InstallAnywhere 2014 Hotfix A: Changes to Authentication and Code-Signing Support for OS X–Based Installers

This article explains the new processes that are involved in authentication and code-signing support for OS X–based installers as of InstallAnywhere 2014 Hotfix A. Hotfix A is available for download from the InstallAnywhere 2014 Hotfix A KB article.

Run-Time Behavior for a Code-Signed Installer that Includes Authentication Support

An installer that includes authentication support consists of the following primary files:

- An authentication wrapper
- A helper tool—the file that is used to launch the installer or uninstaller with elevated privileges
- The installer application—that is, the Java installer
- Optionally, the uninstaller application. Note that InstallAnywhere generates the uninstaller and bundles it into the installer at build time.

The authentication wrapper, helper tool, installer, and uninstaller must all be code signed with the same Developer ID Application certificate.

At run time on target systems, the following process occurs for a properly code-signed installer or uninstaller that includes authentication support:

1. The end user launch the authentication wrapper on an OS X–based system.
2. The authentication wrapper prompts for elevation (if needed for a standard user) and installs the helper tool with root privileges.
3. The authentication wrapper requests that the helper tool launch the LaunchAnywhere.
4. The helper tool launches the LaunchAnywhere, which inherits the elevated privileges from the helper tool.
5. The LaunchAnywhere launches the installer or uninstaller, which inherits the elevated privileges.

At the end of the installation or uninstallation, the authentication wrapper shuts down the helper tool and uninstalls it. If any part of the process is not signed with a matching Developer ID Application certificate, the process fails.

Note: Unexpected results may occur if (a) a user is using VNC to connect remotely to an OS X–based target system, (b) one or more additional users are either logged in physically to that same system or connected remotely using VNC, and (c) one of the users launches a code-signed installer that includes authentication support is launched on that system.

That is, if two users are connected to a target system and at least one of them is using VNC for their connection, and then the second user launches an installer that requires elevation, the prompt for elevated credentials is displayed in the second user's session. However, the installer panels are displayed in the session of the first user.

This is logged as issue IOJ-1718395.

Code-Signing Options for InstallAnywhere Projects

InstallAnywhere supports two different methods of code signing:

- **InstallAnywhere performs the code-signing step at build time on the InstallAnywhere build machine**—The InstallAnywhere user has access to the Developer ID Application certificate and, in the InstallAnywhere project, configures the Code Signing settings (the certificate location and the keystore password). At build time, InstallAnywhere code signs the appropriate output files.
- **The InstallAnywhere build output is code signed on a designated code-signing machine**—The InstallAnywhere user builds an unsigned installer on the build machine. The build output is then code signed on a secure code-signing machine that has the Developer ID Application certificate. **Note:** A limitation of this method is that it is not possible to sign the uninstaller; therefore, the resulting uninstaller cannot be used to uninstall the product. The only way to remove a product whose installer was code signed using this method is to drag it to the Trash.

Requirements for Code Signing

The following requirements must be met for code signing OS X–based installers:

- The OS X–based installer must be built on an OS X–based system.
- All code signing must be done on systems that are running OS X 10.9 or later, since these versions can create version 2 signatures. Version 1 signatures, which are created by earlier versions of OS X, are not recognized by Gatekeeper on systems with OS X 10.9 and later and are considered obsolete. Files that are signed with version 2 signatures will work on OS X 10.8 and later. To learn more, see Technical Note TN2206: OS X Code Signing in Depth in the Mac Developer Library.
- A Developer ID Application certificate must be used to sign the files.
- It is recommended that the Developer ID Application certificate be added to the login keychain—not the system keychain—on the machine that is going to be used for code signing, and that the same user account that was used to add the certificate to the login keychain be used to sign files.
- If you plan on performing builds through the command-line console, ensure that the certificate has been granted access to be used by all applications.
- Ensure that the latest Xcode IDE and all of its default SDKs are installed on the machine that is going to be used for code signing.

