INKA ENTWORKS

appsealing

iOS SDK 1.10.0.0 for Xcode

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

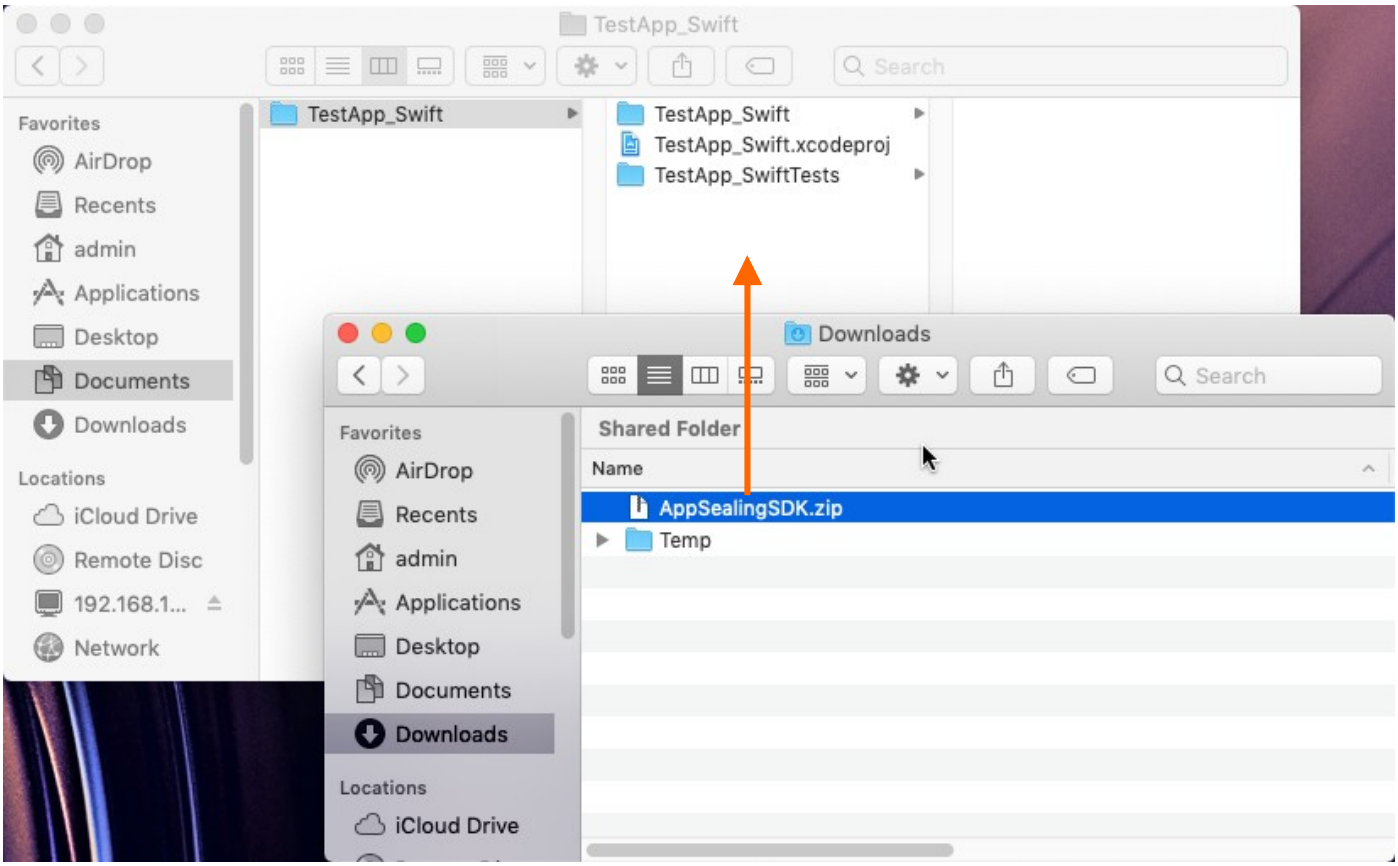
Prerequisite

- 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

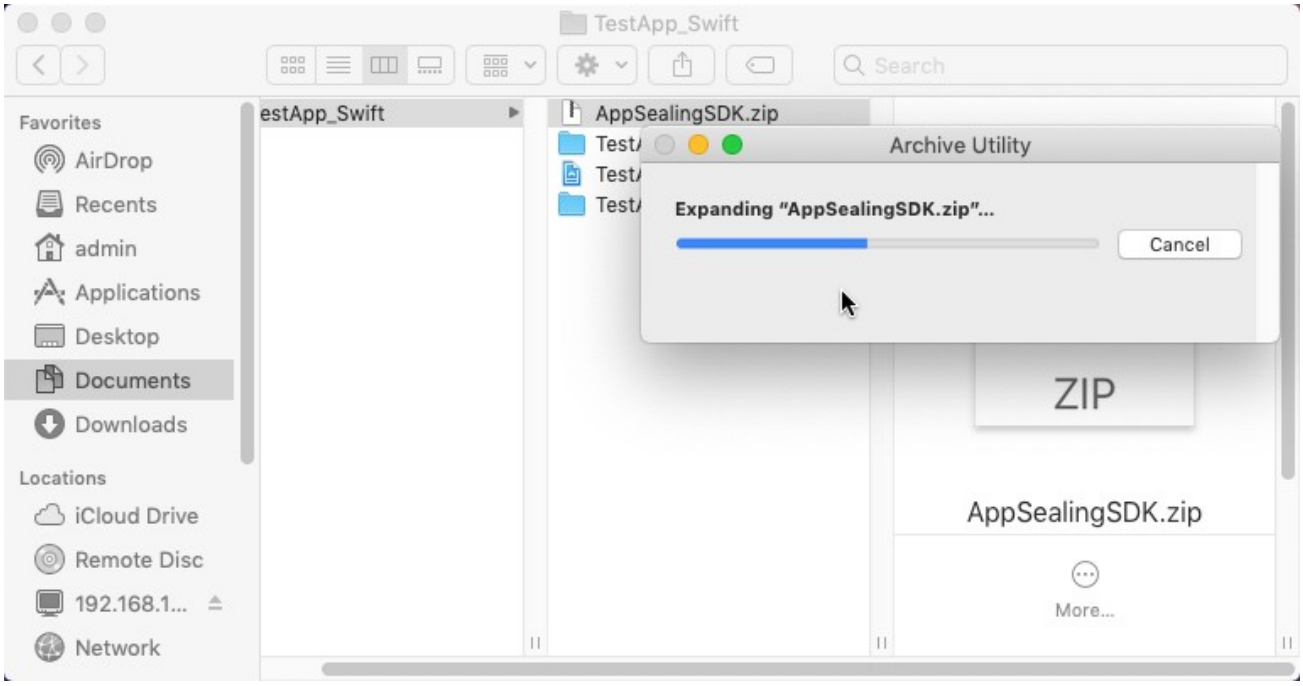
After you downloaded AppSealingSDK zip file, you should un-compress it into your Xcode project folder. This document will use sample Xcode swift project named 'TestApp_Swift' for illustration, and its location is "~/Documents/TestApp_Swift".

1-1 Move downloaded SDK file into your project folder

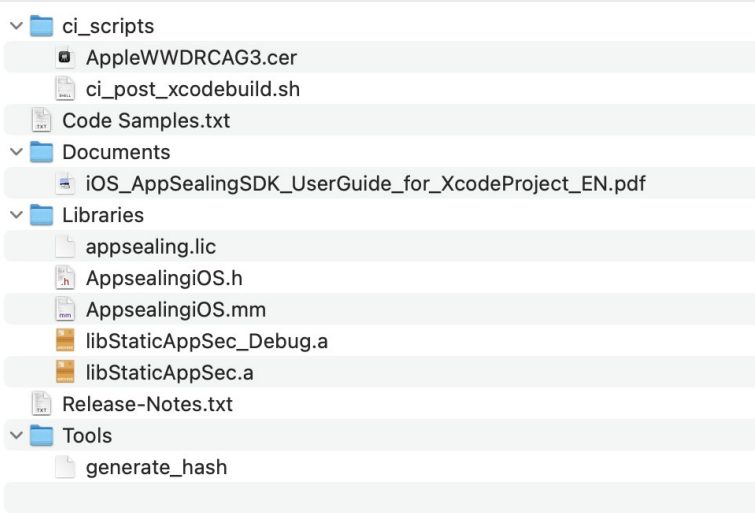


**** Warning: SDK versions that include "AdhocEnabled" in the name of the compressed file are SDKs with weakened security features, enabling app distribution through methods other than the official App Store or TestFlight (such as Ad Hoc, enterprise, MDM, and other third-party stores). Distribution outside of the App Store can make you vulnerable to attacks, so unless absolutely necessary, we recommend downloading a version that does not include AdhocEnabled.**

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



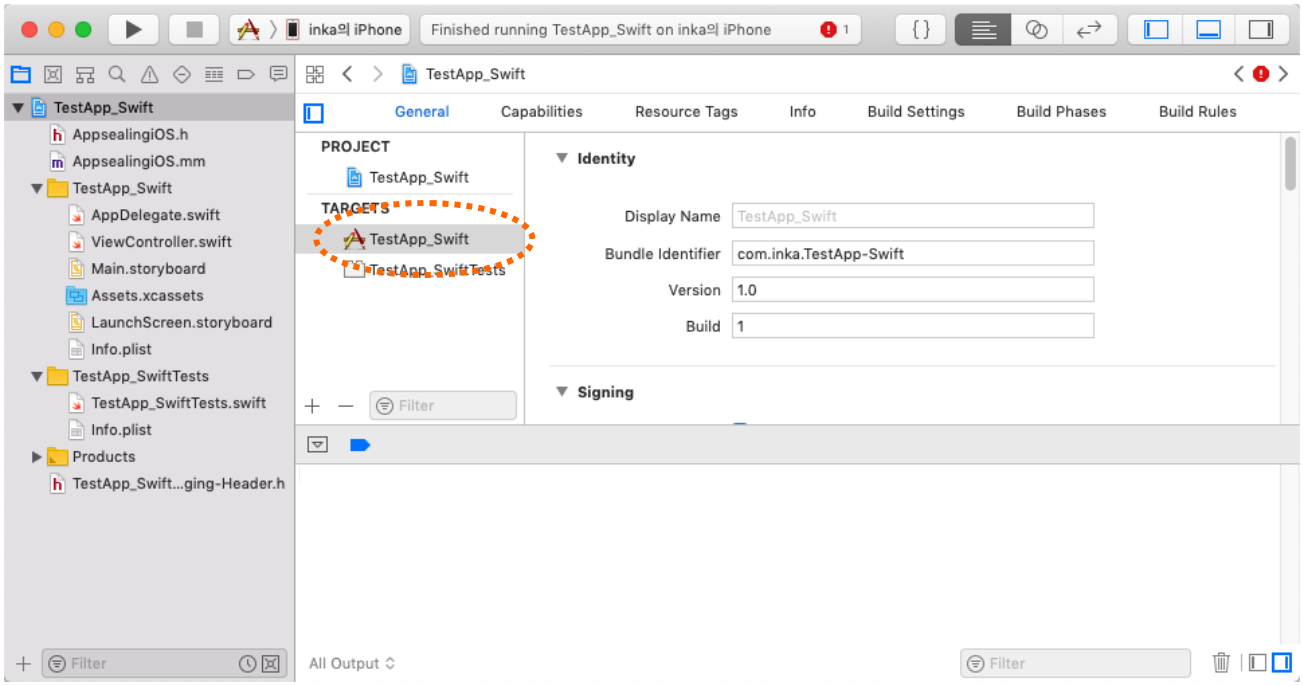
After uncompressing zip file, check whether all files are generated like following structure.



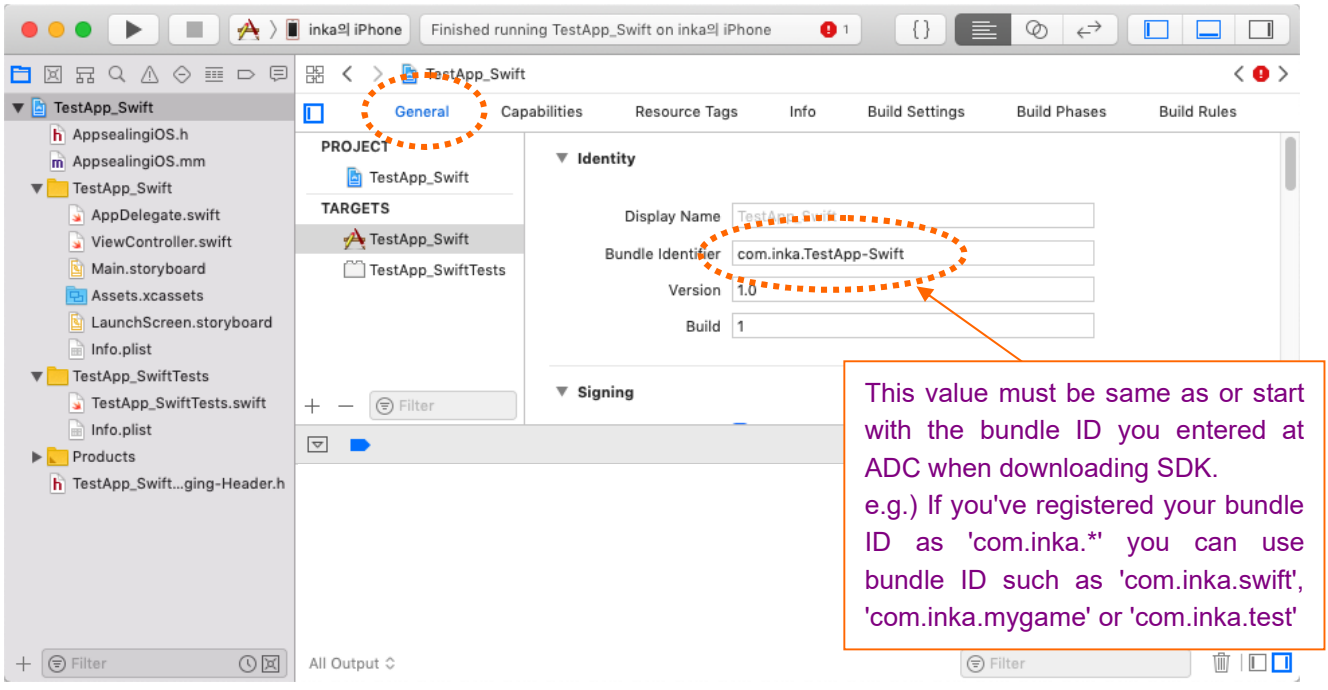
If any file is missing try re-download SDK or try expand zip file again.

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.

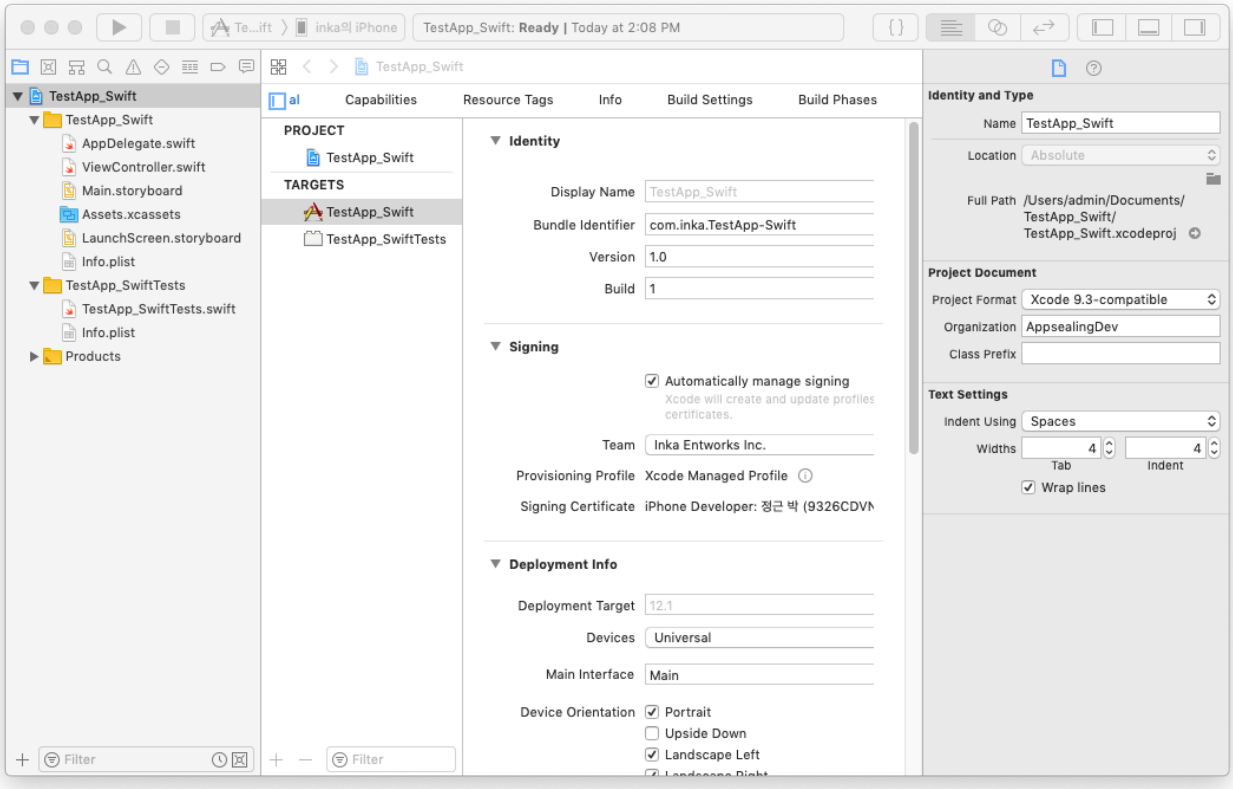


Then select "General" tab and check your bundle Id whether it is same as or starts with the value you entered when downloading SDK at AppSealing Developer Center (ADC).

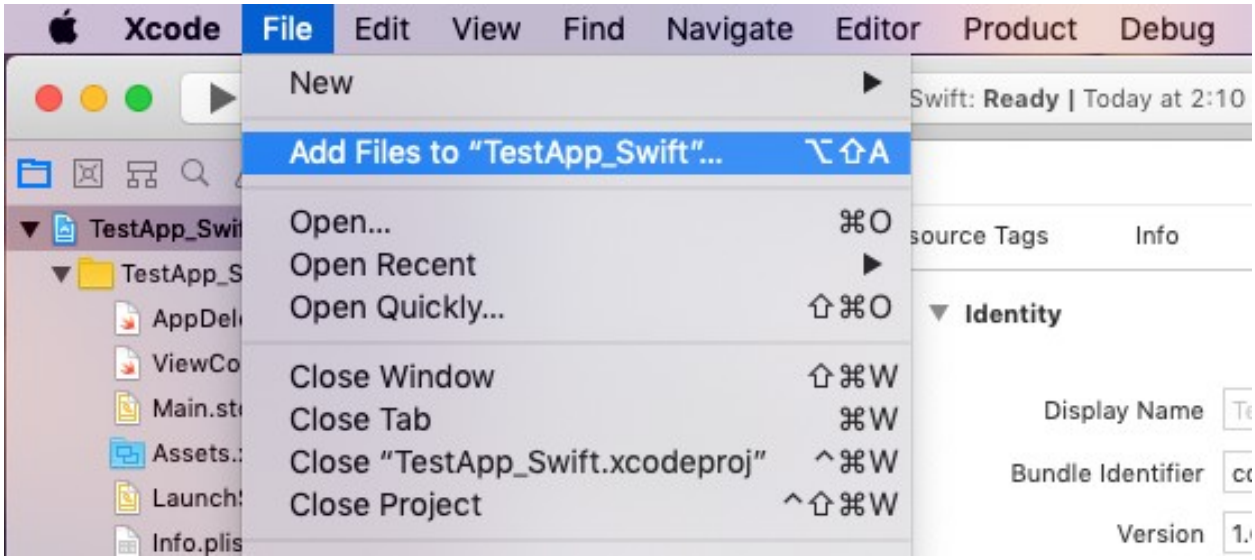


Part 2. Add additional files to your project

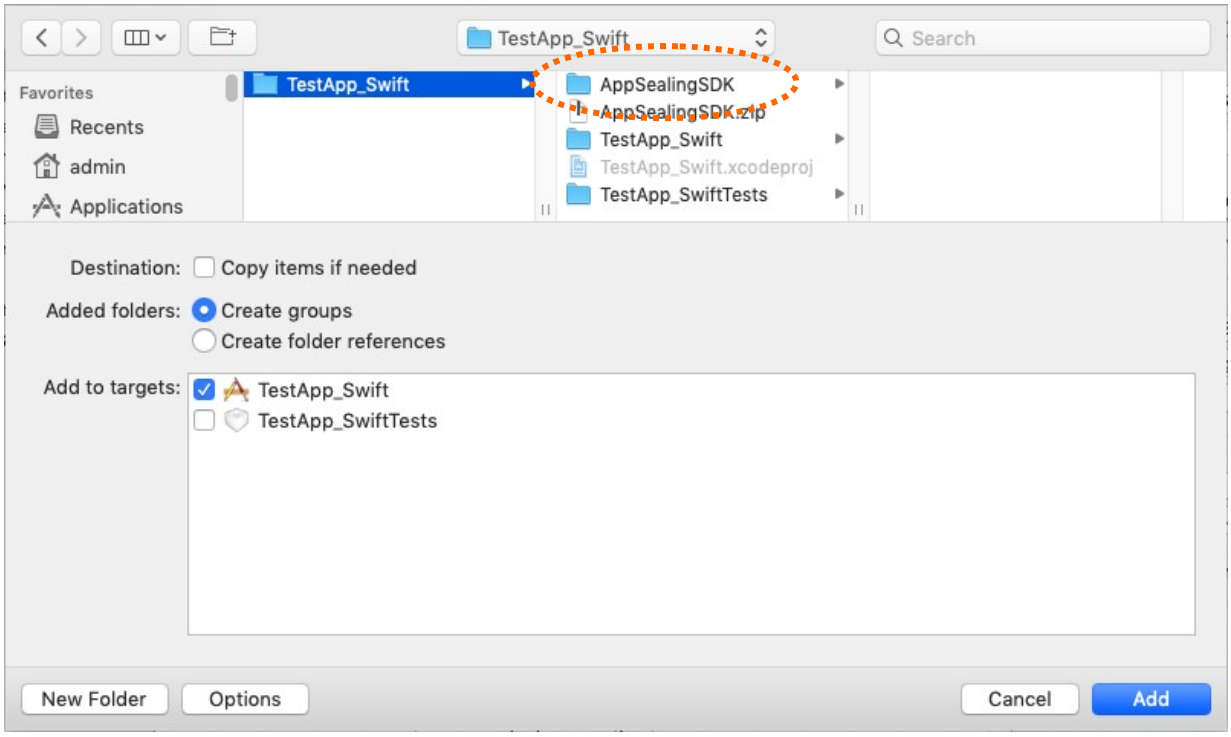
2-1 Open your **Xcode project**

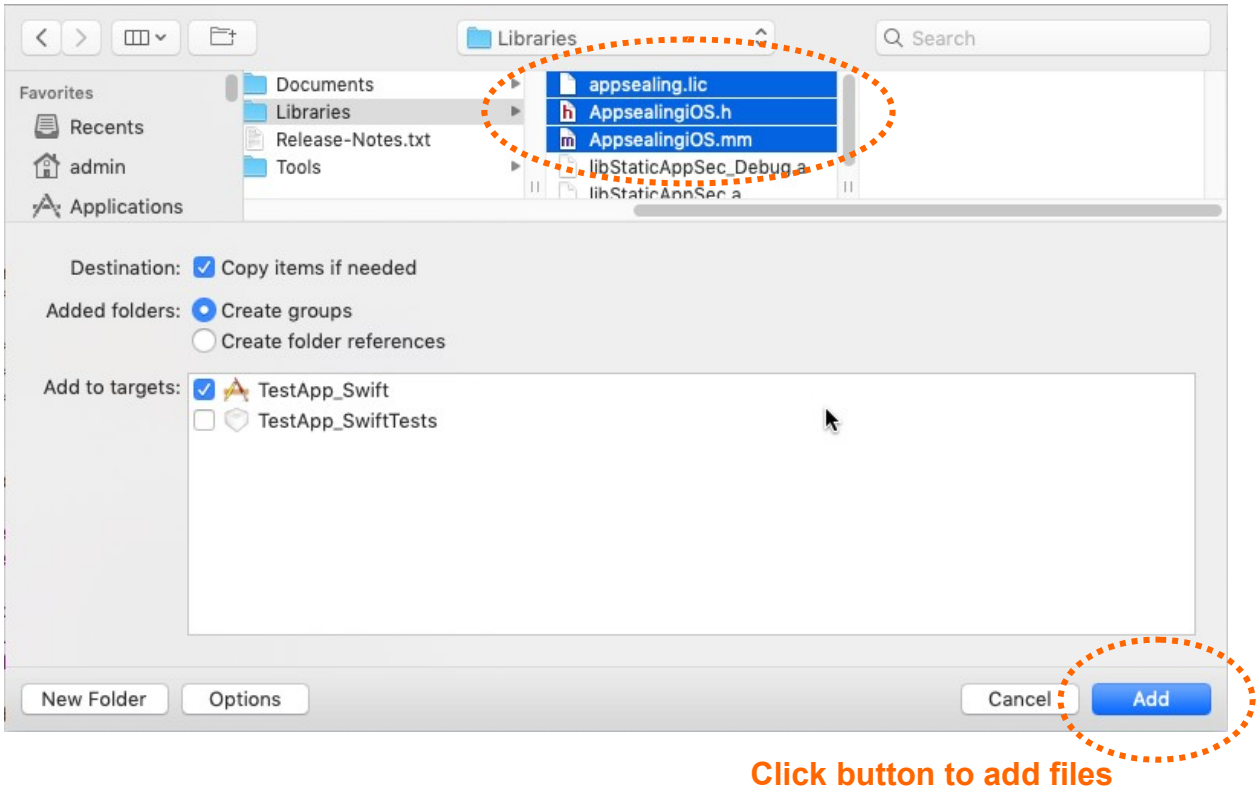


2-2 Perform **'File >> Add Files to "TestApp_Swift"...'** menu action



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





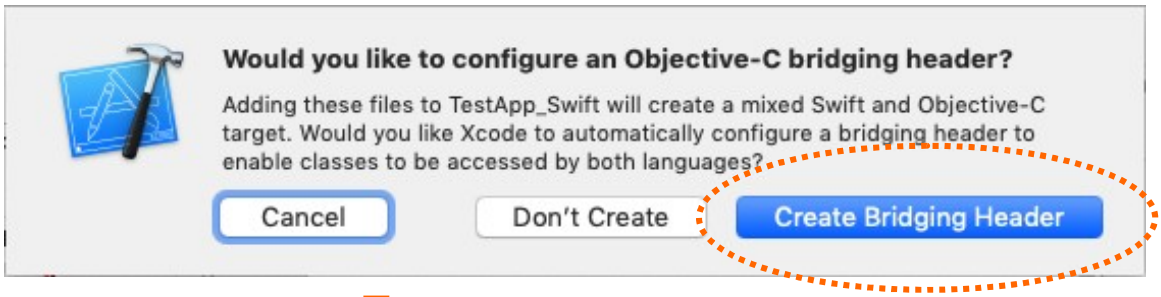
Click button to add files

2-3 Add **Bridging Header into swift project**

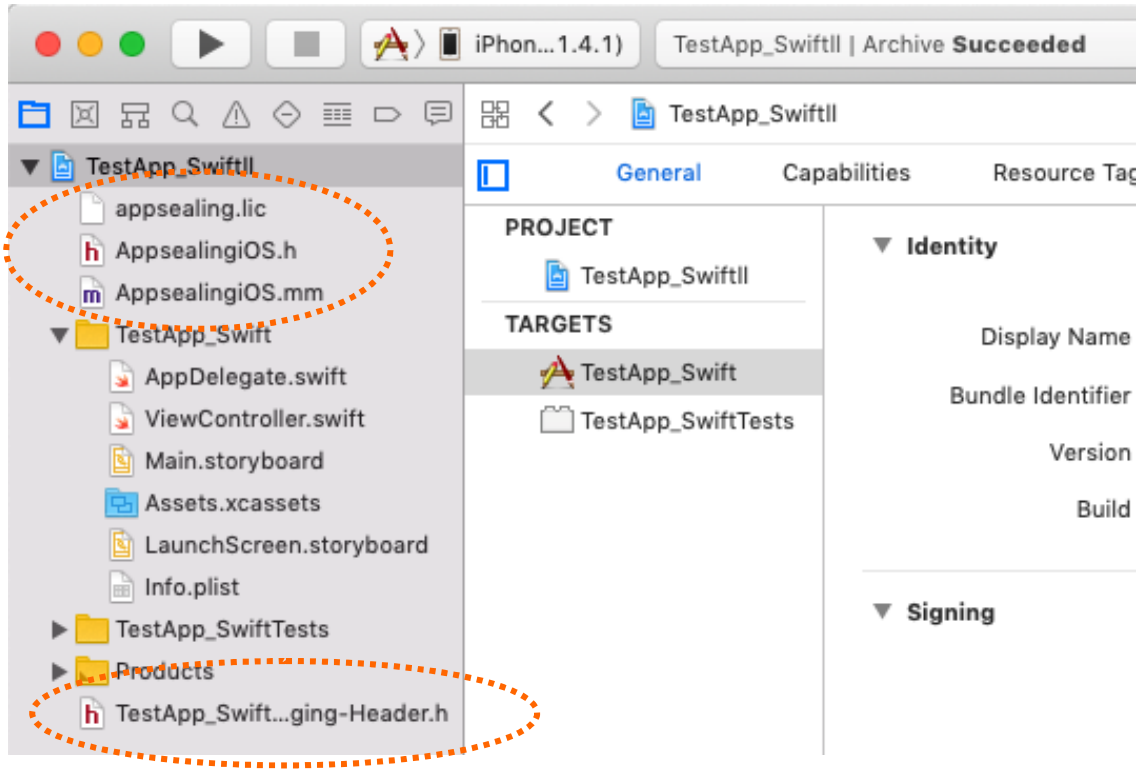
Only if your project is Swift based

(Do not refer this & next pages if your project is objective-c 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".



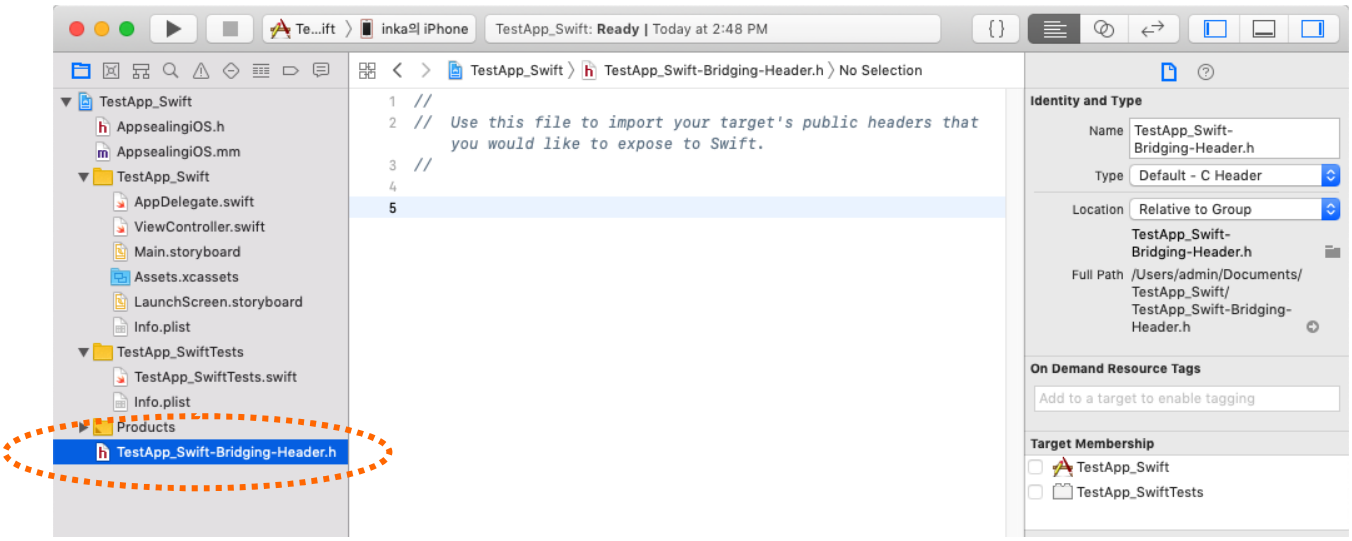
Click this button to create Bridging Header



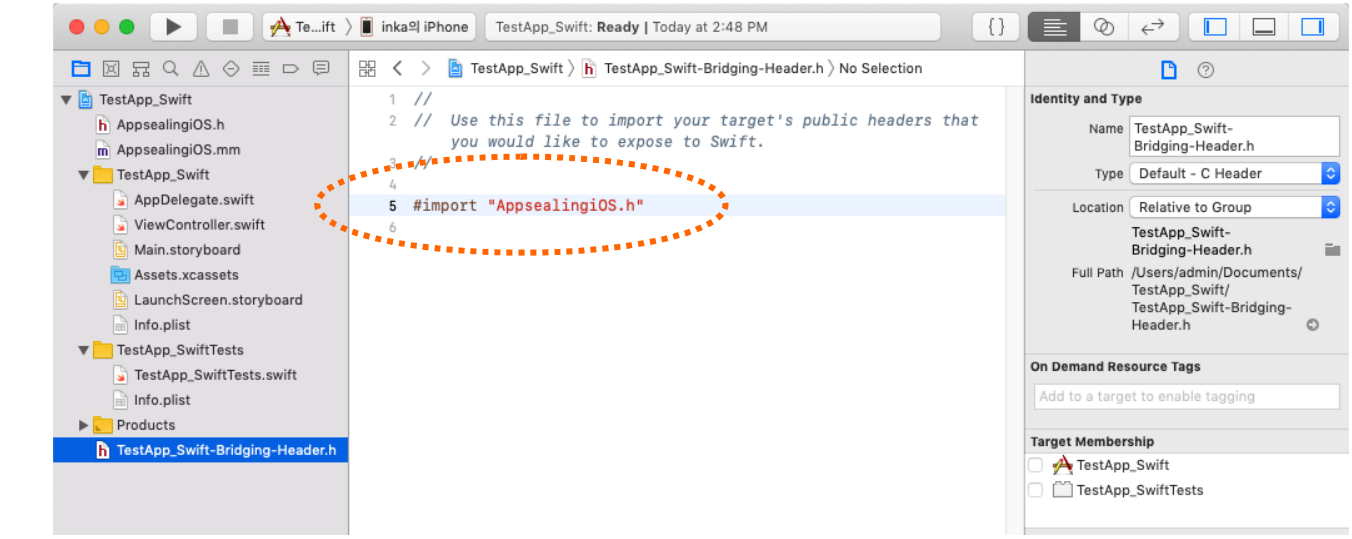
* Supposing your project already has bridging header file, the upper dialog box will not appear and you can use your existing bridging header file.

** If your project is Objective-C based, the bridging header file is not used.

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



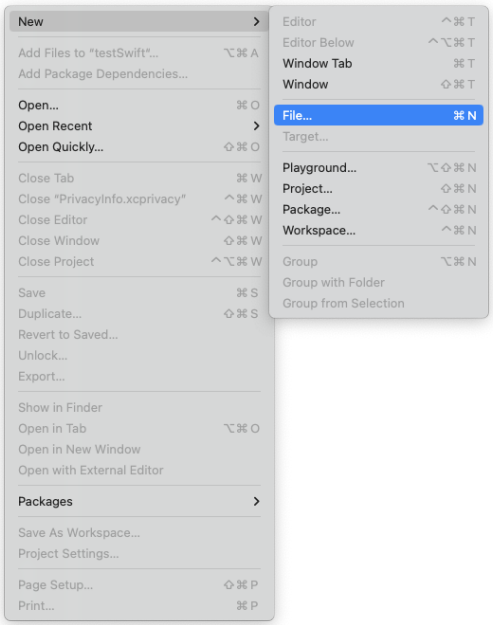
Select newly created "TestApp_Swift-Bridging-Header.h" file (or existing bridging header file) and append *#import "AppsealingiOS.h"* to end of document like below.



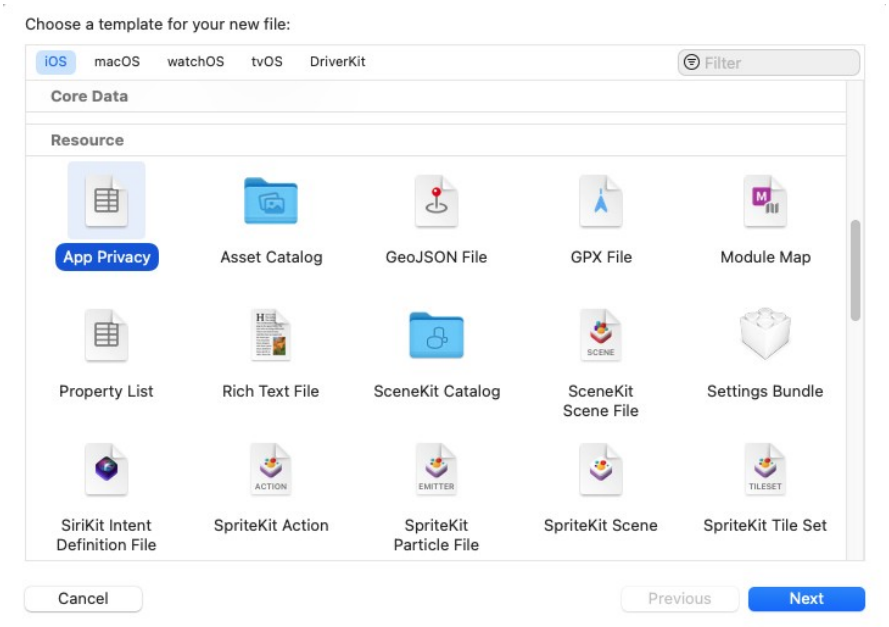
2-5 Add an item to **PrivacyInfo.xcprivacy** file (App Store upload)

Since May 2024, privacy manifest file is required for App Store upload.

If there is no "PrivacyInfo.xcprivacy" file in the project, please open .xcodeproj or .xcworkspace file of the project by Xcode, and then click "File > New > File..." in the menu.



