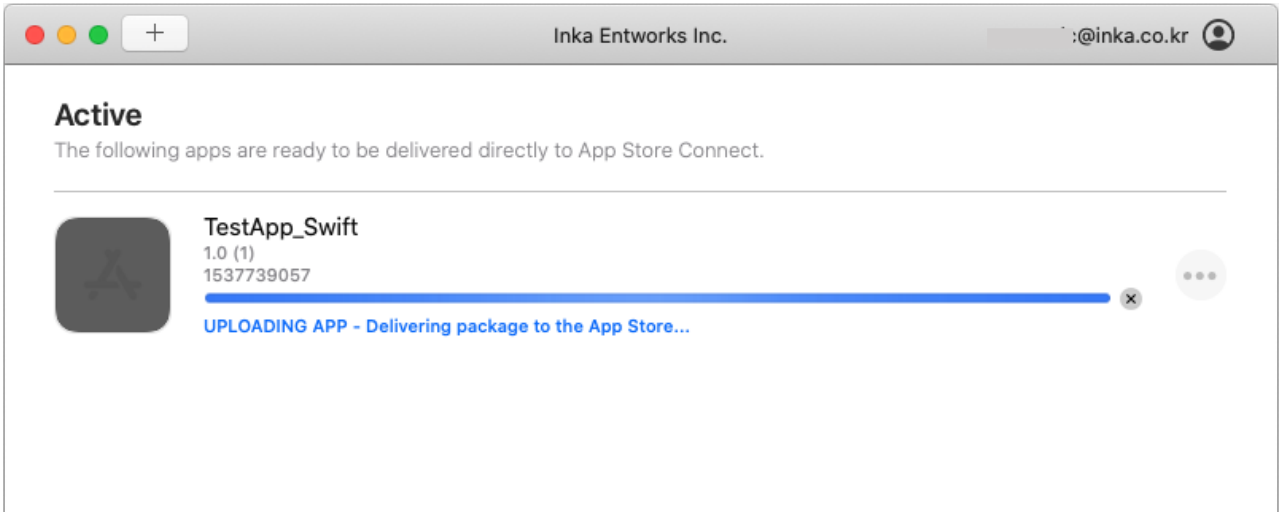

After IPA file has added, click "DELIVER" button then verifying and uploading to App Store Connect process will be in progress.



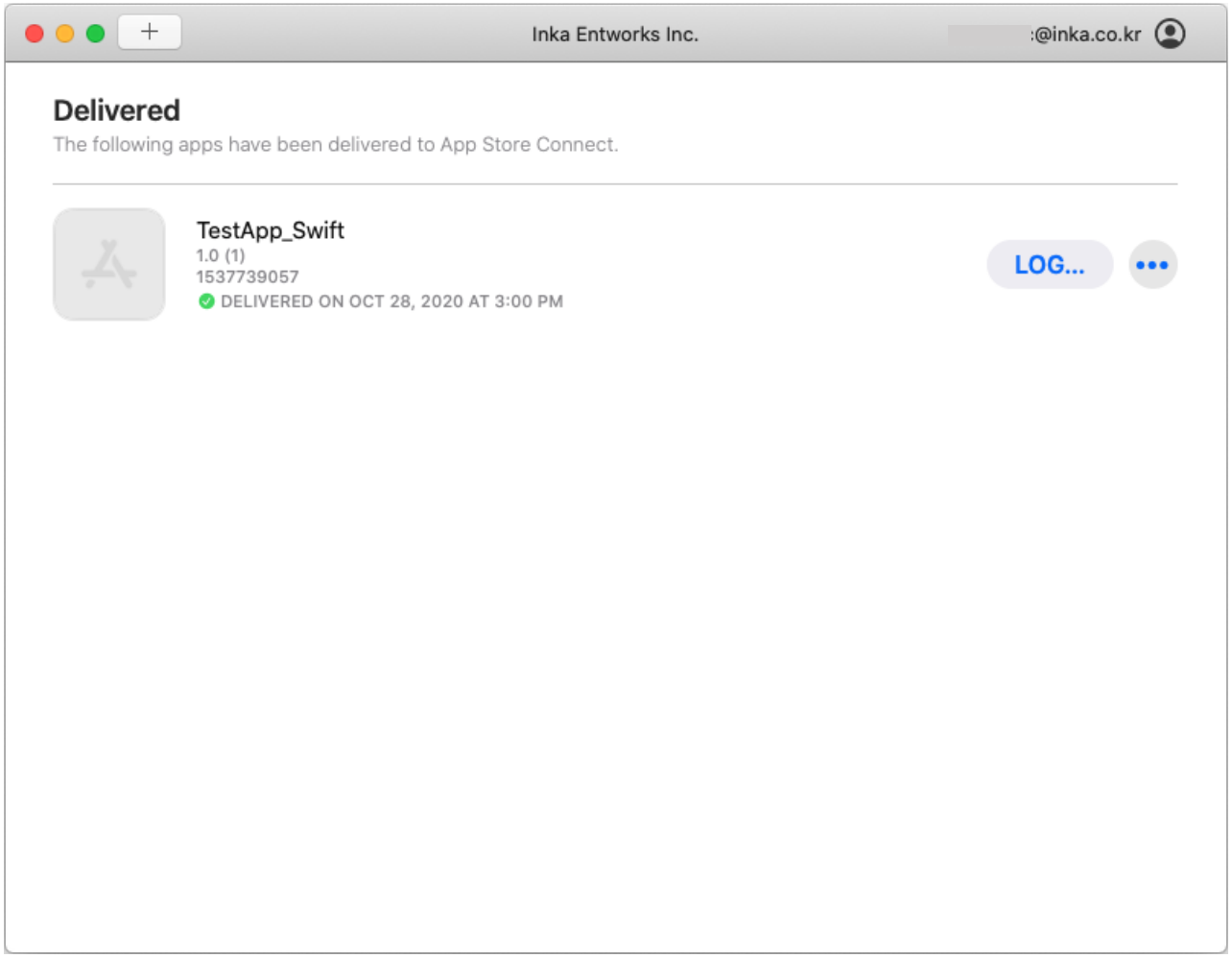


If you encounter below window the upload process has finished and you can submit your build for App Store review or TestFlight distribution.



# Part 5. Acquire AppSealing device unique identifier

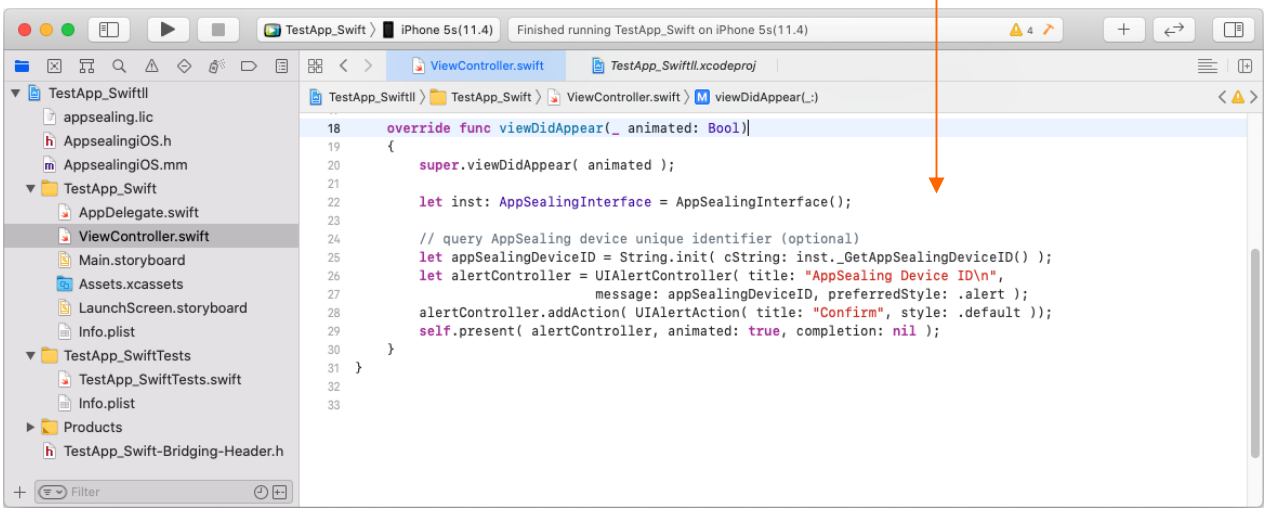
AppSealing SDK generates and manages unique identifier for each device. Customer who use the AppSealing SDK can use the interface of AppSealing to verify the device unique identifier, if necessary. And can be used for the business using the hacking data service provided by AppSealing.

## 5-1 Show acquire device unique identifier

First, open your Xcode project and open "ViewController.swift" file. After you opened the swift file put following code into that (If ViewController.swift file has already included '**viewDidAppear**' method just insert the body of following code below 'super.viewDidAppear( animated );' line.)

Simple UI code into 'ViewController.swift' for **Swift** project

```
override func viewDidAppear(_ animated: Bool)
{
    super.viewDidAppear( animated );

    let inst: AppSealingInterface = AppSealingInterface();
    let appSealingDeviceID = String.init( cString: inst._GetAppSealingDeviceID() );
    let alertController = UIAlertController( title: "AppSealing DeviceID",
                                    message: appSealingDeviceID, preferredStyle: .alert );

    alertController.addAction( UIAlertAction( title: "Confirm", style: .default ));
    self.present( alertController, animated: true, completion: nil );
}
```



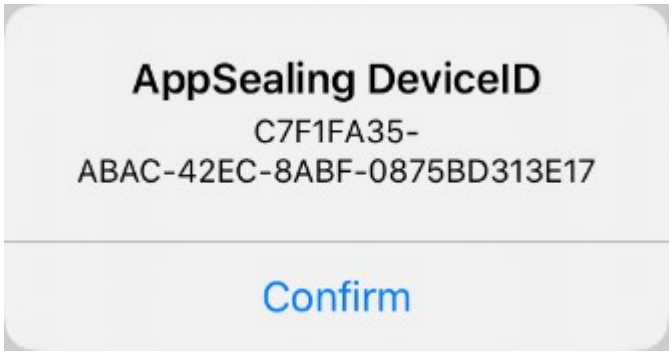
Sample code is also included in "AppSealingiOS.mm" file so you can copy & paste in that file.

If your project is Objective-C based then you can use following code to show simple UI.

Simple UI code into 'ViewController.mm' for **Objective-C** project

```
#include "AppsealingiOS.h"

char _appSealingDeviceID[64];
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];

    if ( ObjC_GetAppSealingDeviceID( _appSealingDeviceID ) == 0 )
    {
        UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"AppSealing DeviceID"
                            message:[[NSString alloc] initWithUTF8String:_appSealingDeviceID]
                            preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction *confirm = [UIAlertAction actionWithTitle:@"Confirm"
                            style:UIAlertActionStyleDefault
                            handler:^(UIAlertAction * _Nonnull action) { }];
        [alert addAction:confirm];
        [self presentViewController:alert animated:YES completion:nil];
    }
}
```

Your app will show simple alert box like below when you run your app.



# Part 6. Enhanced jailbreak-detection using app server

## 6-1 Overview and Necessity of Enhanced Jailbreak Detection Method

One of the main functions within AppSealing SDK is detecting the environment of the jailbroken device and forcibly closes the app. However, there is a possibility that these detection functions will be bypassed by more sophisticated attack methods. This is because, due to the characteristics of the iOS operating system, the code of the loaded dynamic library (dylib) is executed first when the app is launched. An attacker may distribute the code to patch a specific area of the executable file in such a dynamic library.
If this code patch occurs before AppSealing's detection logic is executed, the code that terminates the app has been removed, so even if a jailbreak is detected, the app will remain running.
Of course, not everyone can perform this type of attack easily, but since this type of attack has been confirmed by a group of hackers with specialized hacking knowledge, AppSealing provides an additional jailbreak detection method to overcome such an attack situation.

Since the characteristic of this attack method is to change the code of the running app in advance, no matter how strong detection logic is added to the AppSealing library itself, the situation in which the code is patched by the dynamic library is unavoidable. Therefore, the newly provided jailbreak detection function does not detect in the app, but in a way that rejects all services and actions, such as log-in in or accepting API calls, in the case of a terminal suspected of being jailbroken in the server linked to the app.
The basic method is to obtain server credentials from the app through the AppSealing interface, add them to the existing authentication parameters, and send them to the server.
This method cannot be applied for a client-only app that does not work with the server.

The following sections describe, with example code, how to obtain and validate server credentials.

## 6-2 iOS App Code

Additional process of your app needs verify the server credentials is to call a function in the AppSealing SDK to get the server credential string and send it to the server along with the existing authentication parameters.
Most apps that work with the server will go through a user authentication or login process, and in this process, the account information entered by the user will be transmitted to the server. You can add the server credential string to the parameters you send to the server.