Adding the Code-Signing Capability to Your InstallAnywhere Build Machines or Code-Signing Machines

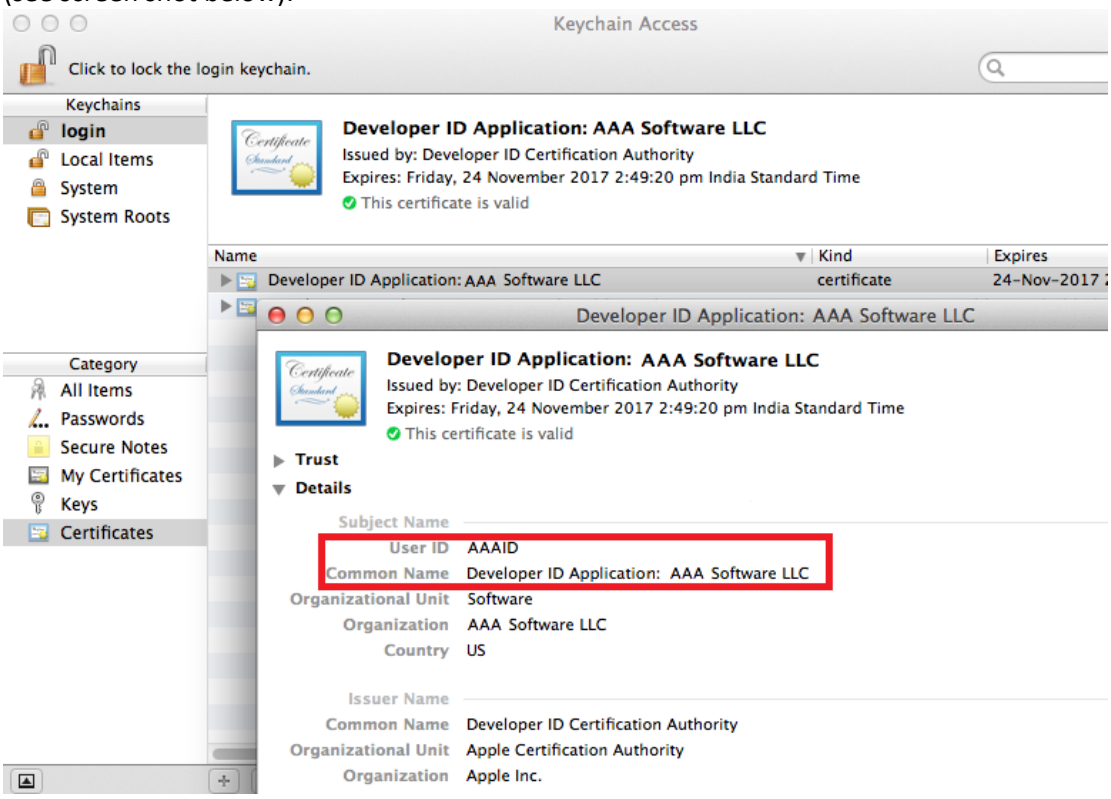
The following instructions explain how to prepare your InstallAnywhere build machine or your code-signing machine so that you can code sign your OS X–based installers and include authentication support.

Note: If you want to prepare your InstallAnywhere build machine or your code-signing machine for code signing your installers, but you do not need to be able to include authentication support in your installers, perform step 1 only.

To add the code-signing capability, as well as the helper tool that is signed with your own certificate, to your machines:

1. Set up the Keychain Access utility on all of your machines that you will be using for code signing (either InstallAnywhere build machines or separate designated signing machines):
   a. Add your Developer ID Application certificate to the login keychain. For information on adding a certificate to a keychain, see Adding certificates to a keychain in the Mac Developer Library.
   b. *If you plan on performing builds through the command-line console:* Ensure that the certificate has been granted access to be used by all applications.
      i. In **Keychain Access**, right-click the certificate and then click **Get Info**.
      ii. On the **Access Control** tab, click the **Allow all applications to access this item** option.
2. Ensure that the latest Xcode IDE and all of its default SDKs are installed.
3. Sign the helper tool with your own Developer ID Application certificate:
   a. *If you are using a separate designated signing machine*, copy the helper tool signing files from your InstallAnywhere build machine to your designated signing machine: Copy the ht-signer folder in the *IA_INSTALL_DIR*`/resource/nativetools/macosx` folder to the designated signing machine.

b. In the ht-signer folder (either in the *IA_INSTALL_DIR*/resource/nativetools/macosx folder on your InstallAnywhere build machine or the folder that you copied to the designated signing machine), find the build-helper-tool.sh file, open it in an editor, and customize the following entries:
   - **CERTIFICATE_ID**—Specify the common name of the certificate. This name is displayed in Keychain Access (see screen shot below). The common name must use syntax such as this: "Developer ID Application: AAA Software LLC"
   - **CERTIFICATE_DEV_ID**— Specify the user ID of the certificate. This ID is displayed in Keychain Access (see screen shot below).