Choose "App Privacy" in the "Resource" section to add privacy manifest file in the project:

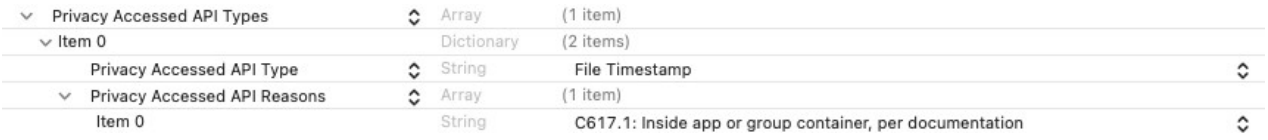


For more detailed guide for the privacy manifest file, please refer to the following link:
https://developer.apple.com/documentation/bundleresources/privacy_manifest_files#4284009

Open "PrivacyInfo.xcprivacy" file and add items so that the file includes the following structure:

- key "NSPrivacyAccessedAPITypes"
  - array
    - key "NSPrivacyAccessedAPIType": string "NSCategoryFileTimestamp"
    - key "NSPrivacyAccessedAPITypeReasons"
      - array
        - string "C617.1"

As the result, when you open "PrivacyInfo.xcprivacy" in Xcode, you can check the following structure is included:



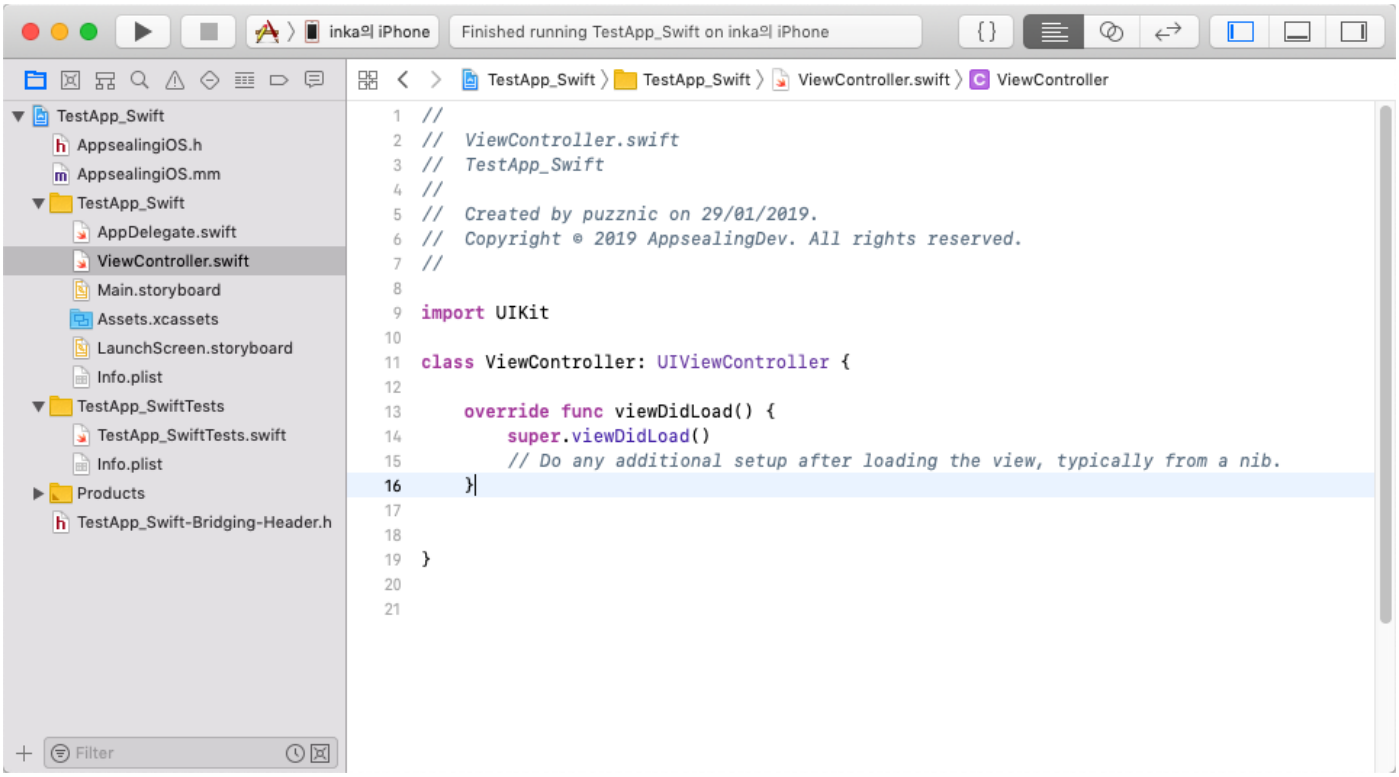
Part 3. Add simple GUI for iOS security intrusion

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. (In case of Objective-C based project, change extension of ViewController.m to mm and modify file)

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

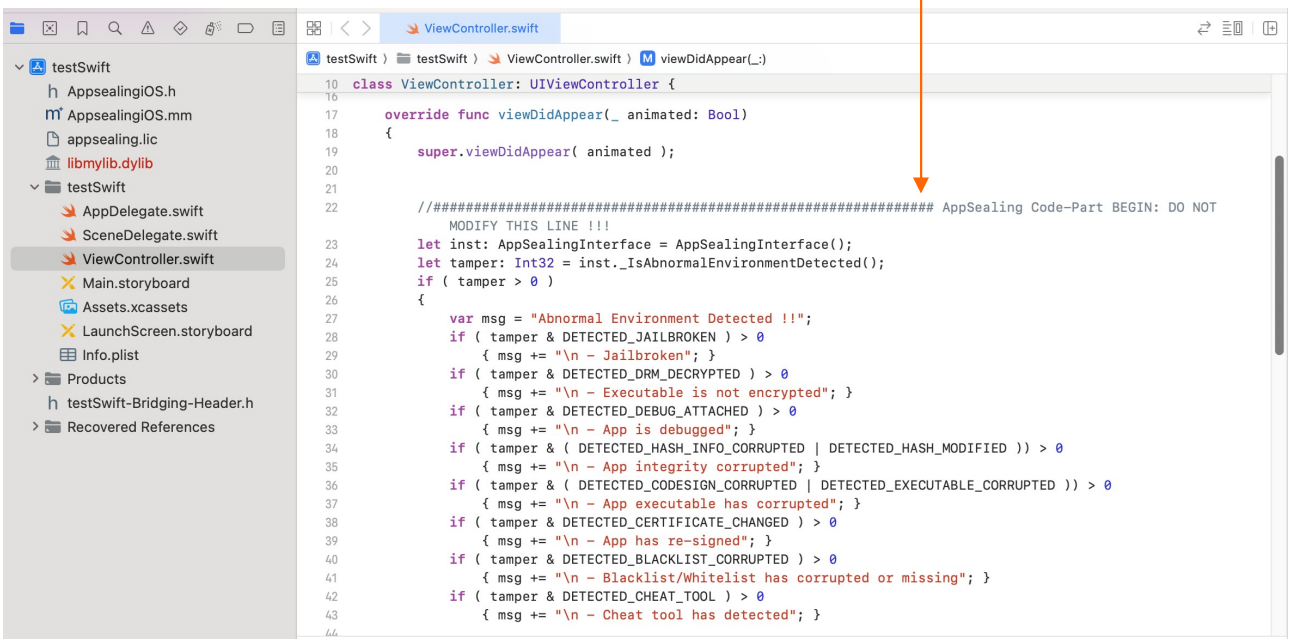
First, open your Xcode project and open ViewController.swift" file.



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 );

    //############################################################# AppSealing Code-Part BEGIN: DO NOT MODIFY THIS LINE !!!
    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"; }
        if ( tamper & DETECTED_BLACKLIST_CORRUPTED ) > 0
            { msg += "\n - Blacklist/Whitelist has corrupted or missing"; }
        if ( tamper & DETECTED_CHEAT_TOOL ) > 0
            { msg += "\n - Cheat tool has detected"; }

        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);
    }

        AppSealingInterface._NotifySwizzlingDetected( { (msg: String?) -> () in
            let alertController = UIAlertController( title: "AppSealing Security", message: msg, preferredStyle: .alert )
            alertController.addAction( UIAlertAction( title: "Confirm", style: .default,
                                                handler: { ( action:UIAlertAction! ) -> Void in
                            #if !DEBUG
                                exit(0);
                            #endif
            } ));
            self.present( alertController, animated: true, completion: nil );
        });
    //############################################################# AppSealing Code-Part END: DO NOT MODIFY THIS LINE !!!
}
```



Sample UI code above is also included in "Code Samples.txt" file so you can copy & paste in that file.

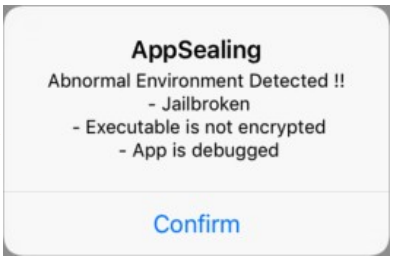
If your project is Objective-C based then you can use following code to show simple UI.

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

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

    //############################################################ AppSealing Code-Part BEGIN: DO NOT MODIFY THIS LINE !!!
    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"];
        if (( tamper & DETECTED_BLACKLIST_CORRUPTED ) > 0 )
            msg = [msg stringByAppendingString:@"\n - Blacklist/Whitelist has corrupted or missing"];
        if (( tamper & DETECTED_CHEAT_TOOL ) > 0 )
            msg = [msg stringByAppendingString:@"\n - Cheat tool has detected"];

        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];
    }


    [AppSealingInterface _NotifySwizzlingDetected:^(NSString* msg){
        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];
    }];
    //############################################################ AppSealing Code-Part END: DO NOT MODIFY THIS LINE !!!
}
```

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

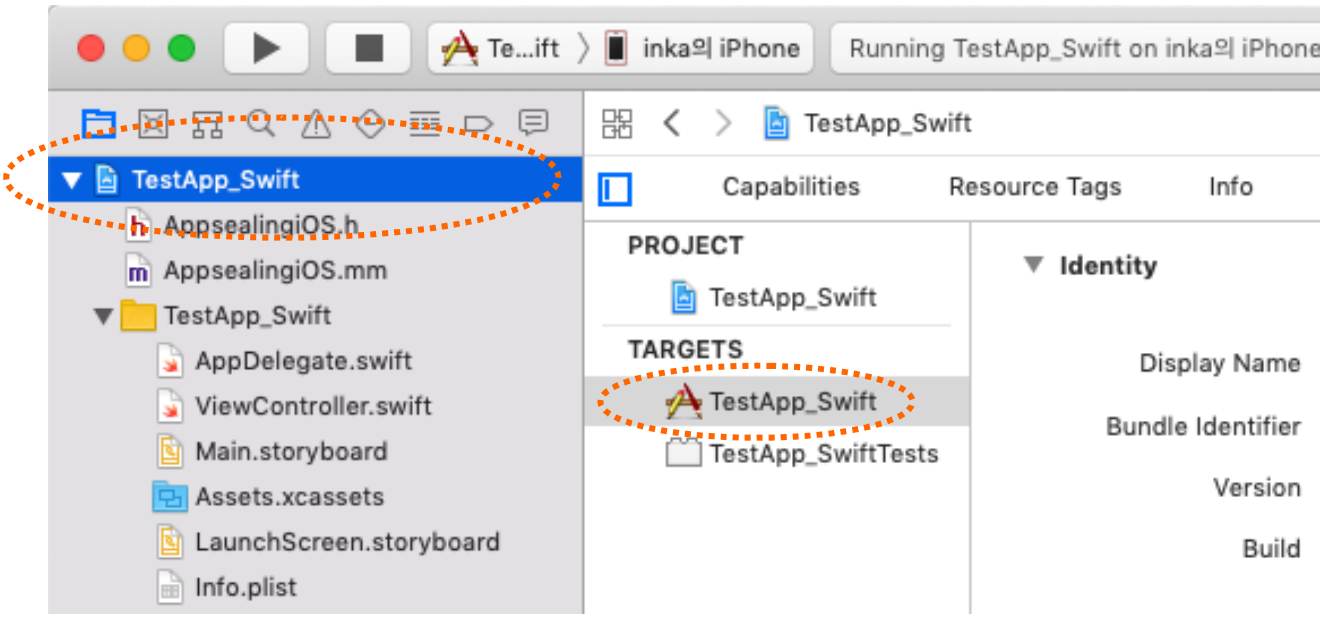
.