The server credential string is obtained in the following way:

Simple UI code into 'ViewController.swift' for **Swift** project

```
func userLogin( userID: String, password: String ) -> Bool
{
    let inst: AppSealingInterface = AppSealingInterface();
    let appSealingCredential = String.init( cString: inst._GetEncryptedCredential() );
     // credential 값을 인증 정보와 함께 서버로 올린다
    let loginResult = processLogin( user: userID, pwd: password, credential: appSealingCredential );
    ...
    ...
}
```

Simple UI code into 'ViewController.mm' for **Objective-C** project

```
- (BOOL)userLogin:(NSString*)userID withPassword:(NSString*)password
{
    char _appSealingCredential[290] = { 0, };
    ObjC_GetEncryptedCredential( _appSealingCredential );
     // credential 값을 인증 정보와 함께 서버로 올린다
    BOOL loginResult = processLogin( userID, password, _appSealingCredential );
    ...
    ...
}
```

If your server fails to validate credential, you should also force the login to fail and the app to not proceed further. However, since code such as checking the login result and closing the app is likely to be tampered by an attacker, the best practice is configuring your server to deny service or response for any requests from that client after the server fails credential validation.
This will be discussed again in the next section.

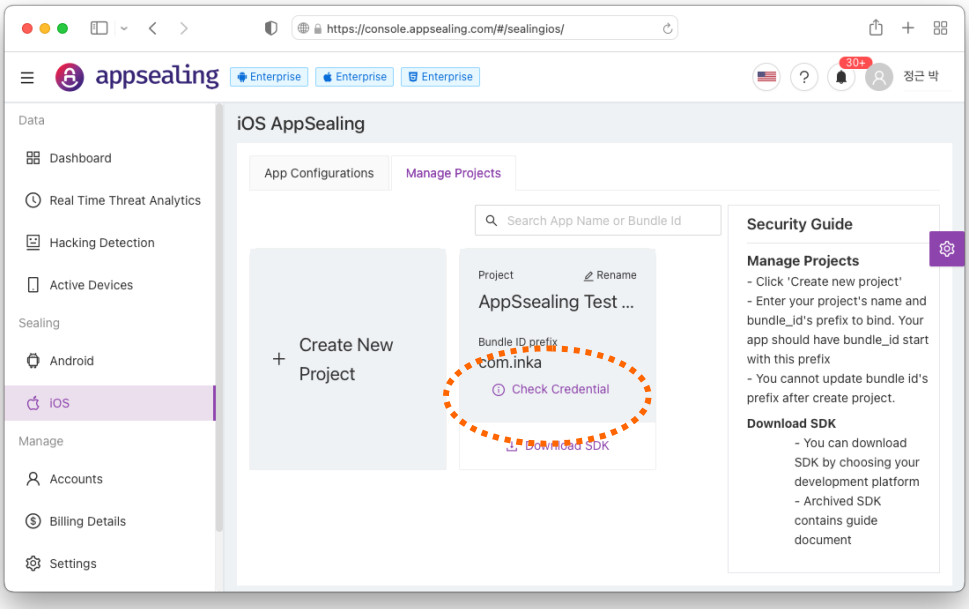
 

## 6-3 Verification at app server

The credential data (hex string) returned from the interface call to the AppSealing module is only valid when the security logic inside AppSealing is normally performed and no dangerous situation is detected in the device.
If code patch attack is made through the dynamic library or the security logic is bypassed by other methods, valid credential data will not be generated, so the server should verify this value and blocks the attack situation of the device.
The app server must check whether the credential value sent by the client (app) is correct, and if it is not correct, it must deny authentication (login) and then deny any services (API call) requested by that client.

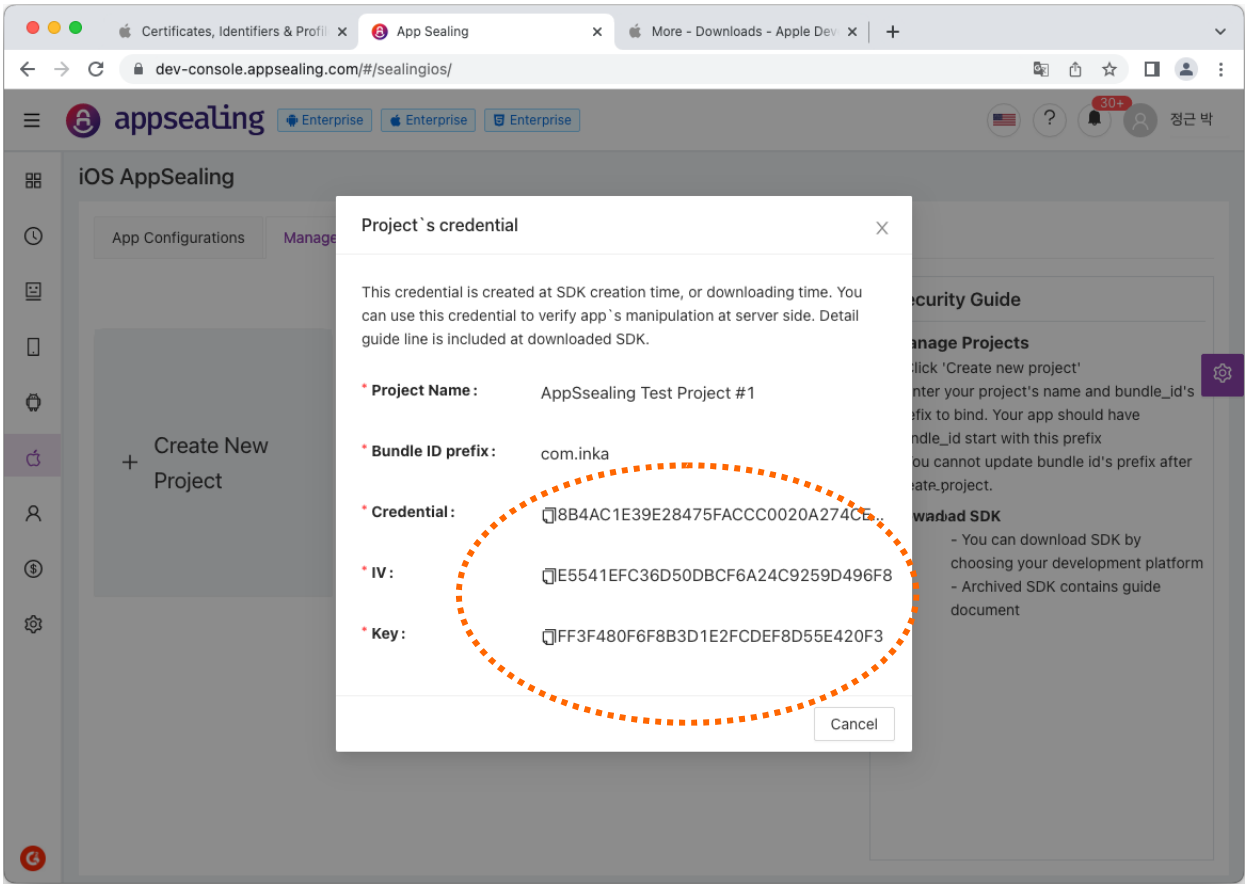
## [Preparation]

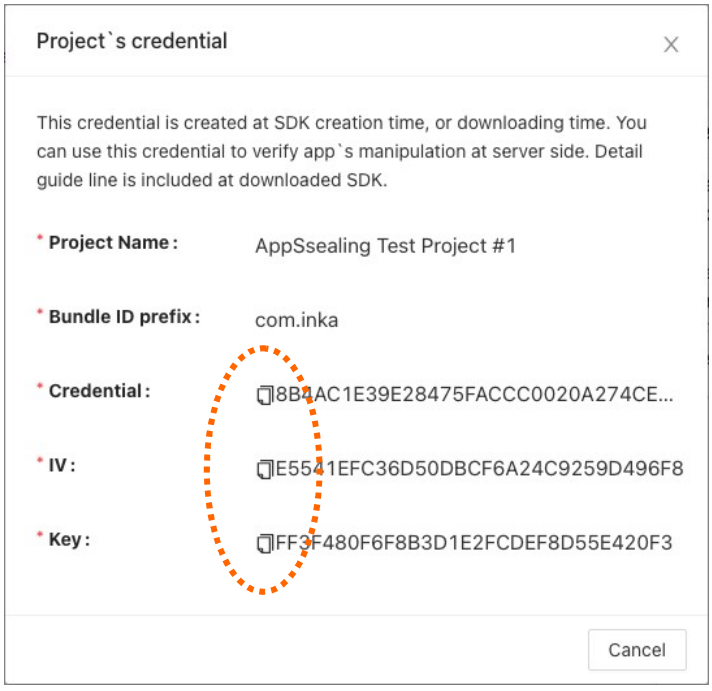
To verify credential data on the server, you need an AES Key and IV to decrypt the data sent from the client, and the original credential data to compare and verify.
All of these values can be acquired through the "Check Credential" button of the project in the ADC. Just copy the Hex string shown here and paste it into the example code and use it. First, connect to ADC as shown in the screen below and click the "Check Credential" button in the project box.
If you click the button, the following window is displayed, where you can check the Credential value and IV and key to be used for decryption. You can use the copy button to the left of the string to copy this value as it is, paste it into the server-side verification code and use it.



If you click the button, the following window is displayed, where you can check the Credential value and IV and AES key to be used for decryption. You can use the copy button to the left of the string to copy this value as it is, paste it into the server-side verification code and use it.



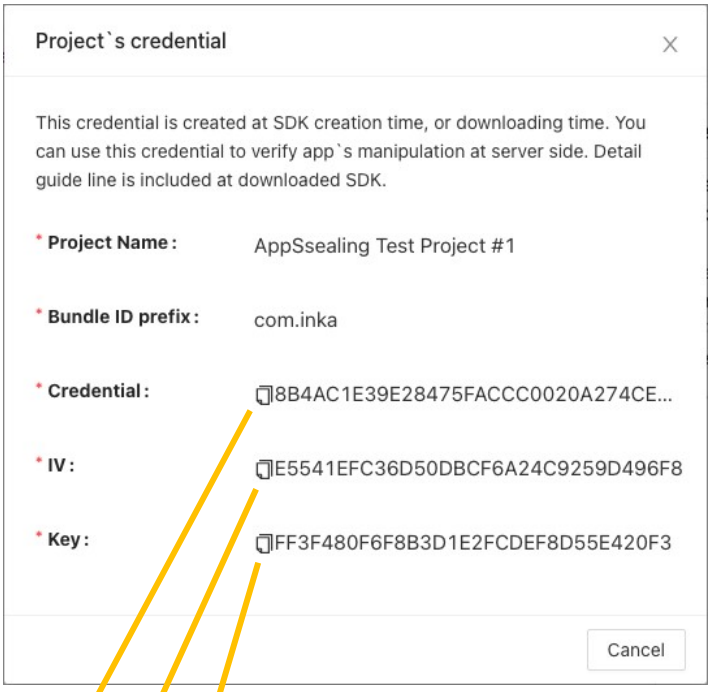



## [In case your server code is using Node.js/Javascript]

Add following code to your existing code and use it for credential data validation. Authenticity can be determined by passing the credential value sent as a parameter in the login or authentication function of the existing server code to the verfityAppSealingCredential function provided in the code below. (The code below is also included in the SDK as a file named appsealing_credential.js).

**** Note: In the code below, ORG_CREDENTIAL, AES_IV, and AES_KEY must be replaced with values obtained through the "Check Credential" function of the corresponding project in the ADC.** If you just take the example code and use it intactly, credential value will not be verified properly.




```
var crypto = require('crypto');

function verifyAppSealingCredential( credential )
{
    // Need to Change : Get From ADC (via 'Check Credential') ---------------------------------------
    const ORG_CREDENTIAL =
"572E0E1459453F2078D6576FF71ECD0DBCA0484430C7FA7FE45B788A37DE3A04204F5A55FEA83AC9AFBA2C688594F75A3828B23972DB34858EC4F6CC3202533E44121
E5F2614B227E18B6419A83810F7511D5E51FCACD5175A1CC550F83CB874A7378ACDAFE78EB2E329CD5D3C384061C4669674F1EE6B1B59FB7D91835DB7EE";

    const AES_IV  = "055772B7434A4174749A   09B1413472";
    const AES_KEY = "71CA94A64A4DEBF5566495AB03F6798F";
    //-----------------------------------------------------------------------------------------------
```

```
    // convert credential from hex string to byte array
    let decrypted_UTC = 0, decrpyted_buffer, aes_key2;

    // decrypt UTC
  try
    {
      const decipher = crypto.createDecipheriv( 'aes-128-ctr', Buffer.from( AES_KEY, 'hex' ), Buffer.from( AES_IV, 'hex' ));
      decrpyted_buffer = Buffer.concat( [decipher.update( credential.substr( 0, 32 ), 'hex' ), decipher.final()] );
              decrypted_UTC = decrpyted_buffer.slice( 0, 8 ).readUInt32LE();
  }
    catch( error )
    {
      throw error;
  }

    // verfity UTC with current time (+/-) 10sec
    const current_UTC = parseInt( Date.now() / 1000 ); // get current UTC in seconds
    if ( Math.abs( current_UTC - decrypted_UTC ) > 10 )
    {
              console.log( "Invalid UTC value has sent, deny login & all services for this client... " + Math.abs( current_UTC -
decrypted_UTC ) );
              return false;
    }
    console.log( "** UTC verified : " + decrypted_UTC + " (current = " + current_UTC + ", diff = " + Math.abs( current_UTC -
decrypted_UTC ) + ")" );

    // get AES KEY2
    aes_key2 = Buffer.concat( [new Uint8Array( decrpyted_buffer.slice( 0, 8 )), new
Uint8Array( Buffer.from( ORG_CREDENTIAL.substring( 52, 52 + 16 ), 'hex' ))] );
    for( let i = 0; i < 16; i++ )
              aes_key2[i] ^= Buffer.from( AES_IV.substring( i * 2, i * 2 + 2 ), 'hex' ).readUInt8();

    // decrypt credential
    let decrypted_credential = [];
  try
    {
      const decipher = crypto.createDecipheriv( 'aes-128-ctr', aes_key2, Buffer.from( AES_IV, 'hex' ));
      decrypted_credential = Buffer.concat( [decipher.update( credential.substr( 32, 256 ), 'hex' ), decipher.final()] );
  }
    catch( error )
    {
      throw error;
  }

    // verfity credential with CREDENTIAL(ADC)
    // return if fail
    if ( ORG_CREDENTIAL.toLowerCase() != decrypted_credential.toString( 'hex' ).toLowerCase() )
    {
              console.log( "Invalid credential value has sent, deny login & all services for this client..." );
              return false;
    }

    console.log( "** Credential verified : PASS" );
    return true;
}
```

## [In case your server code is using Java]

Add following code to your existing code and use it for credential data validation. Authenticity can be determined by passing the credential value sent as a parameter in the login or authentication function of the existing server code to the verfityAppSealingCredential function provided in the code below. (The code below is also included in the SDK as a file named appsealing_credential.java).

**** Note: In the code below, ORG_CREDENTIAL, AES_IV, and AES_KEY must be replaced with values obtained through the "Check Credential" function of the corresponding project in the ADC.** If you just take the example code and use it intactly, credential value will not be verified properly.

```
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.Arrays;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class AppSealingCredential
{
    private static boolean verifyAppSealingCredential( final String credential )
    {
        // Need to Change : Get From ADC (via 'Check Credential') ----------------------------------------
        final String ORG_CREDENTIAL =
"572E0E1459453F2078D6576FF71ECD0DBCA0484430C7FA7FE45B788A37DE3A04204F5A55FEA83AC9AFBA2C688594F75A3828B23972DB34858EC4F6CC
3202533E44121E5F2614B227E18B6419A83810F7511D5E51FCACD5175A1CC550F83CB874A7378ACDAFE78EB2E329CD5D3C384061C4669674F1EE6B1B5
9FB7D91835DB7EE";

        final String AES_IV  = "055772B7434A4174749AFE09B1413472";
        final String AES_KEY = "71CA94A64A4DEBF5566495AB03F6798F";
        //------------------------------------------------------------------------------------------------

        long decrypted_UTC = 0;
        byte[] decryptedUTC = new byte[8];

        // decrypt UTC
        try
        {
            SecretKeySpec key = new SecretKeySpec( DatatypeConverter.parseHexBinary( AES_KEY ), "AES" );
            IvParameterSpec ivSpec = new IvParameterSpec( DatatypeConverter.parseHexBinary( AES_IV ));

            Cipher cipher = Cipher.getInstance( "AES/CTR/NoPadding" );
            cipher.init( Cipher.DECRYPT_MODE, key, ivSpec );

            byte[] encryptedUTC = DatatypeConverter.parseHexBinary( credential.substring( 0, 32 ));
            System.arraycopy( cipher.doFinal( encryptedUTC ), 0, decryptedUTC, 0, 8 );
            System.out.println( DatatypeConverter.printHexBinary( decryptedUTC ));
```

```
            decrypted_UTC = ByteBuffer.wrap( decryptedUTC ).order( ByteOrder.LITTLE_ENDIAN ).getInt() & 0xFFFFFFFFL;
        }
        catch( Exception e )
        {
            System.out.println( "[Error] " + e.getLocalizedMessage() );
        }

        // verfity UTC with current time (+/-) 10sec
        long current_UTC = System.currentTimeMillis() / 1000; // get current UTC in seconds

        if ( Math.abs( current_UTC - decrypted_UTC ) > 10 )
        {
            System.out.println( "Invalid UTC value has sent, deny login & all services for this client..." );
            return false;
        }
        System.out.println( "** UTC verified : " + decrypted_UTC + " (current = " + current_UTC + ", diff = " +
Math.abs( current_UTC - decrypted_UTC ) + ")" );

        // get AES KEY2
        byte[] aes_key2 = Arrays.copyOf( decryptedUTC, 16 );
        byte[] xor = DatatypeConverter.parseHexBinary( ORG_CREDENTIAL.substring( 52, 52 + 16 ));
        System.arraycopy( xor, 0, aes_key2, decryptedUTC.length, xor.length );

        for( int i = 0; i < 16; i++ )
            aes_key2[i] ^= ( byte )DatatypeConverter.parseHexBinary( AES_IV.substring( i * 2, i * 2 + 2 ))[0];
        System.out.println( DatatypeConverter.printHexBinary( aes_key2 ));

        // decrypt credential
        byte[] decrypted_credential = null;
        try
        {
            SecretKeySpec key = new SecretKeySpec( aes_key2, "AES" );
            IvParameterSpec ivSpec = new IvParameterSpec( DatatypeConverter.parseHexBinary( AES_IV ));

            Cipher cipher = Cipher.getInstance( "AES/CTR/NoPadding" );
            cipher.init( Cipher.DECRYPT_MODE, key, ivSpec );

            System.out.println( "########### " + credential.substring( 32, 32 + 256 ) );
            byte[] encrypted_credential = DatatypeConverter.parseHexBinary( credential.substring( 32, 32 + 256 ));
            decrypted_credential = cipher.doFinal( encrypted_credential );
        }
        catch( Exception e )
        {
            System.out.println( "[Error] " + e.getLocalizedMessage() );
        }

        // verfity credential with CREDENTIAL(ADC)
        // return if fail
        if ( !ORG_CREDENTIAL.equalsIgnoreCase( DatatypeConverter.printHexBinary( decrypted_credential )))
        {
            System.out.println( "Invalid credential value has sent, deny login & all services for this client..." );
            return false;
        }

        System.out.println( "** Credential verified : PASS" );
        return true;
    }
}
```

## [In case your server code is using ASP.NET / C#]

Add following code to your existing code and use it for credential data validation. Authenticity can be determined by passing the credential value sent as a parameter in the login or authentication function of the existing server code to the verfityAppSealingCredential function provided in the code below. (The code below is also included in the SDK as a file named appsealing_credential.cs, because C# does not support AES CTR mode, you must include the AesCtrTransform function for CTR processing as shown in the code below.)

**** Note: In the code below, ORG_CREDENTIAL, AES_IV, and AES_KEY must be replaced with values obtained through the "Check Credential" function of the corresponding project in the ADC.** If you just take the example code and use it intactly, credential value will not be verified properly.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace AppSealingSDK
{
    class AppSealingCredential
    {
        public static void AesCtrTransform( byte[] key, byte[] salt, Stream inputStream, Stream outputStream )
        {
            SymmetricAlgorithm aes = new AesManaged { Mode = CipherMode.ECB, Padding = PaddingMode.None };

            int blockSize = aes.BlockSize / 8;
            if ( salt.Length != blockSize )
                throw new ArgumentException( "Salt size must be same as block size " + $"(actual: {salt.Length}, expected:
{blockSize})" );

            byte[] counter = ( byte[] )salt.Clone();

            Queue<byte> xorMask = new Queue<byte>();
            var zeroIv = new byte[blockSize];
            ICryptoTransform counterEncryptor = aes.CreateEncryptor( key, zeroIv );

            int b;
            while(( b = inputStream.ReadByte()) != -1 )
            {
                if ( xorMask.Count == 0 )
                {
                    var counterModeBlock = new byte[blockSize];
```

```
                counterEncryptor.TransformBlock( counter, 0, counter.Length, counterModeBlock, 0 );

                for( var i2 = counter.Length - 1; i2 >= 0; i2-- )
                {
                    if ( ++counter[i2] != 0 )
                        break;
                }
                foreach( var b2 in counterModeBlock )
                    xorMask.Enqueue( b2 );
            }
            var mask = xorMask.Dequeue();
            outputStream.WriteByte(( byte )((( byte )b ) ^ mask ));
        }
    }
    public static byte[] StringToByteArray( String hex )
    {
        return Enumerable.Range( 0, hex.Length / 2 ).Select( x => Convert.ToByte( hex.Substring( x * 2, 2 ),
16 )).ToArray();
    }
    public static bool VerifyAppSealingCredential( String credential )
    {
        // Need to Change : Get From ADC (via 'Check Credential') ---------------------------------------
        const String ORG_CREDENTIAL =
"572E0E1459453F2078D6576FF71ECD0DBCA0484430C7FA7FE45B788A37DE3A04204F5A55FEA83AC9AFBA2C688594F75A3828B23972DB34858EC4F6CC
3202533E44121E5F2614B227E18B6419A83810F7511D5E51FCACD5175A1CC550F83CB874A7378ACDAFE78EB2E329CD5D3C384061C4669674F1EE6B1B5
9FB7D91835DB7EE";

        const String AES_IV  = "055772B7434A4174749AFE09B1413472";
        const String AES_KEY = "71CA94A64A4DEBF5566495AB03F6798F";
        //-----------------------------------------------------------------------------------------------

        long decrypted_UTC = 0;
        byte[] decrypted_buffer = new byte[8];

        // decrypt UTC
        try
        {
            byte[] encrypted_UTC = StringToByteArray( credential.Substring( 0, 32 ));
            byte[] result = new byte[16];
            AesCtrTransform( StringToByteArray( AES_KEY ), StringToByteArray( AES_IV ), new
MemoryStream( encrypted_UTC ), new MemoryStream( result ));
            Array.Copy( result, 0, decrypted_buffer, 0, 8 );
            decrypted_UTC = BitConverter.ToInt64( decrypted_buffer, 0 );
            System.Console.WriteLine( "** UTC = " + decrypted_UTC );
        }
        catch(Exception e )
        {
            System.Console.WriteLine( "[Error] " + e.Message );
        }

        // verfity UTC with current time (+/-) 10sec
        long current_UTC = DateTimeOffset.Now.ToUnixTimeMilliseconds() / 1000;// get current UTC in seconds
        if ( Math.Abs( current_UTC - decrypted_UTC ) > 10 )
        {
            System.Console.WriteLine( "Invalid UTC value has sent, deny login & all services for this client..." );
            return false;
        }
        System.Console.WriteLine( "** UTC verified : " + decrypted_UTC + " (current = " + current_UTC + ", diff = " +
Math.Abs(current_UTC - decrypted_UTC ) + ")" );
```

```
            // get AES KEY2
            byte[] aes_key2 = new byte[16];
            Array.Copy( decrypted_buffer, 0, aes_key2, 0, 8 );
            byte[] xor = StringToByteArray( ORG_CREDENTIAL.Substring( 52, 16 ));
            Array.Copy( xor, 0, aes_key2, decrypted_buffer.Length, xor.Length );

            for(int i = 0; i< 16; i++ )
                aes_key2[i] ^= ( byte )StringToByteArray( AES_IV.Substring( i* 2, 2 ))[0];

            // decrypt credential
            byte[] decrypted_credential = null;
            try
            {
                byte[] encrypted_credential = StringToByteArray( credential.Substring( 32, 256 ));
                decrypted_credential = new byte[encrypted_credential.Length];
                AesCtrTransform( aes_key2, StringToByteArray( AES_IV ), new MemoryStream( encrypted_credential ), new
MemoryStream( decrypted_credential ));
            }
            catch(Exception e )
            {
                System.Console.WriteLine( "[Error] " + e.Message );
            }

            // verfity credential with CREDENTIAL(ADC)
            // return if fail
            if ( !ORG_CREDENTIAL.Equals( BitConverter.ToString( decrypted_credential ).Replace( "-", "" ),
StringComparison.InvariantCultureIgnoreCase ))
            {
                System.Console.WriteLine( "Invalid credential value has sent, deny login & all services for this
client..." );
                return false;
            }

            System.Console.WriteLine( "** Credential verified : PASS" );
            return true;
        }
}
```

## [In case your server code is using python]

Add following code to your existing code and use it for credential data validation. Authenticity can be determined by passing the credential value sent as a parameter in the login or authentication function of the existing server code to the verfityAppSealingCredential function provided in the code below. (The code below is also included in the SDK as a file named appsealing_credential.py)

**** Note: In the code below, ORG_CREDENTIAL, AES_IV, and AES_KEY must be replaced with values obtained through the "Check Credential" function of the corresponding project in the ADC.** If you just take the example code and use it intactly, credential value will not be verified properly.