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

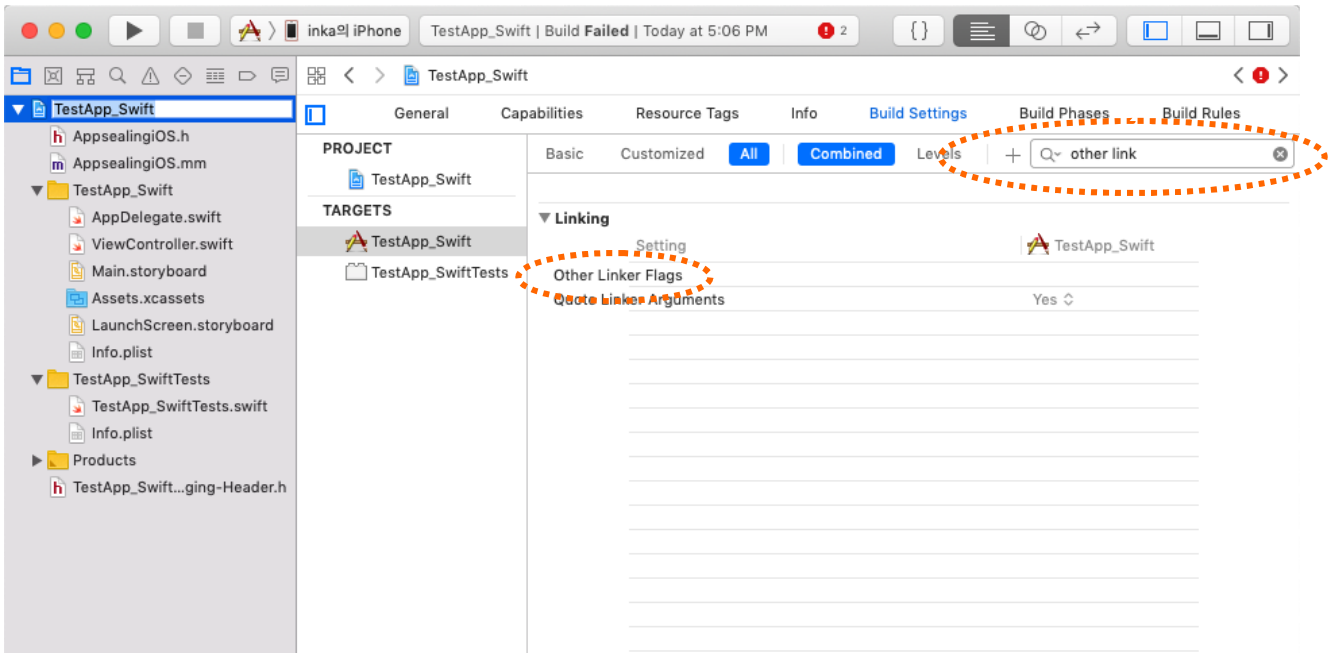
```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
     launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool
{
    // Override point for customization after application launch.

      //############################################################# AppSealing Code-Part BEGIN: DO NOT MODIFY THIS LINE !!!
      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"; }
          if ( tamper & DETECTED_BLACKLIST_CORRUPTED ) > 0
              { msg += "\n - Blacklist/Whitelist has corrupted or missing"; }
          if ( tamper & DETECTED_CHEAT_TOOL ) > 0
              { msg += "\n - Cheat tool has detected"; }

          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
                                } ));
          DispatchQueue.main.async {
              self.window?.rootViewController?.present( alertController, animated: true, completion: nil)
          };
      }

      AppSealingInterface._NotifySwizzlingDetected( { (msg: String?) -> () in
          let alertController = UIAlertController( title: "AppSealing Security", message: msg, preferredStyle: .alert );
          alertController.addAction( UIAlertAction( title: "Confirm", style: .default,
                                                    handler: { ( action:UIAlertAction! ) -> Void in
  #if DEBUG
  #else
              exit(0);
  #endif
          } ));

          DispatchQueue.main.async {
              self.window?.rootViewController?.present( alertController, animated: true, completion: nil )
          };
      });

      //############################################################# AppSealing Code-Part END: DO NOT MODIFY THIS LINE !!!
    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

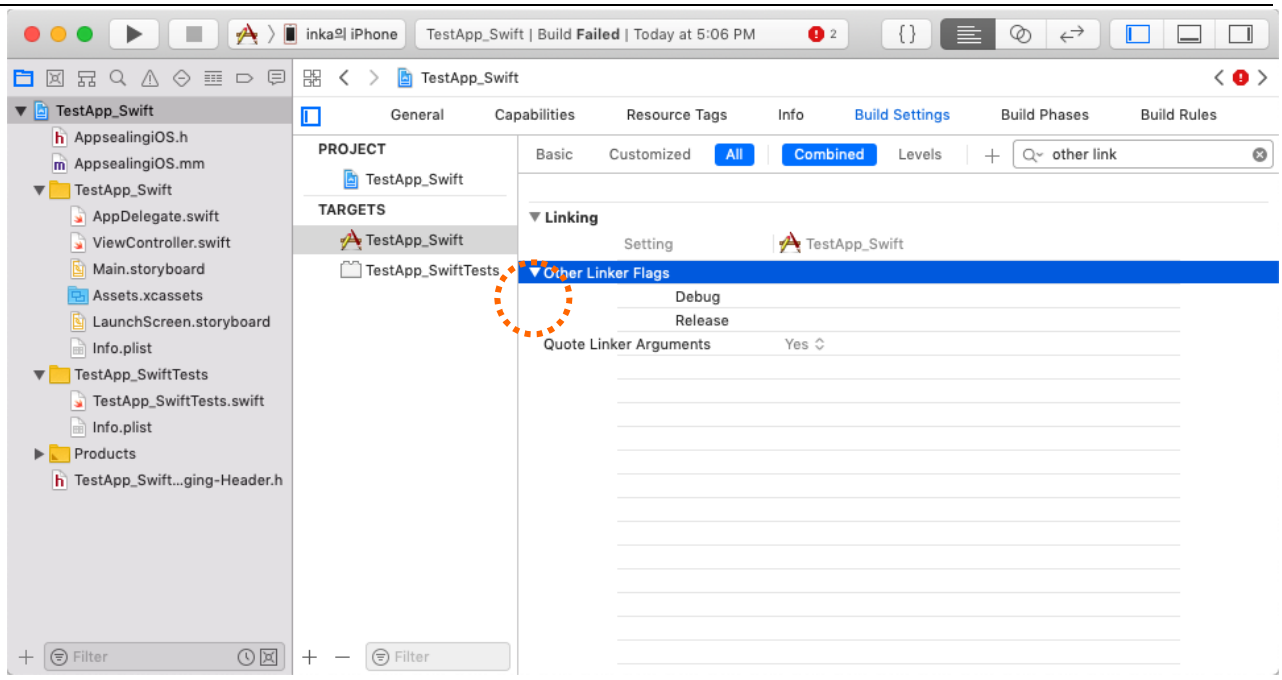


4-2 Select "**Build Settings**" tab and type "**Other link**" in search box



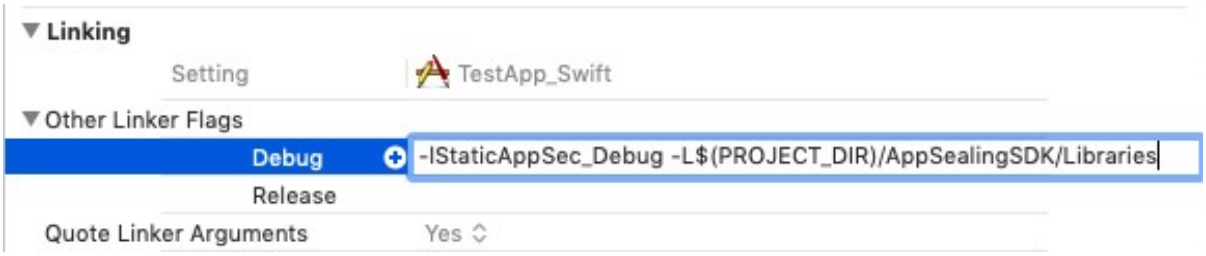
Then you will see **"Other Linker Flags"** field. Now, fill this settings by following steps.

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



$(PROJECT_DIR) is an environment variable specifying the current project path. To replace this value with an absolute path, you must change the entire $(PROJECT_DIR), not PROJECT_DIR in parentheses. For example, if the target project path is /Users/myname/MyProject/SampleProject1, the following string should be added to the value of Other Linker Flags.

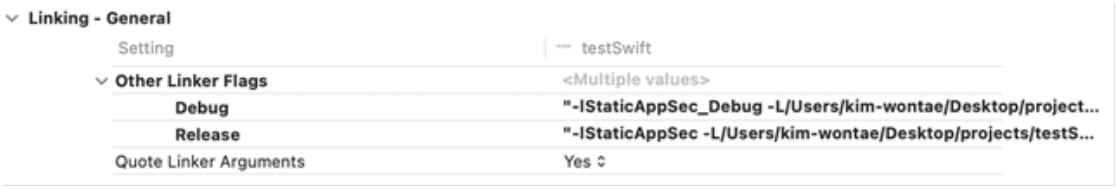
-L/Users/myname/MyProject/SampleProject1/AppSealingSDK/Libraries

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

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

When using an absolute path even for Release, be careful not to change only the path in parentheses while leaving $( ) as it is.

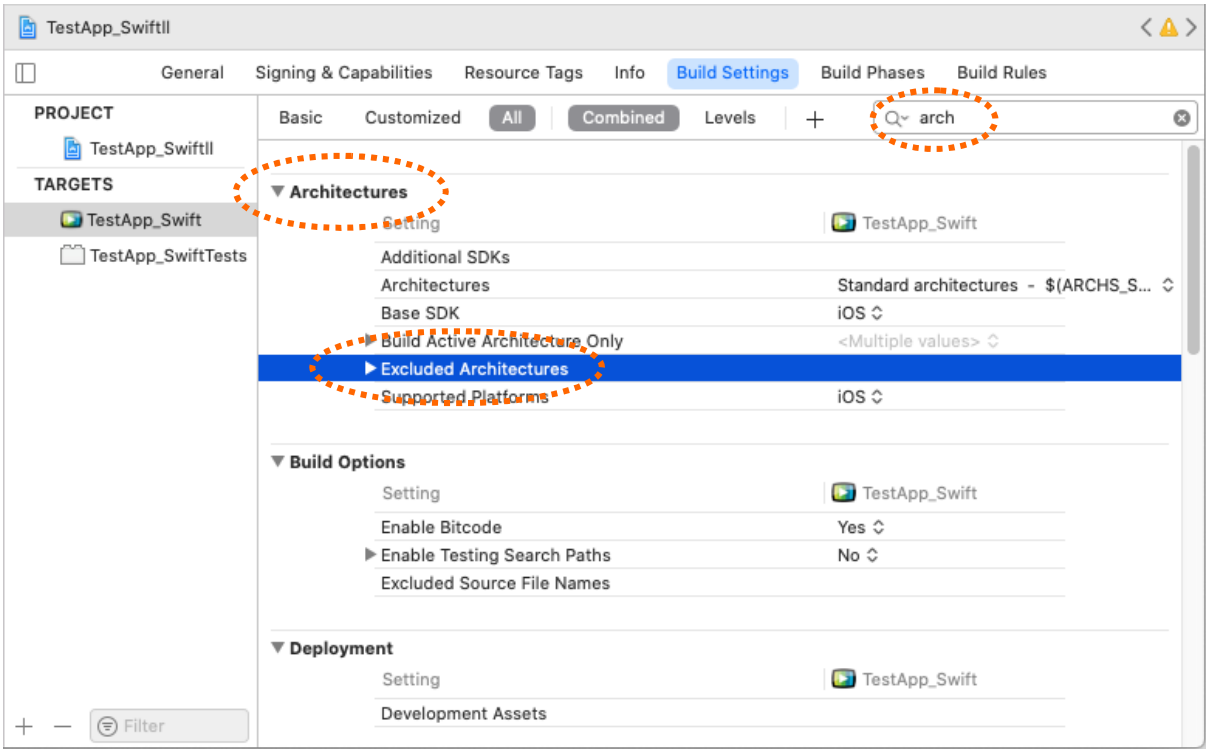
**** Warning: Depending on how you paste the options, quotes may be added as shown below. In this case, you should enter the options by splitting them at each space so that there are no quotes in the final input.**



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.

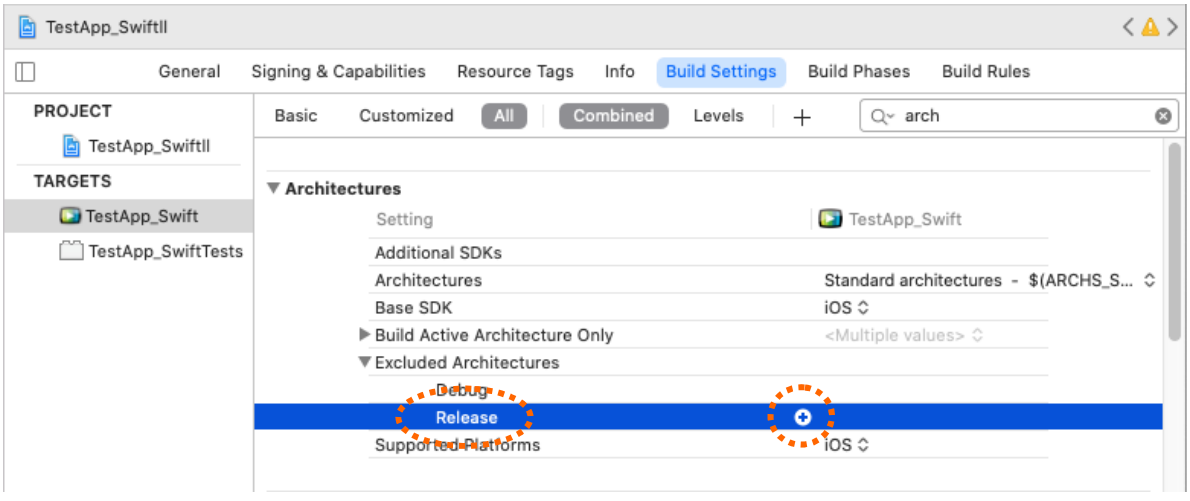
4-3 Configure **Build Settings** "**Architectures – Excluded Architectures**"

Search "arch" keyword at Build Settings panel.

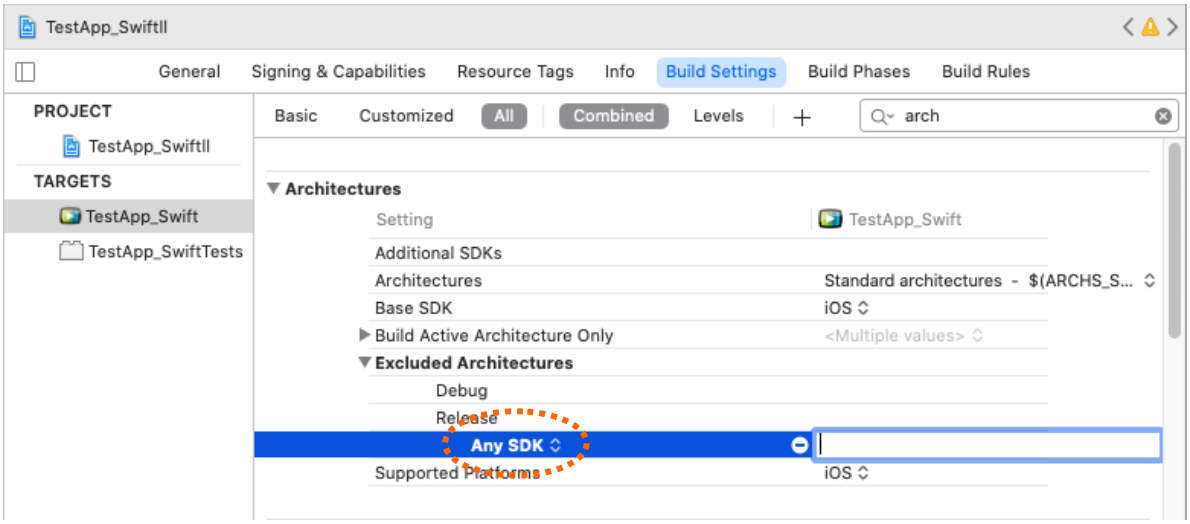


Follow next steps after **"Architectures"** field has shown.

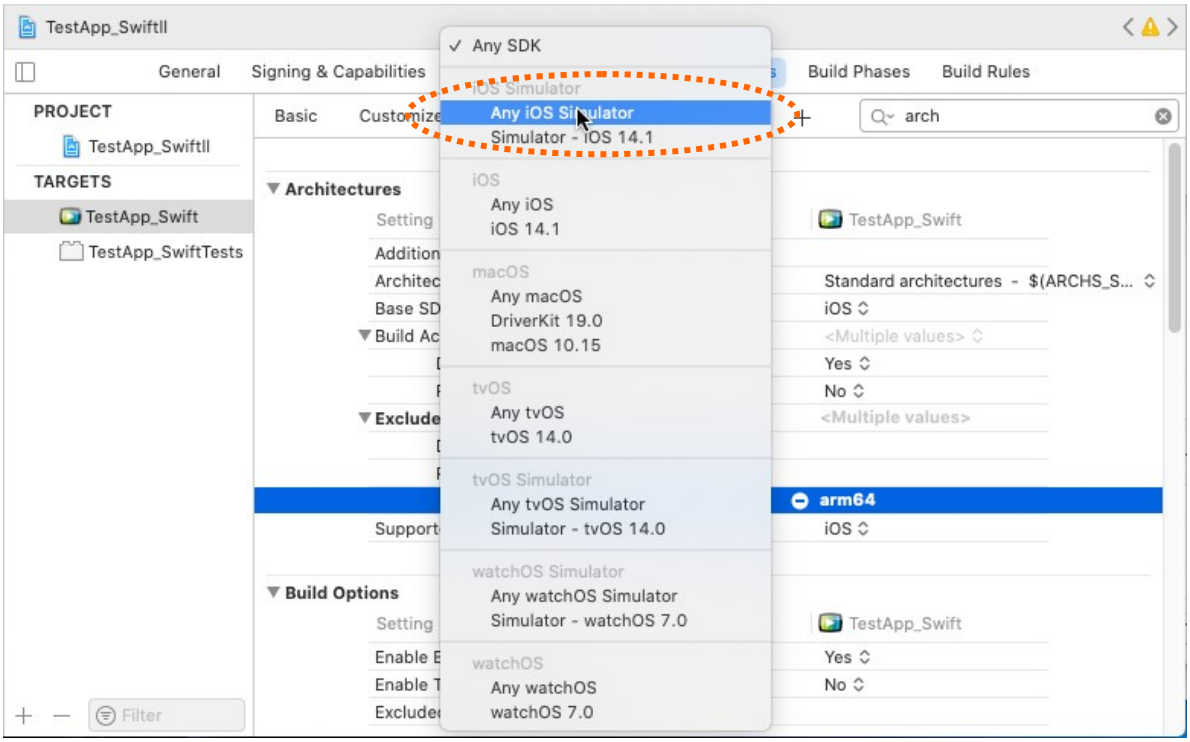
1) Expand "Excluded Architectures" by clicking the left-side triangle icon.



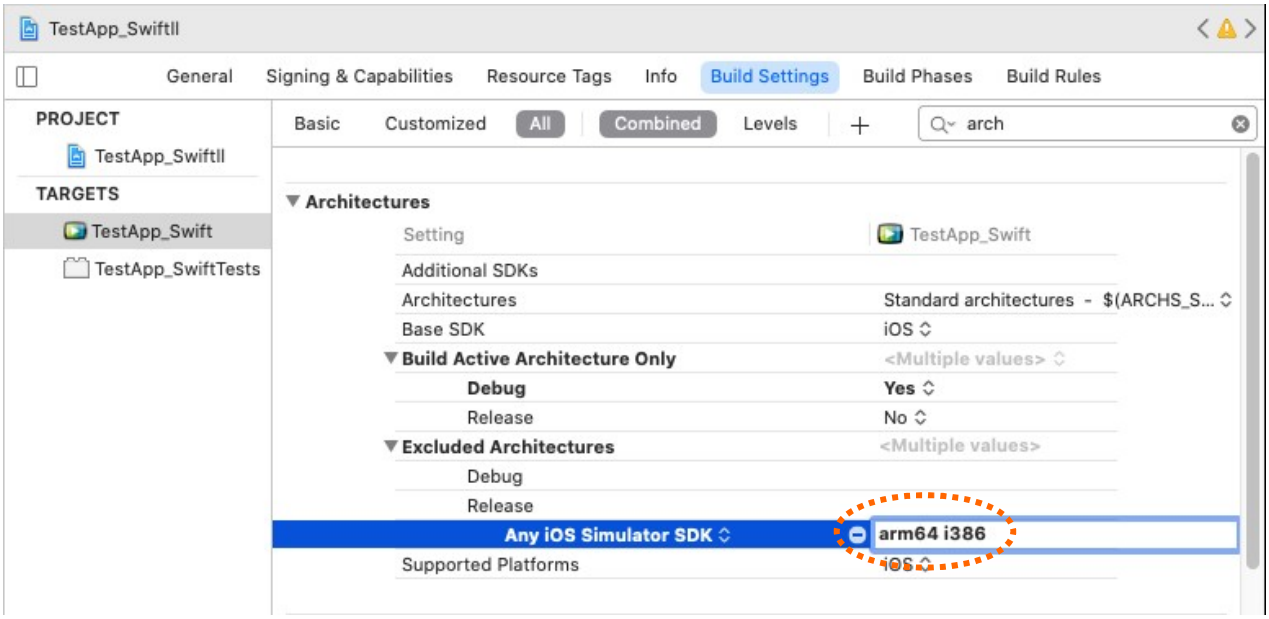
2) Select "Release" item and click right-side "+" icon.



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

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

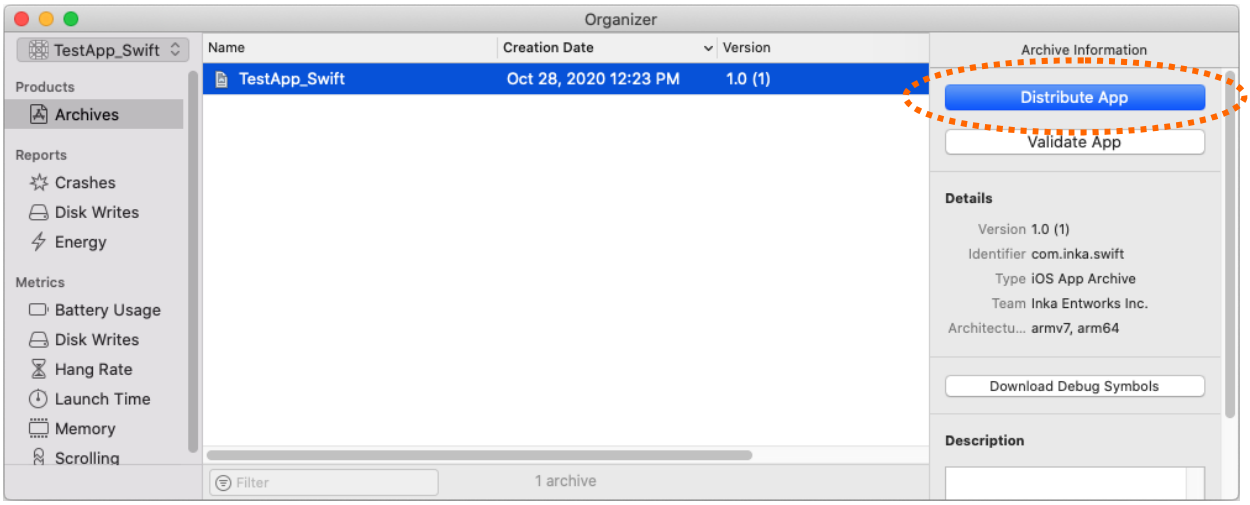
You will build the app as Release mode when distributing to the App Store. If you test AppSealing with Release mode, your app should be distributed to App Store or 'TestFlight'. If not, the executable file will be detected as not encrypted, so the app will be closed.

4-5 Generate App integrity & certificate verification snapshot

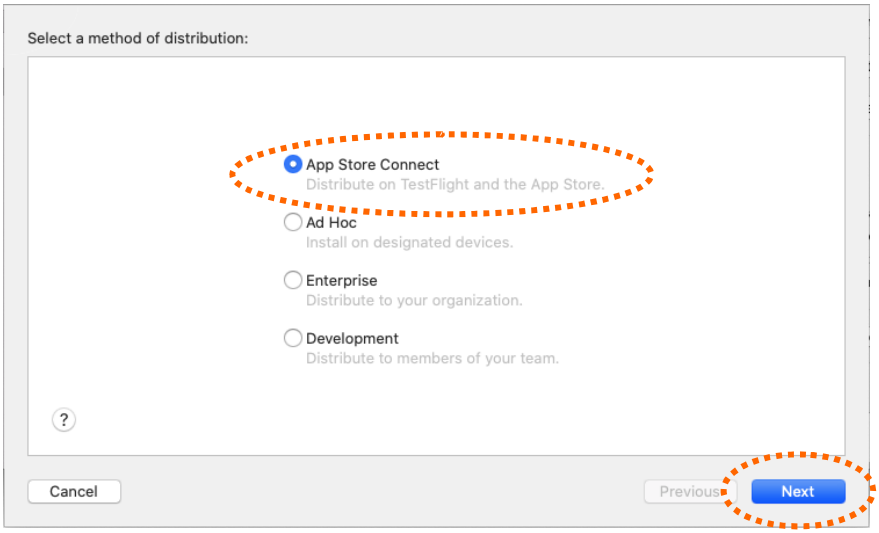
There is additional process to verify app integrity & certificate when you test your app or distribute app through app store. If you skip this step the app running on device will be terminated after few seconds for broken app integrity.

You should process this step when you distribute your app through TestFlight or App Store, and when you use Ad Hoc distribution with AdhocEnabled SDK.

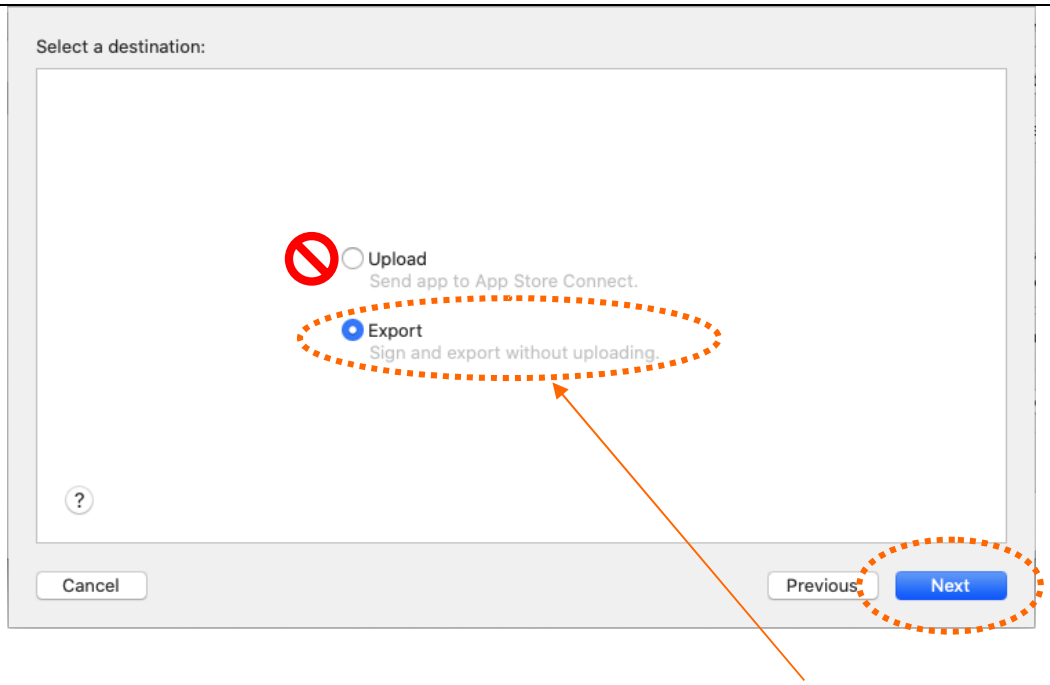
Let's see the upload process to App Store or TestFlight step by step. Below is Organizer window after Archive from Xcode.



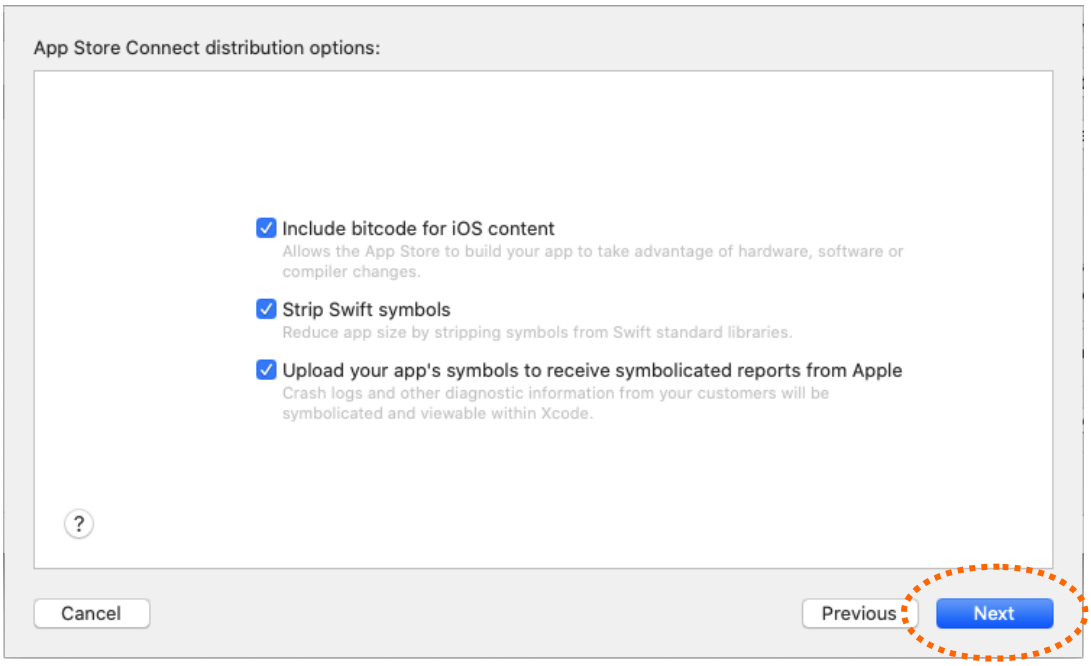
Click "Distribute App" button to generate IPA for uploading to App Store.



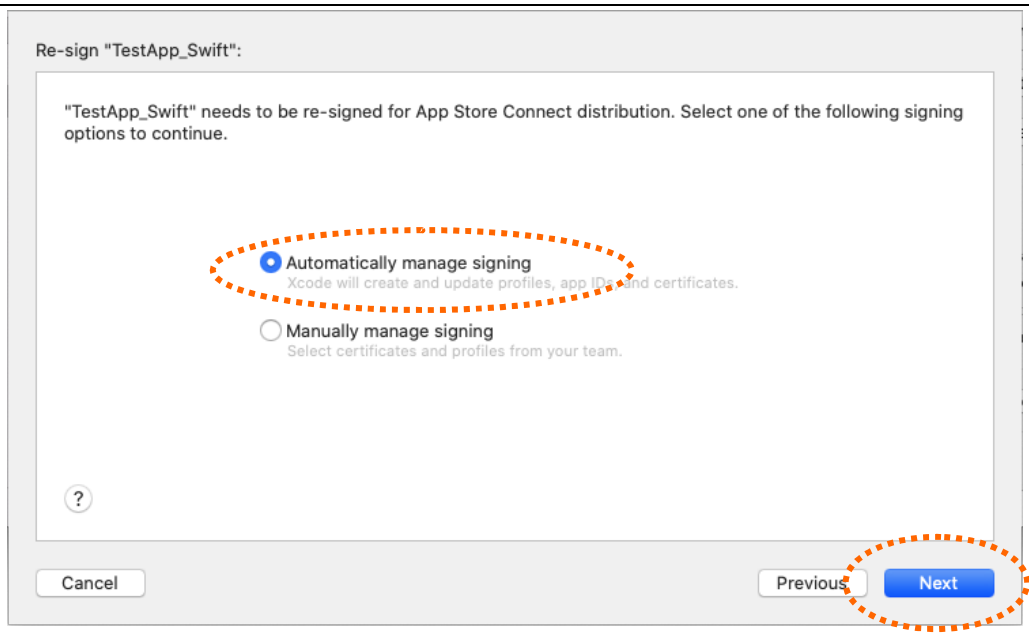
Click "Next" button with "App Store Connect" is selected. (If you are using AdhocEnabled SDK, "Ad Hoc" is also an available option.)



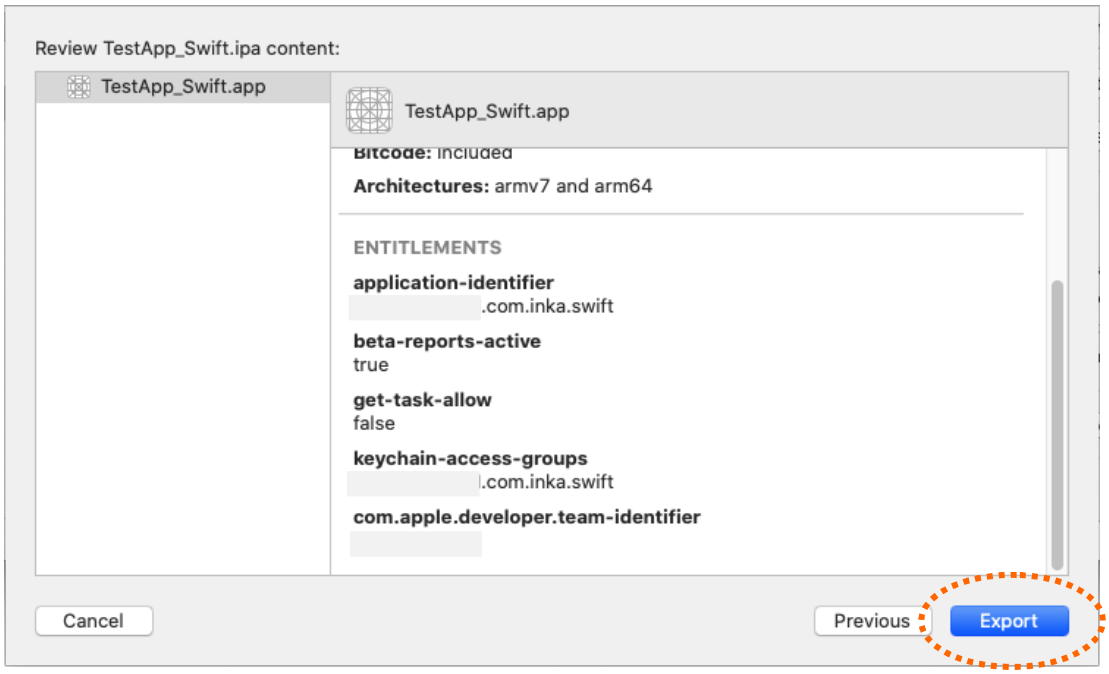
You usually selected "Upload" almost but you **must select "Export" to apply AppSealing**. This is because taking snapshot for app integrity and certificate is needed and your **app will not run normally on device without this process**. Click "Next" button with "Export" is selected.



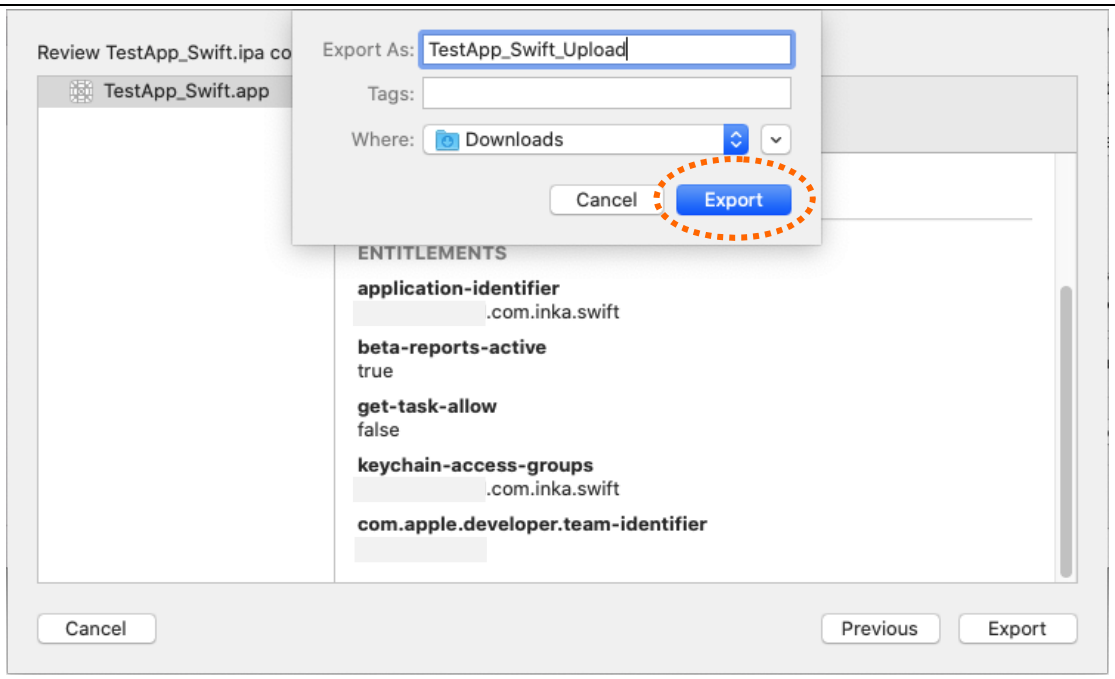
Click "Next" button with all options keep default.



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

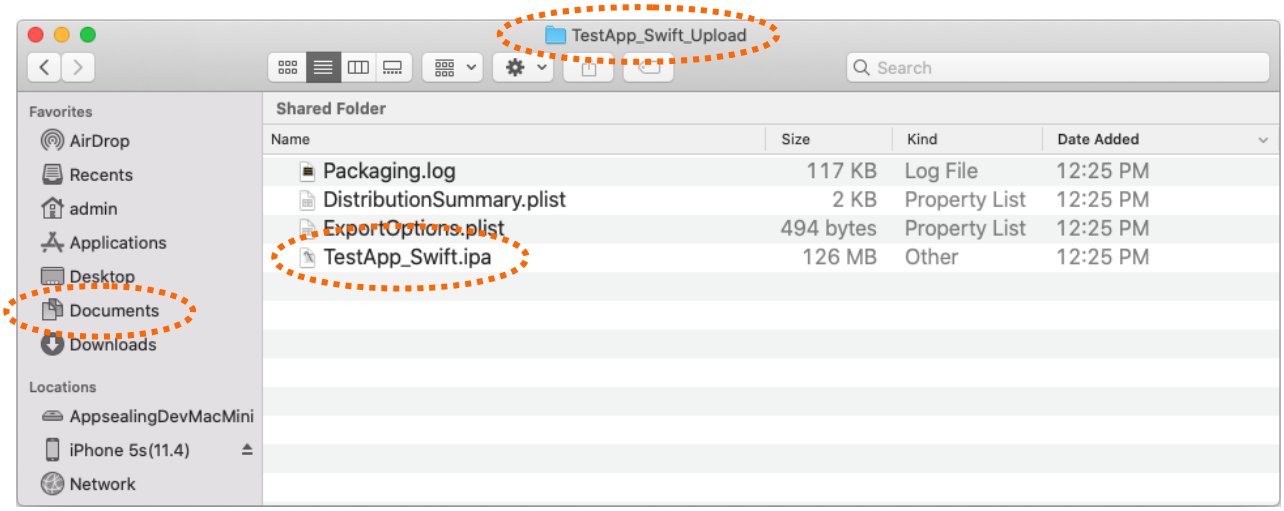


Verify the brief contents and click "Export" button.



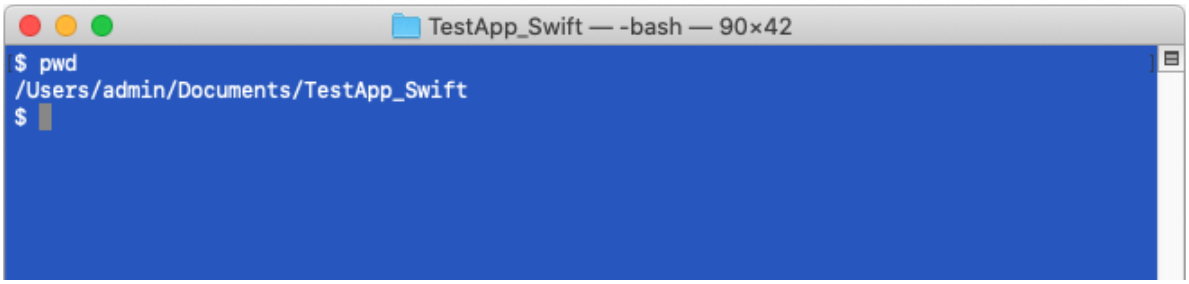
When destination dialog appear select store location and click "Export" button. This document used folder named "~/Downloads/TestApp_Swift_Upload"

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.



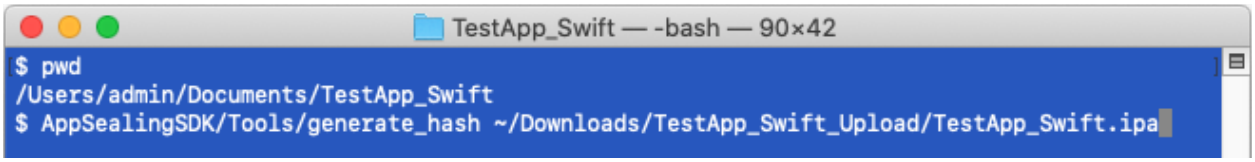
This document used project path for unity-exported Xcode project as "~/Documents/TestApp_Swift". You can verify the path name by pwd command like above picture.

Run add permission command like below.

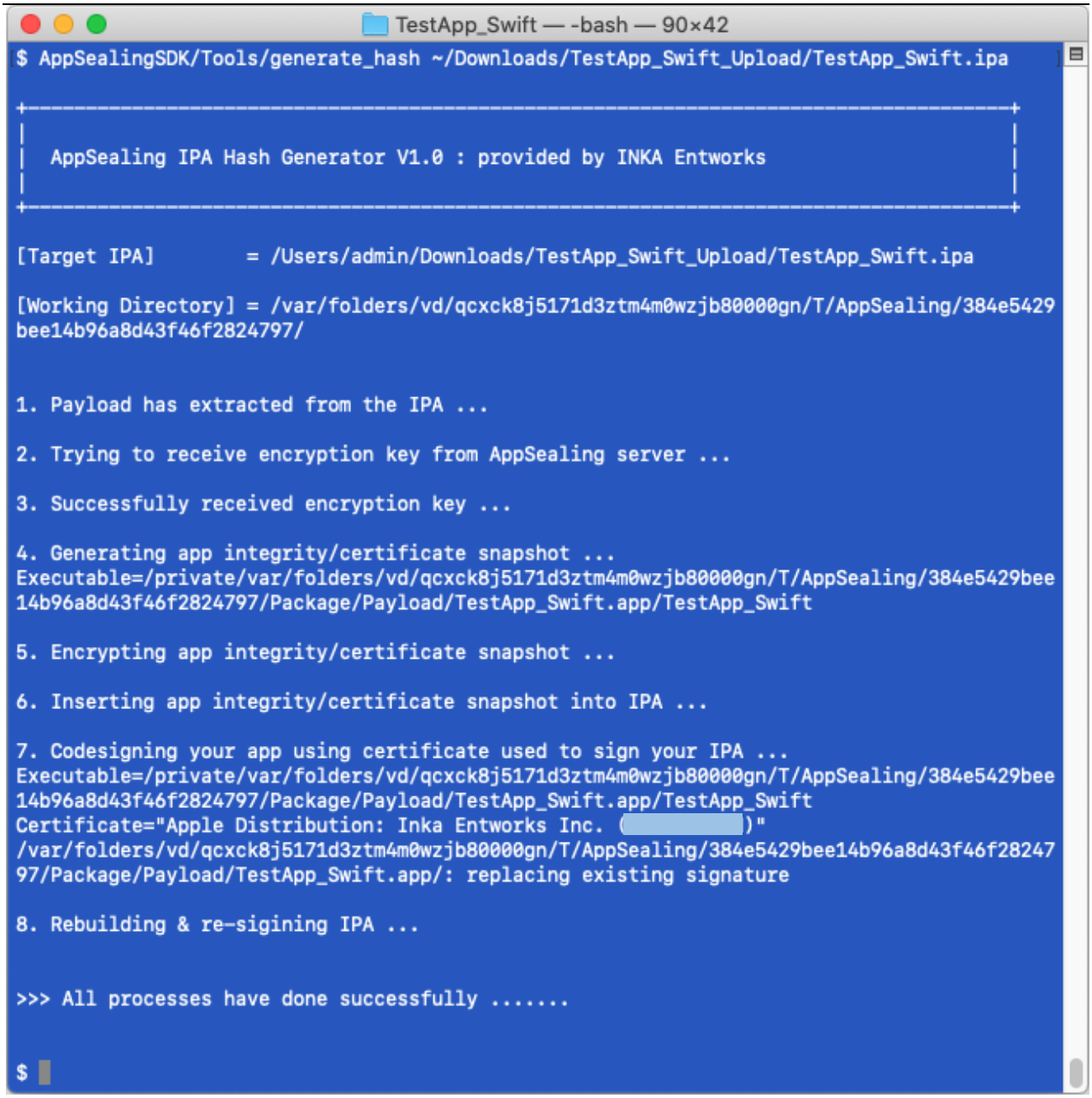
```
$ chmod +x AppSealingSDK/Tools/*
```

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 Controlling anti-swizzling / anti-hooking features

Method swizzling is a technique used primarily in Objective-C to swap the functionality of an existing method of a class with the functionality of a new method, so that when the original method is called, the code of the new method is actually executed. This allows for flexible modification or extension of the behavior of existing code.

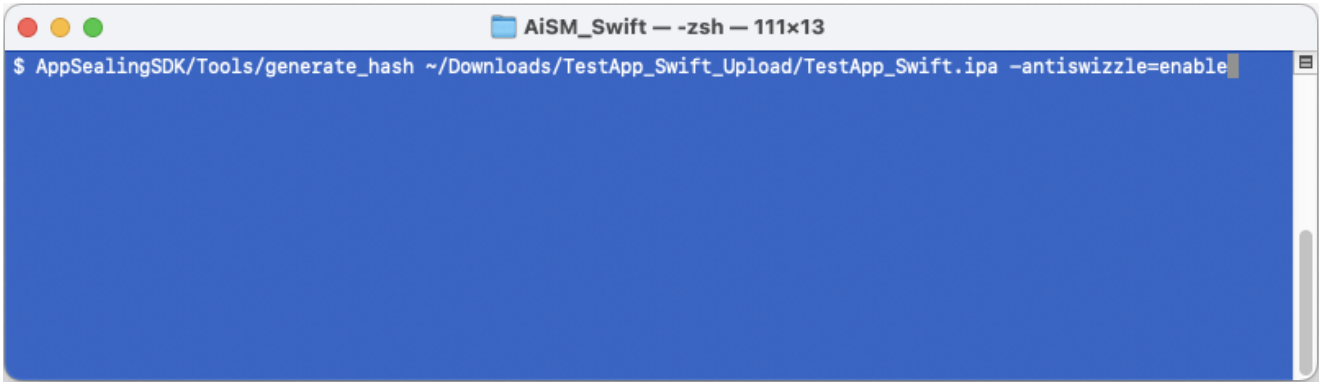
Method swizzling is a useful technique, but it also allows attackers to replace existing methods with their own malicious code. For example, they can modify methods that process user input to steal data or perform unwanted actions.

You can also swizzle the system's logging methods so that important events are not logged, thereby preventing malicious behavior from being detected.

The AppSealing SDK adds the ability to detect this method swizzling behavior, but since method swizzling is not always used as a means of attack, you can specify whether or not you want to detect it.

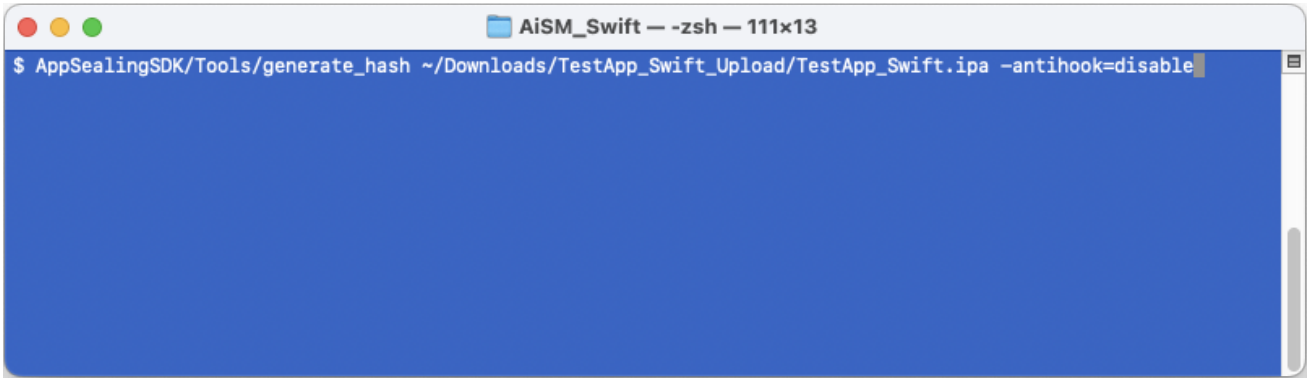
By default, the SDK is distributed with method swizzling detection disabled to prevent malfunctions, but you can enable it when running the generate_hash script. If you receive a security warning during the app testing phase that method swizzling has been detected, you should not enable method swizzling detection, as it may be that another library included in your app is intentionally using method swizzling.

To enable method swizzling detection within script phase, run generate_hash script with the "**antiswizzle=enable**" parameter, as follows:

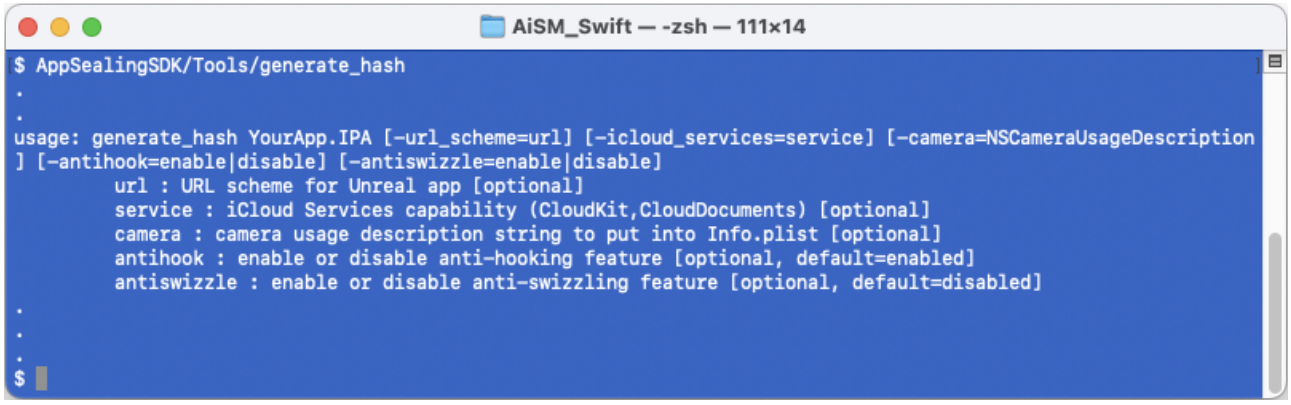


Method hooking is an attack method used with method swizzling. Method hooking is similar to method swizzling, but the difference is that it directly manipulates and controls the execution flow of the method rather than swapping methods. Method swizzling can be used in apps or specific libraries as needed, but method hooking is not used in normal apps and libraries, so the AppSealing SDK has method hooking detection enabled by default, but this feature can be disabled if needed.

To disable method hooking detection, run the generate_hash script with the "**antihook=disable**" parameter as follows:

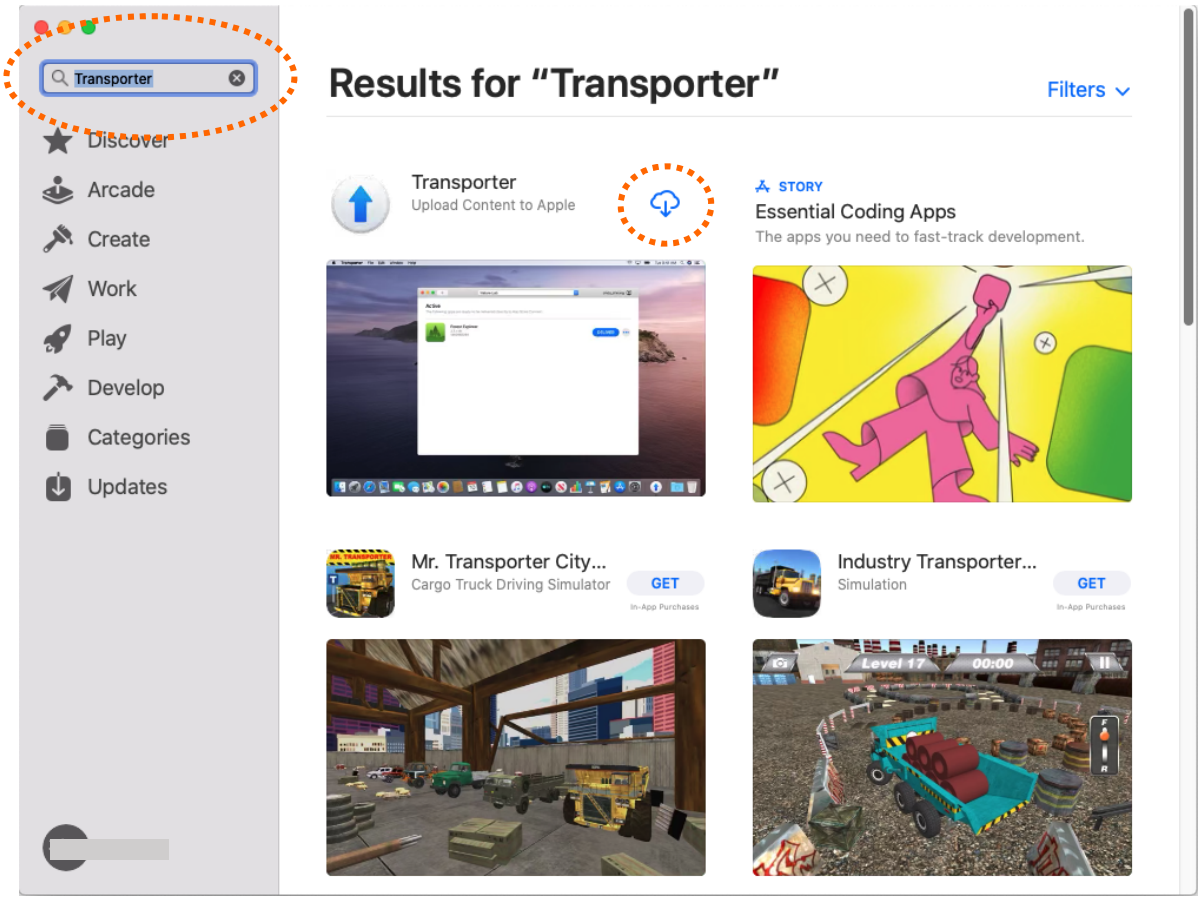


If you want to check all parameters of the generate_hash script, you can run the script without the IPA path name and check all available parameters as shown in the screen below.

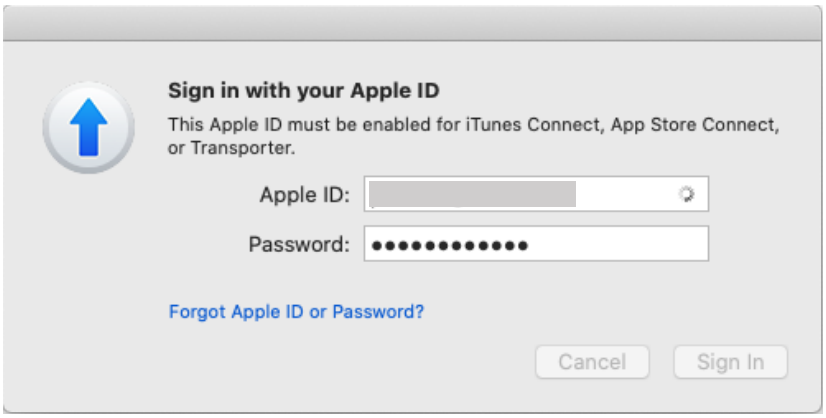


4-7 Upload re-signed IPA to App Store Connect

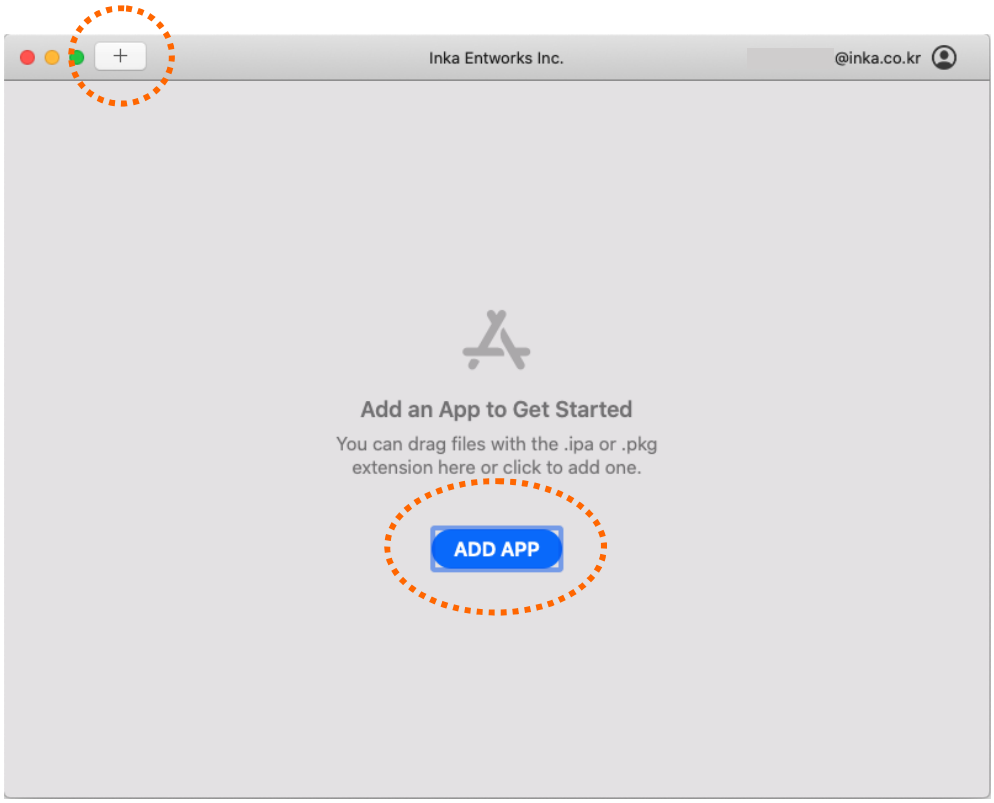
Now you can upload re-signed IPA to App Store Connect. (Of course, if you use Ad Hoc distribution by AdhocEnabled SDK, this step is not required.) This document uses Transporter app (MAC) for convenient uploading. If the Transporter app has not installed in your MAC you can open Mac AppStore, search "Transporter" and install.

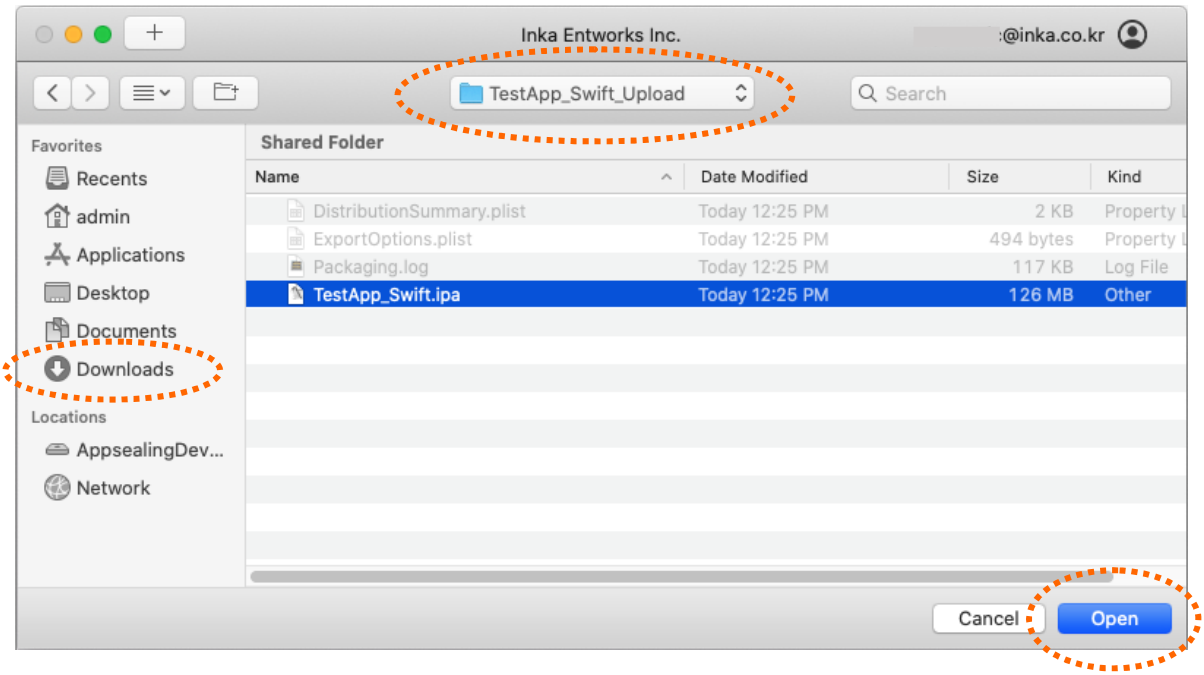


Launch Transporter after installation you are requested for Apple ID like below. Enter your Apple ID and password. (This step is required only once for the first time)

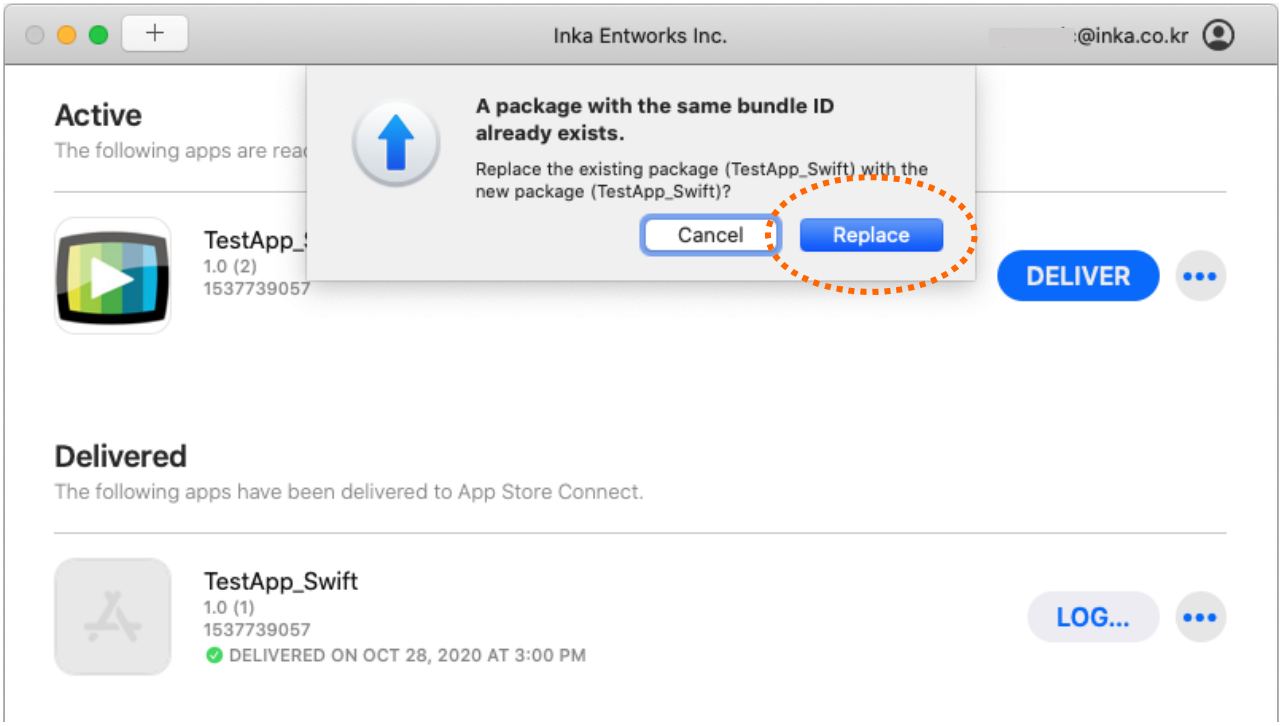


After you login with your ID and password you can see the Transporter window like below. Click the "+" button upper-left or "ADD APP" button in the middle to select IPA to be uploaded and select the re-signed IPA in previous step.

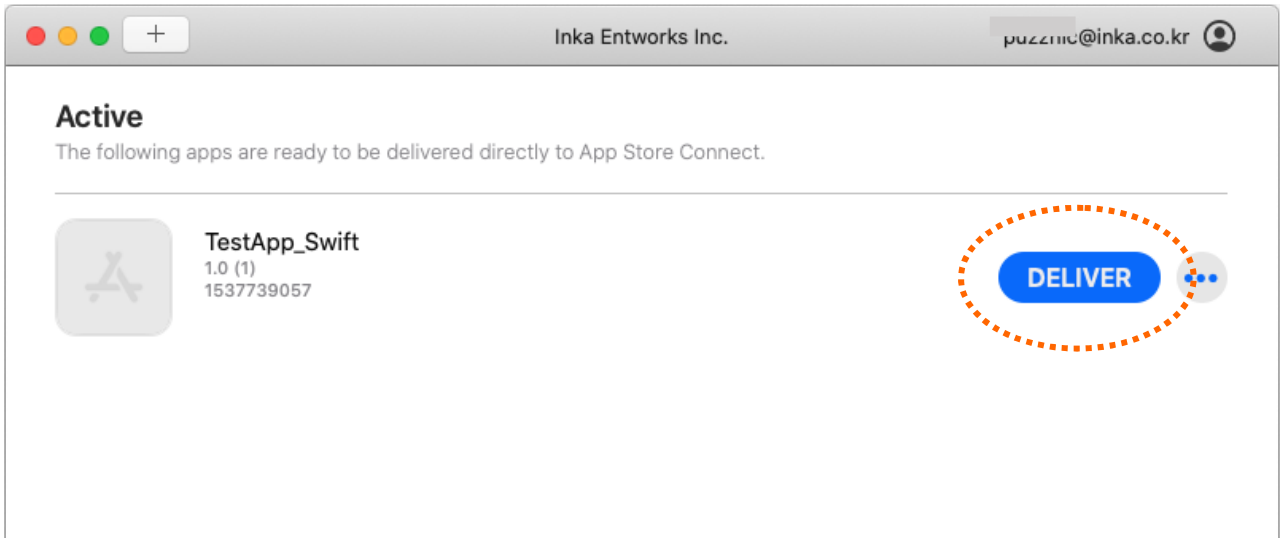


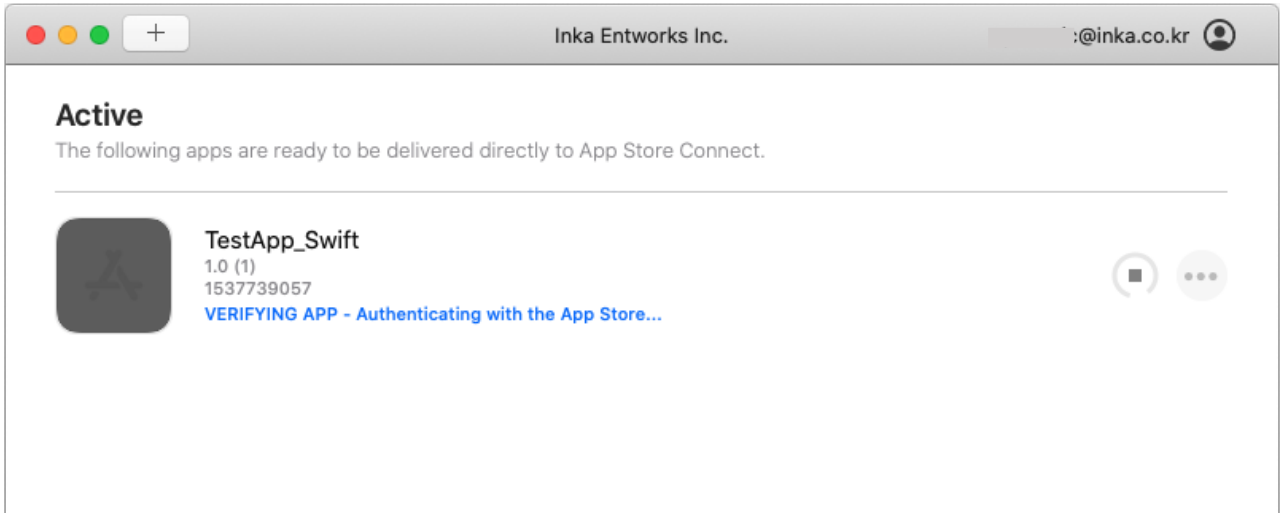


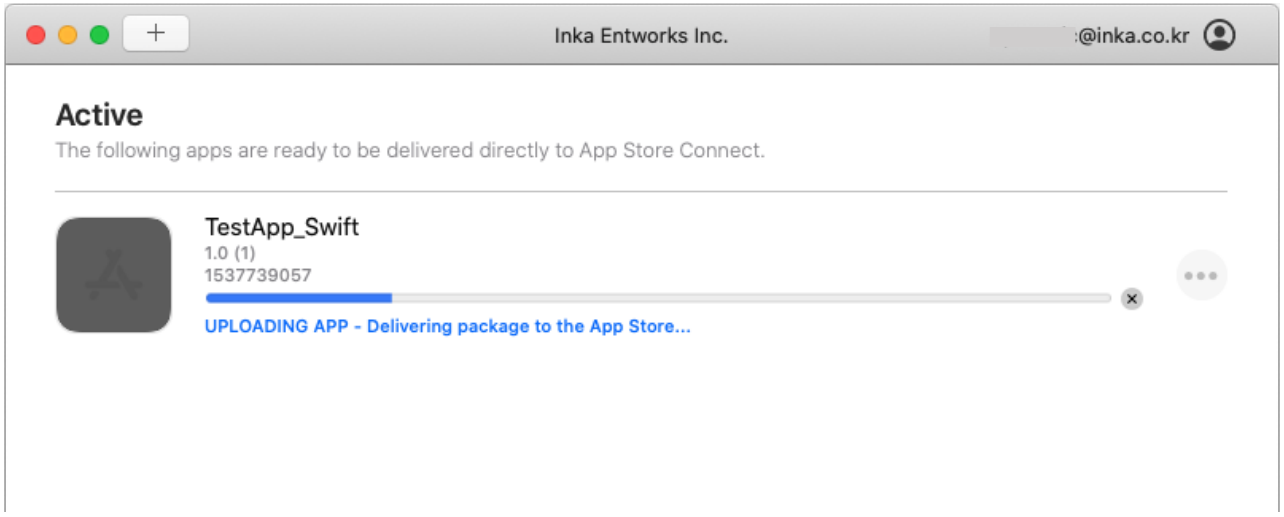
When you update your app by adding IPA file with new version or higher build number a warning dialog can appear like below because of same bundle ID. In this case just click "Replace" button to upload new IPA.

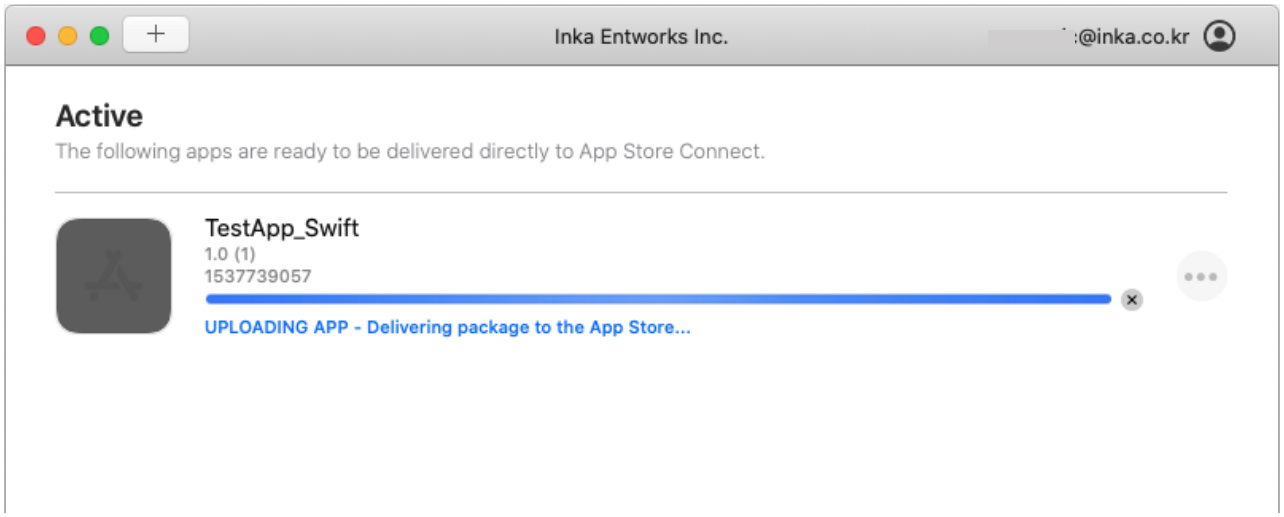


After IPA file has added, click "DELIVER" button then verifying and uploading to App Store Connect process will be in progress.

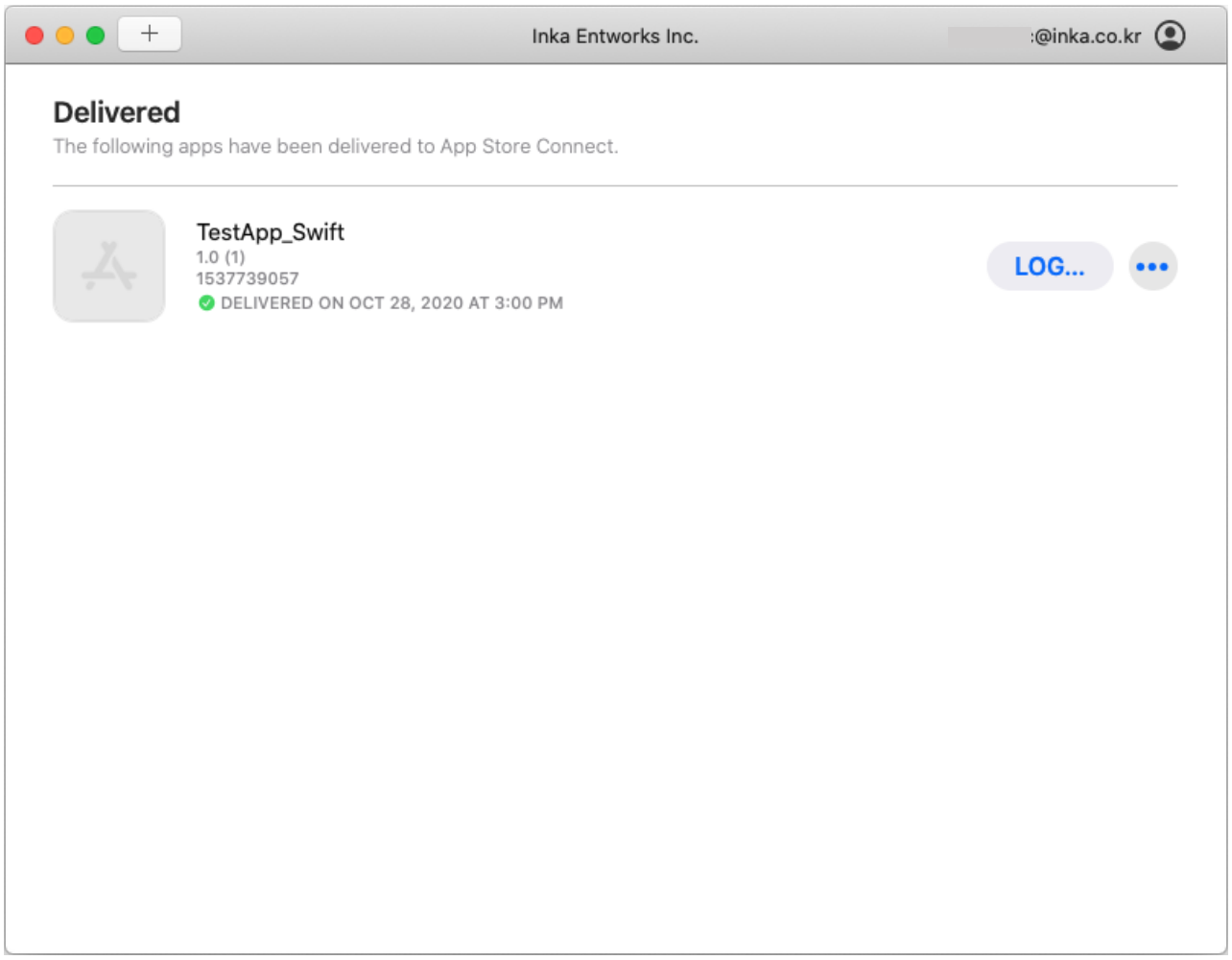








If you encounter below window the upload process has finished and you can submit your build for App Store review or TestFlight distribution.



Part 5. Acquire AppSealing device unique identifier

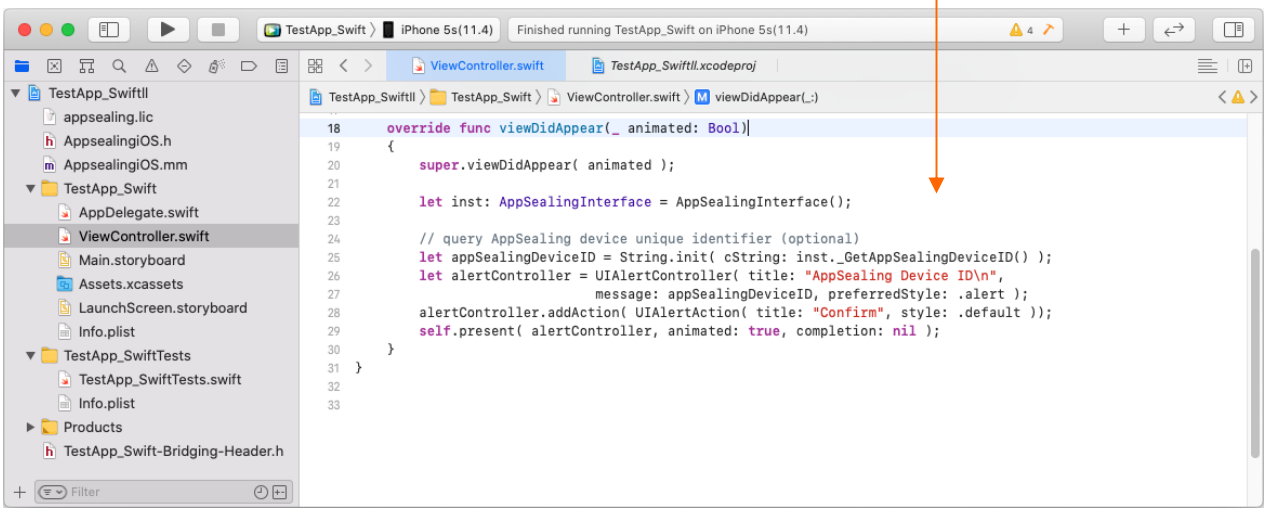
AppSealing SDK generates and manages unique identifier for each device. Customer who use the AppSealing SDK can use the interface of AppSealing to verify the device unique identifier, if necessary. And can be used for the business using the hacking data service provided by AppSealing.

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 );
}
```