```
from Crypto.Cipher import AES
from Crypto.Util import Counter
import time

#============================================================
def verifyAppSealingCredential( credential ):

    # Need to Change : Get From ADC (via 'Check Credential') ------------------------------------------
    ORG_CREDENTIAL =
"572E0E1459453F2078D6576FF71ECD0DBCA0484430C7FA7FE45B788A37DE3A04204F5A55FEA83AC9AFBA2C688594F75A3828B23972DB34858EC4F6CC
3202533E44121E5F2614B227E18B6419A83810F7511D5E51FCACD5175A1CC550F83CB874A7378ACDAFE78EB2E329CD5D3C384061C4669674F1EE6B1B5
9FB7D91835DB7EE"
    AES_IV  = "055772B7434A4174749AFE09B1413472"
    AES_KEY = "71CA94A64A4DEBF5566495AB03F6798F"
    #-------------------------------------------------------------------------------------------------

          # decrypt UTC
    try:
        counter = Counter.new( 128, initial_value = int.from_bytes( bytes.fromhex( AES_IV ), "big" ))
        cipher = AES.new( bytes.fromhex( AES_KEY ), AES.MODE_CTR, counter = counter )
        descrypted_buffer = cipher.decrypt( bytes.fromhex( credential[:32] ))

        decrypted_UTC = int.from_bytes( descrypted_buffer[:8], "little" )
    except Exception as e:
        print( '[Error] ', e )
        return 0;

    # verfity UTC with current time (+/-) 10sec
    current_UTC = round( time.time() * 1000 );     # get current UTC in seconds

    if abs( current_UTC - decrypted_UTC ) > 10:
        print( "Invalid UTC value has sent, deny login & all services for this client..." )
        #return 0

    print( "** UTC verified : ", decrypted_UTC, " (current = ", current_UTC, ", diff = ", abs( current_UTC -
decrypted_UTC ), ")" )
```

```
    # get AES KEY2
    aes_key2 = bytearray( 16 )
    aes_key2[0:8] = descrypted_buffer[0:8]
    aes_key2[8:8] = bytes.fromhex( ORG_CREDENTIAL[52:68] )
    print( aes_key2[:16].hex() )

    for i in range( 0, 16 ):
        aes_key2[i] ^= bytes.fromhex( AES_IV[i * 2:i * 2 + 2] )[0]
    print( aes_key2[:16].hex() )

    # decrypt credential
    try:
        counter2 = Counter.new( 128, initial_value = int.from_bytes( bytes.fromhex( AES_IV ), "big" ))
        cipher2 = AES.new( aes_key2[:16], AES.MODE_CTR, counter = counter2 )
        print( credential[32:288] )
        decrypted_credential = cipher2.decrypt( bytes.fromhex( credential[32:288] ))
    except Exception as e:
        print( '[Error] ', e )
        return 0;

    # verfity credential with CREDENTIAL(ADC)
    # return if fail
    if ORG_CREDENTIAL.casefold() != decrypted_credential.hex().casefold():
        print( "Invalid credential value has sent, deny login & all services for this client..." )
        return 0

    print( "** Credential verified : PASS" )
    return 1
```

## [In case your server code is using ruby script]

Add following code to your existing code and use it for credential data validation. Authenticity can be determined by passing the credential value sent as a parameter in the login or authentication function of the existing server code to the verfityAppSealingCredential function provided in the code below. (The code below is also included in the SDK as a file named appsealing_credential.rb)

**** Note: In the code below, ORG_CREDENTIAL, AES_IV, and AES_KEY must be replaced with values obtained through the "Check Credential" function of the corresponding project in the ADC.** If you just take the example code and use it intactly, credential value will not be verified properly.

```
require 'securerandom'
require 'net/https'
require 'json'

# Need to Change : Get From ADC (via 'Check Credential') ------------------------------------------
ORG_CREDENTIAL =
"572E0E1459453F2078D6576FF71ECD0DBCA0484430C7FA7FE45B788A37DE3A04204F5A55FEA83AC9AFBA2C688594F75A3828B23972DB34858EC4F6CC
3202533E44121E5F2614B227E18B6419A83810F7511D5E51FCACD5175A1CC550F83CB874A7378ACDAFE78EB2E329CD5D3C384061C4669674F1EE6B1B5
9FB7D91835DB7EE"
AES_IV  = "055772B7434A4174749AFE09B1413472"
AES_KEY = "71CA94A64A4DEBF5566495AB03F6798F"
#-------------------------------------------------------------------------------------------------

def verifyAppSealingCredential( credential )

    # decrypt UTC
    begin
        aes = OpenSSL::Cipher::AES.new( "128-CTR" )
        aes.decrypt
        aes.key = [AES_KEY].pack( 'H*' )
        aes.iv = [AES_IV].pack( 'H*' )
        decrypted_buffer = aes.update( [credential[0..31]].pack( 'H*' )) + aes.final

        decrypted_UTC = decrypted_buffer.slice( 0, 8 ).unpack( 'V' ).first
    rescue => e
        puts '[Error] ' + e.to_s
        return 0
           end

    # verfity UTC with current time (+/-) 10sec
    current_UTC = Time.now.strftime( '%s%L' ).to_i;          # get current UTC in seconds

    if ( current_UTC - decrypted_UTC ).abs > 10
        puts "Invalid UTC value has sent, deny login & all services for this client..."
        return 0
    end
```

```
    puts "** UTC verified : " + decrypted_UTC.to_s + " (current = " + current_UTC.to_s + ", diff = " + ( current_UTC -
decrypted_UTC ).abs.to_s + ")"

    # get AES KEY2
    aes_key2 = decrypted_buffer.bytes.slice( 0, 8 ) + [ORG_CREDENTIAL[52..67]].pack( 'H*' ).bytes
    for i in 0..15 do
        aes_key2[i] ^= [AES_IV[i * 2..i * 2 + 1]].pack( 'H*' ).bytes[0]
    end

    # decrypt credential
    begin
        aes = OpenSSL::Cipher::AES.new( "128-CTR" )
        aes.decrypt
        aes.key = aes_key2.pack( 'C*' )
        aes.iv = [AES_IV].pack( 'H*' )
        decrypted_credential = aes.update( [credential[32..287]].pack( 'H*' )) + aes.final
    rescue => e
        puts '[Error] ' + e.to_s
        return 0
           end

    # verfity credential with CREDENTIAL(ADC)
    # return if fail
    if ORG_CREDENTIAL.casecmp( decrypted_credential.unpack( 'H*' ).first ) != 0
        print( "Invalid credential value has sent, deny login & all services for this client..." )
        return 0
    end

    print( "** Credential verified : PASS" )
    return 1

end
```

## [In case your server code is using C++]

Add following code to your existing code and use it for credential data validation. Authenticity can be determined by passing the credential value sent as a parameter in the login or authentication function of the existing server code to the verfityAppSealingCredential function provided in the code below. (The code below is also included in the SDK as a file named appsealing_credential.cpp with aes.hpp/aes.cpp)

**** Note: In the code below, ORG_CREDENTIAL, AES_IV, and AES_KEY must be replaced with values obtained through the "Check Credential" function of the corresponding project in the ADC.** If you just take the example code and use it intactly, credential value will not be verified properly.

```
#include <iostream>
#include <vector>
#include <time.h>
#include "aes.cpp"

typedef std::vector<unsigned char> bytes;
bytes HexToBytes( const std::string& hex )
{
    std::vector<unsigned char> bytes;
    for( unsigned int i = 0; i < hex.length(); i += 2 )
    {
        std::string byteString = hex.substr( i, 2 );
        unsigned char byte = ( unsigned char )strtol( byteString.c_str(), NULL, 16 );
        bytes.push_back( byte );
    }
    return bytes;
}

static bool verifyAppSealingCredential( const char* credential )
{
    // Need to Change : Get From ADC (via 'Check Credential') ------------------------------------------
    const char* ORG_CREDENTIAL =
"572E0E1459453F2078D6576FF71ECD0DBCA0484430C7FA7FE45B788A37DE3A04204F5A55FEA83AC9AFBA2C688594F75A3828B23972DB34858EC4F6CC
3202533E44121E5F2614B227E18B6419A83810F7511D5E51FCACD5175A1CC550F83CB874A7378ACDAFE78EB2E329CD5D3C384061C4669674F1EE6B1B5
9FB7D91835DB7EE";
    const char* AES_IV  = "055772B7434A4174749AFE09B1413472";
    const char* AES_KEY = "71CA94A64A4DEBF5566495AB03F6798F";
    //-------------------------------------------------------------------------------------------------

    unsigned char decrypted_buffer[16], decrypted_credential[128];

    // decrypt UTC
    struct AES_ctx ctx;
    try
    {
```

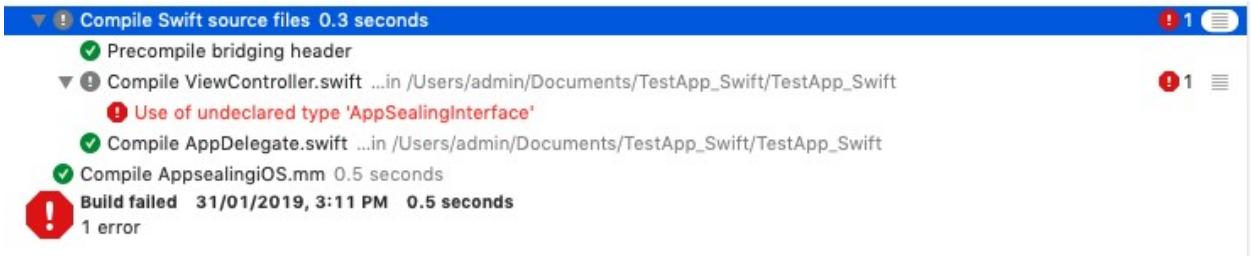
 

```
        memcpy( decrypted_buffer, HexToBytes( std::string( credential ).substr( 0, 32 )).data(), 16 );

        AES_init_ctx_iv( &ctx, HexToBytes( AES_KEY ).data(), HexToBytes( AES_IV ).data() );
        AES_CTR_xcrypt_buffer( &ctx, decrypted_buffer, 16 );

        long decrypted_UTC = *(( long* )decrypted_buffer );

        std::cout << "** UTC = " << decrypted_UTC << "\n";

        // verfity UTC with current time (+/-) 10sec
        time_t current_UTC = time( 0 );
        current_UTC = 1664439768;//REMOVE
        if ( abs( current_UTC - decrypted_UTC ) > 10 )
        {
            std::cout << "Invalid UTC value has sent, deny login & all services for this client...\n";
            return false;
        }
        std::cout << "** UTC verified : " << decrypted_UTC << " (current = " << current_UTC << ", diff = " <<
abs(current_UTC - decrypted_UTC ) << ")\n";

        // get AES KEY2
        unsigned char aes_key2[16];
        memcpy( aes_key2, decrypted_buffer, 8 );
        bytes XOR = HexToBytes( std::string( ORG_CREDENTIAL ).substr( 52, 16 ));
        memcpy( aes_key2 + 8, XOR.data(), XOR.size() );

        for( int i = 0; i < 16; i++ )
            aes_key2[i] ^= HexToBytes( std::string( AES_IV ).substr( i * 2, 2 )).data()[0];

        // decrypt credential
        memcpy( decrypted_credential, HexToBytes( std::string( credential ).substr( 32, 256 )).data(), 128 );

        AES_init_ctx_iv( &ctx, aes_key2, HexToBytes( AES_IV ).data() );
        AES_CTR_xcrypt_buffer( &ctx, decrypted_credential, 128 );
    }
    catch( const std::out_of_range& e )
    {
        std::cout << "pos exceeds string size\n";
    }

    // verfity credential with CREDENTIAL(ADC)
    // return if fail
    if ( memcmp( HexToBytes( ORG_CREDENTIAL ).data(), decrypted_credential, 128 ) != 0 )
    {
        std::cout << "Invalid credential value has sent, deny login & all services for this client...";
        return false;
    }
    std::cout << "** Credential verified : PASS";
}
```

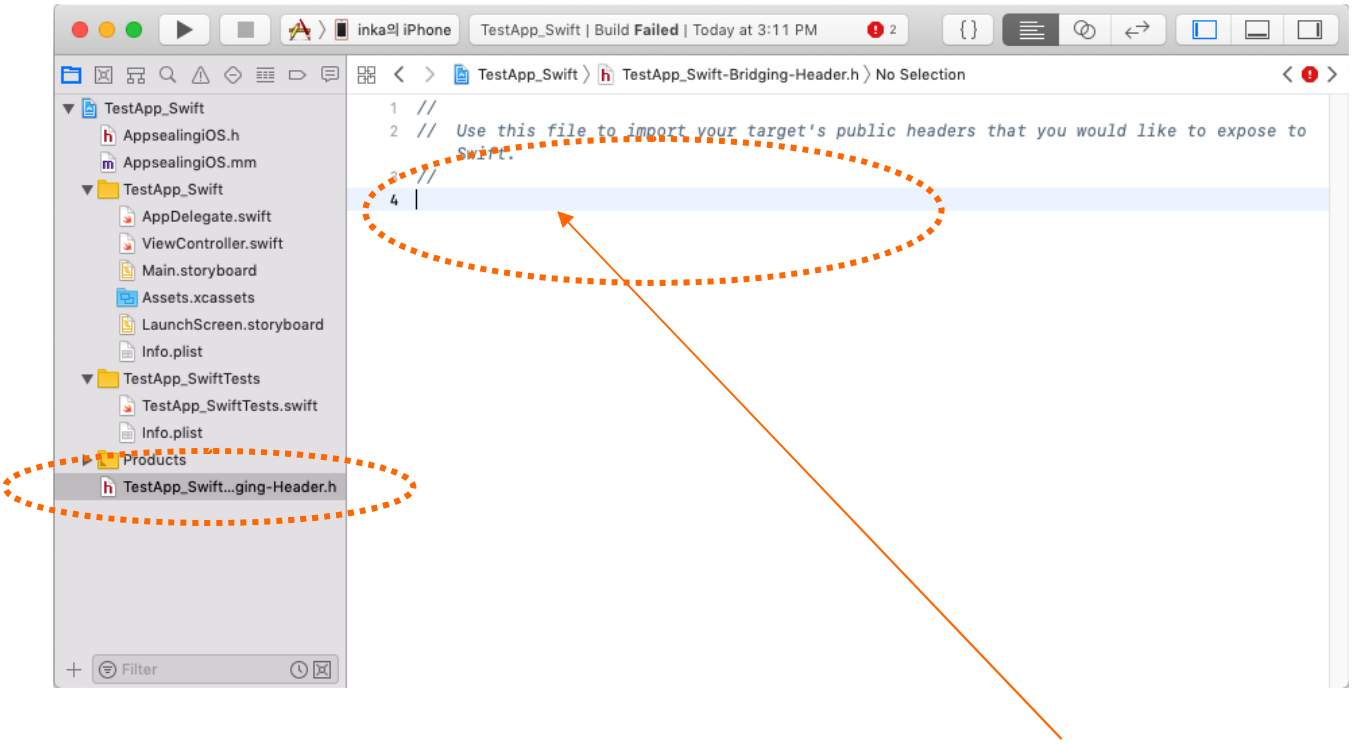
 

# Part 7. Troubleshooting

## 7-1 Xcode Build error (1) **Use of undeclared type 'AppSealingInterface'**




You will gen an error message like above when you omit or miss-typed '#import "AppsealingiOS.h"' sentence in Bridging Header file.

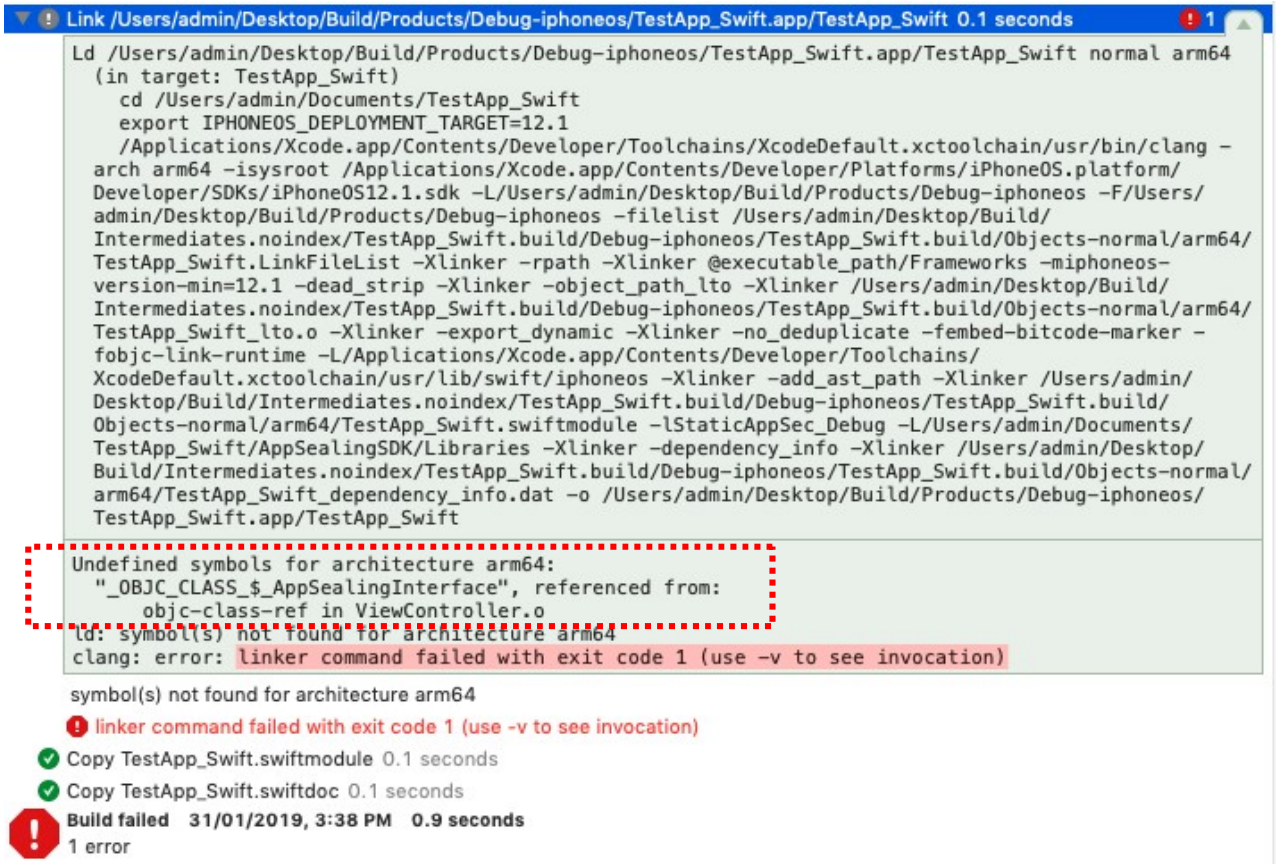



Select "TestApp_Swift-Bridging-Header.h" and append '**#import "AppsealingiOS.h**"' at the end of document. Be sure that "AppsealingiOS.h" file exist in the designated folder (Xcode project/AppSealingSDK/Libraries/AppsealingiOS.h)

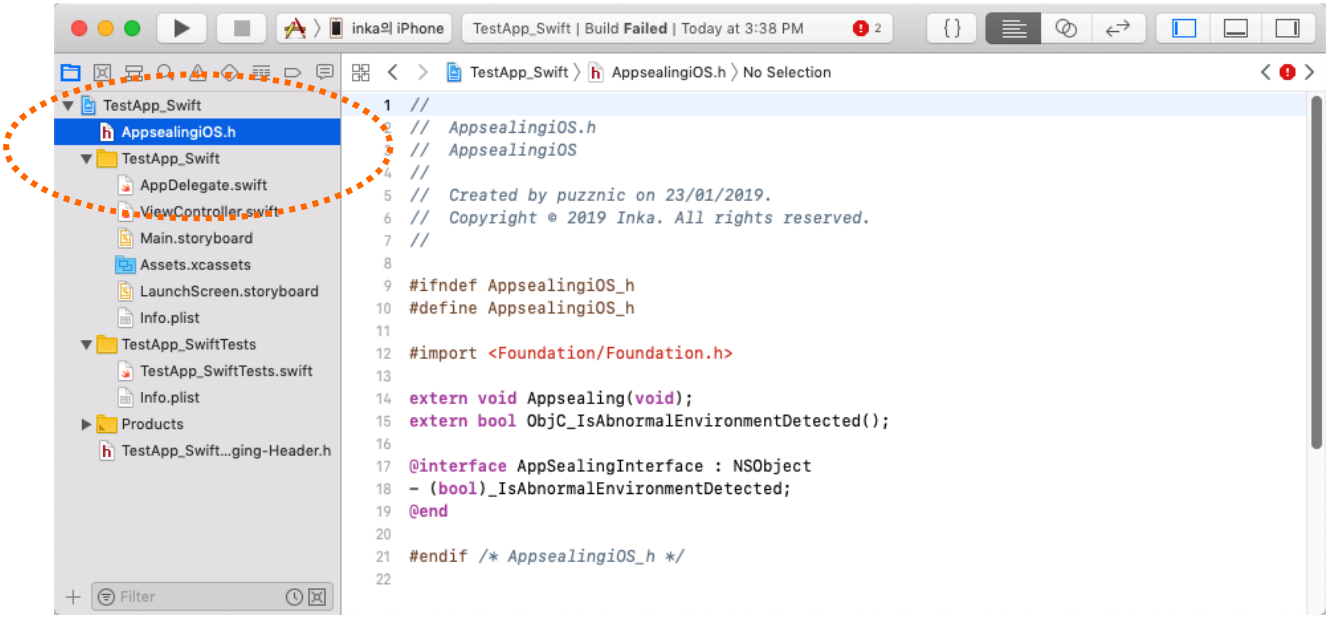
Also, the "AppsealingiOS.h" file must be included in your project with "AppsealingiOS.mm" file.

## 7-2 Xcode Build error (2) **Undefined symbols for architecture arm64: "_OBJC_CLASS_$_AppSealingInterface"**

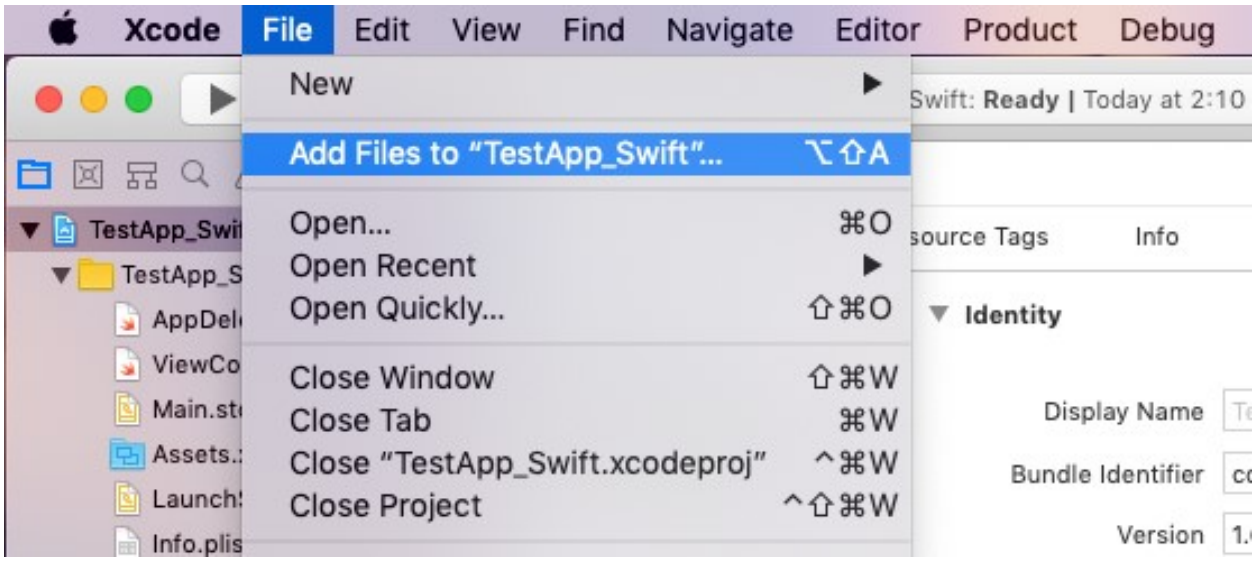


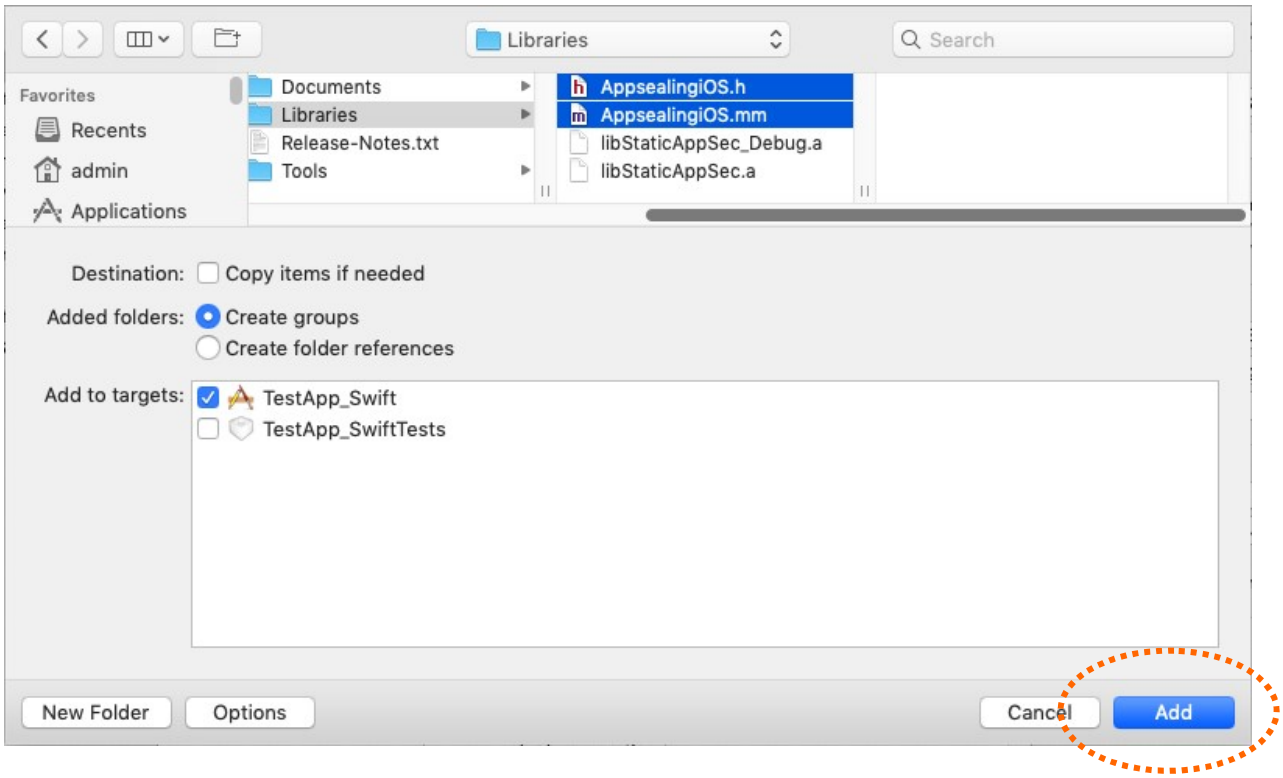

This linker error occurs when the "AppsealingiOS.mm" is not included in your project. Check project tree whether the file has added or not.



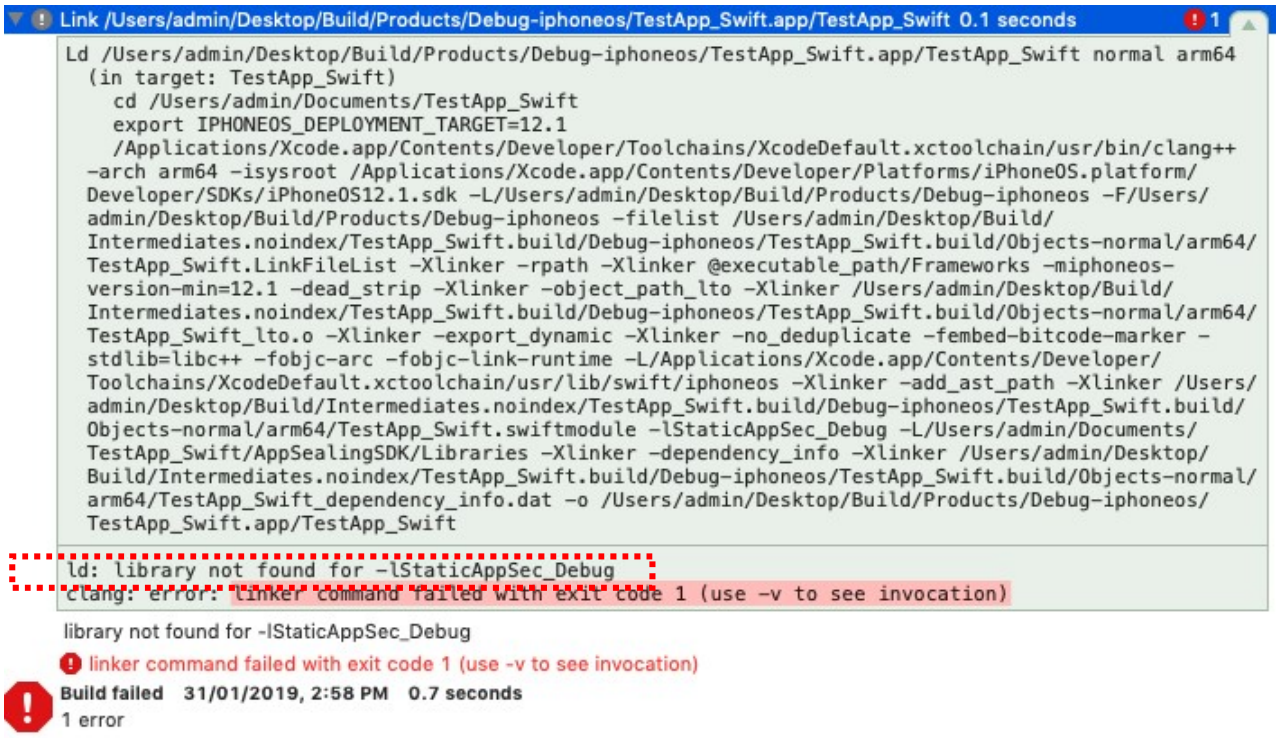
If the "AppsealingiOS.mm" file is not included in your project, perform "File > Add Files to "TestApp_Swift" ... " menu action.



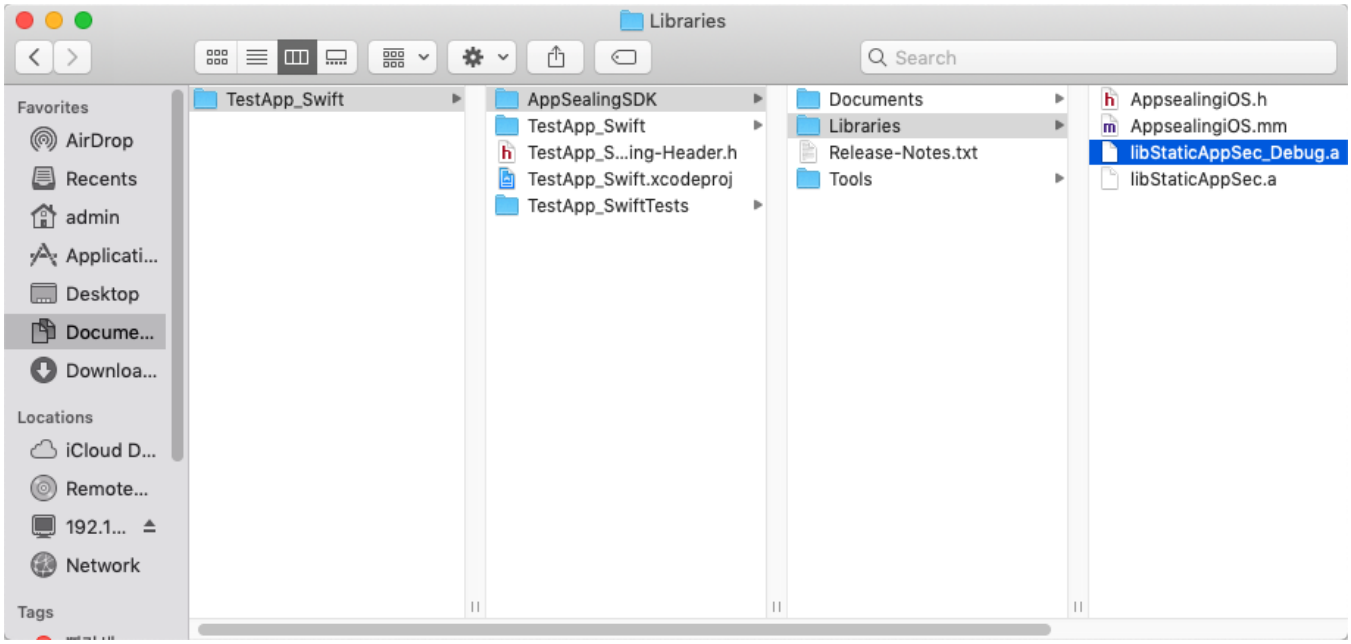

 