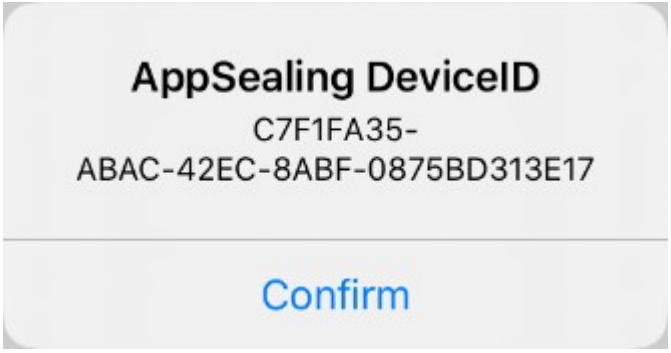


If your project is Objective-C based then you can use following code to show simple UI.

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

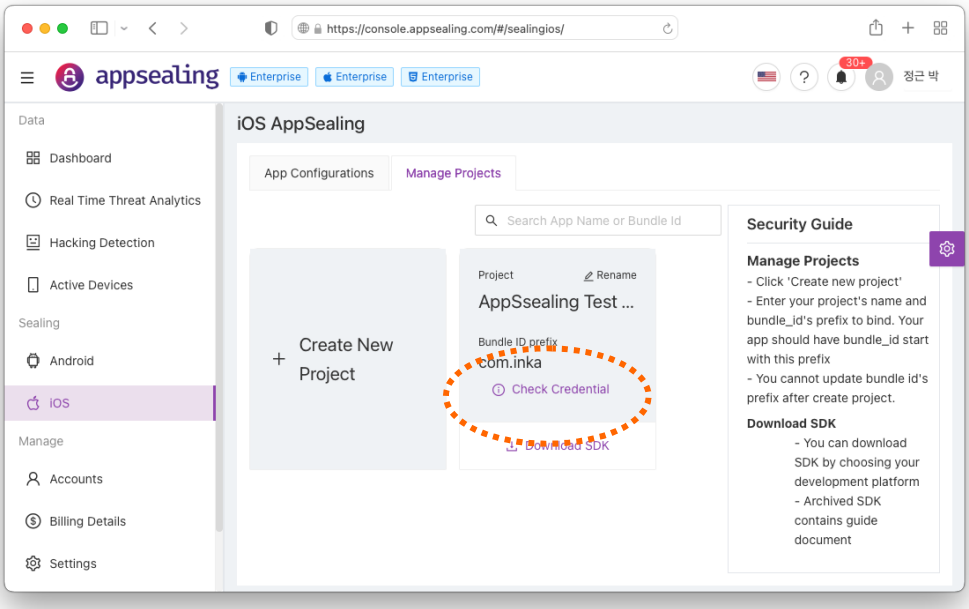
If your server fails to validate credential, you should also force the login to fail and the app to not proceed further. However, since code such as checking the login result and closing the app is likely to be tampered by an attacker, the best practice is configuring your server to deny service or response 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

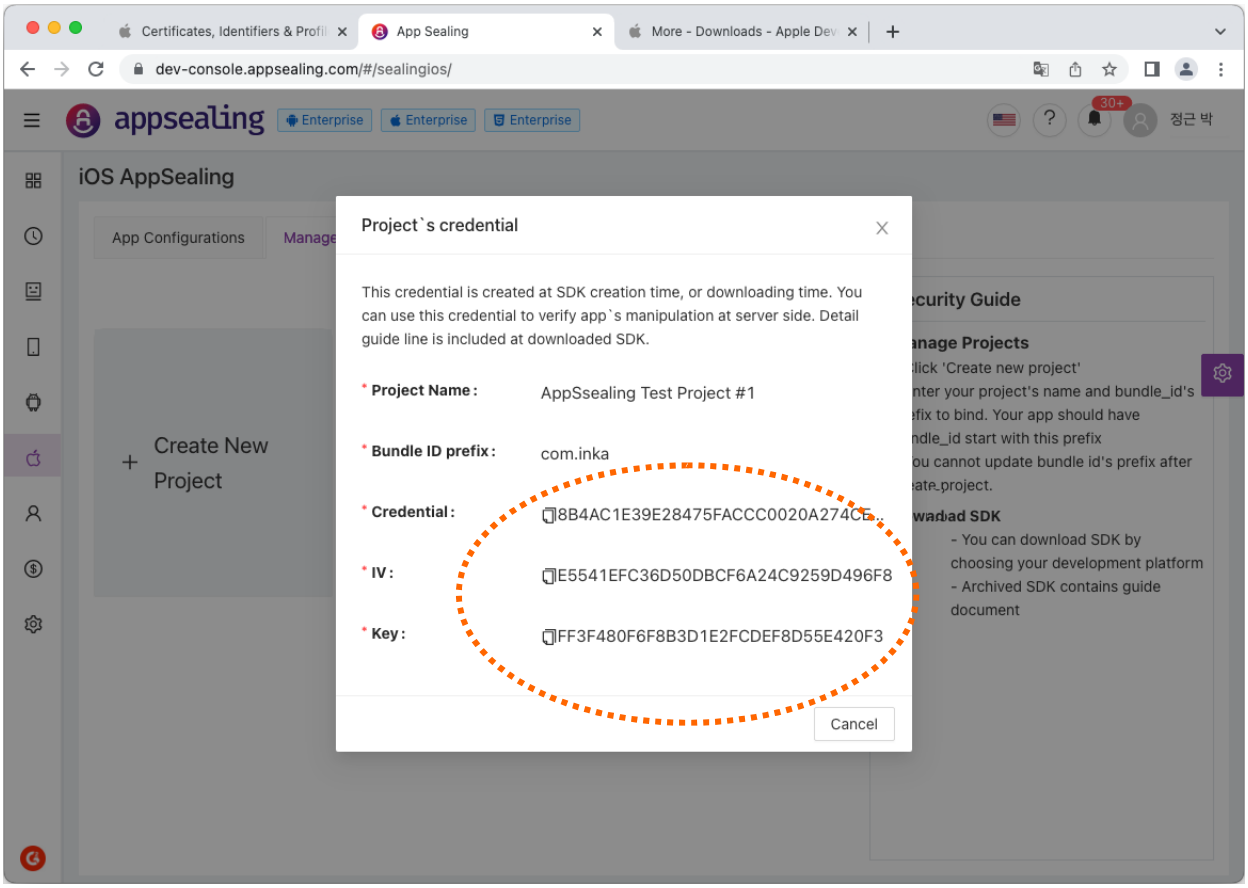
The credential data (hex string) returned from the interface call to the AppSealing module is only valid when the security logic inside AppSealing is normally performed and no dangerous situation is detected in the device.
If code patch attack is made through the dynamic library or the security logic 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.

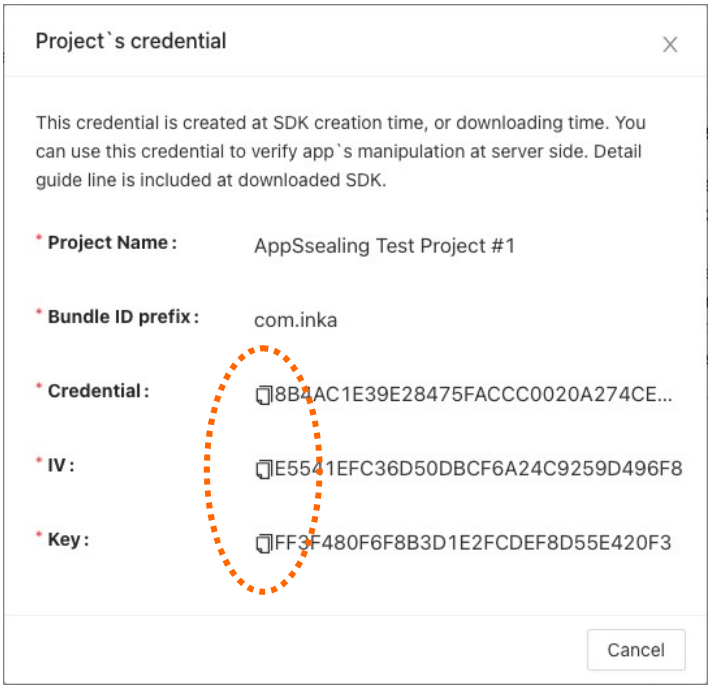
[Preparation]

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.

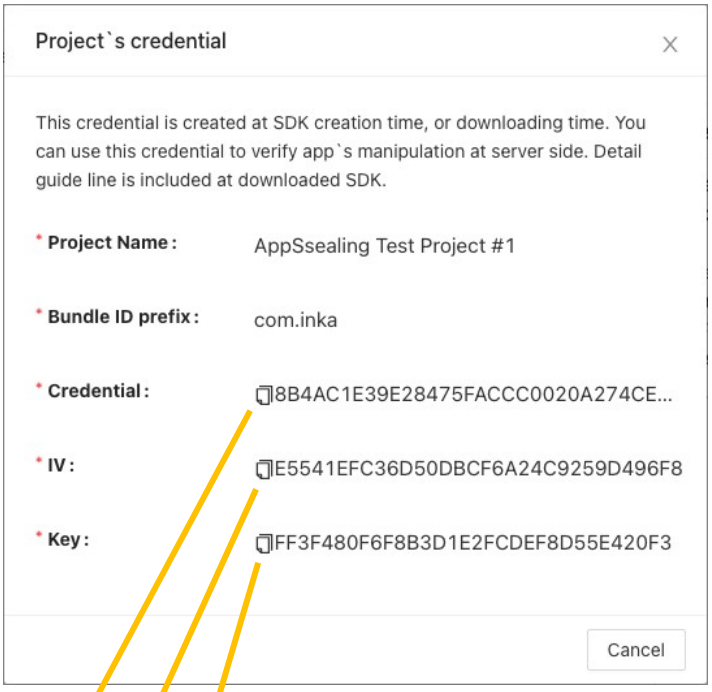




[In case your server code is using Node.js/Javascript]

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
    {
```



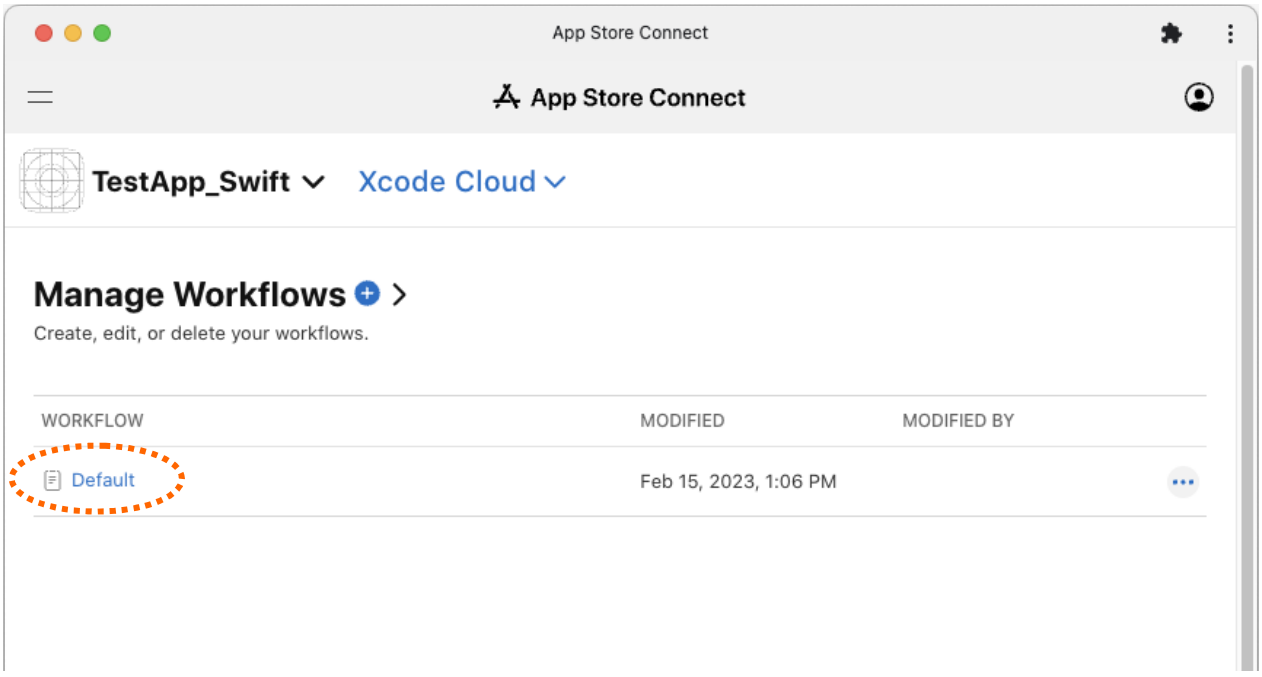
```
        memcpy( decrypted_buffer, HexToBytes( std::string( credential ).substr( 0, 32 )).data(), 16 );

        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. Resign and upload from Xcode Cloud

AppSealing SDK can be applied to projects using Xcode Cloud. After adding the necessary files for the SDK and pushing them to the Git repository, the app is archived and exported as an IPA in the way Xcode Cloud works, and then the IPA with security-related features added is uploaded to App Store Connect by performing a re-signing script. will do.
However, in this process, some actions must be performed differently from the original Xcode Cloud operation, so additional settings and preparations are required. Follow the steps outlined below to prepare the process.

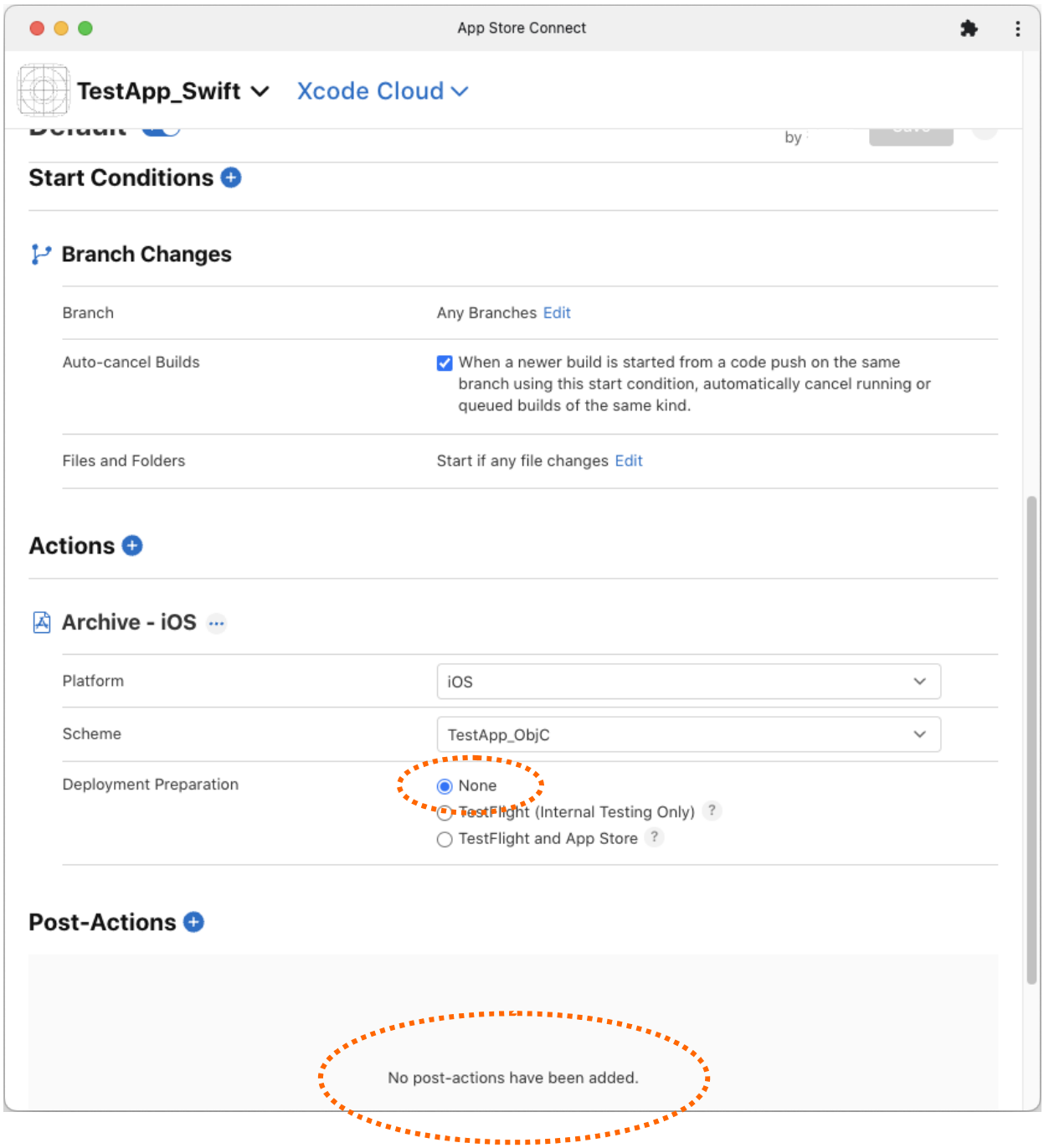
7-1 Configure Xcode Cloud

Integration of Xcode Cloud and project Git repository is assumed to be completed as described in the Xcode Cloud documentation. First you need to change the workflow settings in Xcode Cloud. The project name used in this document is "TestApp_Swift" and the workflow name is "Default". Connect to "App Store Connect" and go to the "Xcode Cloud" page of the project.



Click on the workflow name to go to the settings screen.

When the setting screen is displayed, leave other items as they are, scroll the page to the bottom, select "None" as the value of "Actions - Deployment Preparation" and leave "Post-Actions" blank so that no action is entered.

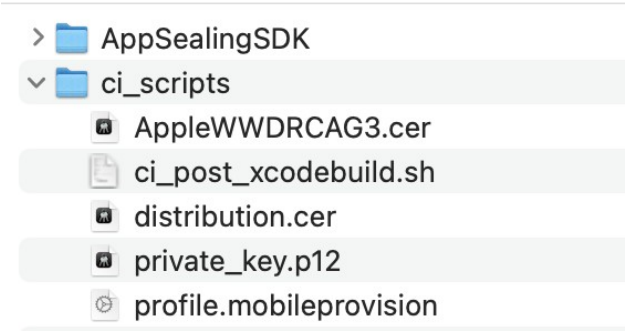


If you do not select "None" as the value of the "Deployment Preparation" item, the app will not run normally because the built IPA is uploaded to App Store Connect without going through the re-signing process. The same goes for Post-Actions, so these two must be left blank or set to "None"

7-2 ci_scripts preparation process

To re-sign the built IPA with security settings file in the Xcode Cloud environment, a custom script file, a signing certificate & private key, and a provisioning profile for distribution are additionally required to upload the modified IPA to App Store Connect.
All of these files should be included in the ci_scripts folder of your project path. In addition to the ci_scripts configuration items provided by the AppSealing SDK, the following items are additionally required and added by yourself.

- Cerfiticate for App Store distribution : distribution.cer (CER format) or distribution.p12 (PKCS#12 format, password: "" [empty])
- Private key for distribution certificate : private_key.p12 (password: '' [empty], needed only when certificate is in CER format)
- Provisioning profile for App Store distribution : profile.mobileprovision
- Apple ID & App-Specific password (to be written in script)



The ci_scripts configuration of the AppSealing SDK includes the Apple root certificate "AppleWWDRCAG3.cer" certificate file and "ci_post_xcodebuild.sh" ruby script file. These files should not be renamed either.

The three previously mentioned files (distribution.p12 or distribution.cer/ private_key.p12, profile.mobile provision) are added to these basic files, resulting in a total five files as shown in the picture above.

Certificate for App Store distribution

The ci_scripts folder must contain a distribution certificate. This certificate must be a CER format file downloaded from the Apple Developer site or a PKCS#12 format file containing both the certificate and the private key, and the file name must be fixed as "distribution.cer" or "distribution.p12." When creating a certificate in PKCS#12 format, the password must be an empty string, and if a password is specified, the script must be modified accordingly. If a certificate other than the App Store distribution certificate is added or the file name is different, re-signing will not be performed properly.

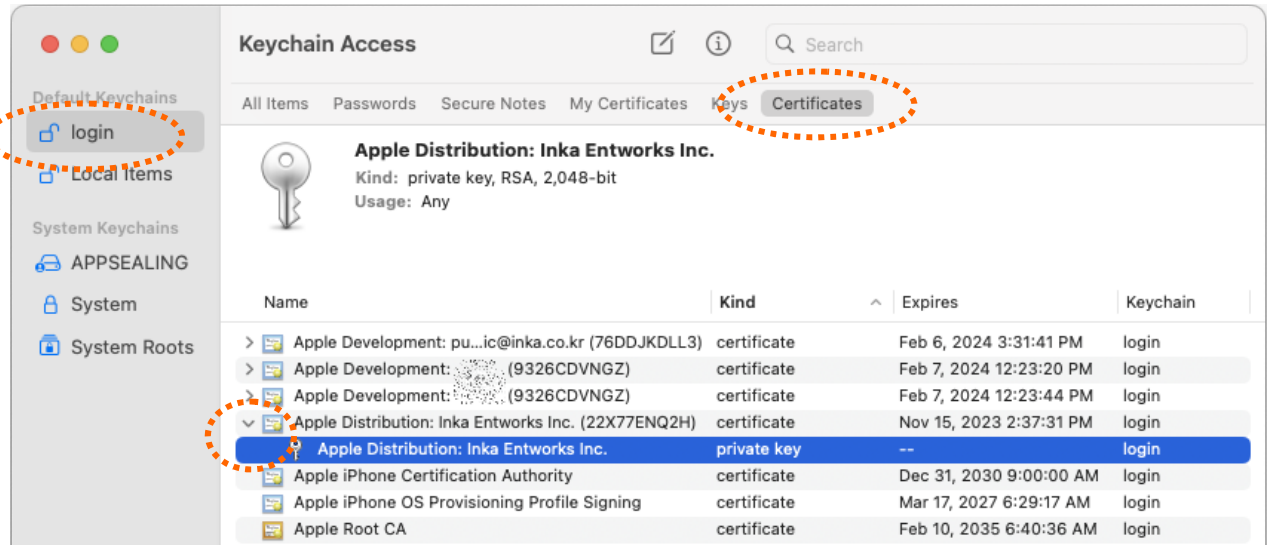
Private key for distribution certificate

If the distribution certificate included above is in CER format (distribution.cer), the corresponding private key file must also be included in the ci_scripts folder. The private key file can be exported from the Mac Keychain app, and the format must be set to PKCS#12 (extension .p12), and the password for the private key file must be left blank as an empty string. The file name of the private key file must also be fixed as "private_key.p12".
If the file name does not match or a non-empty value is set in the password, re-signing will not proceed properly. If you want to set a password in the private key file, proceed with additional steps in the script modification section below.
Below is a step-by-step explanation of the private key file creation process. If you have already created a private key file, skip these instructions.

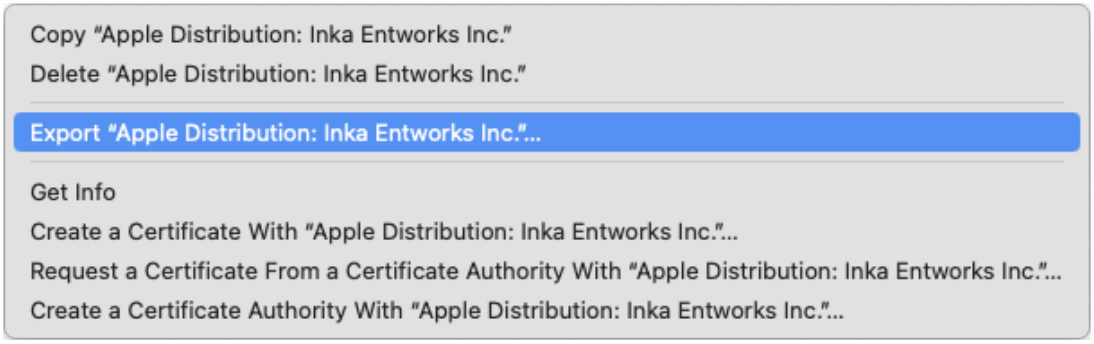
STEP Private key file creation process in Mac Keychain app

If you go to "Etc." in Mac's launchpad, the "Keychain Access" app will appear. Run this app and when the screen below appears, select the "login" item from the left tab.

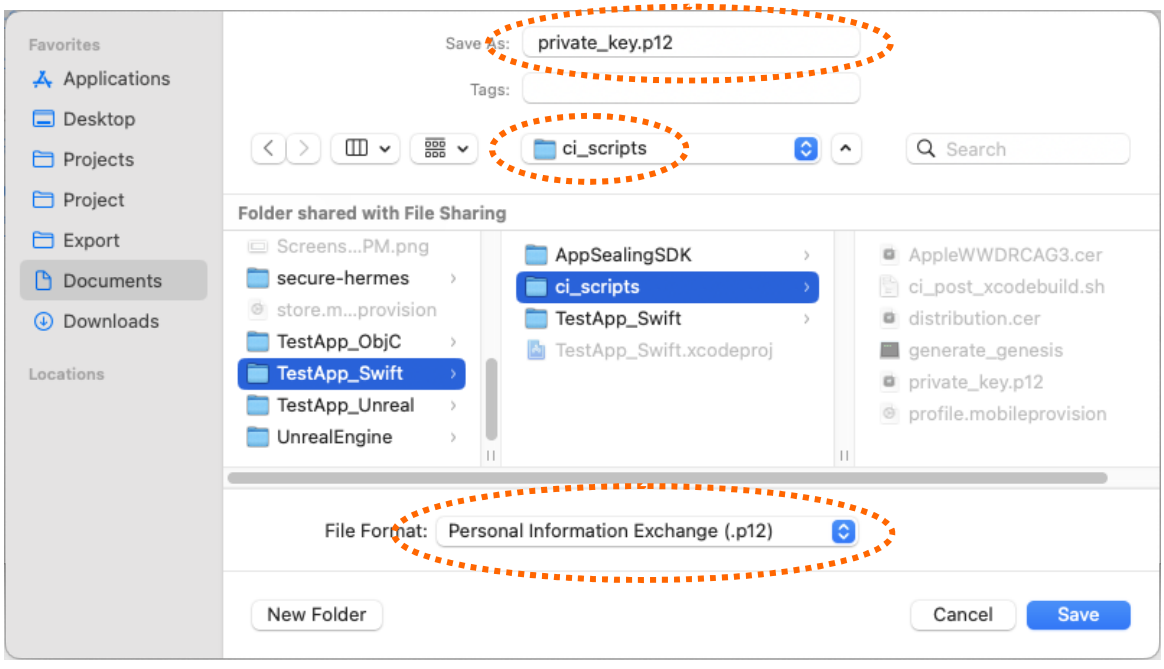
Select "Certificates" from the top tab menu in the right pane.
Click the right-angle bracket (>) to the left of the distribution certificate registered in the keychain to display the private key of the certificate.



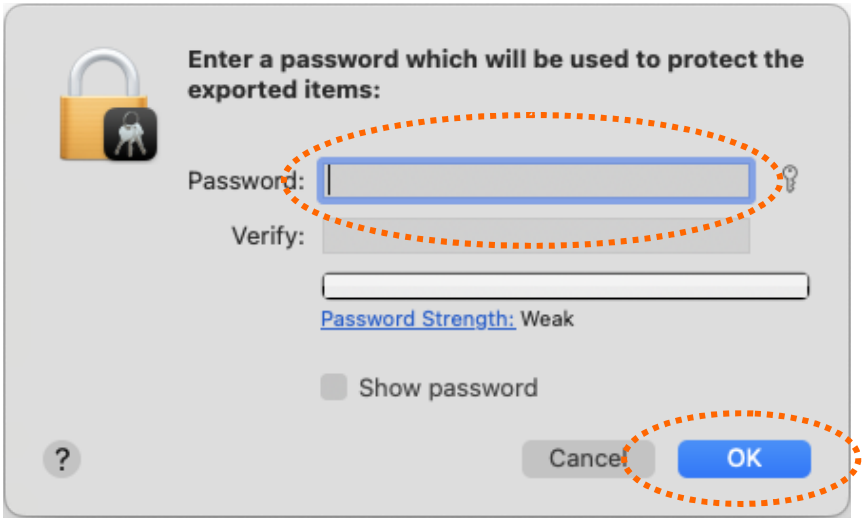
After selecting the private key, right-click the mouse and when the context menu appears, select "Export... " item.



Select the "ci_scripts" folder of the project as the save location, the file name as "private_key.p12", and the file format as "Personal Information Exchange."



When the password input window appears, leave the password blank and save it.