## 7-3 Xcode Build error (3) **ld: library not found for -lStaticAppSec_Debug**




This linker error occurs when the libStaticAppSec_Debug.a file is not exist in the designated folder or corrupted. The libStaticAppSec_Debug.a file should be exist in following path.

*"TestApp_Swift (Project folder) > AppSealingSDK > Libraries > libStaticAppSec_Debug.a"*



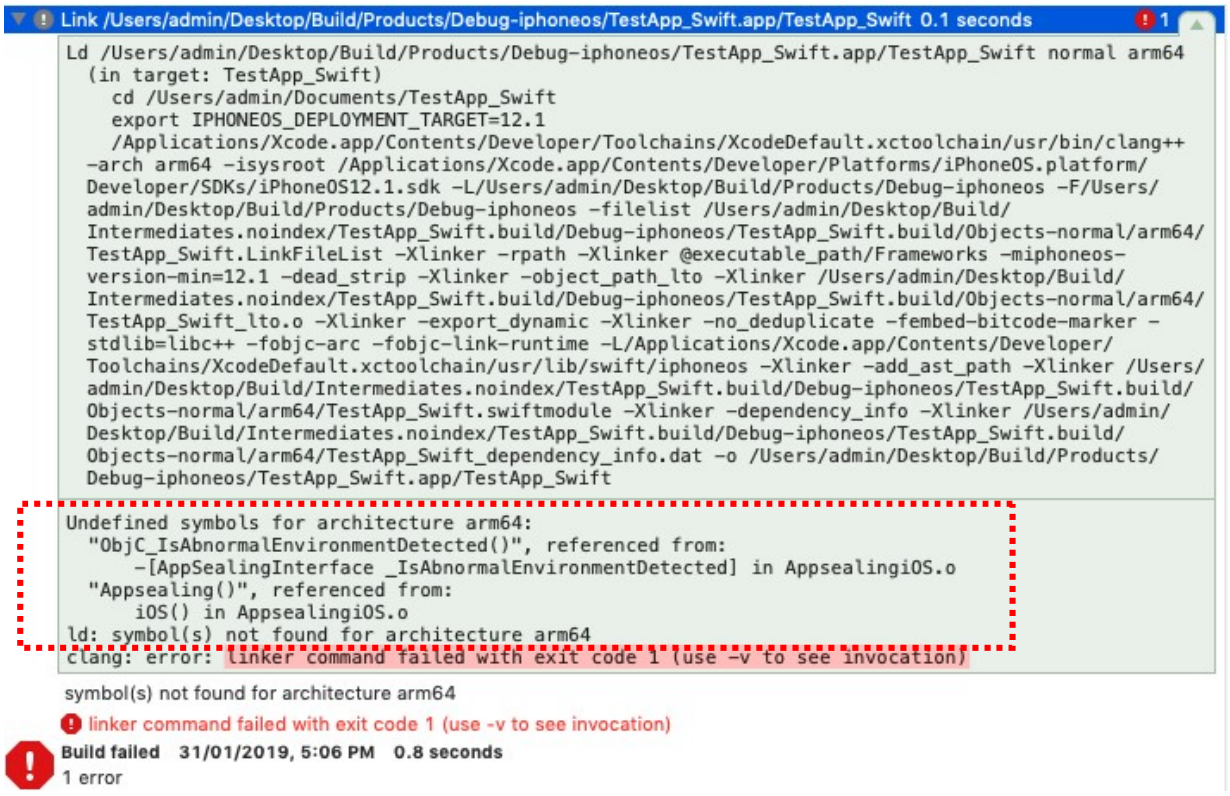
If the file is missing (or maybe corrupted) try to re-download or re-expand AppSealingSDK zip file.

This step is also applied to the linker error that says "ld: library not found for -lStaticAppSec" by just replacing "lStaticAppSec_Debug" to "lStaticAppSec". For Release builds, the file “libStaticAppSec.a” must exist in the correct location.

*"TestApp_Swift (Project folder) > AppSealingSDK > Libraries > libStaticAppSec.a"*

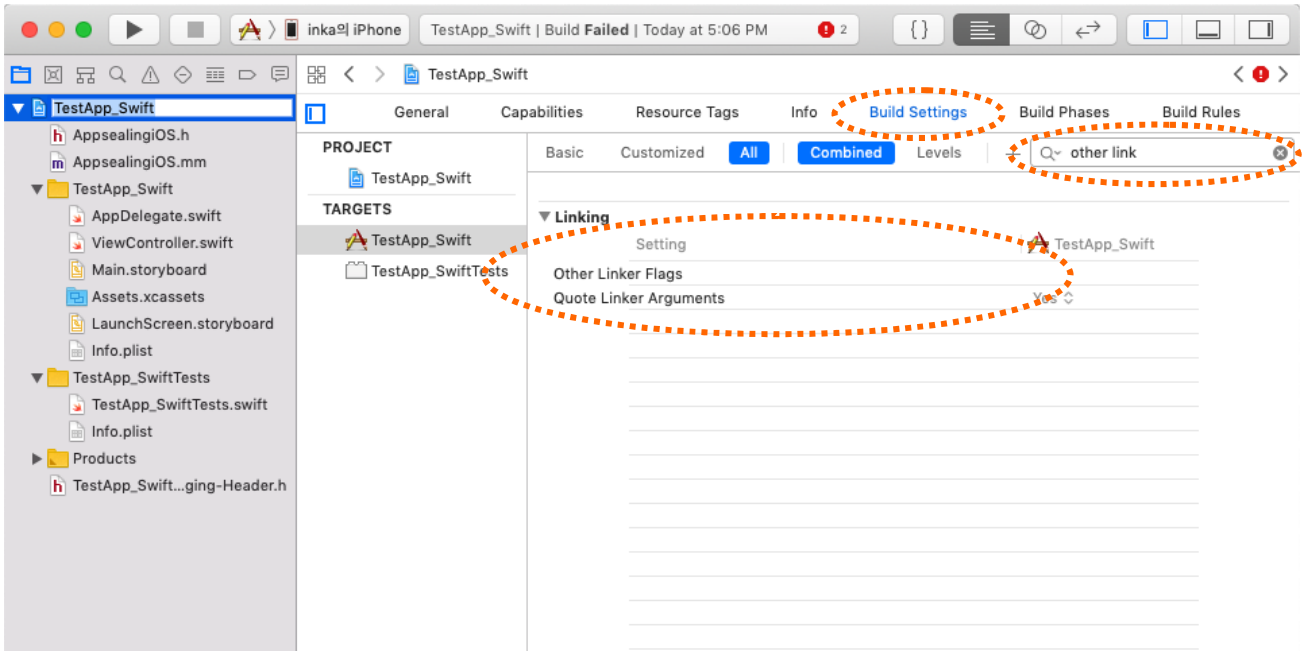
 

## 7-4 Xcode Build error (4) **Undefined symbols for architecture arm64: "ObjC_IsAbnormalEnvironmentDetected()", "Appsealing()"**




This linker error occurs when the value of "Other Linker Flags" field within "Build Settings" configuration section is missing or invalid.

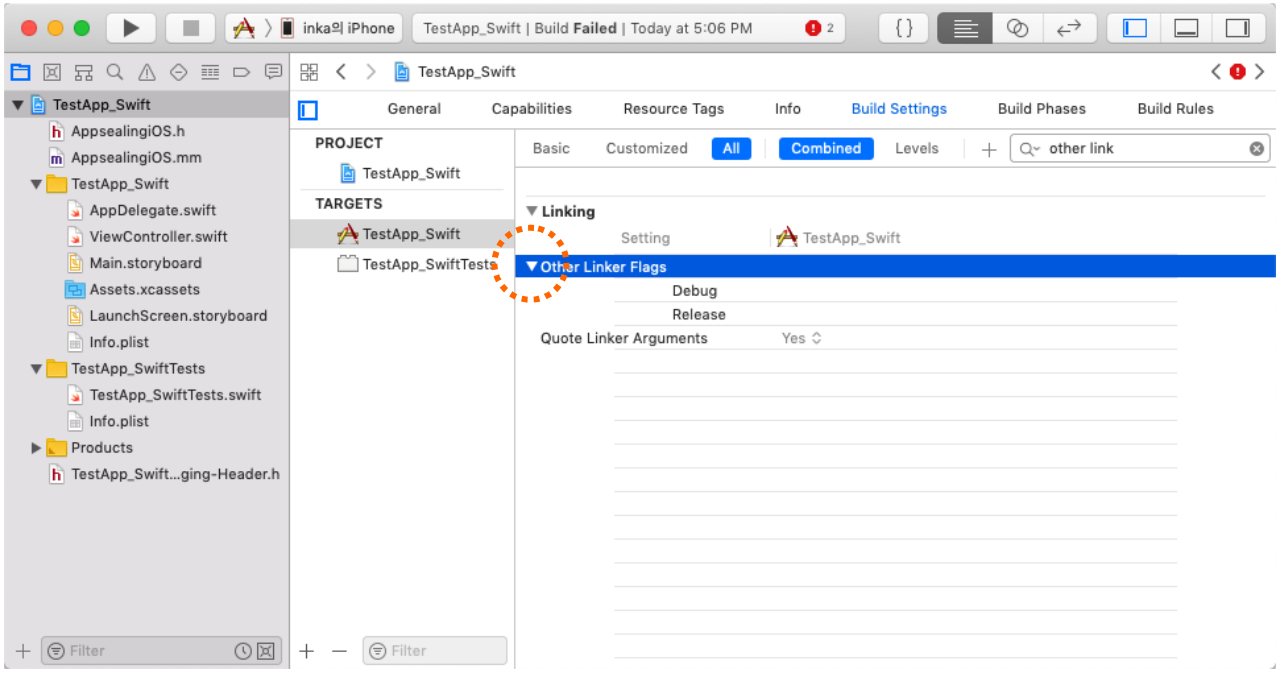
First, check "Build Settings" for "**Other Linker Flags**" field value.



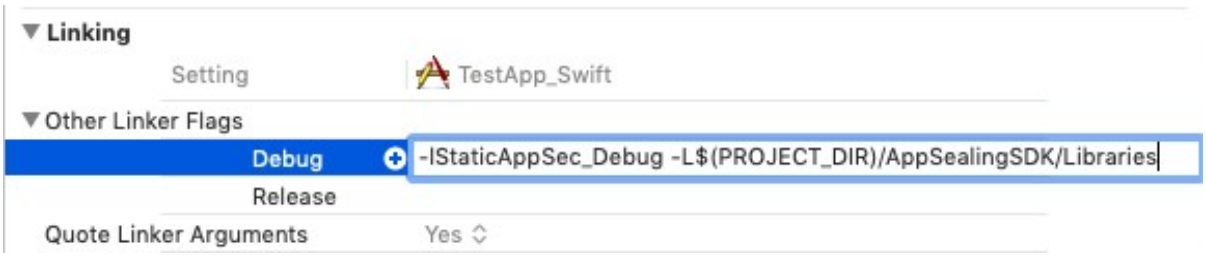
Now, restore the "Other Linker Flags" settings by following steps.

1) Clear the setting value : select "Other Linker Flags" row and press 'Delete' key.
2) Expand "Other Linker Flags" by clicking the triangle icon left to "Other Linker Flags"




3) Select "Debug" item and click to edit/insert value, then type in following text.