If you specify a password, you must edit the ci_post_xcodebuild.sh file.
Open the file with an editor and go to line 445. You can see that the password for the private key file is left blank in this line of commands.

```
441    system( 'security create-keychain -p 0000 ' + APPSEALING_KEYCHAIN )
442    system( 'security list-keychains -d user -s login.keychain ' + APPSEALING_KEYCHAIN )
443    system( 'security import ./AppleWWDRCAG3.cer -k ' + APPSEALING_KEYCHAIN + ' -t cert -A -P ""' )
444    system( 'security import ./distribution.cer -k ' + APPSEALING_KEYCHAIN + ' -t cert -A -P ""' )
445    system( 'security import ./private_key.p12 -k ' + APPSEALING_KEYCHAIN + ' -t priv -A -P ""' )
446    system( 'security default-keychain -d user -s ' + APPSEALING_KEYCHAIN )
447    system( 'security unlock-keychain -p 0000 ' + APPSEALING_KEYCHAIN )
448    system( 'security set-keychain-settings ' + APPSEALING_KEYCHAIN )
449    system( 'security set-key-partition-list -S apple-tool:,apple:,codesign: -s -k 0000 ' + APPSEALING_KEYCHAIN + ' > /dev/null' )
```

Write down the password you specified and save the file in this location. If you saved your password as the string "your_password", you must modify it as follows.

```
441    system( 'security create-keychain -p 0000 ' + APPSEALING_KEYCHAIN )
442    system( 'security list-keychains -d user -s login.keychain ' + APPSEALING_KEYCHAIN )
443    system( 'security import ./AppleWWDRCAG3.cer -k ' + APPSEALING_KEYCHAIN + ' -t cert -A -P ""' )
444    system( 'security import ./distribution.cer -k ' + APPSEALING_KEYCHAIN + ' -t cert -A -P ""' )
445    system( 'security import ./private_key.p12 -k ' + APPSEALING_KEYCHAIN + ' -t priv -A -P "your_password"' )
446    system( 'security default-keychain -d user -s ' + APPSEALING_KEYCHAIN )
447    system( 'security unlock-keychain -p 0000 ' + APPSEALING_KEYCHAIN )
448    system( 'security set-keychain-settings ' + APPSEALING_KEYCHAIN )
449    system( 'security set-key-partition-list -S apple-tool:,apple:,codesign: -s -k 0000 ' + APPSEALING_KEYCHAIN + ' > /dev/null' )
```

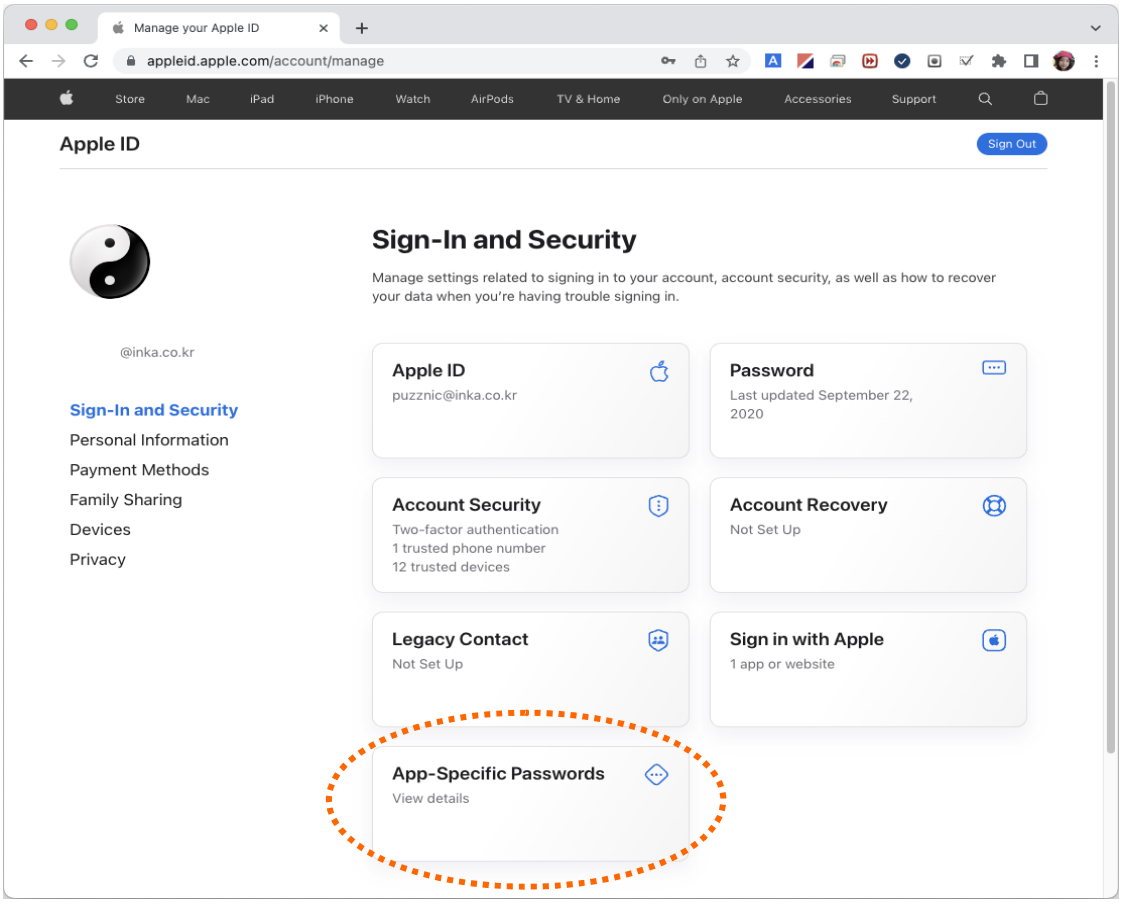

Provisioning profiles for App Store distribution

The ci_scripts folder must contain a provisioning profile for App store deployment. This file should contain the profile downloaded from the Apple Developer site with the name "profile.mobileprovision". If the file name does not match or the profile is not correct, re-signing may fail or the app may not install/run properly.

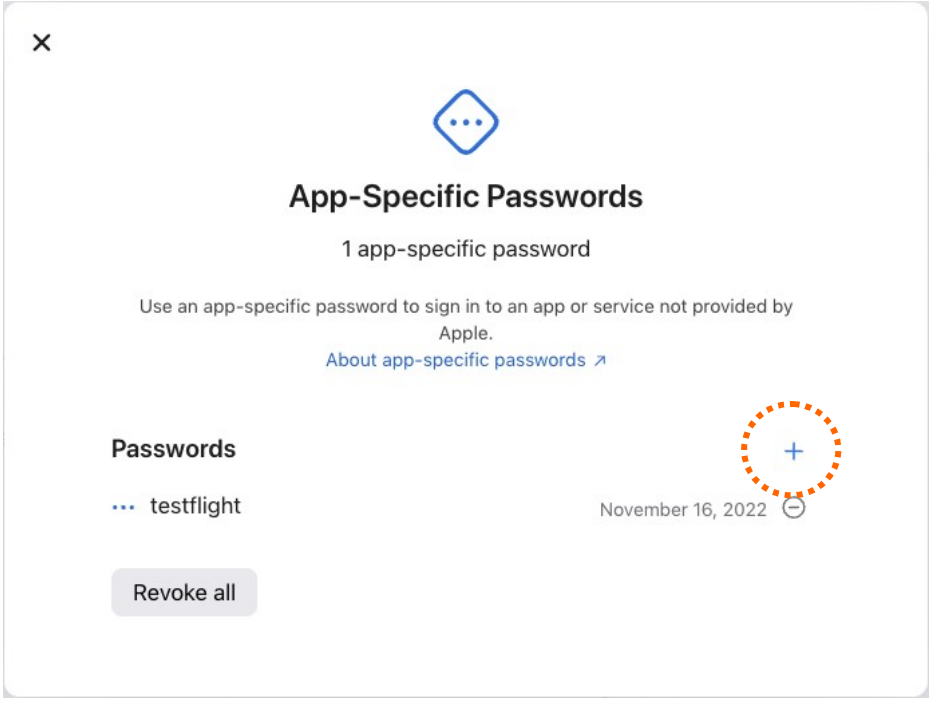
Custom scripts for post-build processing

The ci_scripts folder contains the ci_post_xcodebuild.sh script file by default. The content of the script includes a process for uploading the re-signed IPA to App Store Connect. To perform this operation, the app developer's Apple account and app password are required. (Not password for Apple account ID!)

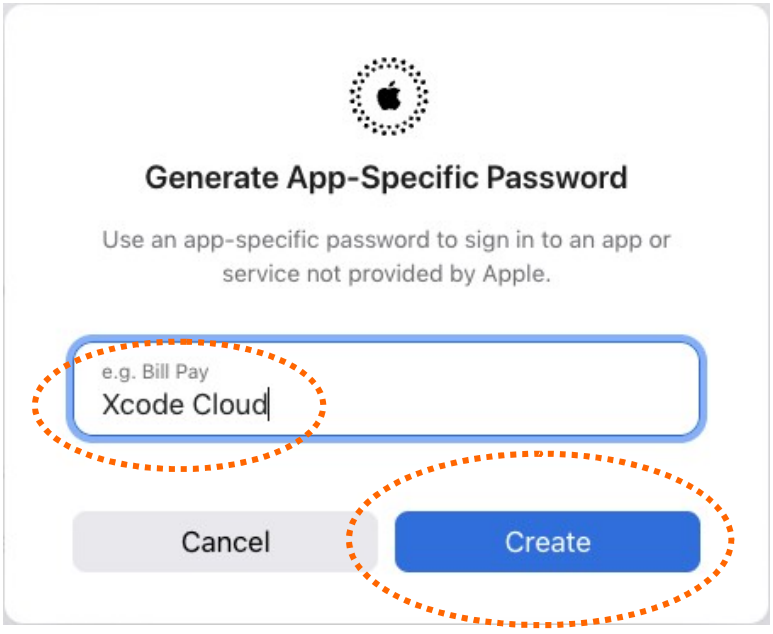
If you do not have an app password, you can create one on below Apple's page.
https://appleid.apple.com/account/manage



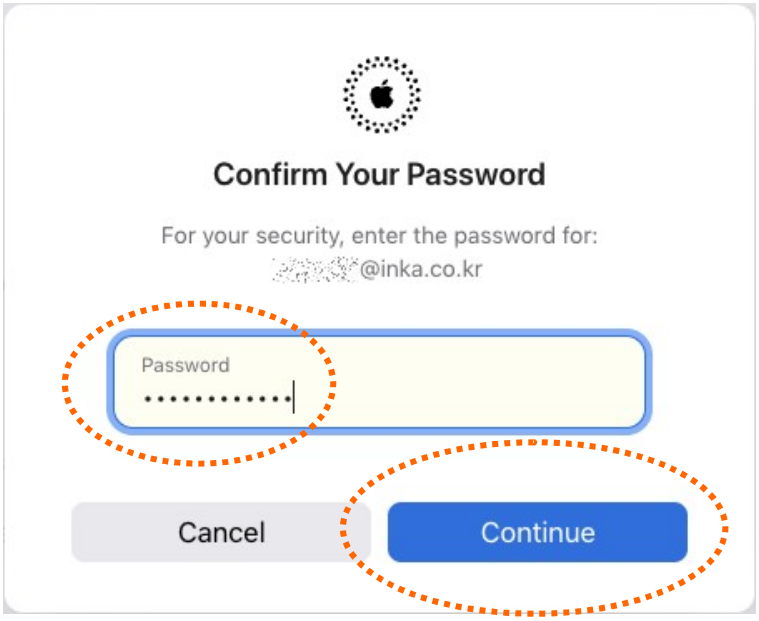
When you select the "App-Specific Passwords" item of the "Sign-In and Security" page, the "App-Specific Passwords" window will appear as shown below. Here, click the + button to start creating a new app password.



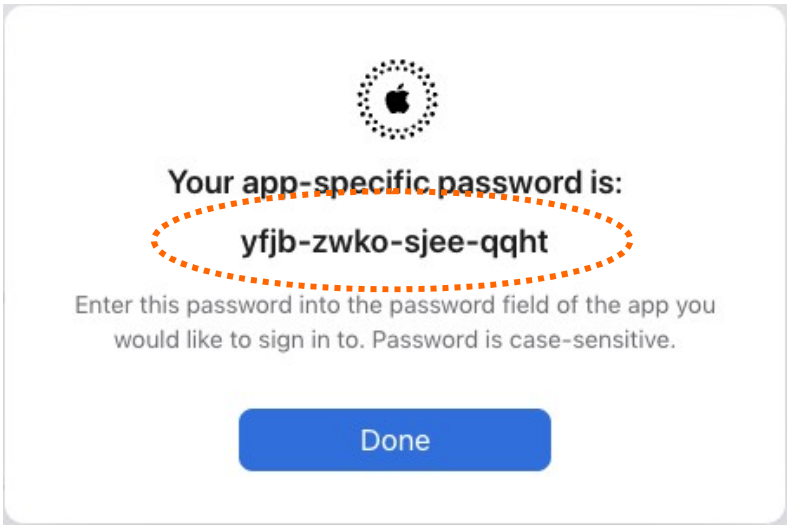
When you click the + button, a window will appear where you can enter the name of the app password. Please enter a name that is easy to recognize and suit the purpose. Here we arbitrarily entered the name "Xcode Cloud". Now click on the "Create" button.



Now, when a window asking you to enter your Apple account password appears, enter the account password of your Apple ID and click the "Continue" button.



You will now see your newly created 16-digit app password as shown in the window below. Passwords created in this way cannot be verified later and can only be discarded, so they must be well recorded.



Now, write your Apple account and app password in lines 20 to 21 at the top of the ci_post_xcodebuild.sh file and save the file.

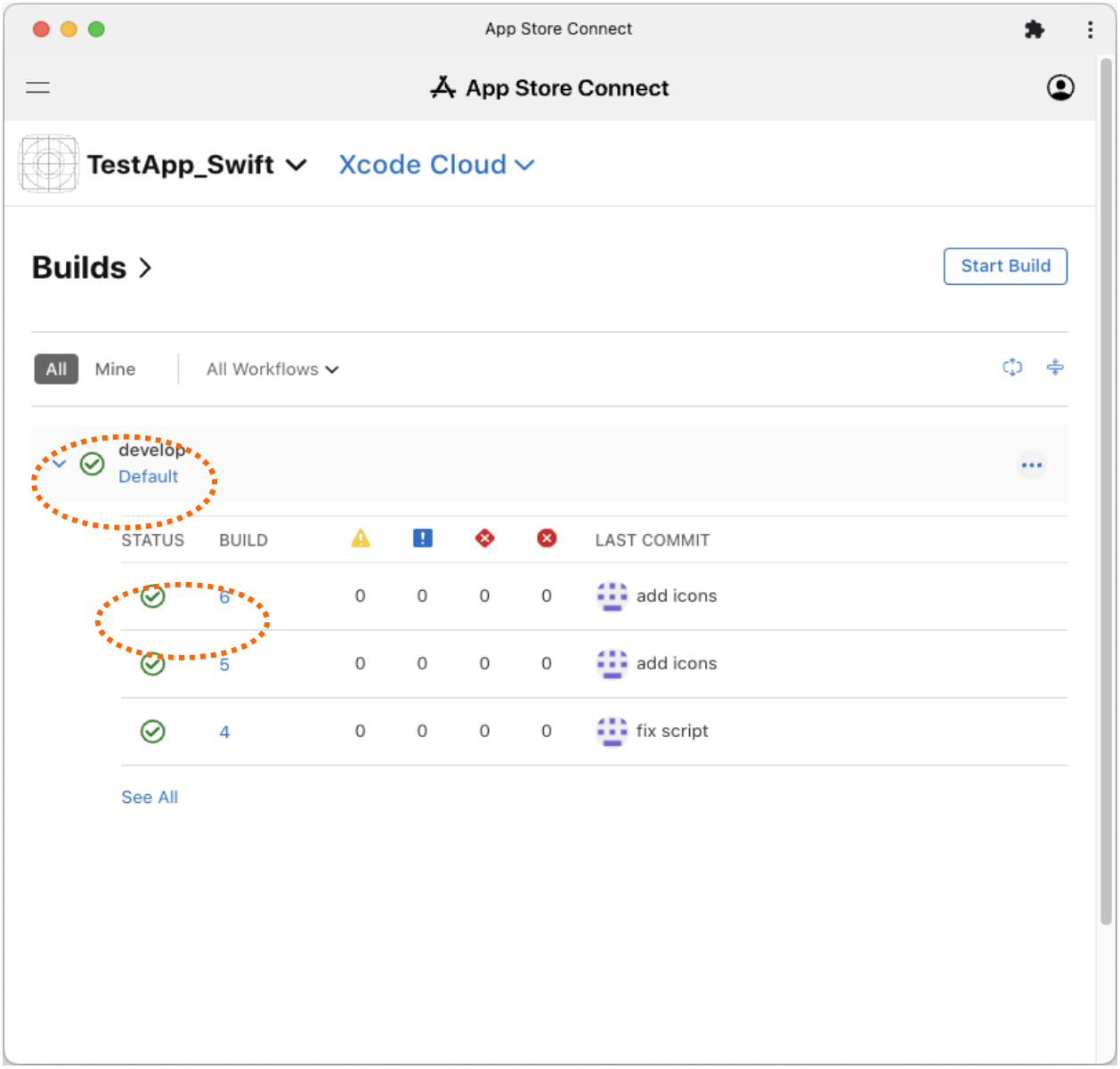
```
1  #!/usr/bin/env ruby
2
3  =begin
4  ================================================================================
5  |  AppSealing iOS SDK Hash Generator V1.2.2                                    |
6  |                                                                              |
7  |  * presented by Inka Entworks                                                |
8  ================================================================================
9  =end
10
11 require 'pathname'
12 require 'tmpdir'
13 require 'securerandom'
14 require 'net/https'
15 require 'json'
16 require 'io/console'
17 require 'open-uri'
18
19 #-------------------------------------------------------------------------------------------------------------
20 APPLE_ID = "support@inka.co.kr"                  # replace with your apple developer ID
21 APPLE_APP_PASSWORD = "aaaa-bbbb-cccc-dddd"      # replace with your apple application password (https://applei
22                                                  # NOT ACCOUNT PASSWORD !
23 #-------------------------------------------------------------------------------------------------------------
24
```

If the Apple account is "myappleid@company.com" and the issued app password is "yfjb-zwko-sjee-qqht", lines 20 to 21 of the script file should be changed as follows.

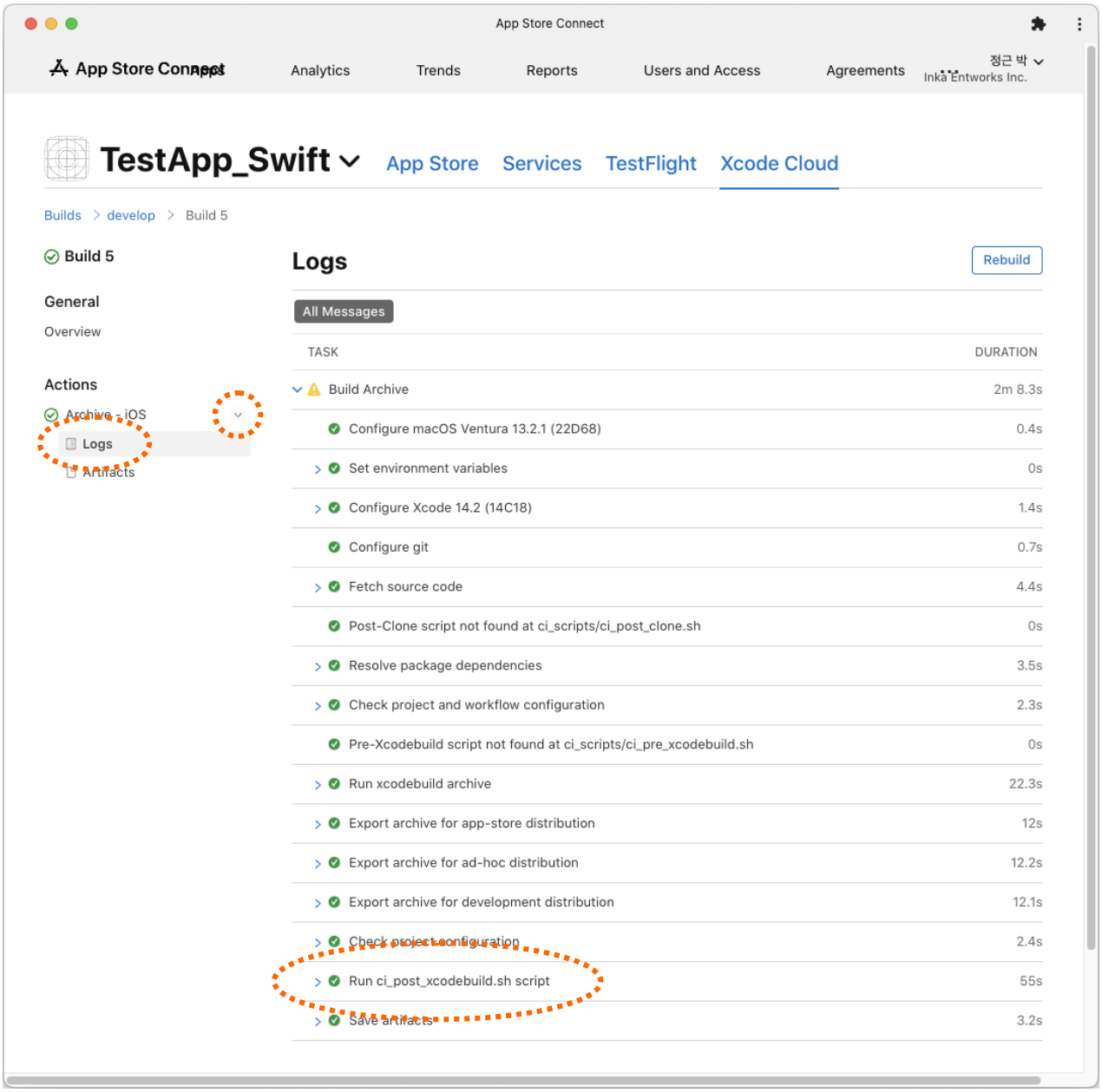
```
13 #-------------------------------------------------------------------------------------------------------------
14 APPLE_ID = "myappleid@company.com"          # replace with your apple developer ID
15 APPLE_APP_PASSWORD = "yfjb-zwko-sjee-qqht" # replace with your apple application password
16                                            # NOT ACCOUNT PASSWORD !
17 #-------------------------------------------------------------------------------------------------------------
```


7-3 Check the build and upload process

Once you have completed setting up the Xcode Cloud environment and preparing the files required for the ci_scripts folder, push the files added to ci_scripts to the git repository. Now connect to App Store Connect and confirm that the build of the target branch (in this document, develop) has been completed.



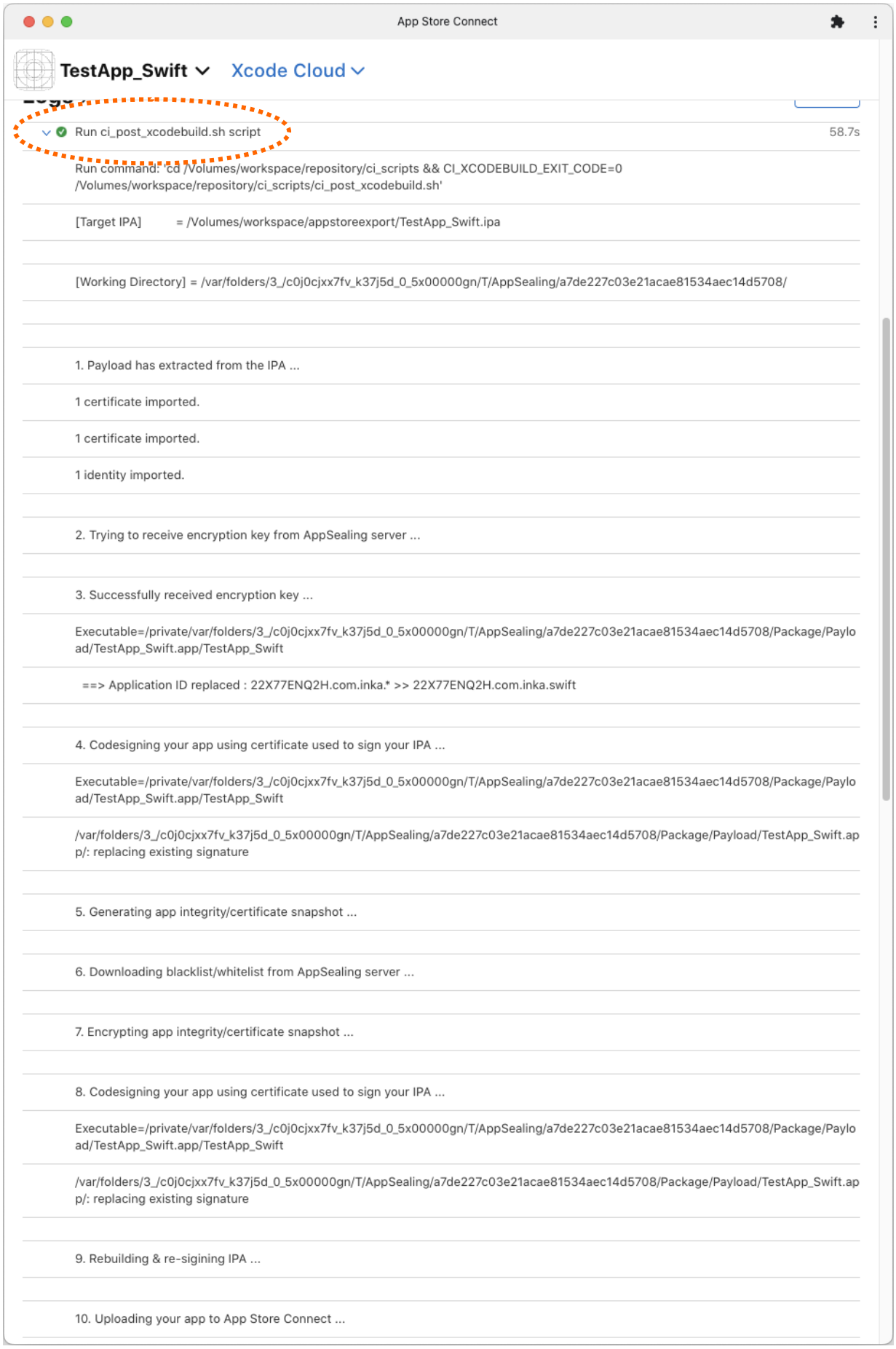
Click the build number to go to the build overview screen. Click the arrow to the right of the "Archive – iOS" item under "Actions" on the left to expand the menu and check the log.

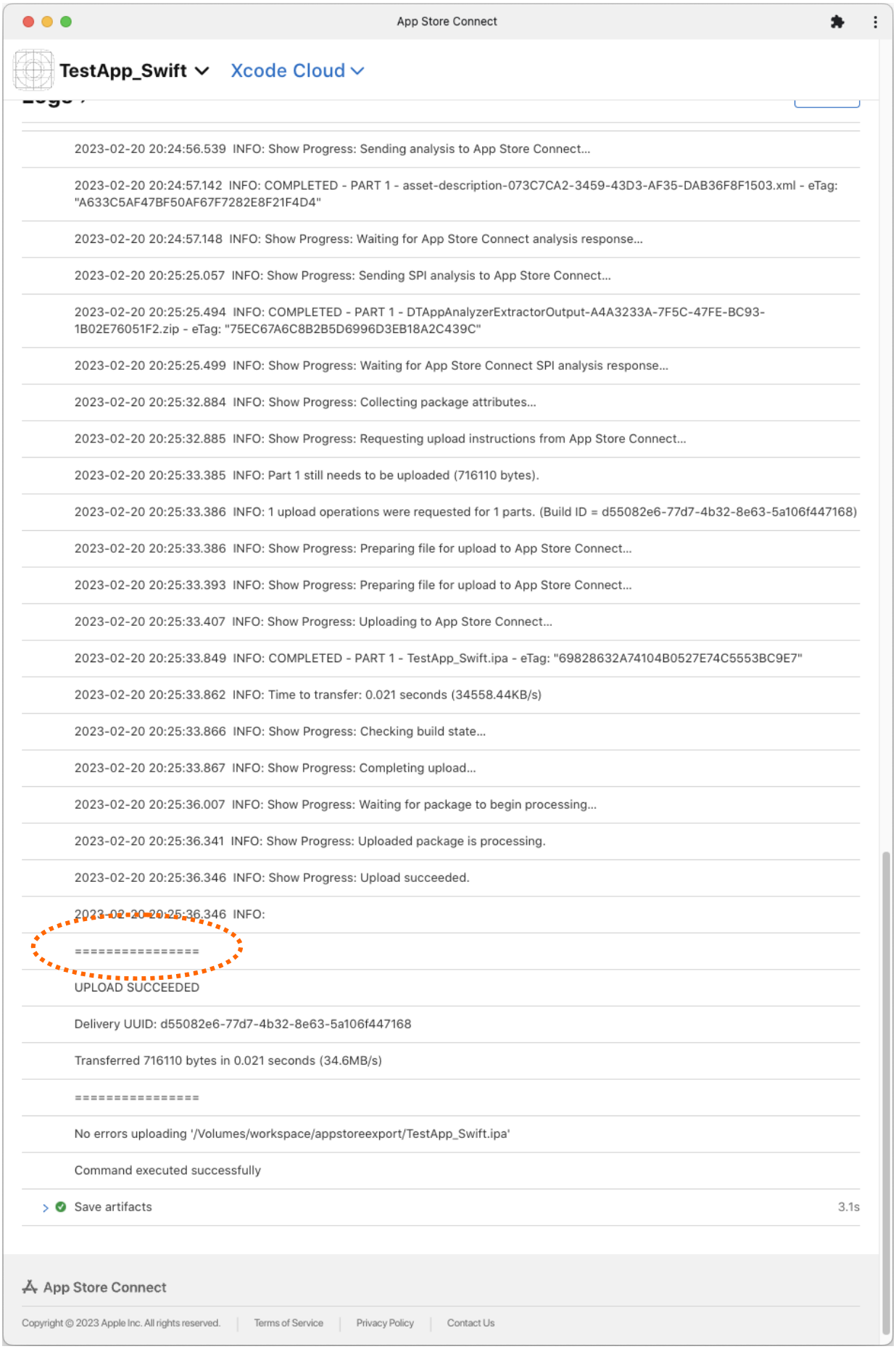


As shown in the screen above, all tasks under "Build Archive" should be performed normally. In particular, no further operations should be performed after "Run ci_post_xcodebuild.sh script" and "Save artifacts". If there is a progress log for another task, you must refer to section 7-1 to modify the workflow settings and then proceed with the build again.

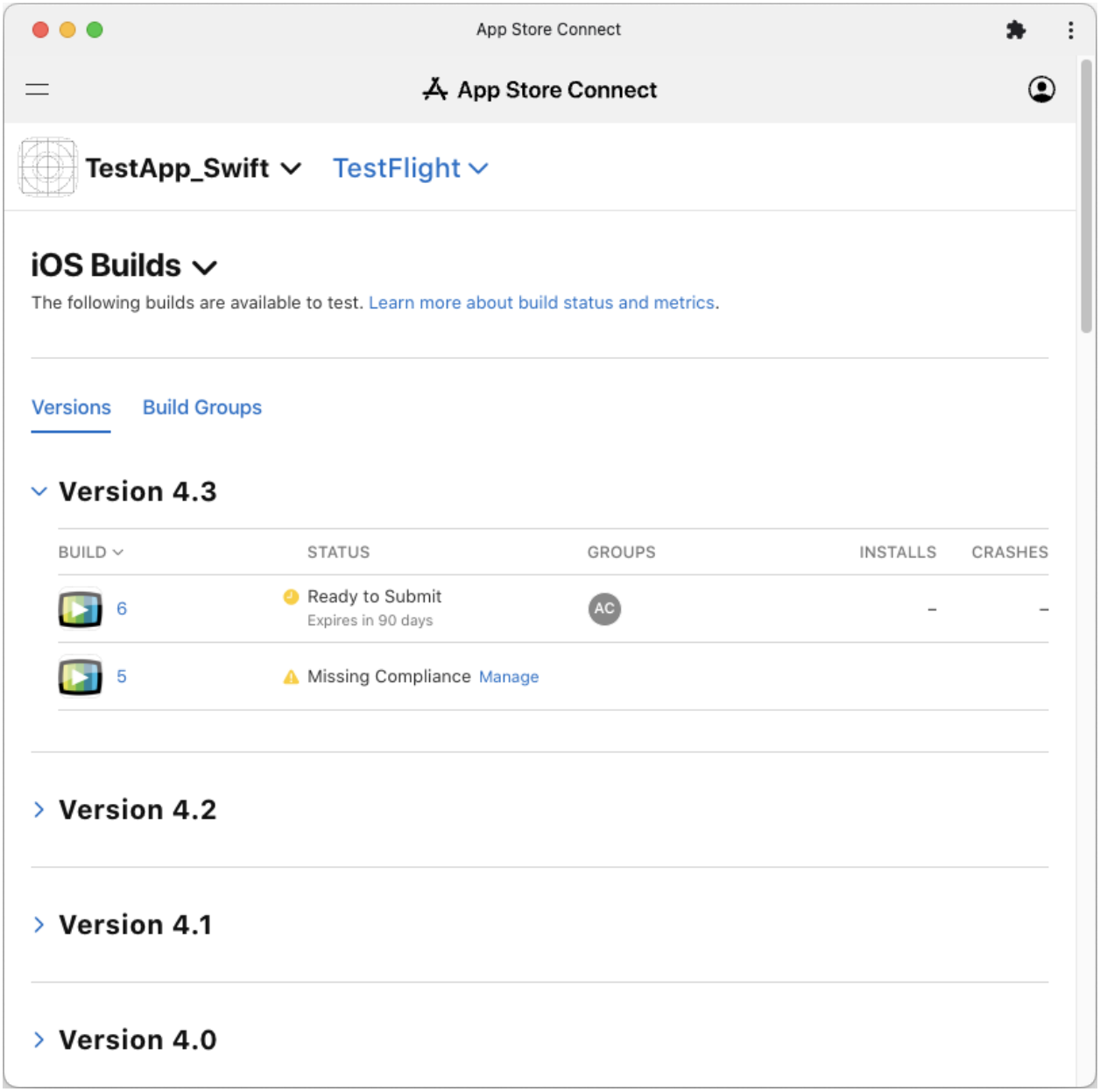
Now click on the left angle sign (>) of the "Run ci_post_xcodebuild.sh script" item to expand the script log. You can check the log where the ci_post_xcodebuild.sh script was executed for the exported IPA. Scroll to the bottom of the log to see that the IPA

was successfully uploaded.




App Store Connect
TestApp_Swift
Xcode Cloud
2023-02-20 20:24:56.539 INFO: Show Progress: Sending analysis to App Store Connect...
2023-02-20 20:24:57.142 INFO: COMPLETED - PART 1 - asset-description-073C7CA2-3459-43D3-AF35-DAB36F8F1503.xml - eTag: "A633C5AF47BF50AF67F7282E8F21F4D4"
2023-02-20 20:24:57.148 INFO: Show Progress: Waiting for App Store Connect analysis response...
2023-02-20 20:25:25.057 INFO: Show Progress: Sending SPI analysis to App Store Connect...
2023-02-20 20:25:25.494 INFO: COMPLETED - PART 1 - DTAppAnalyzerExtractorOutput-A4A3233A-7F5C-47FE-BC93-1B02E76051F2.zip - eTag: "75EC67A6C8B2B5D6996D3EB18A2C439C"
2023-02-20 20:25:25.499 INFO: Show Progress: Waiting for App Store Connect SPI analysis response...
2023-02-20 20:25:32.884 INFO: Show Progress: Collecting package attributes...
2023-02-20 20:25:32.885 INFO: Show Progress: Requesting upload instructions from App Store Connect...
2023-02-20 20:25:33.385 INFO: Part 1 still needs to be uploaded (716110 bytes).
2023-02-20 20:25:33.386 INFO: 1 upload operations were requested for 1 parts. (Build ID = d55082e6-77d7-4b32-8e63-5a106f447168)
2023-02-20 20:25:33.386 INFO: Show Progress: Preparing file for upload to App Store Connect...
2023-02-20 20:25:33.393 INFO: Show Progress: Preparing file for upload to App Store Connect...
2023-02-20 20:25:33.407 INFO: Show Progress: Uploading to App Store Connect...
2023-02-20 20:25:33.849 INFO: COMPLETED - PART 1 - TestApp_Swift.ipa - eTag: "69828632A74104B0527E74C5553BC9E7"
2023-02-20 20:25:33.862 INFO: Time to transfer: 0.021 seconds (34558.44KB/s)
2023-02-20 20:25:33.866 INFO: Show Progress: Checking build state...
2023-02-20 20:25:33.867 INFO: Show Progress: Completing upload...
2023-02-20 20:25:36.007 INFO: Show Progress: Waiting for package to begin processing...
2023-02-20 20:25:36.341 INFO: Show Progress: Uploaded package is processing.
2023-02-20 20:25:36.346 INFO: Show Progress: Upload succeeded.
2023-02-20 20:25:36.346 INFO:
==========
UPLOAD SUCCEEDED
Delivery UUID: d55082e6-77d7-4b32-8e63-5a106f447168
Transferred 716110 bytes in 0.021 seconds (34.6MB/s)
==========
No errors uploading '/Volumes/workspace/appstoreexport/TestApp_Swift.ipa'
Command executed successfully
Save artifacts
3.1s
App Store Connect
Copyright © 2023 Apple Inc. All rights reserved.
Terms of Service
Privacy Policy
Contact Us

Now if we go to TestFlight, we can see that the version we just built (6) is registered.



Part 8. Apply source code string encryption

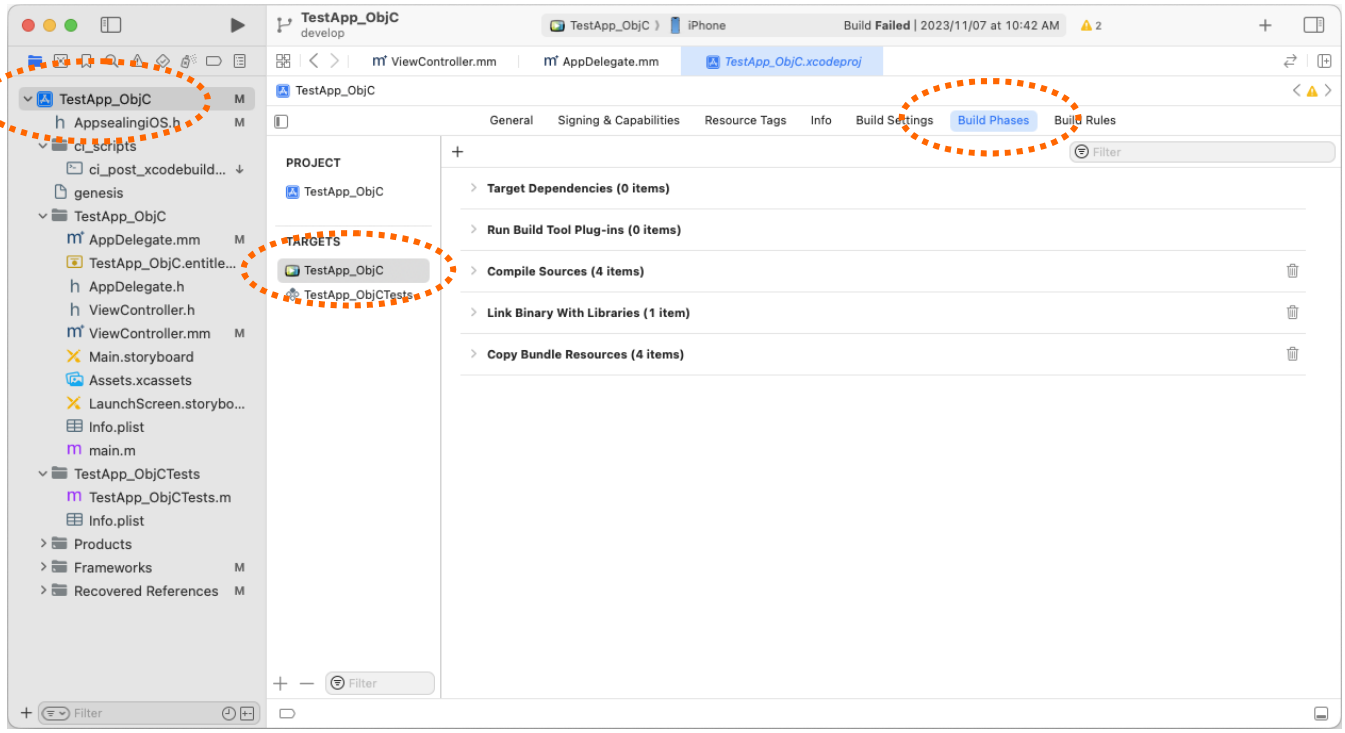
8-1 Setting project's Build Phase

AppSealing SDK provides the ability to encrypt source code strings, completely hiding them from the executable. In general, it is not necessary to encrypt all strings in the source code, but in the case of strings containing serial numbers or sensitive information, or if it is necessary to prevent string copying, the string can be protected through encryption.

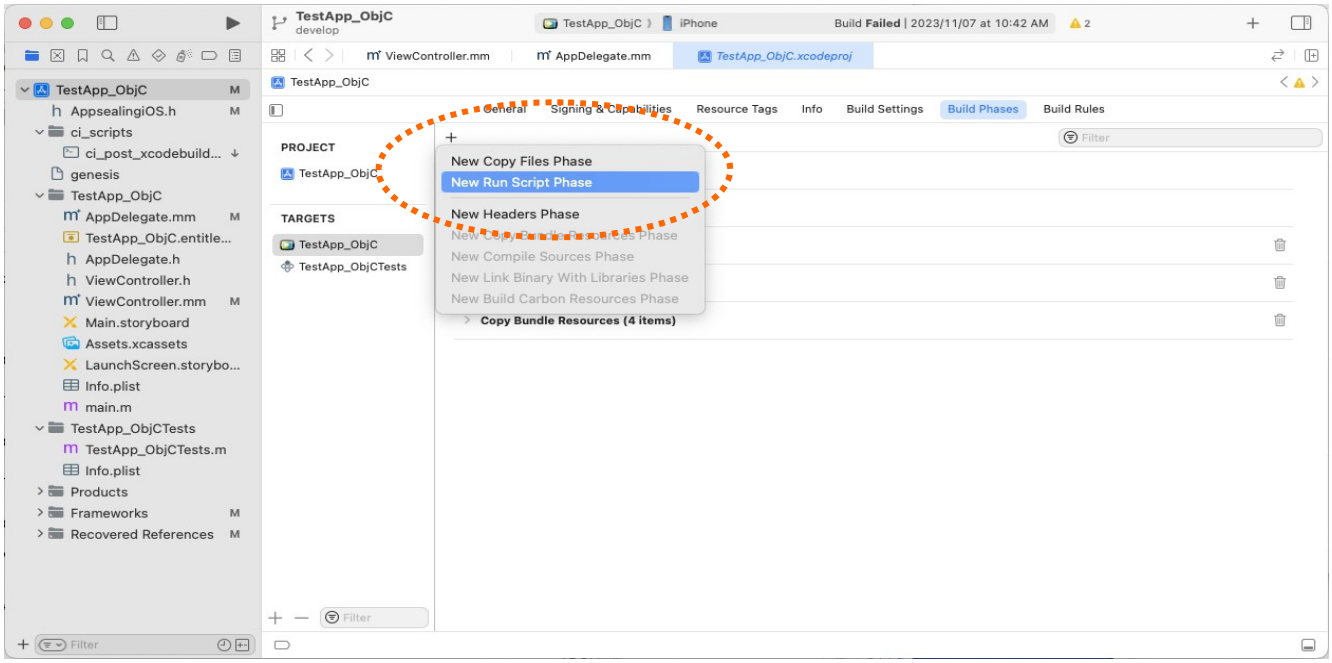
Therefore, AppSealing's string encryption function does not encrypt all strings in the source code, but app developer should append a designated tag only to strings that require protection in the source code. The tagged string preserves its original string form and is then replaced with an encrypted string just before compilation. When compilation is completed, the original tagged string form is restored. However, this requires additional settings during the Xcode build process. Follow the steps outlined below to set up the build script.

This chapter uses an Objective-C-based project named "TestApp_ObjC" as a sample project.

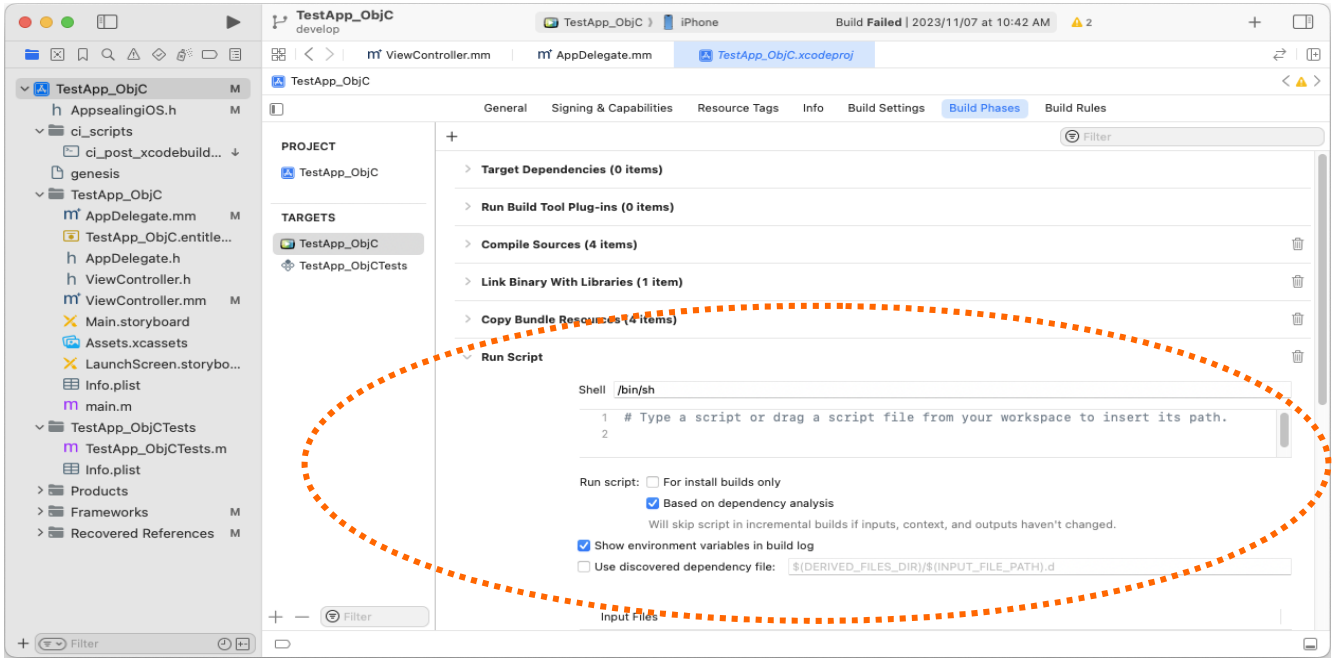
First, select the project in the Project Navigator and select the main target "TestApp_ObjC" in TARGETS. Select the "Build Phase" tab from the tabs displayed on the right.



When the "Build Phase" tab screen is displayed, click the "+" icon in the upper left corner to display the menu and select "New Run Script Phase" from there.

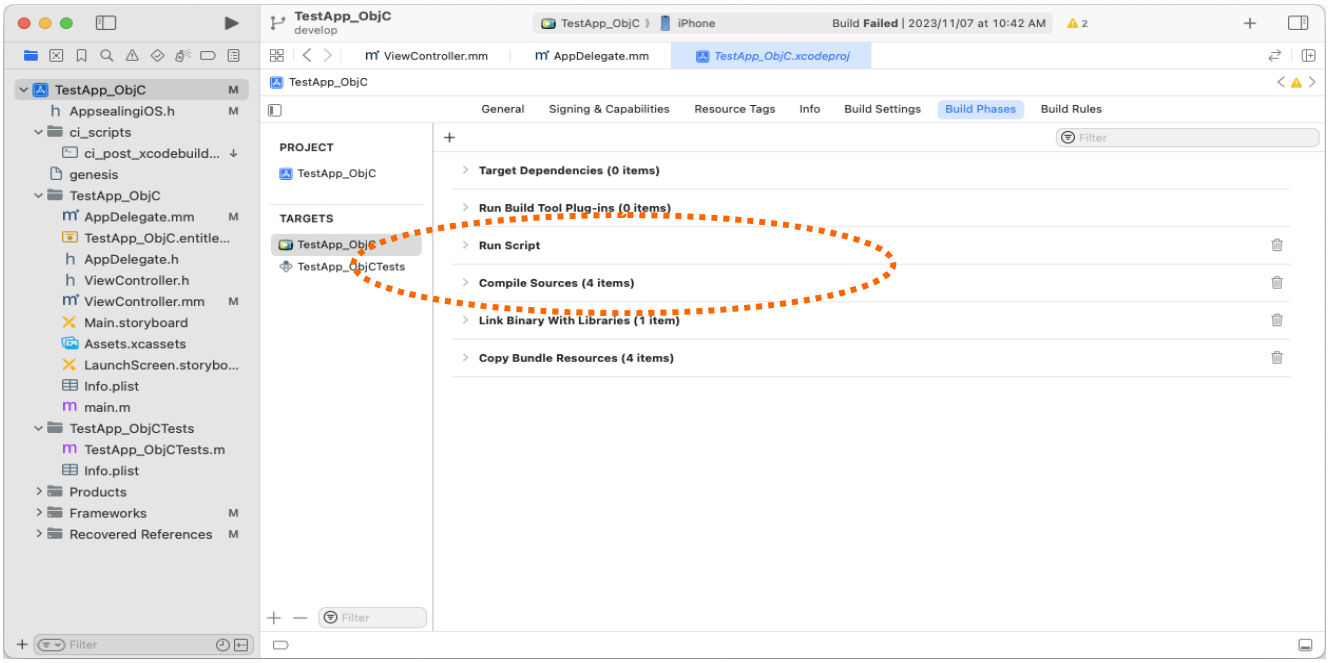


As shown in the screen below, a "Run Script" step is added and a box where you can add script commands is displayed.



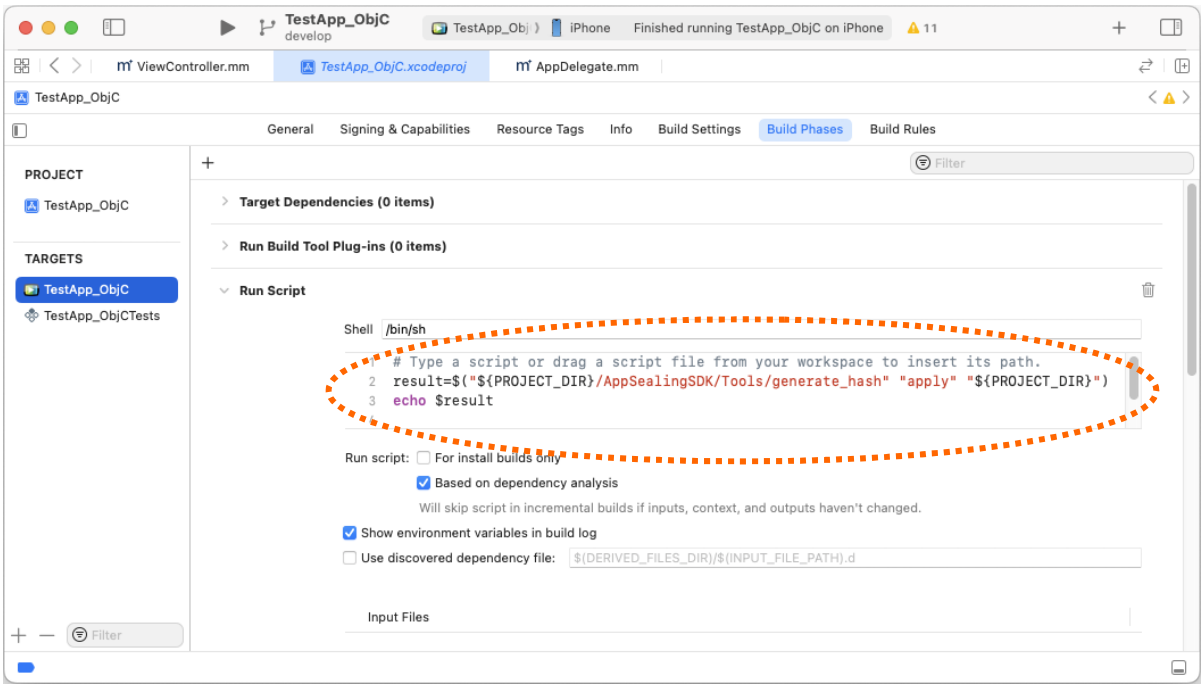
Click the inverted caret (∨) icon to the left of "Run Script" to collapse the expanded script

contents, then grab and drag the "Run Script" item with the left mouse button and move it above the "Compile Sources (## items)" item. It should look like the screen below.

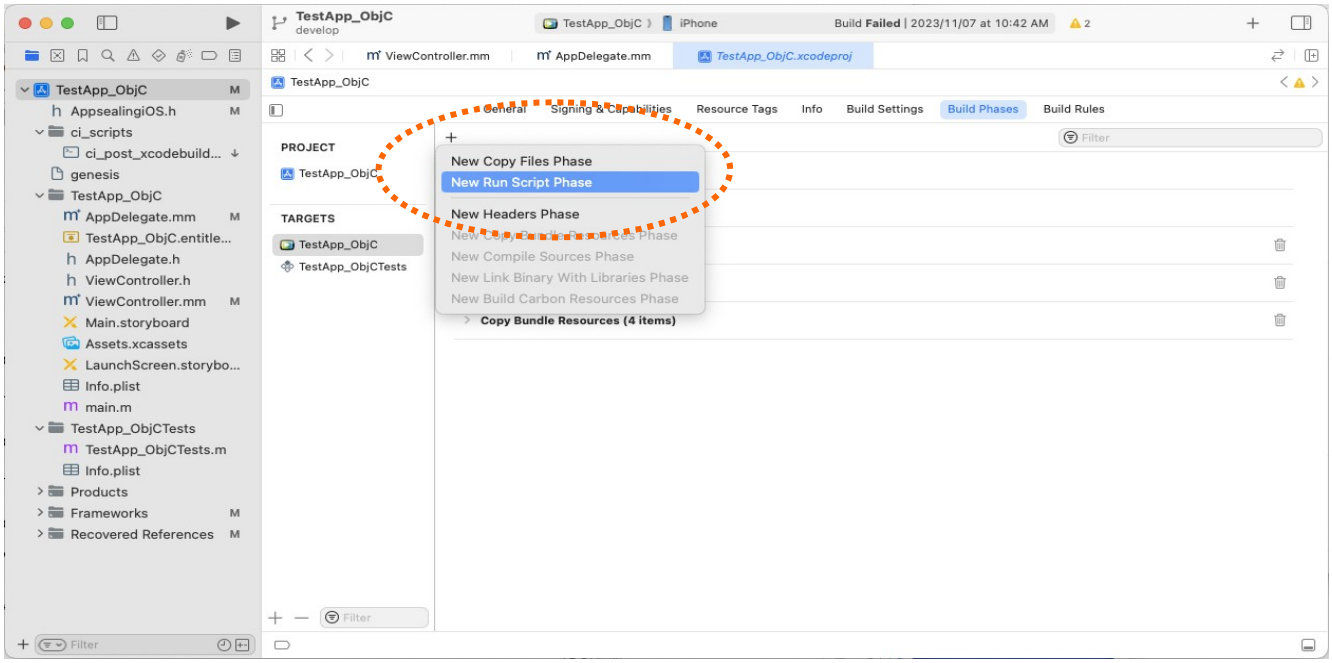


Click the ">" icon to the left of "Run Script" again to expand the script items, then copy and paste the contents below into the script box.

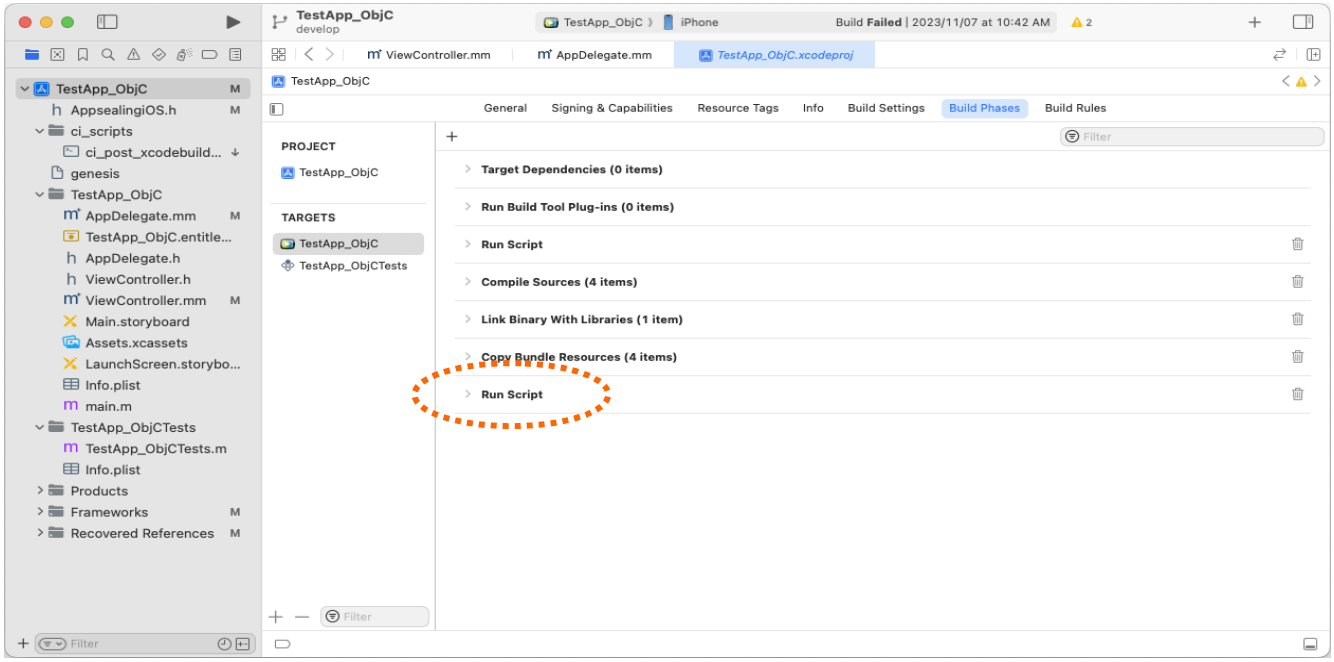
```
result=$("${PROJECT_DIR}/AppSealingSDK/Tools/generate_hash " "apply" "${PROJECT_DIR}")
echo $result
```



Next, click the "+" icon in the upper left corner once more to display the menu and additionally select the "New Run Script Phase" item.

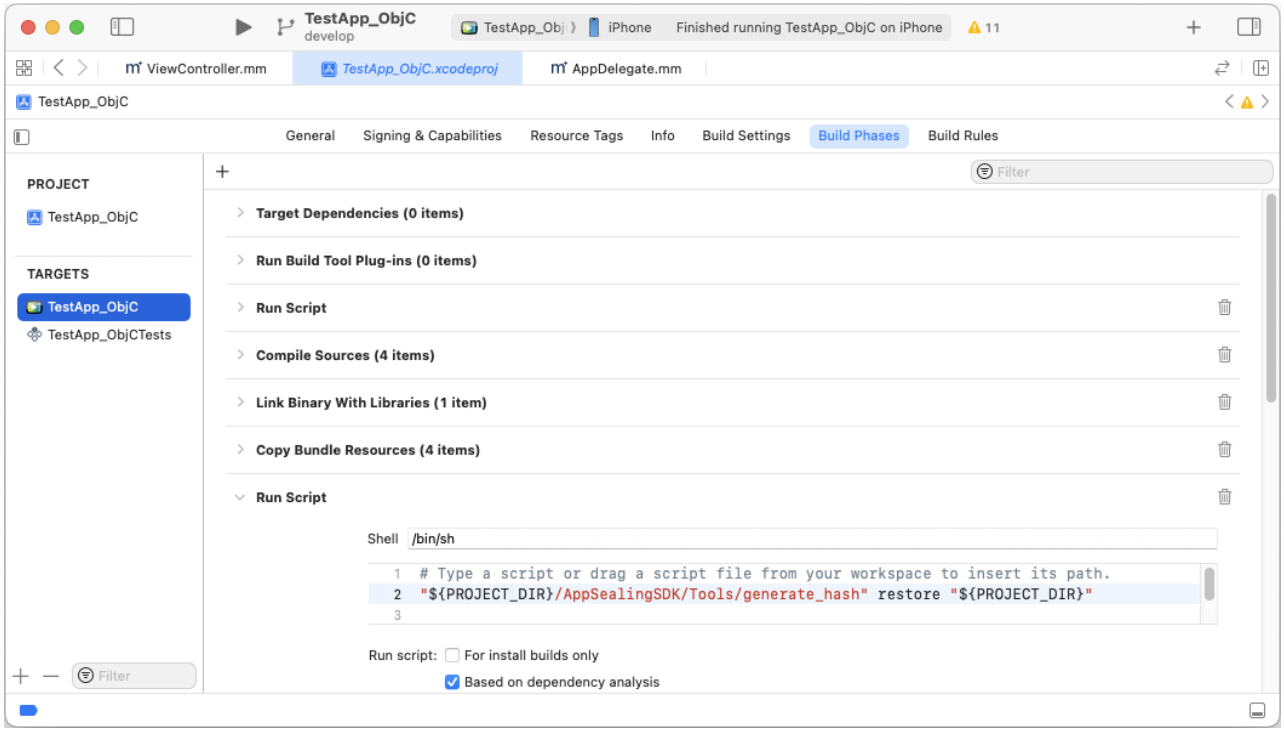


Now, one more "Run Script" item should be added at the bottom as shown in the screen below.



Click the ">" icon to the left of "Run Script" to expand the script items, then copy and paste the contents below into the script box.

"${PROJECT_DIR}/AppSealingSDK/Tools/generate_hash " "restore" "${PROJECT_DIR}"



8-2 Append encryption tag to strings

Now, all Build Phase settings for string encryption have been completed. You need to add tags to the strings that you want to apply encryption to in source code. The encryption tag is "\0x1\0x1\0x1\0x1" and must be added to the end of the string.

e.g)

"This is my string" → "This is my string\0x1\0x1\0x1\0x1"

"Special char !@#$%^\"\\" → "Special char !@#$%^\"\\\0x1\0x1\0x1\0x1"

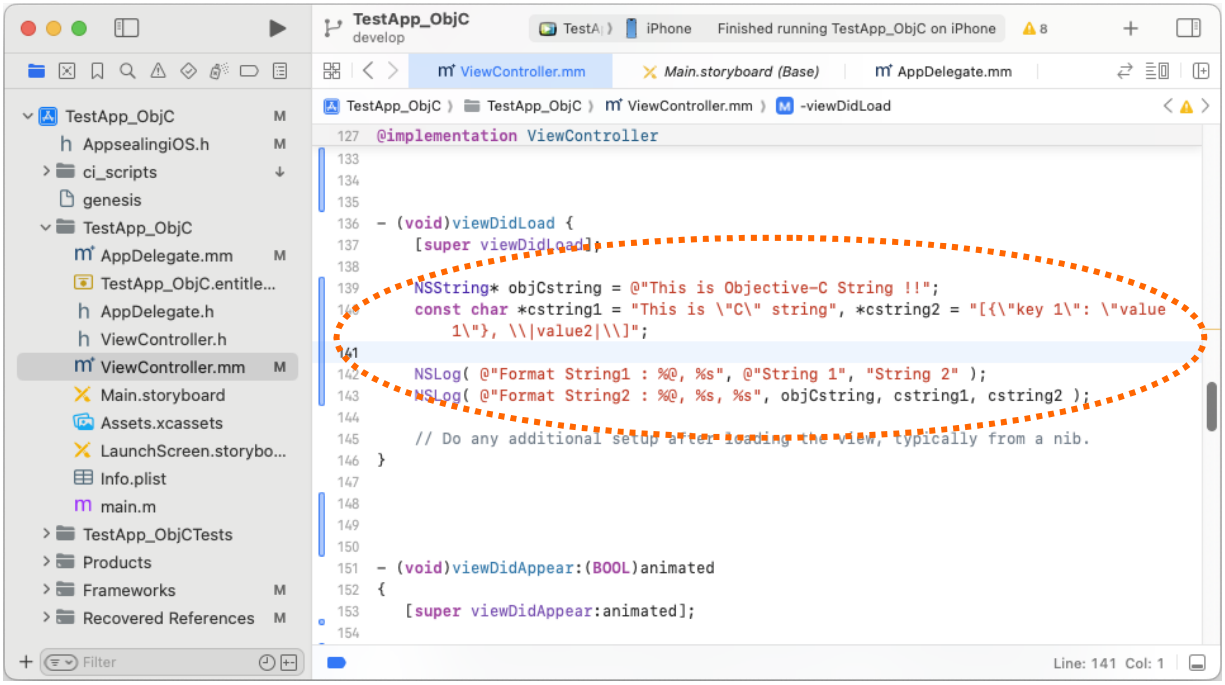
"문자열을 암호화 합니다" → "문자열을 암호화 합니다\0x1\0x1\0x1\0x1"

"文字列を暗号化します" → "文字列を暗号化します\0x1\0x1\0x1\0x1"

"加密字符串" → "加密字符串\0x1\0x1\0x1\0x1"

"Cifrar la cadena" → "Cifrar la cadena\0x1\0x1\0x1\0x1"

Let's apply tags to a string in actual source code. The TestApp_ObjC project is based on the Objective-C language, so it can include both C/C++ strings and Objective-C strings in the source code.



Let's apply encryption to four Objective-C strings and three C strings in the viewDidLoad function in the code above.

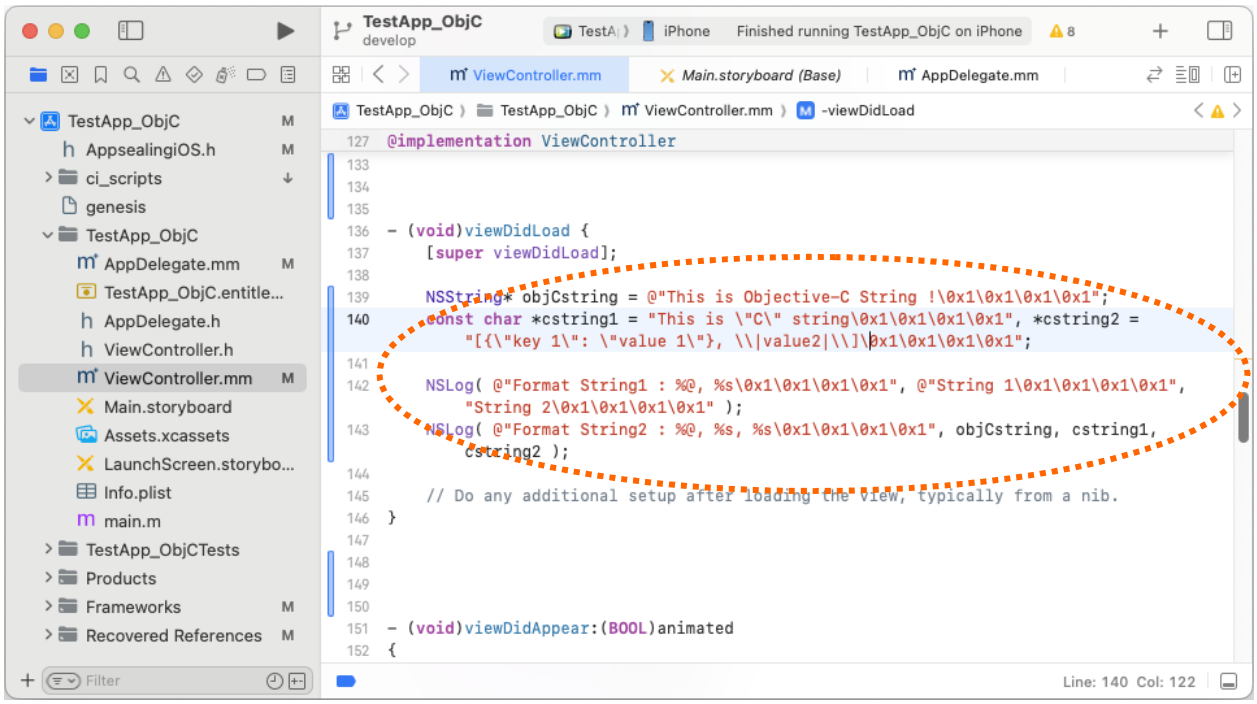
[Objective-C strings]

```
@"This is Objective-C String !" ➔ @"This is Objective-C String !\0x1\0x1\0x1\0x1"

@"Format String1 : %@, %s" ➔ @"Format String1 : %@, %s\0x1\0x1\0x1\0x1"

@"String 1" ➔ @"String 1\0x1\0x1\0x1\0x1"

@"Format String2 : %@, %s, %s" ➔ @"Format String2 : %@, %s, %s\0x1\0x1\0x1\0x1"
```

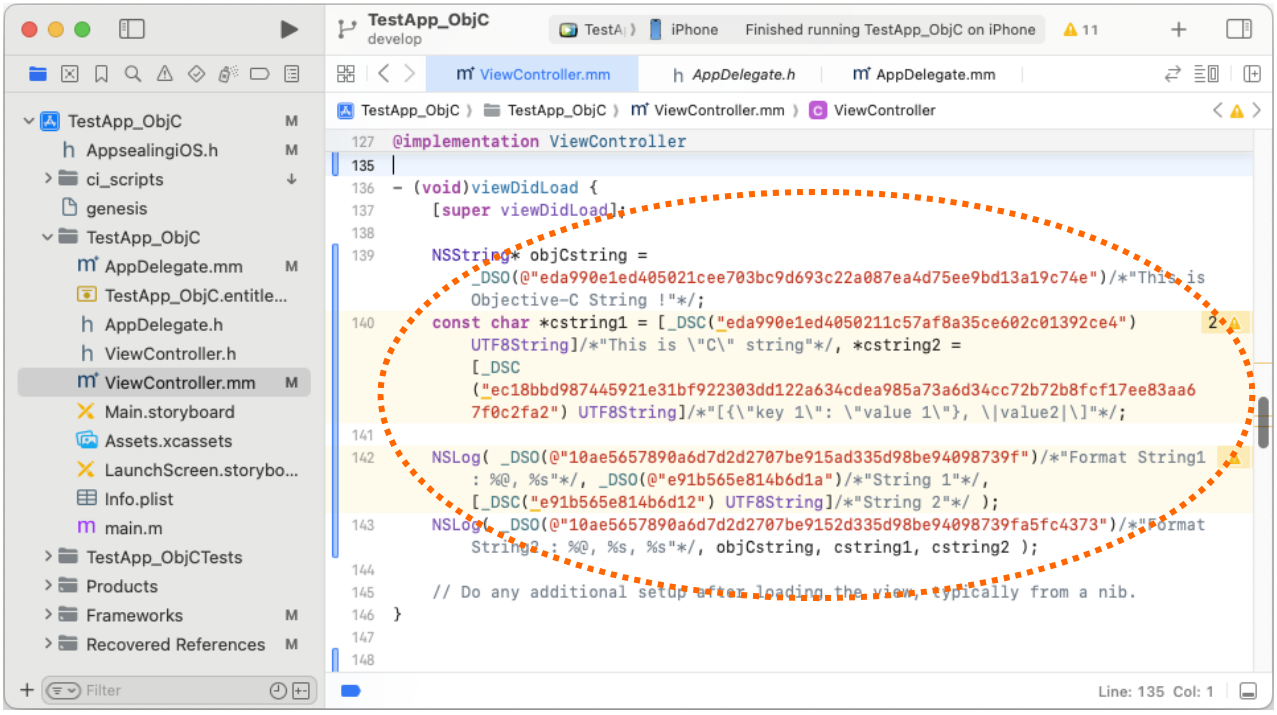
[C strings]

```
"This is \"C\" string" ➔ "This is \"C\" string\0x1\0x1\0x1\0x1"

"[{\"key 1\": \"value 1\"}, \\|value2|\\]" ➔
                        "[{\"key 1\": \"value 1\"}, \\|value2|\\]\0x1\0x1\0x1\0x1"

"String 2" ➔ "String 2\0x1\0x1\0x1\0x1"
```

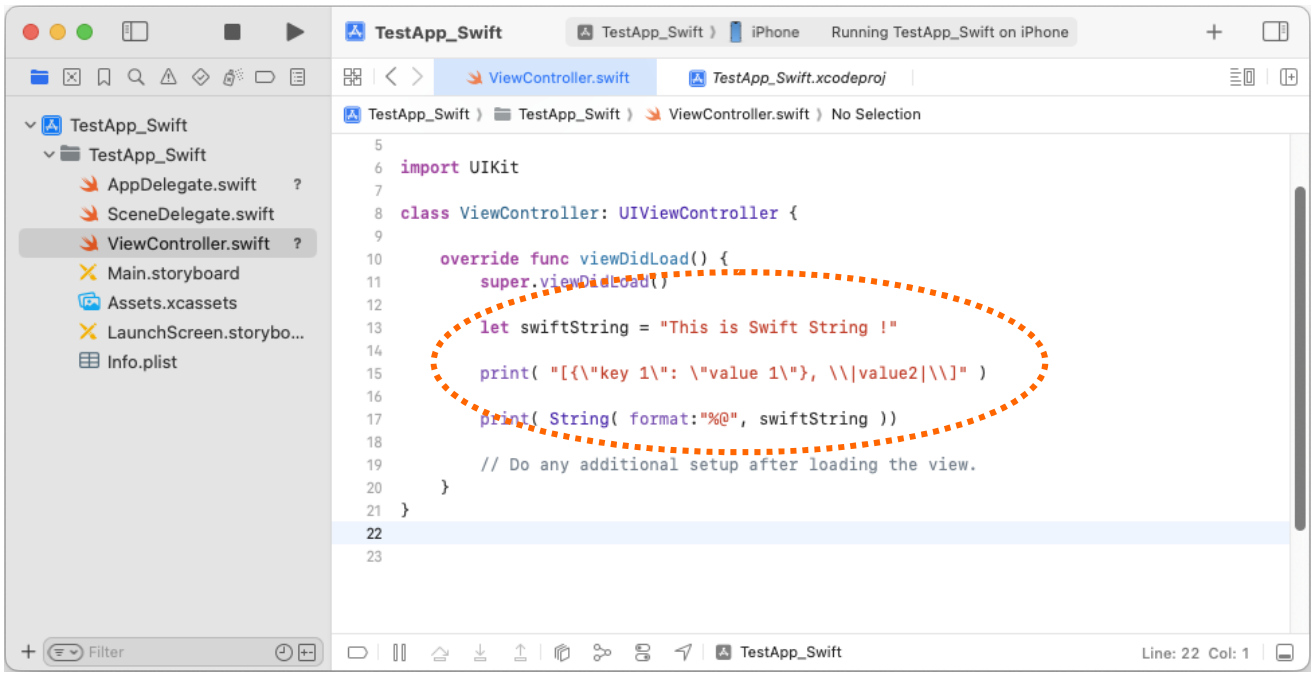

The code with all tags applied is as follows.



Objective-C code with this tag applied is converted to an encrypted string as shown below at the time of compilation, and when compilation is completed, it is restored to the original tagged string as shown in the picture above. (So Objective-C code containing encrypted strings like the one below won't be visible under normal circumstances)



For Swift source code, tag application is done in the same way.

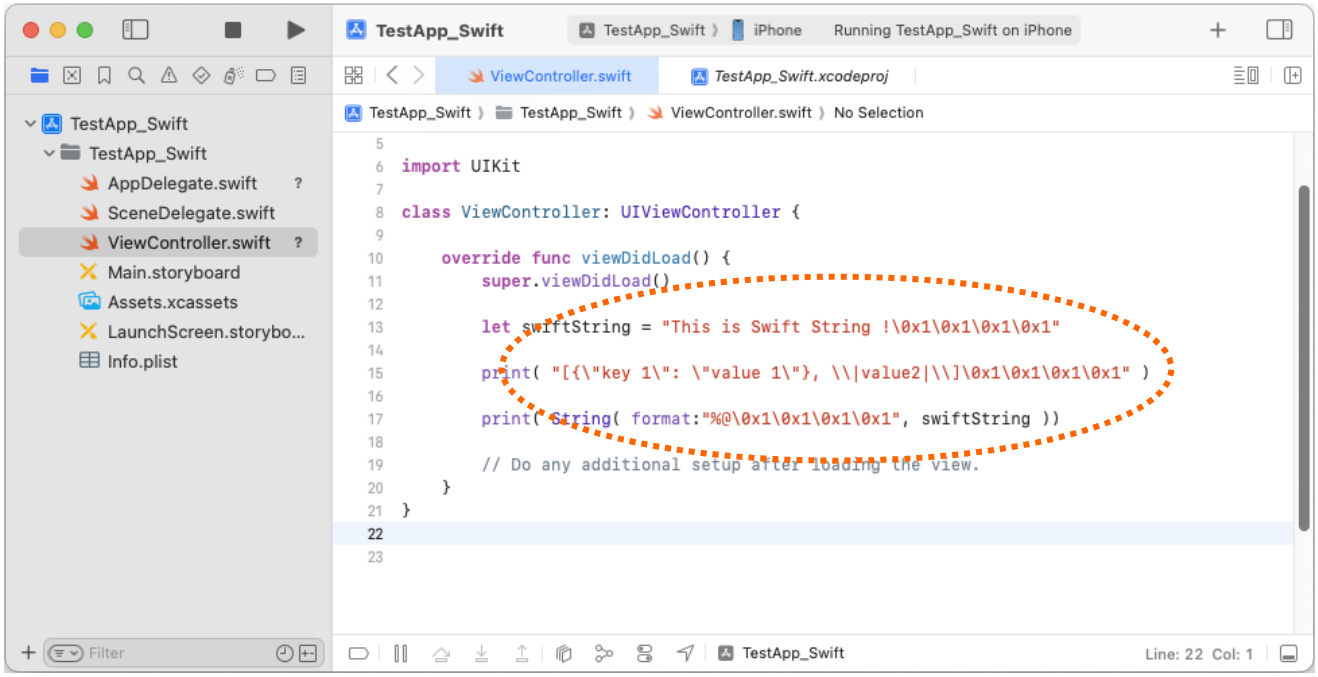


"This is Swift string" ➔ "This is Swift string\0x1\0x1\0x1\0x1"

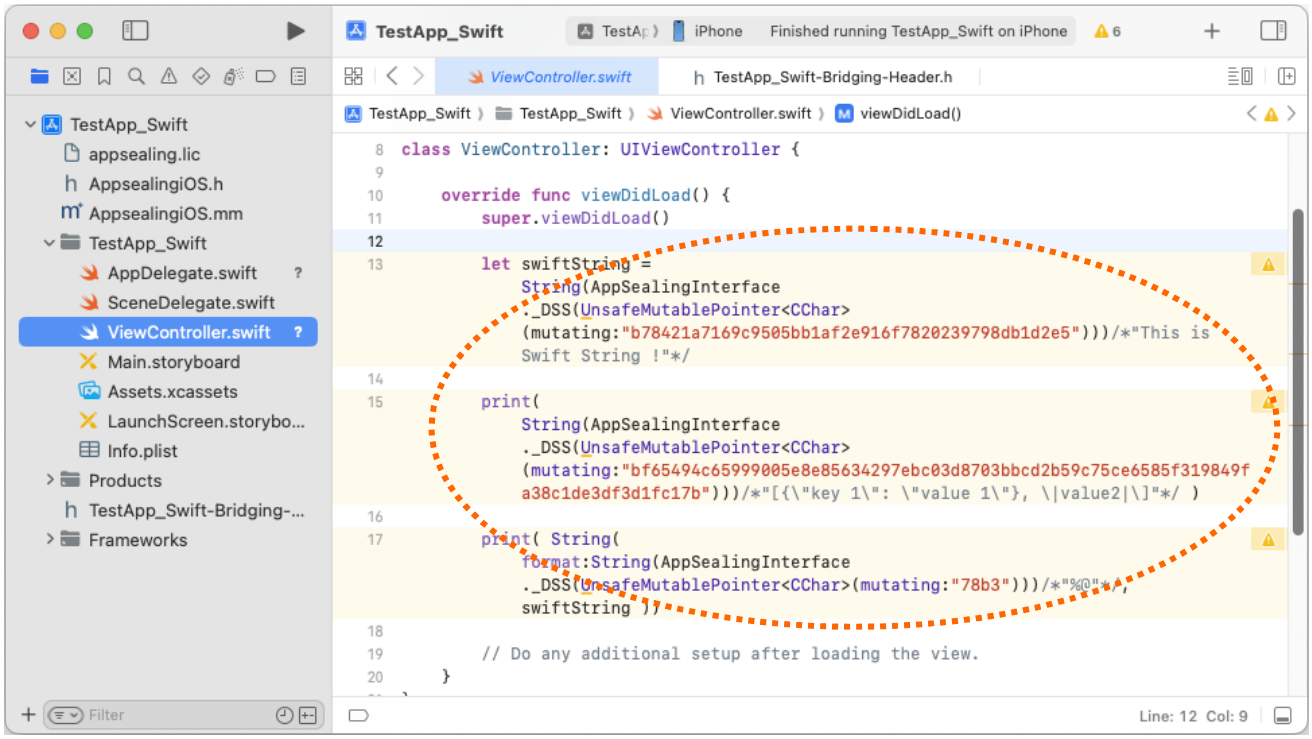
"[{\"key 1\": \"value 1\"}, \\|value2|\\]" ➔

"[{\"key 1\": \"value 1\"}, \\|value2|\\]\0x1\0x1\0x1\0x1"

"%@" ➔ "%@\0x1\0x1\0x1\0x1"



The tagged Swift code is converted to an encrypted string as shown below at the time of compilation, and when compilation is completed, it is restored to the original tagged string as shown above. (Similarly, Swift code containing encrypted strings like the one below is also not visible under normal circumstances)



8-3 Precautions for Swift string encryption

- When using the string enumeration like the following, tags cannot be added to strings specified as raw values. If you add tags, a compilation error will occur.

```
enum StringEnum: String
{
  case first = "FIRST"
  case second = "SECOND"
  case third = "THIRD"
}
```

```
enum StringEnum: String
{
  case first = "FIRST\0x1\0x1\0x1\0x1"
  case second = "SECOND\0x1\0x1\0x1\0x1"
  case third = "THIRD\0x1\0x1\0x1\0x1"
}
```

- You cannot add tags to multiline strings like the following.

```
let str = """
    This is multi
    line string,
    do not use tag
    """
```

```
let str = """
    This is multi\0x1\0x1\0x1\0x1
    line string,\0x1\0x1\0x1\0x1
    do not use tag\0x1\0x1\0x1\0x1
    """
```

- Tags cannot be added to strings that have string interpolation applied, as shown below.

```
var city = "Seoul"
var message = "Welcome to \(city)!"
print( "Message : \(message)" )
```

```
var city = "Seoul"
var message = "Welcome to \(city)!\0x1\0x1\0x1\0x1"
print( "Message : \(message)\0x1\0x1\0x1\0x1" )
```

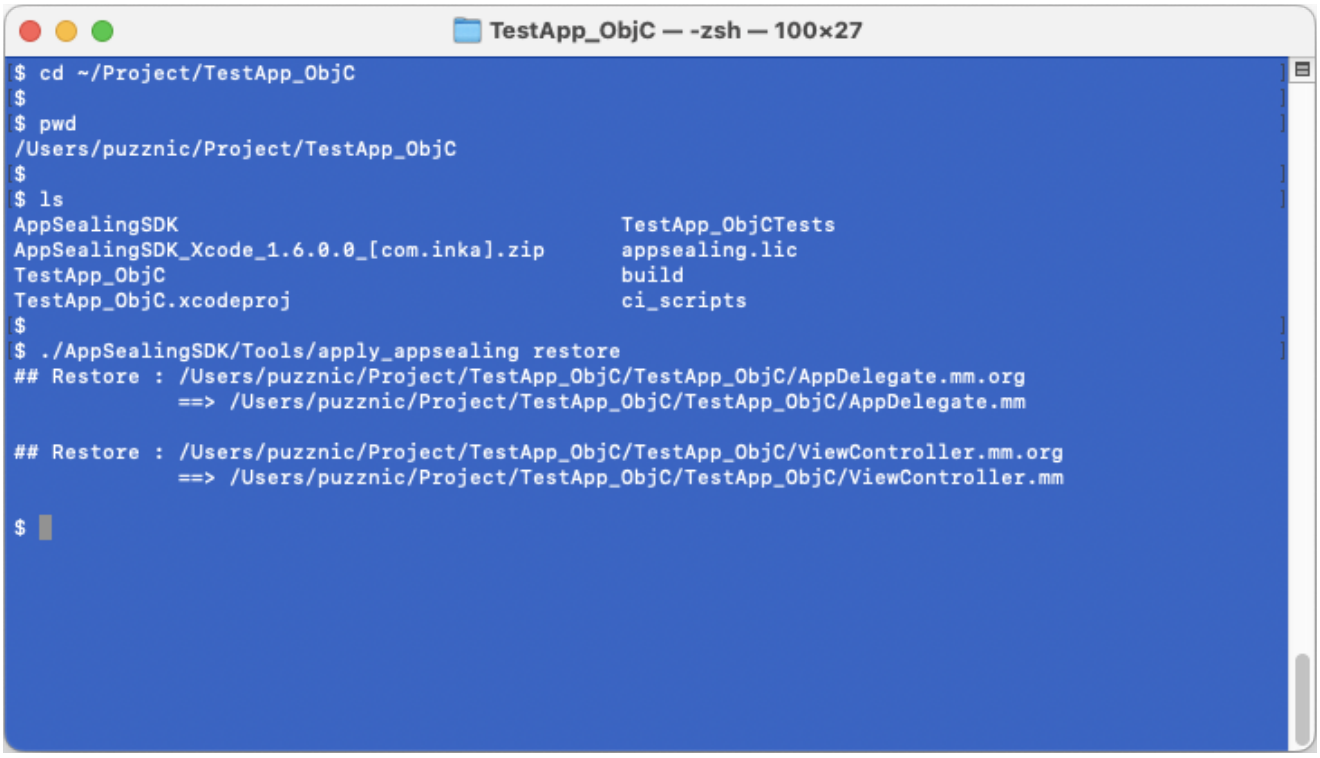

8-4 Handle compile error: Restore encrypted string

The tagged string is changed to an encrypted string just before compilation and restored to the original string after compilation (8-1 Run Script). However, if a compilation error occurs or the compilation process is interrupted for any reason, the script after compilation will not executed, this will lead to a problem where the encrypted string will remain in the source code without being restored.