*-lStaticAppSec_Debug -L$(PROJECT_DIR)/AppSealingSDK/Libraries*

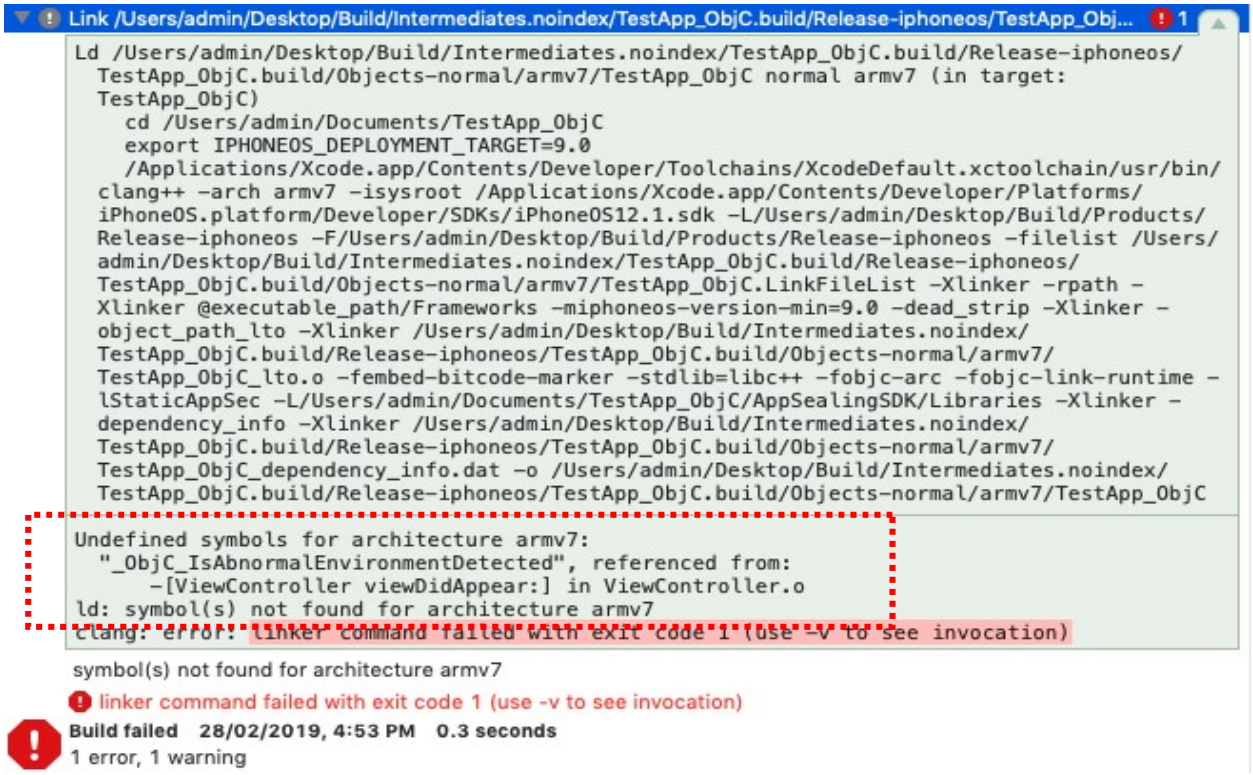



4) Select "Release" item and click to edit/insert value, then type in following text.

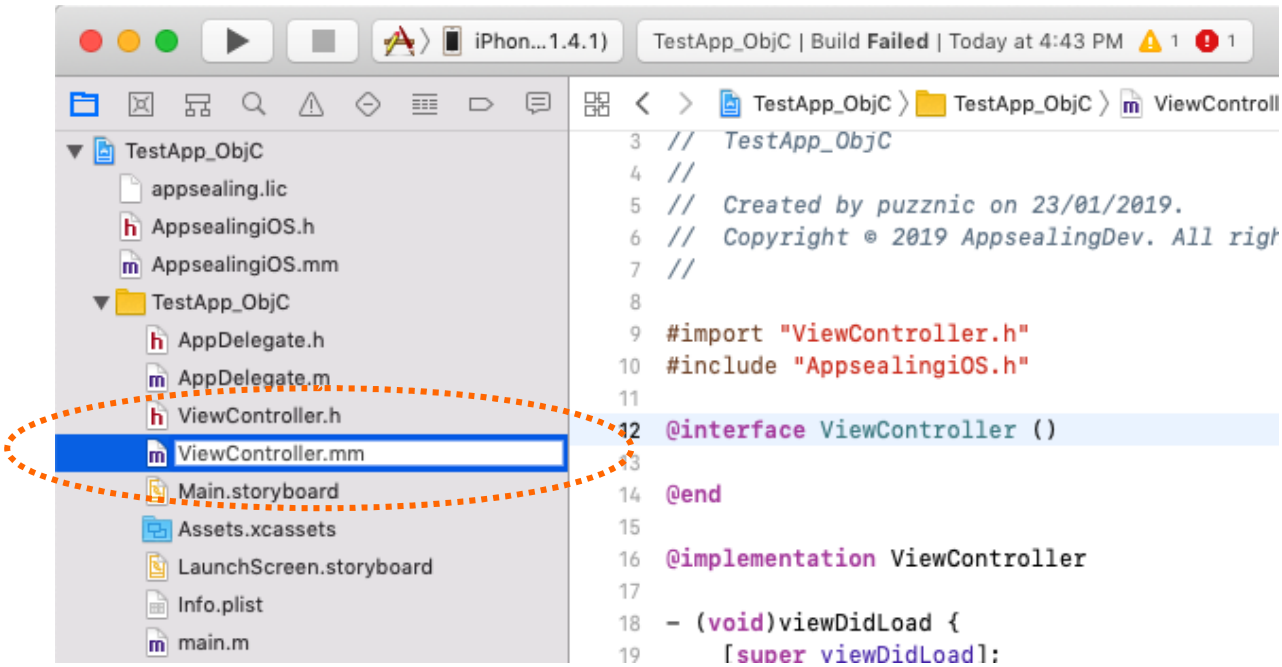
*-lStaticAppSec -L$(PROJECT_DIR)/AppSealingSDK/Libraries*

## 7-5 Xcode Build error (5) **Undefined symbols for architecture arm64: "ObjC_IsAbnormalEnvironmentDetected()"** (Objective-C project only)



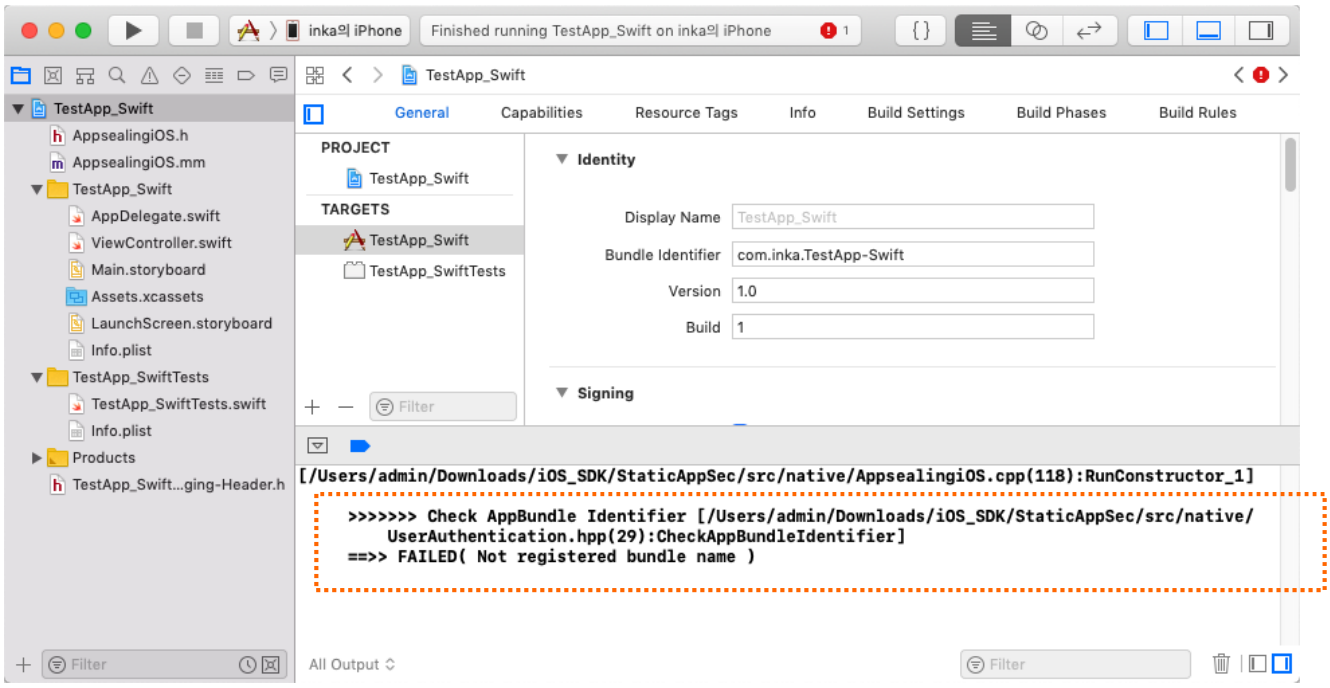

This linker error occurs when your project is Objective-C based and ViewController source code file has extension "**.m**". Just change the extension to "**.mm**".



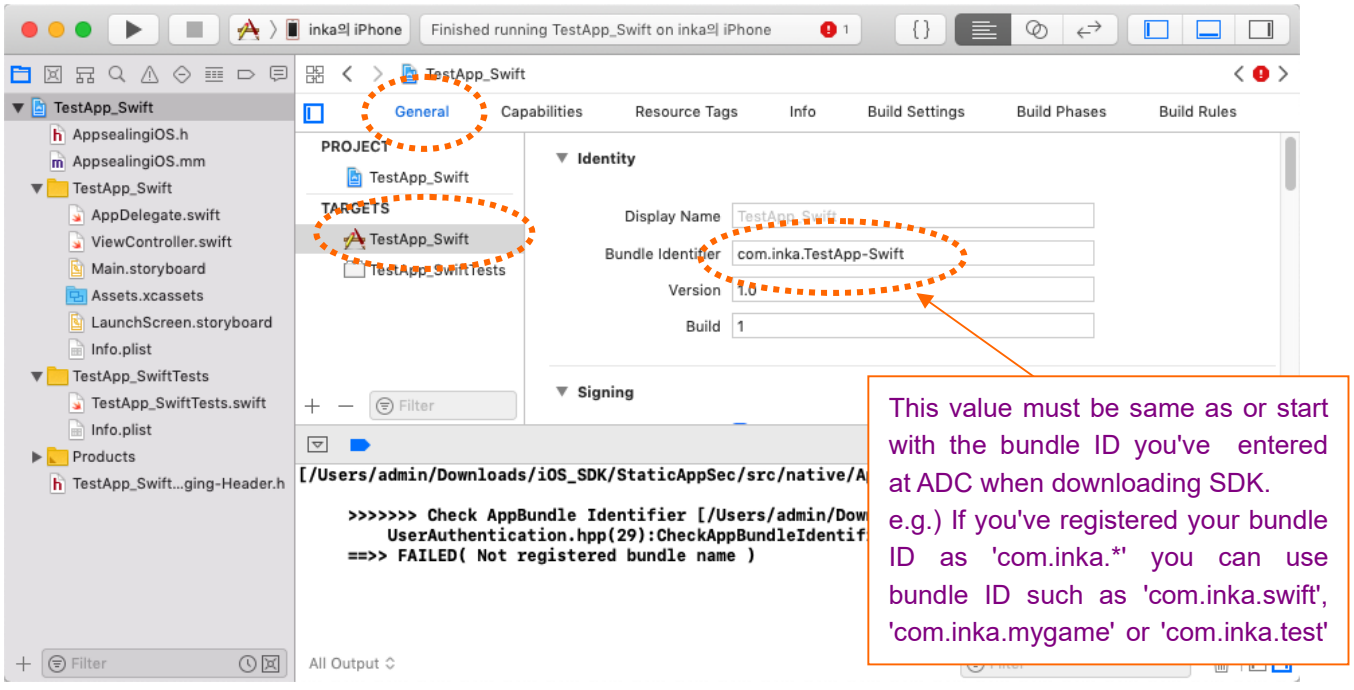

 

## 7-6 Execution error (I) **App terminated immediately after launch**

If your app has terminated right after launching in device, you should check the execution log message whether your bundle ID is valid.



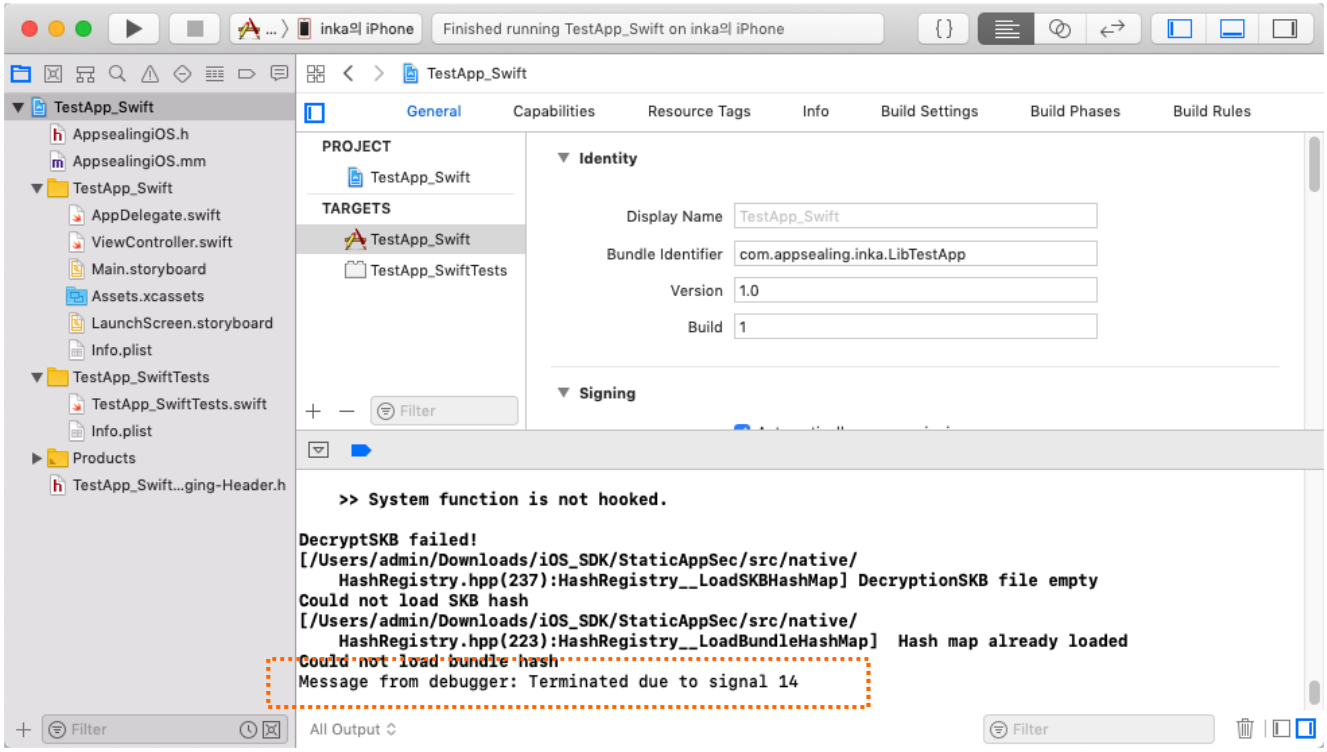

If the log message contains string like " ==>> FAILED( Not registered bundle name )" then it tells the bundle Id you used is not registered properly at AppSealing Developer Center (ADC) while downloading AppSealing SDK file. Check your bundle Id if it is same as the value you entered when downloading SDK at ADC.



## 7-7 Execution error (II) **App terminated suddenly while running**

If your app has terminated suddenly while running about after 20 seconds from launching you should check the execution log message whether you have run your app with Release configuration.



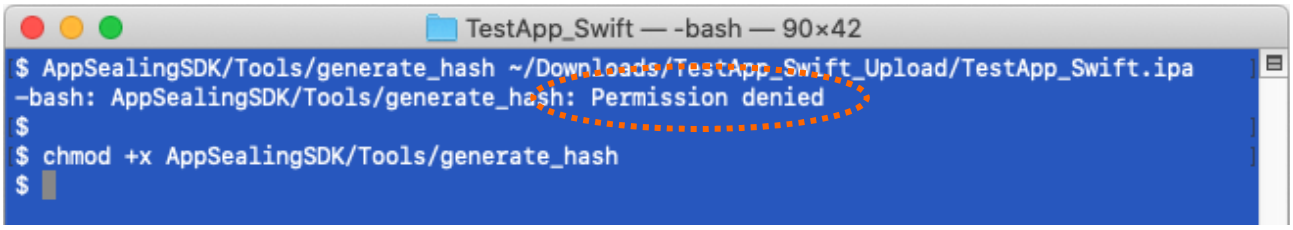