In this case, there are two ways to restore the encrypted string to its original state.

The first is restoration through script execution. Launch the Terminal app and navigate to your project directory. Then run the "AppSealingSDK/Tools/apply_appsealing" script with the "restore" parameter to restore the original code.

```
$ ./AppSealingSDK/Tools/apply_appsealing restore
```

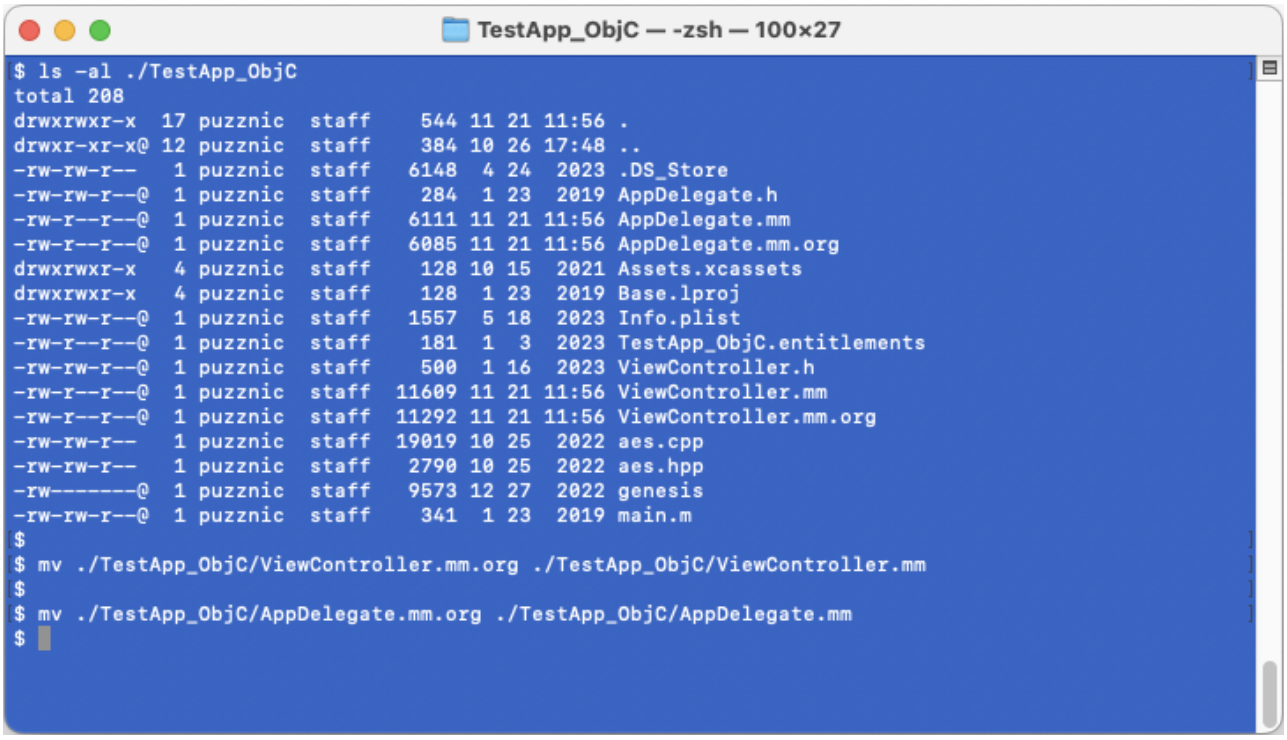


When you run the script, you will see a message displayed that the two source codes of the TestApp_ObjC project used as an example, AppDelegate.mm and ViewController.mm, have been restored.

The second method is to return the backed-up code file to its original name by directly entering a shell command in the terminal. Taking the TestApp_ObjC project as an example, if you list the contents of the TestApp_ObjC subfolder under the TestApp_ObjC folder, you can see that two source codes are backed up with the extension ".org", as shown in the screen below.

Now enter the command below to return the two files to their original names.

```
$ mv ./TestApp_ObjC/ViewController.mm.org ./TestApp_ObjC/ViewController.mm

$ mv ./TestApp_ObjC/AppDelegate.mm.org ./TestApp_ObjC/AppDelegate.mm
```



Part 9. Apply to Fastlane project

9-1 Configure project's Fastfile

AppSealing SDK can also be used in projects that have applied Fastlane. Typically, the reason for using the Fastlane tool in an Xcode project is to automate the app build and packaging process, and also to automate the subsequent distribution process. If you apply AppSealing SDK to an Xcode project, there are additional steps that need to be performed after building the app, but if Fastlane is applied, you can maintain the existing automation process by adding the AppSealing application process to the Fastfile script.
This chapter explains how to modify your scripts to maintain Fastlane's automated build and distribution process.

Here is the Fastfile code for a project named "TestApp_Swfit" with Fastlane enabled (excluding comments). It includes the steps to build, sign, and upload the app to Testflight. The script for uploading to the Apple Store has the same structure, so this guide will only explain Testflight case.

Default Fastfile of project

```
default_platform(:ios)

platform :ios do
  desc "Push a new beta build to TestFlight"
  lane :beta do
    increment_build_number(xcodeproj: "TestApp_Swift.xcodeproj")
    build_app(scheme: "TestApp_Swift")
    upload_to_testflight
  end
end
```

In general, you will build and deploy by running the "fastlane beta" command without making any major modifications to this file. If you apply the AppSealing SDK here, additional work is required between the build and distribution, so you need to modify the fastfile script as follows. In the original script, the project name and file name are directly entered as string values, but the script to be modified is designed to extract

the necessary values from the project folder instead of directly specifying the values so that it can be applied to all other projects. Therefore, even if the project name is not "TestApp_Swift", you can apply the same Fastfile script to all projects.

Open Fastfile with a text editor and replace all codes with the code below. However, you must replace the FASTLANE_USER, FASTLANE_APPLE_APPLICATION_SPECIFIC_ PASSWORD, and PROFILE values in the code below with the values you actually use. FASTLANE_USER should be your Apple account, and FASTLANE_APPLE_APPLICATION_SPECIFIC_ PASSWORD should be your app-specific password. For information on how to issue an app-specific password, please refer to page 63.

The PROFILE value should be the name of the provisioning profile you use in the distribution step.

Fastfile when using AppSealing SDK

```
default_platform(:ios)

before_all do
  ENV["FASTLANE_USER"] = "#YOUR_APPLE_ID#"
  ENV["FASTLANE_APPLE_APPLICATION_SPECIFIC_PASSWORD"] = "#YOUR_APP_SPECIFIC_PASSWORD#"
  ENV['PROFILE'] = "#YOUR_DISTRIBUTION_PROVISIONING_PROFILE_NAME#"
end

platform :ios do
  desc "Push a new beta build to TestFlight"
  lane :beta do
    # Get project root path
    project_root = File.expand_path("..", __dir__)

    # Dynamically retrieve Xcode project or workspace files
    xcode_file_path = Dir.glob(File.join(project_root, "*.xcworkspace")).first ||
Dir.glob(File.join(project_root, "*.xcodeproj")).first
    if xcode_file_path.nil?
      UI.user_error!("[AppSealing] No Xcode project or workspace found in the project
root directory: #{project_root}")
    end

    # Automatically extract project name and scheme
    project_name = File.basename(xcode_file_path, File.extname(xcode_file_path))

    # Auto-search scheme
    scheme_name = ""
    Dir.chdir(File.dirname(xcode_file_path)) do
      scheme_name = sh("xcodebuild -list -#{xcode_file_path.end_with?('.xcworkspace') ?
```



```
'workspace' : 'project'} #{File.basename(xcode_file_path)} | grep 'Schemes:' -A 1 |
tail -n 1").strip
    end

    if scheme_name.empty?
      UI.user_error!("[AppSealing] Failed to retrieve scheme name from Xcode project or
workspace.")
    end

    # Dynamically extract bundle ID
    bundle_id = ""
    Dir.chdir(File.dirname(xcode_file_path)) do
      bundle_id_command = "xcodebuild -showBuildSettings -scheme #{scheme_name} | grep
'PRODUCT_BUNDLE_IDENTIFIER' | awk -F ' = ' '{print $2}'"
      bundle_id_output = sh(bundle_id_command).lines.map(&:strip).reject { |line|
line.include?("WARNING") || line.empty? }
      bundle_id = bundle_id_output.last.strip
    end

    if bundle_id.empty?
      UI.user_error!("[AppSealing] Failed to retrieve Bundle ID from Xcode project.")
    end

    # Dynamically extract Team ID
    team_id = ""
    Dir.chdir(File.dirname(xcode_file_path)) do
      team_id_command = "xcodebuild -showBuildSettings -scheme #{scheme_name} | grep
'DEVELOPMENT_TEAM' | awk -F ' = ' '{print $2}'"
      team_id_output = sh(team_id_command).lines.map(&:strip).reject { |line|
line.include?("WARNING") || line.empty? }
      team_id = team_id_output.last.strip
    end

    if team_id.empty?
      UI.user_error!("[AppSealing] Failed to retrieve Team ID from Xcode project.")
    end

    UI.message "[AppSealing] Project Name: #{project_name}"
    UI.message "[AppSealing] Scheme Name: #{scheme_name}"
    UI.message "[AppSealing] Bundle ID: #{bundle_id}"
    UI.message "[AppSealing] Team ID: #{team_id}"

    # Build and generate IPA file
    archive_path = File.join(project_root, "build", "#{project_name}.xcarchive")
    ipa_output_path = File.join(project_root, "build")

    # Separate handling of workspace and project options when calling build_ios_app
    build_options = {
      scheme: scheme_name,
      export_method: "app-store",
      clean: true,
```



```
      output_directory: ipa_output_path,
      output_name: "#{project_name}.ipa",
      export_options: {
        provisioningProfiles: {
          bundle_id => ENV["PROFILE"]
        }
      }
    }

    if xcode_file_path.end_with?(".xcworkspace")
      build_options[:workspace] = xcode_file_path
      build_options[:project] = nil
    else
      build_options[:project] = xcode_file_path
      build_options[:workspace] = nil
    end

    build_ios_app(build_options)

    # Set .ipa file path
    ipa_path = File.join(ipa_output_path, "#{project_name}.ipa")

    unless File.exist?(ipa_path)
      UI.user_error!("[AppSealing] IPA file not found at path: #{ipa_path}")
    end

    UI.message "[AppSealing] IPA Path: #{ipa_path}"

    # Dynamically find and execute the generate_hash script path
    generate_hash_script = Dir.glob(File.join(project_root, "**/generate_hash")).first
|| File.join(project_root, "AppSealingSDK", "Tools", "generate_hash")

    unless File.exist?(generate_hash_script)
      UI.user_error!("[AppSealing] generate_hash script not found at path:
#{generate_hash_script}")
    end

    unless File.executable?(generate_hash_script)
      sh("chmod +x '#{generate_hash_script}'")
    end

    sh("#{generate_hash_script} #{File.absolute_path(ipa_path)}")

     # Upload your IPA file to TestFlight
     begin
       upload_to_testflight(
         ipa: ipa_path,
         skip_waiting_for_build_processing: true, # Skip build processing wait time
       )
       UI.success("[AppSealing] Upload to TestFlight completed successfully!")
```



```
    rescue => e
      UI.error("[AppSealing] Upload to TestFlight failed with error: #{e.message}")
      raise e

    ensure
      # Delete build directory
      if Dir.exist?(ipa_output_path)
        UI.message("[AppSealing] Deleting build directory: #{ipa_output_path}")
        FileUtils.rm_rf(ipa_output_path)
        UI.message("[AppSealing] Build directory deleted.")
      else
        UI.message("[AppSealing] Build directory not found or already deleted.")
      end
    end
  end
end
```

The above code is also included in the CodeSamples.txt file included in the SDK, so you can copy and use it from there.

If you modify the code and run the "fastlane beta" command as before, it will build the app, export it as an IPA, automatically run the generate_hash script, and upload the AppSealing-enabled IPA to testflight.

9-2 Use Github Action and Fastlane simultaneously

You can build a deployment pipeline to automatically build and distribute when your Xcode project code is uploaded to Github and code is pushed using Github Actions. In this case, you will add a build script file in yaml format as shown below for Action definition.

This file defines the source code branch name as "develop" and includes the Fastlane installation step to run the fastfile included in the project. In addition, the APP_STORE_CONNECT_KEY_ID, APP_STORE_CONNECT_ISSUER_ID, and APP_STORE_CONNECT_PRIVATE_KEY values are specified in env to set the environment variables required to run fastfile. In the step of applying Fastlane to a local project or using Xcode Cloud, an app-only password was used, but in this step, the App Store Connect API is used. To use the App Store Connect API, you must obtain an API Key from the Apple account page, and the private key (APP_STORE_CONNECT_PRIVATE_KEY), key ID (APP_STORE_CONNECT_KEY_ID), and issuer ID (APP_STORE_CONNECT_ISSUER_ID)

issued here must be passed to the fastfile script. The actual values passed are not entered directly into the script, but are written in a way that they are stored in the secrets variable of Github Action and then passed by referencing the variable.

Default Github Action build script of project (ios_build.yml)

```
name: iOS Build & Deploy
on:
  push:
    branches:
      - develop

jobs:
  release-ios:
    name: Build and release iOS app
    runs-on: macos-latest
    steps:
      - uses: actions/checkout@v2

      - uses: actions/setup-ruby@v1
        with:
          ruby-version: '3.1.2'

      - name: Install Fastlane
        run: cd ios && bundle install

      - name: Install pods
        run: cd ios && pod install

      - name: Execute Fastlane
        env:
          APP_STORE_CONNECT_KEY_ID: ${{ secrets.KEY_ID }}
          APP_STORE_CONNECT_API_KEY: ${{ secrets.API_KEY }}
          APP_STORE_CONNECT_PRIVATE_KEY: ${{ secrets.PRIVATE_KEY }}
        run: cd ios && fastlane release
```

Below is the Fastfile code that will take these environment variables and perform the actual fastlane task. It includes the setup steps for using the App Store Connect API, as well as the steps for building and uploading the app.

Default Fastfile script of github project

```
default_platform(:ios)

platform :ios do
  desc "Release to App Store"
```



```
  lane :release do
    # App Store Connect API Authentication
    app_store_connect_api_key(
      key_id: ENV["APP_STORE_CONNECT_KEY_ID"],
      issuer_id: ENV["APP_STORE_CONNECT_ISSUER_ID"],
      key_content: ENV["APP_STORE_CONNECT_PRIVATE_KEY"]
    )

    # build app
    build_app(
      scheme: "scheme_name_of_project",
      export_method: "app-store"
    )

    # upload app
    upload_to_testflight(
      skip_waiting_for_build_processing: true,
      distribute_external: true
    )
  end
end
```

If you have not already been issued an App Store Connect API Key, please refer to the link below to obtain a key and then proceed with the steps below.
(https://developer.apple.com/documentation/appstoreconnectapi/creating-api-keys-for-app-store-connect-api)

Here's what to do if you apply the AppSealing SDK in this state. If you apply the AppSealing SDK, you'll need to perform script work on the IPA generated through the archive and export steps, and since the IPA will be re-signed during this process, you'll need an additional signing certificate and provisioning profile. The certificate must be a distribution certificate for store upload, and the profile must also have a distribution profile. Therefore, you'll need to convert the certificate to PKCS#12 format and include it in the project, and push it to github with the provisioning profile included. At this time, the file name of the certificate should be "certificate.p12" and the file name of the provisioning profile should be "distribution.mobileprovision". If you use different file names, you'll need to modify the file names in the script below accordingly. The name of the provisioning profile can be modified by directly entering a value in the PROFILE environment variable.
When saving the certificate in PKCS#12 format, the password is assumed to be set to "123456". If the password is different, you must also modify the password string in the script below. Modify the "ios_build.yml" script as follows to install the certificate

uploaded to Github into the keychain and install the provisioning profile.

ios_build.yml when applying Github Action + AppSealing SDK

```
name: iOS Build with Xcode 16

on:
  push:
    branches:
      - develop

jobs:
  build-ios:
    runs-on: macos-15

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup Xcode 16
        uses: maxim-lobanov/setup-xcode@v1
        with:
          xcode-version: '16.0'

      - name: Set up Signing Credentials
        run: |
          security create-keychain -p "temp_password" temp.keychain
          security import ./certificate.p12 -k temp.keychain -P "123456" -T
/usr/bin/codesign
          security set-key-partition-list -S apple-tool:,apple:,codesign: -s -k
"temp_password" temp.keychain
          security list-keychains -s temp.keychain
          security unlock-keychain -p "temp_password" temp.keychain
          security set-keychain-settings -lut 3600 temp.keychain

      - name: Set up Provisioning Profile from Project Folder
        run: |
          mkdir -p ~/Library/Developer/Xcode/UserData/Provisioning\ Profiles
          cp ./distribution.mobileprovision
~/Library/Developer/Xcode/UserData/Provisioning\ Profiles/

      # 5. Install Fastlane using Homebrew
      - name: Install Fastlane
        run: |
          brew install fastlane

      # 6. Run Fastlane lane for TestFlight upload
      - name: Run Fastlane Lane
        env:
          PROFILE: “#YOUR_DISTRIBUTION_PROVISIONING_PROFILE_NAME#“
          APP_STORE_CONNECT_KEY_ID: ${{ secrets.APP_STORE_CONNECT_KEY_ID }}
```



```
          APP_STORE_CONNECT_ISSUER_ID: ${{ secrets.APP_STORE_CONNECT_ISSUER_ID }}
          APP_STORE_CONNECT_PRIVATE_KEY:
${{ secrets.APP_STORE_CONNECT_PRIVATE_KEY }}
        run: |
          fastlane beta --verbose
```

이 스크립트에는 "runs-on: macos-15"와 "xcode-version: '16.0'" 항목이 포함되어 있는데 이는 프로비저닝 프로파일 저장 경로가 Xcode 16 버전부터 변경되었기 때문입니다. Xcode 버전을 16으로 지정하지 않으면 프로파일이 잘못된 경로로 복사되고 이로 인해 IPA 후처리 단계에서 재서명에 실패하게 됩니다.

이제 앱 빌드 및 재서명, 업로드를 위한 작업에 AppSealing SDK 관련 작업이 반영되도록 Fastfile도 다음과 같이 수정합니다.

Github Action + AppSealing SDK 적용 시의 Fastfile

```
default_platform(:ios)

platform :ios do
  desc "Push a new beta build to TestFlight"
  lane :beta do
    require 'net/http'
    require 'uri'
    require 'json'
    require 'jwt'

    # Get project root path (parent directory relative to Fastfile)
    project_root = File.expand_path("..", __dir__)
    fastlane_path = File.expand_path(".", __dir__)

    KEY_ID = ENV['APP_STORE_CONNECT_KEY_ID'] # Import from GitHub Actions Secret
    ISSUER_ID = ENV['APP_STORE_CONNECT_ISSUER_ID'] # Import from GitHub Actions
Secret
    PRIVATE_KEY_CONTENT = ENV['APP_STORE_CONNECT_PRIVATE_KEY'] # Import private key
contents into environment variable

    unless KEY_ID && ISSUER_ID && PRIVATE_KEY_CONTENT
      UI.user_error!("Missing required App Store Connect credentials. Ensure
APP_STORE_CONNECT_KEY_ID, APP_STORE_CONNECT_ISSUER_ID, and
APP_STORE_CONNECT_PRIVATE_KEY are set as environment variables.")
    end

    # Automatically detects Xcode project (.xcodeproj) or workspace (.xcworkspace)
files
    xcode_file_path = Dir.glob(File.join(project_root, "*.xcworkspace")).first ||
```



```
Dir.glob(File.join(project_root, "*.xcodeproj")).first
    if xcode_file_path.nil?
      UI.user_error!("[AppSealing] No Xcode project or workspace found in the
project root directory: #{project_root}")
    end

    # Automatically extract project name and scheme
    project_name = File.basename(xcode_file_path, File.extname(xcode_file_path))

    # Auto-search scheme
    scheme_name = ""
    Dir.chdir(File.dirname(xcode_file_path)) do
      scheme_name = sh("xcodebuild -list -
#{xcode_file_path.end_with?('.xcworkspace') ? 'workspace' : 'project'}
#{File.basename(xcode_file_path)} | grep 'Schemes:' -A 1 | tail -n 1").strip
    end

    if scheme_name.empty?
      UI.user_error!("[AppSealing] Failed to retrieve scheme name from Xcode
project or workspace.")
    end

    UI.message "[AppSealing] Project Name: #{project_name}"
    UI.message "[AppSealing] Scheme Name: #{scheme_name}"

    # Dynamically extract bundle ID
    bundle_id = ""
    Dir.chdir(File.dirname(xcode_file_path)) do
      bundle_id_command = "xcodebuild -showBuildSettings -scheme #{scheme_name} |
grep 'PRODUCT_BUNDLE_IDENTIFIER' | awk -F ' = ' '{print $2}'"
      bundle_id_output = sh(bundle_id_command).lines.map(&:strip).reject { |line|
line.include?("WARNING") || line.empty? }
      bundle_id = bundle_id_output.last.strip
    end

    if bundle_id.empty?
      UI.user_error!("[AppSealing] Failed to retrieve Bundle ID from Xcode
project.")
    end

    UI.message "[AppSealing] Bundle ID: #{bundle_id}"

    # Dynamically extract Team ID
    team_id = ""
    Dir.chdir(File.dirname(xcode_file_path)) do
      team_id_command = "xcodebuild -showBuildSettings -scheme #{scheme_name} |
grep 'DEVELOPMENT_TEAM' | awk -F ' = ' '{print $2}'"
      team_id_output = sh(team_id_command).lines.map(&:strip).reject { |line|
line.include?("WARNING") || line.empty? }
      team_id = team_id_output.last.strip
    end
```



```
    if team_id.empty?
      UI.user_error!("[AppSealing] Failed to retrieve Team ID from Xcode project.")
    end

    UI.message "[AppSealing] Team ID: #{team_id}"

    # JWT token generation function
    def generate_jwt_token(private_key_content)
      private_key = OpenSSL::PKey::EC.new(private_key_content)
      payload = {
        iss: ISSUER_ID,
        exp: Time.now.to_i + 20 * 60, # 20 minutes validity period
        aud: "appstoreconnect-v1"
      }
      header = { kid: KEY_ID }
      JWT.encode(payload, private_key, 'ES256', header)
    end

    # Get CFBundleShortVersionString (Marketing Version) function in Xcode project
    def fetch_version_from_xcode(xcodeproj_path, scheme_name)
      version_command = "xcodebuild -project #{xcodeproj_path} -scheme
#{scheme_name} -showBuildSettings | grep MARKETING_VERSION | sed
's/[ ]*MARKETING_VERSION = //'"
      version = `#{version_command}`.strip

      if version.empty?
        UI.user_error!("Failed to retrieve MARKETING_VERSION from Xcode project.")
      end

      UI.message("[AppSealing] Retrieved MARKETING_VERSION: #{version}")
      version
    end

    # Function to get the latest build number
    def fetch_latest_build_number(jwt_token, bundle_id, version)
      # Get the App ID (if required)
      app_id = fetch_app_id(jwt_token, bundle_id)

      # API request URL
      uri =
URI("https://api.appstoreconnect.apple.com/v1/builds?filter[app]=#{app_id}&filter[p
reReleaseVersion.version]=#{version}&sort=-version")
      request = Net::HTTP::Get.new(uri)
      request['Authorization'] = "Bearer #{jwt_token}"

      # Execute API request
      response = Net::HTTP.start(uri.hostname, uri.port, use_ssl: true) do |http|
        http.request(request)
      end
```



```
      # Response processing
      if response.code.to_i == 200
        builds = JSON.parse(response.body)['data']
        if builds && !builds.empty?
          latest_build_number = builds.first['attributes']['version'].to_i
          return latest_build_number
        else
          UI.message("[AppSealing] No builds found for version #{version}.")
        end
      else
        UI.error("[AppSealing] Failed to fetch builds: #{response.code} -
#{response.body}")
      end

      nil # Returns nil if no build is found.
    end

    # Get App ID function (if needed)
    def fetch_app_id(jwt_token, bundle_id)
      uri =
URI("https://api.appstoreconnect.apple.com/v1/apps?filter[bundleId]=#{bundle_id}")
      request = Net::HTTP::Get.new(uri)
      request['Authorization'] = "Bearer #{jwt_token}"

      response = Net::HTTP.start(uri.hostname, uri.port, use_ssl: true) do |http|
        http.request(request)
      end

      if response.code.to_i == 200
        apps = JSON.parse(response.body)['data']
        if apps && !apps.empty?
          app_id = apps.first['id']
          return app_id
        else
          UI.user_error!("[AppSealing] No app found with bundle ID: #{bundle_id}")
        end
      else
        UI.error("[AppSealing] Failed to fetch app ID: #{response.code} -
#{response.body}")
        UI.user_error!("[AppSealing] Unable to retrieve app ID for bundle ID:
#{bundle_id}")
      end

      nil # Returns nil if the app ID is not found.
    end

    # Generate JWT token and get latest build number
    jwt_token = generate_jwt_token(PRIVATE_KEY_CONTENT)
    version = fetch_version_from_xcode(xcode_file_path, scheme_name)
    latest_build_number = fetch_latest_build_number(jwt_token, bundle_id, version)
```



```
    if latest_build_number.nil?
      UI.message("[AppSealing] No builds found for version #{version}. Starting
with build number 1.")
      new_build_number = 1
    else
      new_build_number = latest_build_number + 1
      UI.message("[AppSealing] Latest Build Number: #{latest_build_number}")
    end

    # Set a new build number (applies to Xcode projects)
    increment_build_number(
      build_number: new_build_number,
      xcodeproj: xcode_file_path # Specify Xcode project path
    )

    UI.message "[AppSealing] Updated Build Number: #{new_build_number}"

    # Build and generate IPA file
    archive_path = File.join(project_root, "build", "#{project_name}.xcarchive")
    ipa_output_path = File.join(project_root, "build")

    build_ios_app(
      scheme: scheme_name,
      export_method: "app-store",
      clean: true,
      output_directory: ipa_output_path,
      output_name: "#{project_name}.ipa",
      workspace: xcode_file_path.end_with?(".xcworkspace") ? xcode_file_path : nil,
      project: xcode_file_path.end_with?(".xcodeproj") ? xcode_file_path : nil,
      export_options: {
        provisioningProfiles: {
          bundle_id => ENV["PROFILE"]
        }
      },
      xcargs: "CODE_SIGN_STYLE=Manual DEVELOPMENT_TEAM=#{team_id}
PROVISIONING_PROFILE_SPECIFIER=\"#{ENV['PROFILE']}\"",
      codesigning_identity: "Apple Distribution"
    )

    # Set .ipa file path
    ipa_path = File.join(ipa_output_path, "#{project_name}.ipa")

    unless File.exist?(ipa_path)
      UI.user_error!("[AppSealing] IPA file not found at path: #{ipa_path}")
    end

    UI.message "[AppSealing] IPA Path: #{ipa_path}"

    # Dynamic search for generate_hash script path
    generate_hash_script = File.join(project_root, "AppSealingSDK", "Tools",
"generate_hash")
```

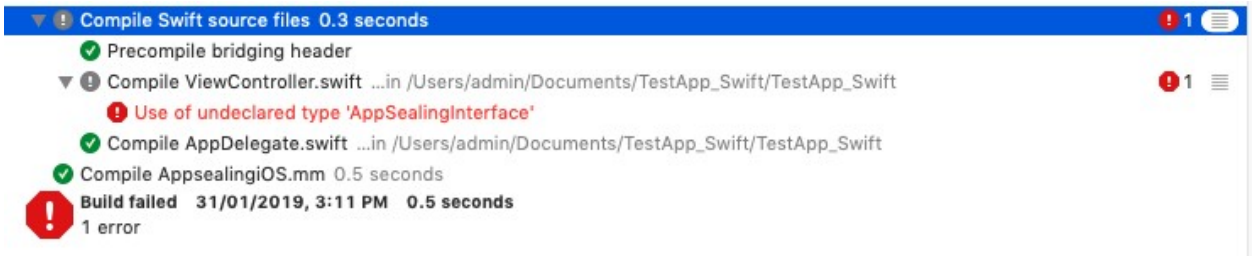


```
    unless File.exist?(generate_hash_script)
      UI.user_error!("[AppSealing] generate_hash script not found at path:
#{generate_hash_script}")
    end

    # Check and set execution permissions
    unless File.executable?(generate_hash_script)
      sh("chmod +x '#{generate_hash_script}'")
    end
    sh("'#{generate_hash_script}' '#{File.absolute_path(ipa_path)}'")

     # Step 2: Upload the IPA to TestFlight with callback
    begin
      upload_to_testflight(
        ipa: ipa_path,
        api_key: {
          key_id: KEY_ID,
          issuer_id: ISSUER_ID,
          key: PRIVATE_KEY_CONTENT
        },
        skip_waiting_for_build_processing: true, # Skip build processing wait time
      )
      UI.success("[AppSealing] Upload to TestFlight completed successfully!")

    rescue => e
      UI.error("[AppSealing] Upload to TestFlight failed with error: #{e.message}")
      raise e

    ensure
      # Always delete the build directory, even if an error occurs
      if Dir.exist?(ipa_output_path)
        UI.message("[AppSealing] Deleting build directory: #{ipa_output_path}")
        FileUtils.rm_rf(ipa_output_path)
        UI.message("[AppSealing] Build directory deleted.")
      else
        UI.message("[AppSealing] Build directory not found or already deleted.")
      end
    end
  end
end
```

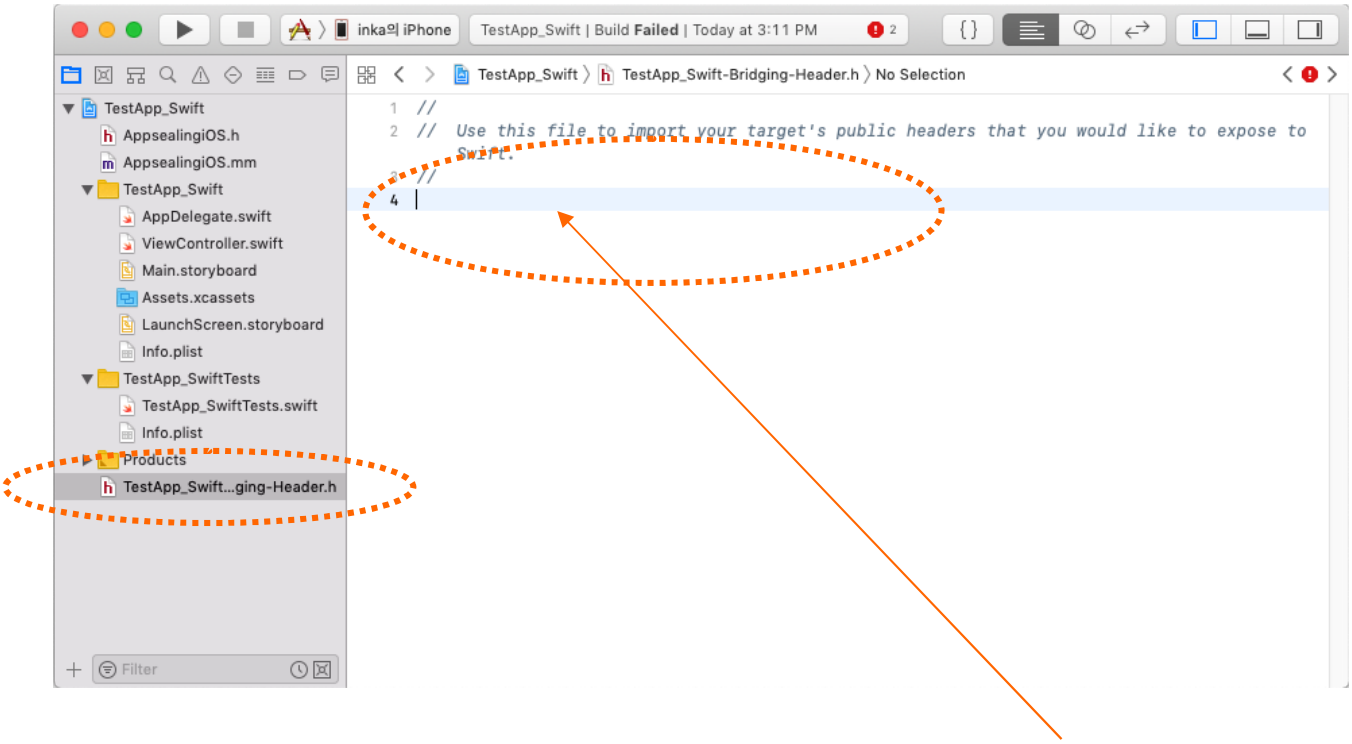
This script includes the entire process of getting the app information from App Store Connect, automatically increasing the build number, building the app, exporting it as an IPA, applying the generate_hash script, re-signing it, and uploading it to App Store Connect. All values required for this process, such as project name, scheme, target, and bundle ID, are automatically extracted and used from the project file, so the same script can be used for projects other than TestApp_Swift.

Part 10. Troubleshooting

10-1 Xcode Build error (1) **Use of undeclared type 'AppSealingInterface'**



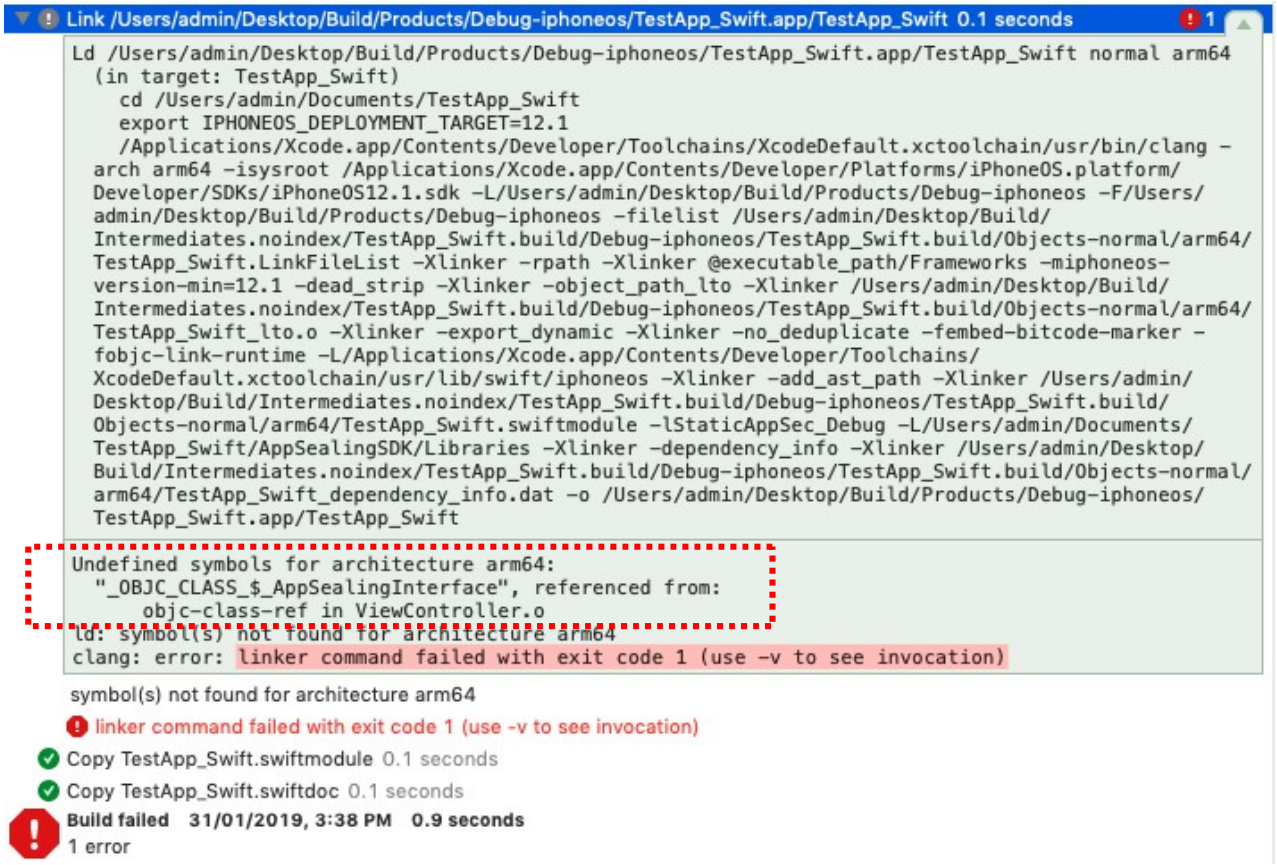
You will gen an error message like above when you omit or miss-typed '#import "AppsealingiOS.h"' sentence in Bridging Header file.