If the log message contains string like "Message from debugger: Terminated due to signal 14" then it tells that you have built & run your app in Release configuration and so AppSealing detection logic has activated. AppSealing logic in Release configuration checks some security intrusion such as jailbreak, debugger attachment, execution file encryption flag and if there is some problem it will close the app after 20 seconds irrespectively of user action or other circumstances.

Only in Release configuration the AppSealing logic will be activated and will terminate your app, so you should run your app with Debug configuration until you distribute the app to AppStore or TestFlight because the built executable file doesn't have encrypted with FairPlay DRM before it is installed from AppStore to device. AppSealing logic will detect that executable file has decrypted abnormally and it will terminate the app after 20 seconds while running your app in device at Xcode.

## 7-8 Cannot execute "generate_hash" : Permission denied

It may happens that script execution in step 3-5 "Generate App integrity and certificate snapshot" fails by "Permission denied" error message like below.




In this situation run add permission command like below and try again.

```
$ chmod +x AppSealingSDK/Tools/generate_hash
```

## 7-9 Using AppSealing SDK in **Continuous Integration** Tools

AppSealing SDK can be integrated naturally into CI (continues integration) tools such as "Buddy" or "Jenkins" etc. Once the AppSealing SDK has applied to your Xcode project, you can archive and export the project with command line shell script as usual.