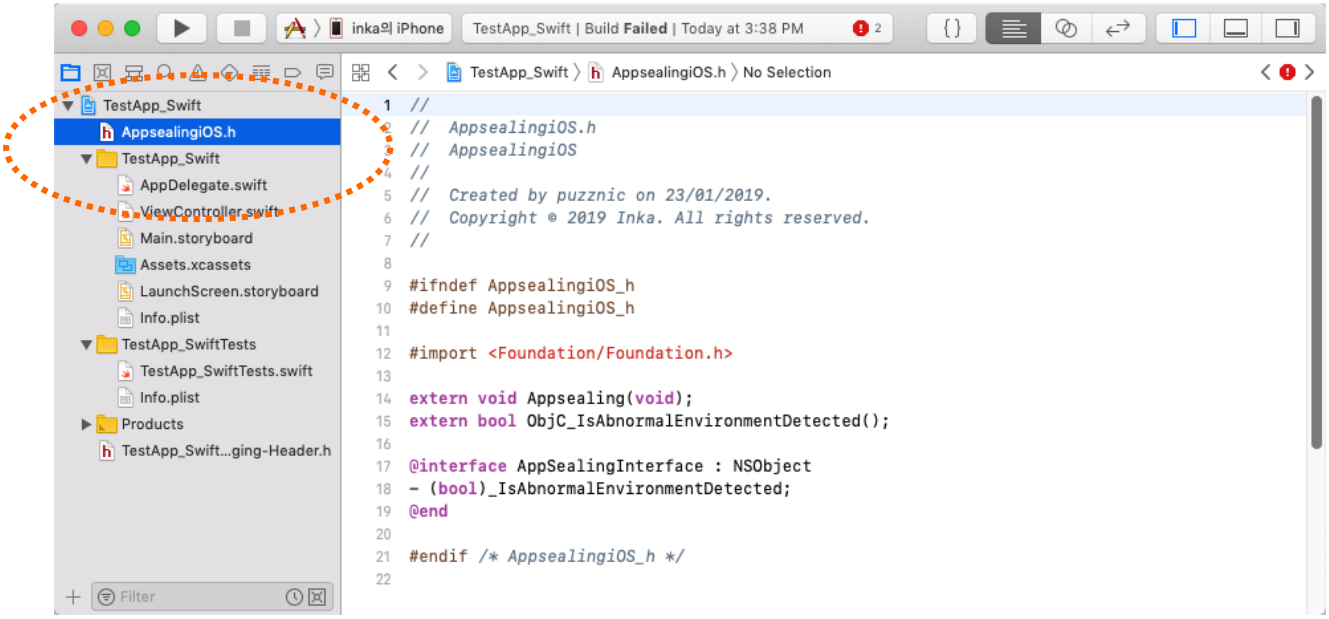
Select "TestApp_Swift-Bridging-Header.h" and append '**#import "AppsealingiOS.h**"' at the end of document. Be sure that "AppsealingiOS.h" file exist in the designated folder (Xcode project/AppSealingSDK/Libraries/AppsealingiOS.h)

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

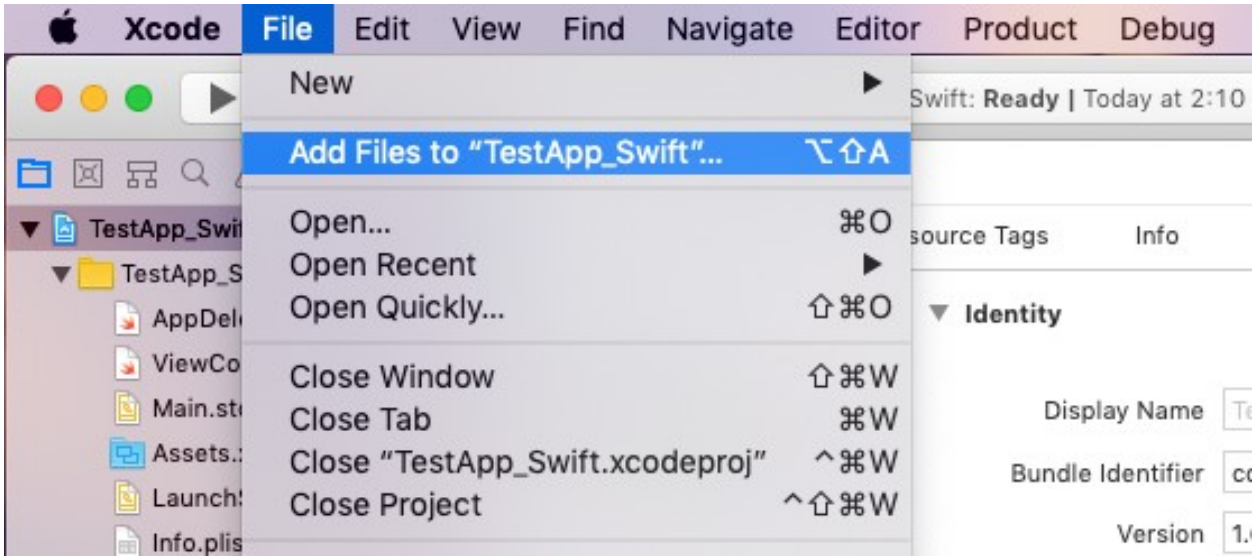
10-2 Xcode Build error (2) **Undefined symbols for architecture arm64: "_OBJC_CLASS_$_AppSealingInterface"**

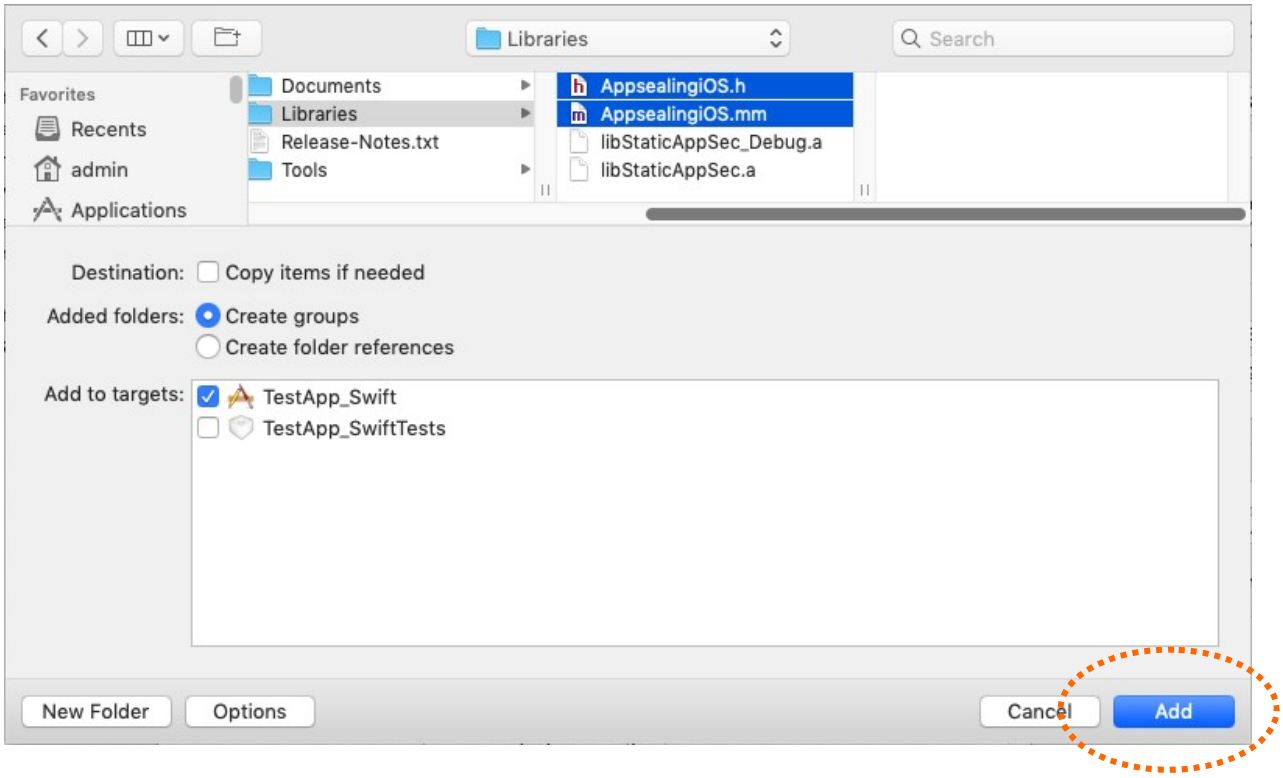


This linker error occurs when the "AppsealingiOS.mm" is not included in your project. Check project tree whether the file has added or not.

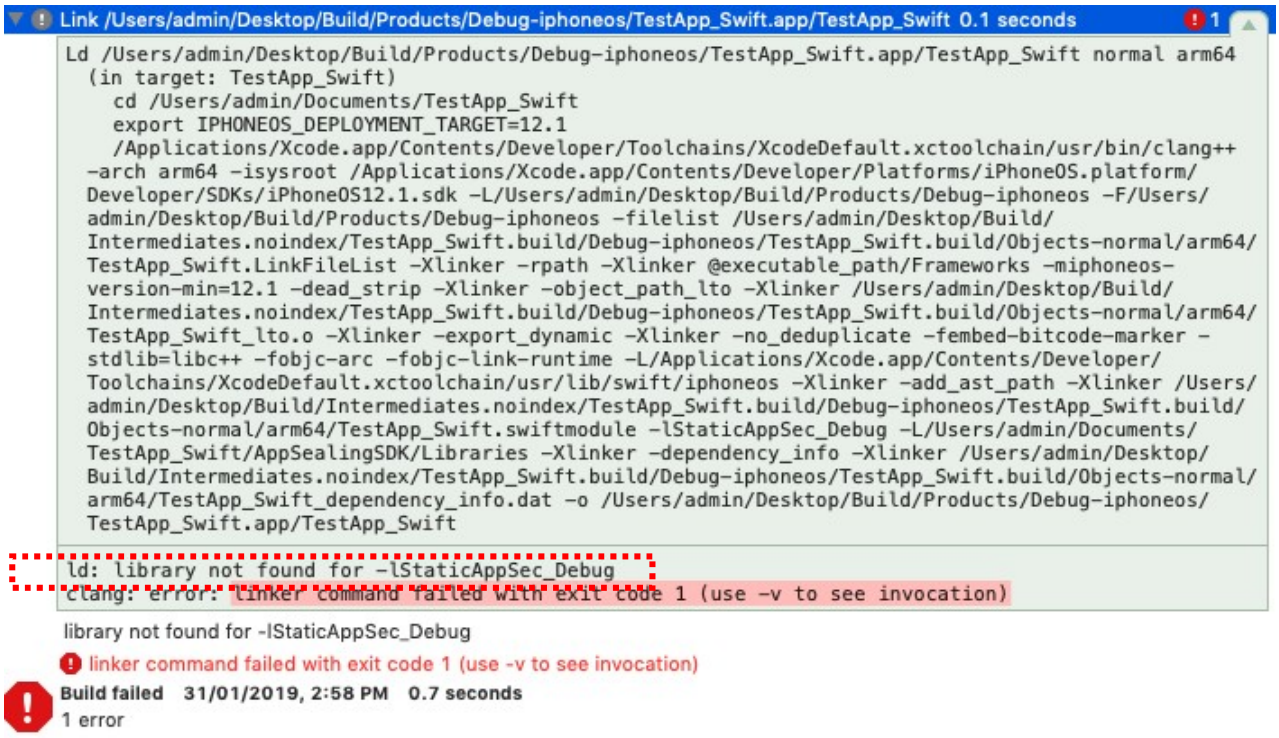


If the "AppsealingiOS.mm" file is not included in your project, perform "File > Add Files to "TestApp_Swift" ... " menu action.



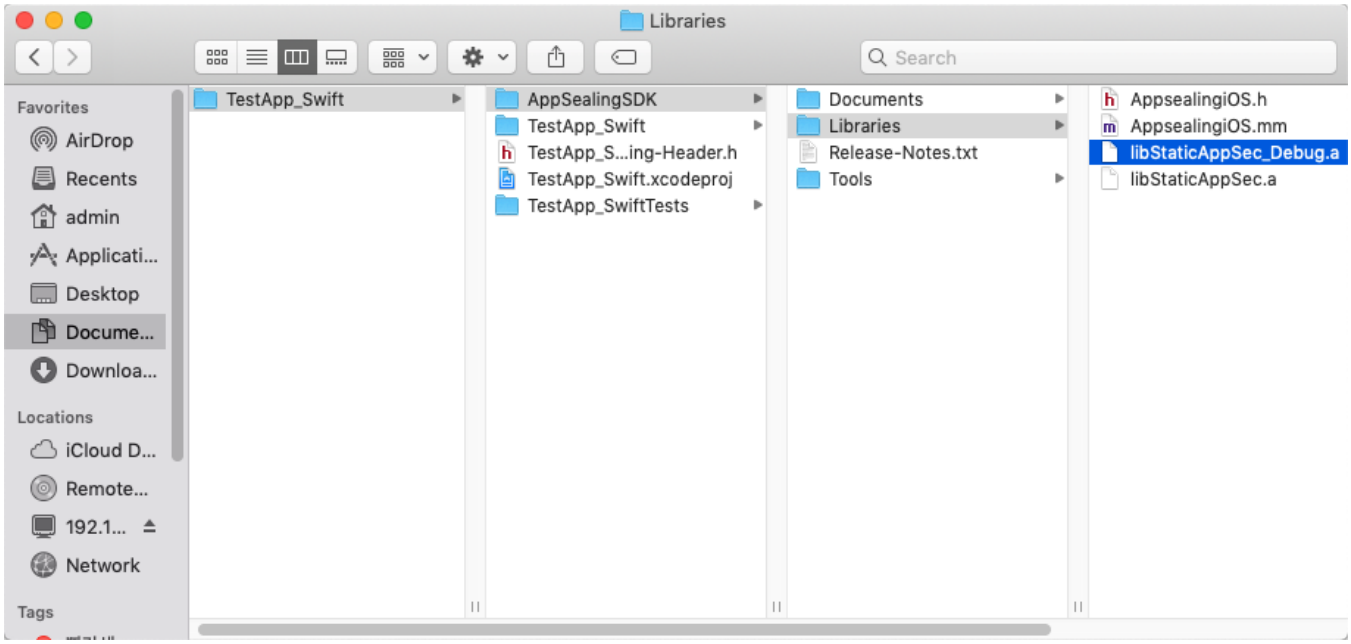


10-3 Xcode Build error (3) **ld: library not found for -lStaticAppSec_Debug**



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"

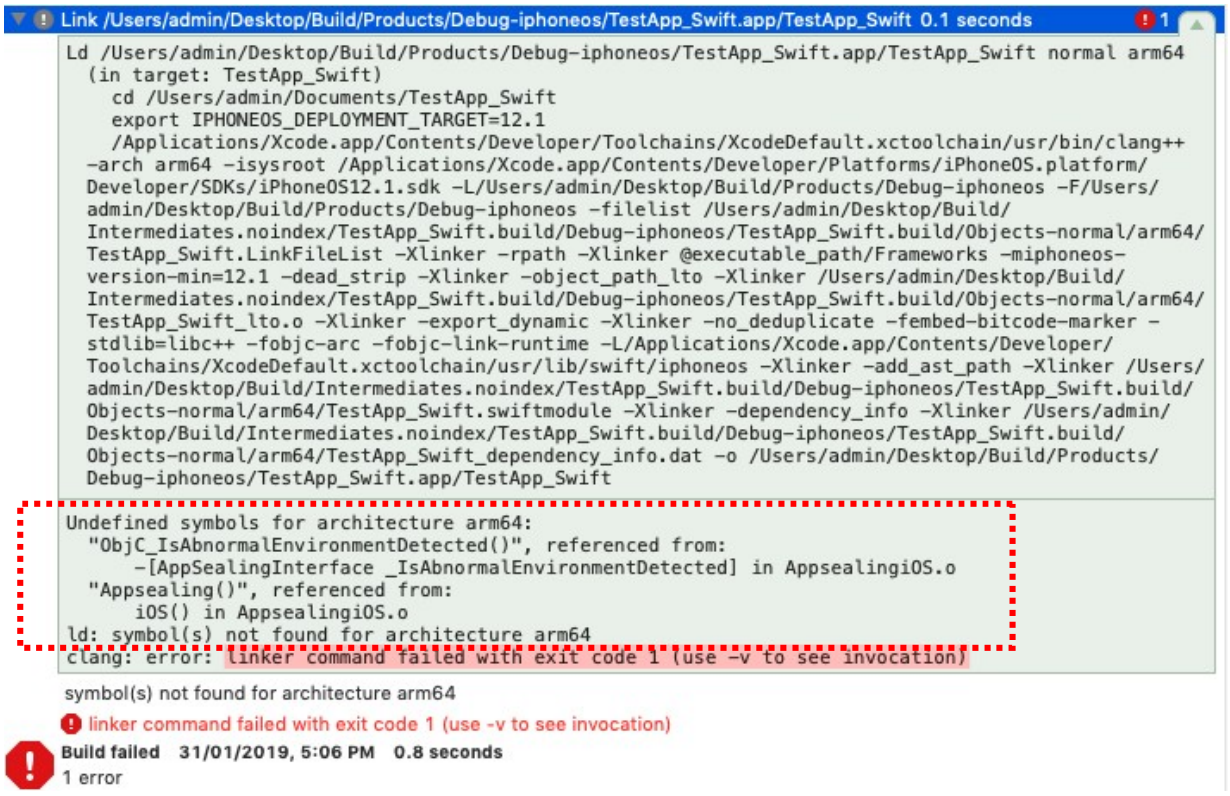


If the file is missing (or maybe corrupted) try to re-download or re-expand AppSealingSDK zip file.

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.

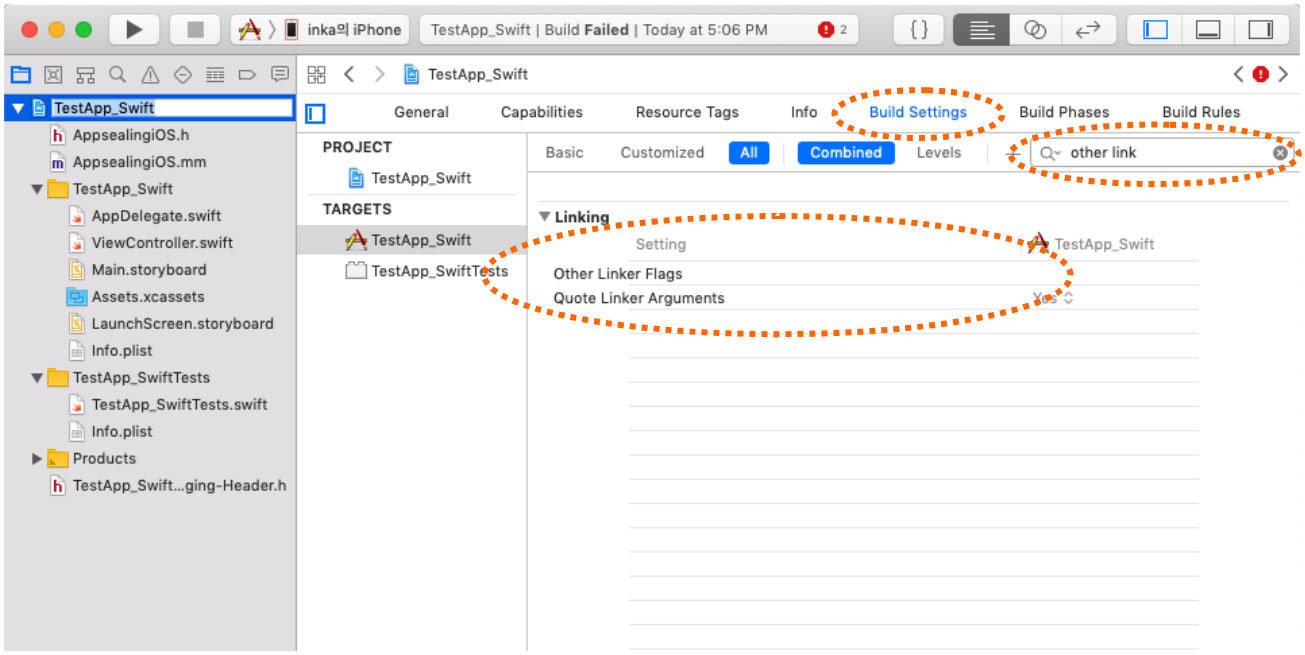
"TestApp_Swift (Project folder) > AppSealingSDK > Libraries > libStaticAppSec.a"

10-4 Xcode Build error (4) **Undefined symbols for architecture arm64: "ObjC_IsAbnormalEnvironmentDetected()", "Appsealing()"**



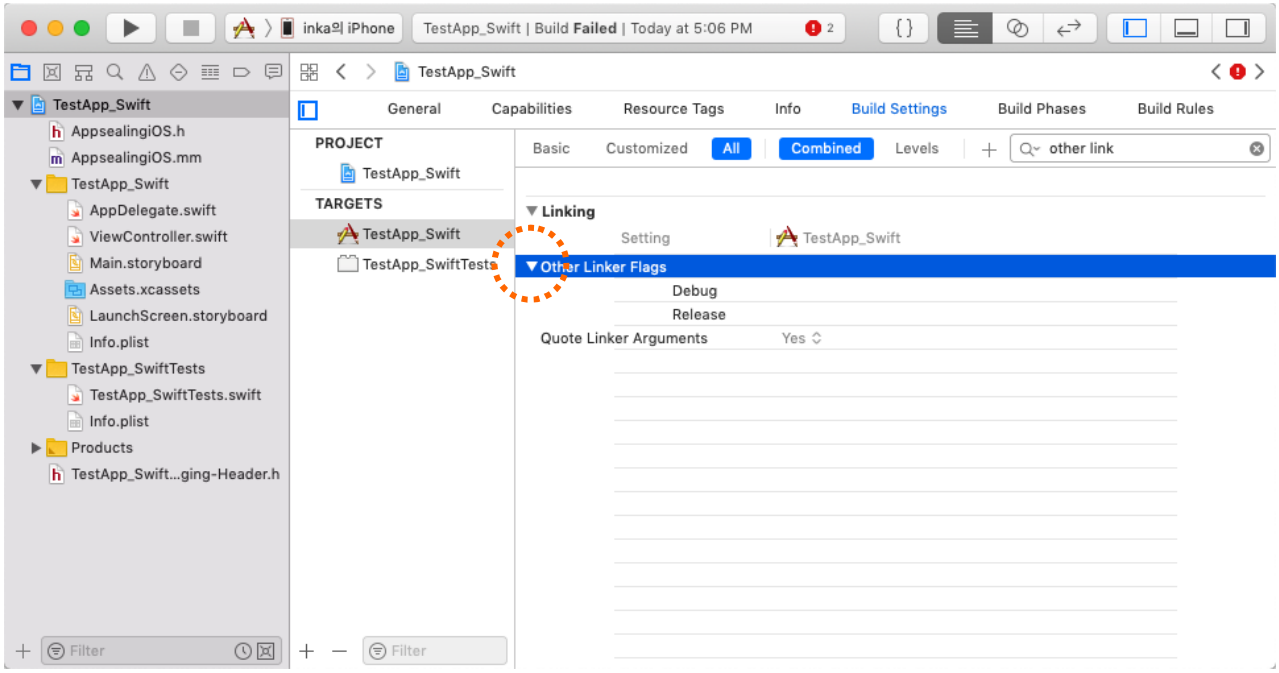
This linker error occurs when the value of "Other Linker Flags" field within "Build Settings" configuration section is missing or invalid.

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



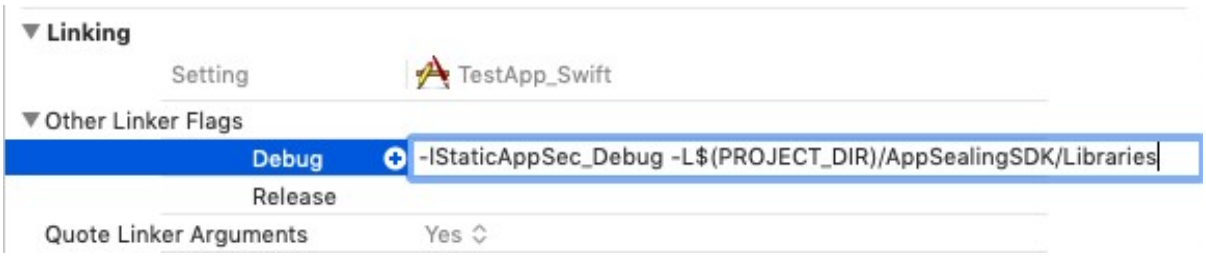
Now, restore the "Other Linker Flags" settings by following steps.

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.

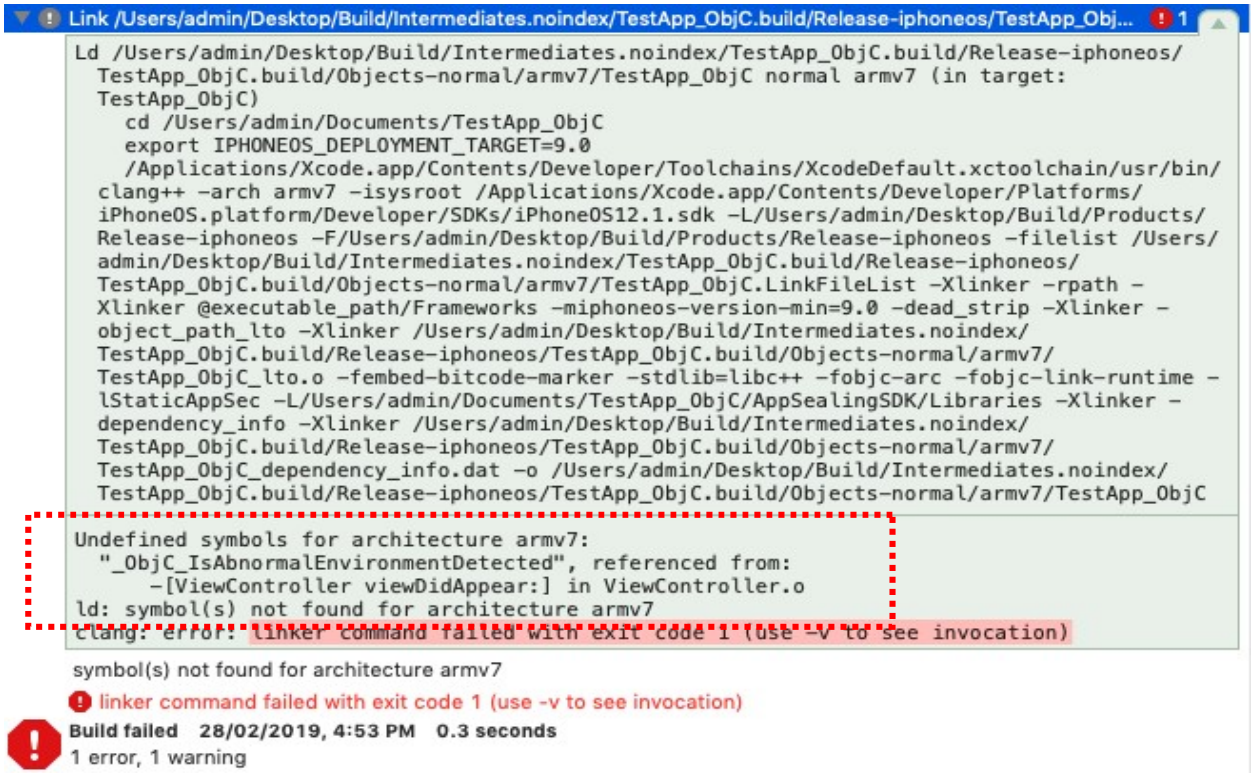
-lStaticAppSec_Debug -L$(PROJECT_DIR)/AppSealingSDK/Libraries



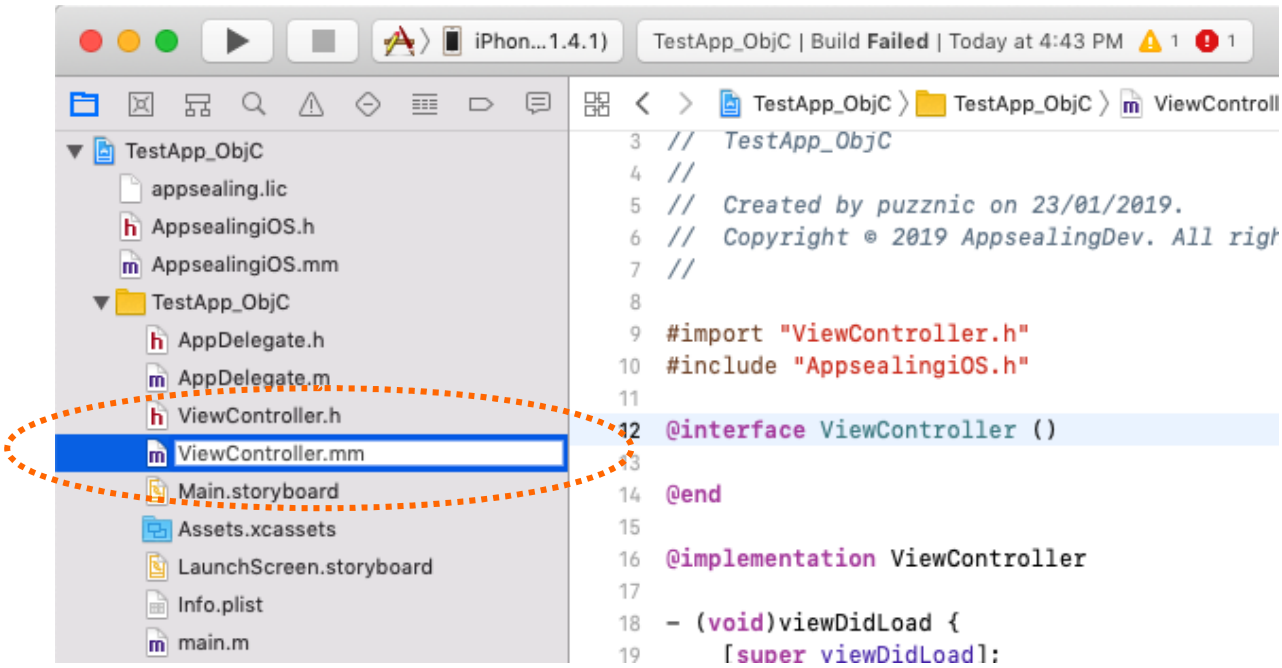
4) Select "Release" item and click to edit/insert value, then type in following text.

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

10-5 Xcode Build error (5) **Undefined symbols for architecture arm64: "ObjC_IsAbnormalEnvironmentDetected()"** (Objective-C project only)

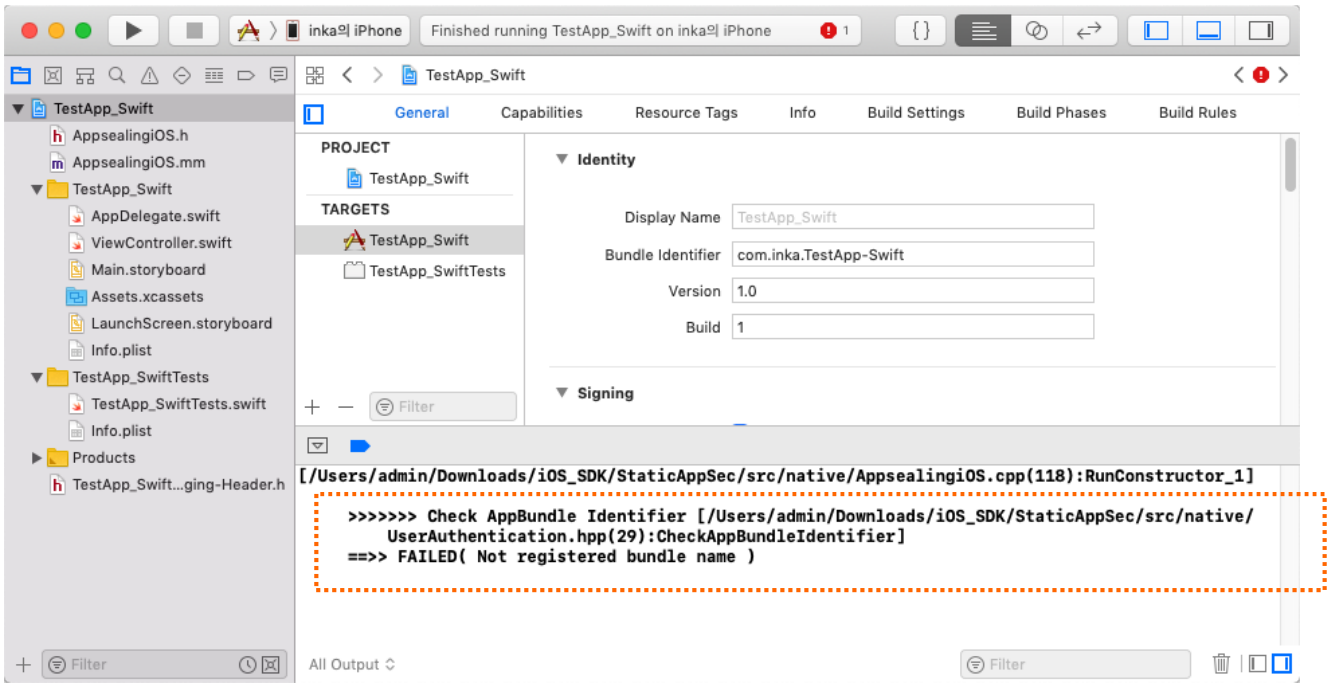


This linker error occurs when your project is Objective-C based and ViewController source code file has extension "**.m**". Just change the extension to "**.mm**".

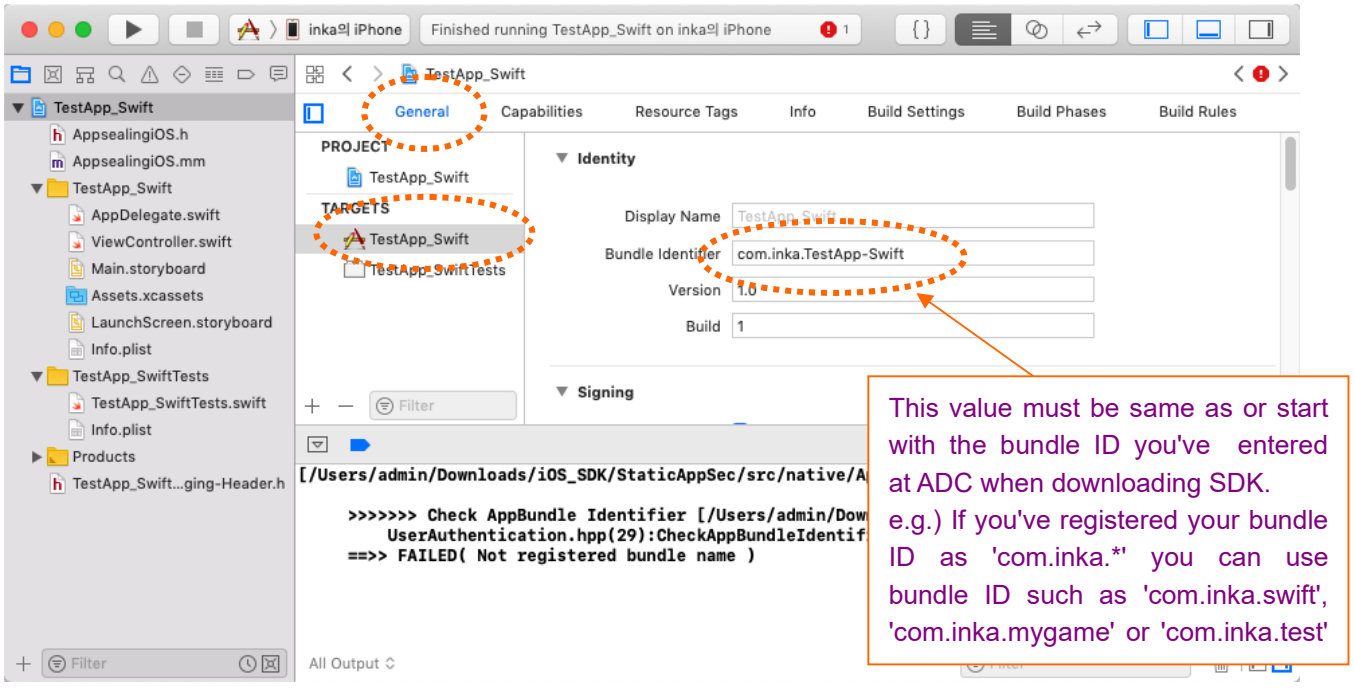


10-6 Execution error (I) **App terminated immediately after launch**

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

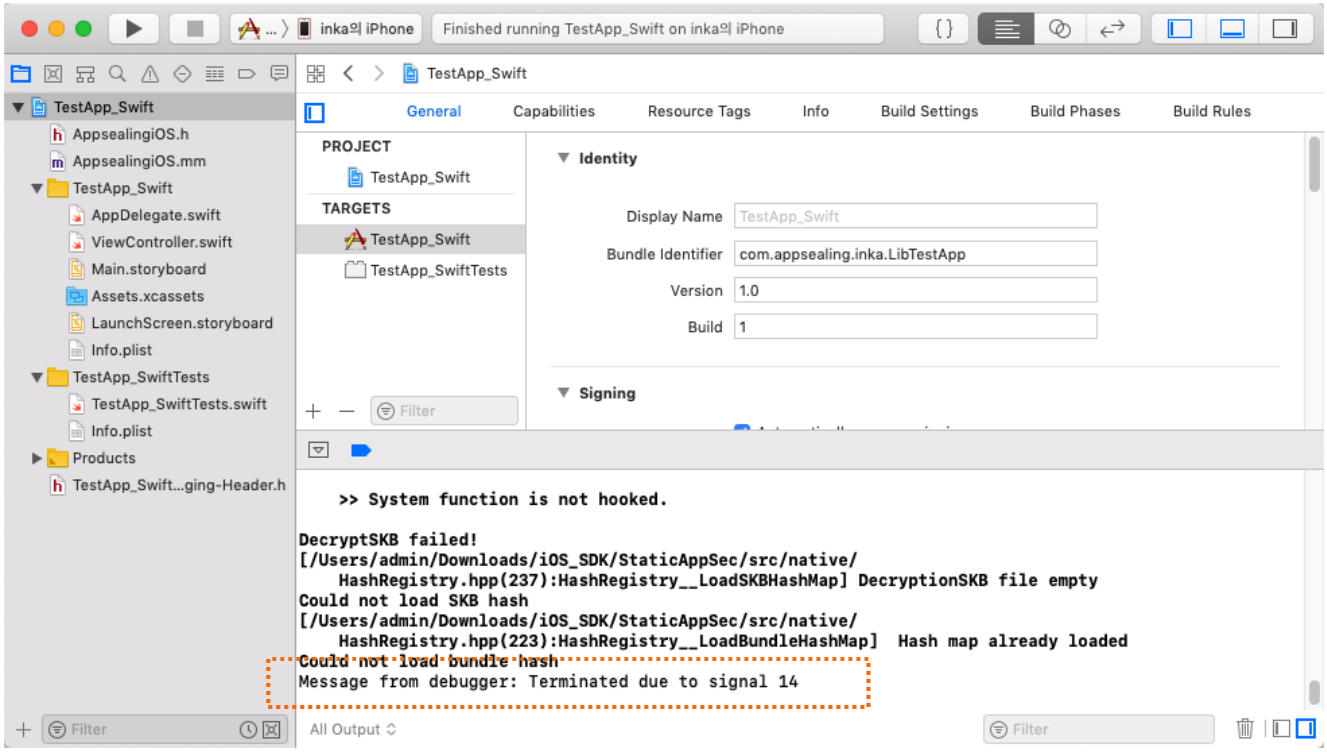


If the log message contains string like " ==>> FAILED(Not registered bundle name)" then it tells the bundle Id you used is not registered properly at AppSealing Developer Center (ADC) while downloading AppSealing SDK file. Check your bundle Id if it is same as the value you entered when downloading SDK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



10-7 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.

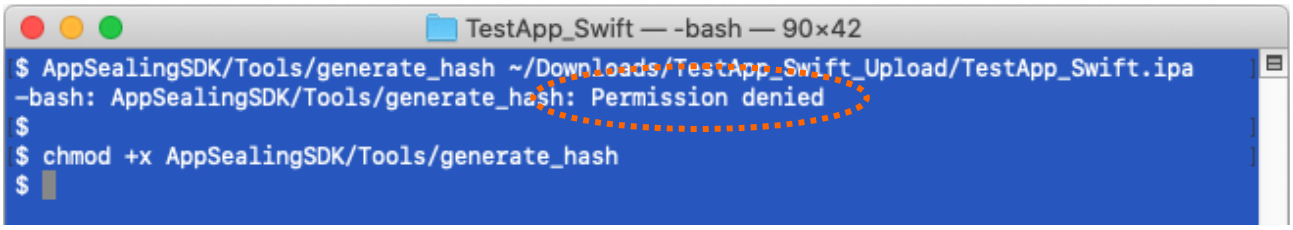


If the log message contains string like "Message from debugger: Terminated due to signal 14" then it tells that you have built & run your app in Release configuration and so AppSealing detection logic has activated. AppSealing logic in Release configuration checks some security intrusion such as jailbreak, 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.

10-8 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
```