   - **OUTPUT_DIR**— Specify the output directory where the helper tool will be copied. (If you are working on the InstallAnywhere build machine, you can specify the *IA_INSTALL_DIR*/resource/nativetools/macosx directory. Doing so will replace the helper tool that shipped in InstallAnywhere 2014 Hotfix A with the one that is signed with your certificate.
   - **SDK_PATH**—Specify the Xcode SDK directory. If Xcode SDK is available in the default path, you can leave this blank. If you are logged in on an OS X 10.9–based machine, SDK 10.9 is the default SDK. If you want to specify a separate SDK, or if the SDK is not installed, provide the absolute path of the SDK. For example: SDK_PATH="/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.9.sdk"

c. Change the current directory to the ht-signer directory (either in the *IA_INSTALL_DIR*/resource/nativetools/macosx folder on your InstallAnywhere build machine or the ht-signer folder that you copied to the designated signing machine), and execute the revised build-helper-tool.sh script in that directory. The signed helper tool—called com.flexera.ia.helper—is created in the output directory that you specified for OUTPUT_DIR.

4. Verify that the helper tool was successfully signed. To learn how, see Verifying that Your Output Files Are Working as Expected.

5. Make the verified, signed helper tool available to all of your InstallAnywhere build machines: Copy the com.flexera.ia.helper tool to the *IA_INSTALL_DIR*/resource/nativetools/macosx folder on each InstallAnywhere build machine.

Your machines are now ready to code sign your installers. The next time that you build an installer that includes authentication support, InstallAnywhere includes your signed helper tool with your installer.

If you install a new version of InstallAnywhere, or if you are replacing a Developer ID Application certificate with a new one, the aforementioned process needs to be repeated.

Code Signing Your Installers and Including Authentication Support

The process for code signing your installers varies, depending on whether you are performing the code-signing step at build time on the InstallAnywhere build machine or on a separate designated code-signing machine.

Option 1: Code Signing the Authentication Wrapper, Installer, and Uninstaller While Building on the InstallAnywhere Build Machine

Once you have added the code-signing capability to one of your InstallAnywhere build machines, you can use that machine to build authenticated installers for OS X–based target systems.

To configure your InstallAnywhere project to code sign your build output and include authentication support:

1. In the Advanced Designer, on the **Project** page, click **Platforms**. The **Platforms** view opens.
2. In the **Mac OS X** area, in the **Code Signing** setting, select the **Code Sign the Generated Installer** check box.
3. Specify the location and password of the certificate. The use of build-time variables for the certificate location and password is highly recommended for security purposes.
   a. For the **PKCS #12 File** setting, specify the fully qualified path for your PKCS #12 file (.p12).
   b. In the **Keystore Password** setting, specify the certificate's password.

   Note: The certificate you specify must be the same Developer ID Application certificate that was used to add the code-signing capability to your machines, as described in Adding the Code-Signing Capability Your InstallAnywhere Build Machines or Code-Signing Machines.
4. Under the **Authentication** category, in the **Requires an Administrator Name and Password to Install** setting, select **Yes**.
5. Optionally change the value of the **Always Show GUI** setting.

When you build OS X–based installers, they are code signed and include authentication support. Monitor the stderr and stdout streams for any code-signing errors that occur at build time. The code-sign command should not exit with a non-zero exit code.

Before you release these installers, test them and verify that they are properly built. To learn how, see Verifying that Your Output Files Are Working as Expected.

Option 2: Building an Installer with Authentication Support on the InstallAnywhere Build Machine and then Code Signing the Output on a Designated Code-Signing Machine

Once you have added your verified, signed helper tool to an InstallAnywhere build machine and prepared your separate designated code-signing machine, you can build and code sign authenticated installers for OS X–based target systems.

To configure your InstallAnywhere project to include authentication support in your OS X–based installers:

1. In the Advanced Designer, on the **Project** page, click **Platforms**. The **Platforms** view opens.
2. In the **Mac OS X** area, under the **Authentication** category, in the **Requires an Administrator Name and Password to Install** setting, select **Yes**.
3. Optionally change the value of the **Always Show GUI** setting.
4. In the **Code Signing** setting, ensure that the **Code Sign the Generated Installer** check box is cleared.

When you build OS X–based installers, they include authentication support and are ready to be code signed on your designated code-signing machine.

To code sign the build output (install.zip file) on a separate code-signing machine:

1. Customize the two information property list files (one for the installer and one for the authentication wrapper) in the build output so that they include the user ID from your Developer ID Application certificate:
   a. Extract the install.zip file to a folder on your code-signing machine.
   b. Grant write-execute permission to the extracted application so that you can edit values in its Info.plist files. The following command demonstrates this:
      `chmod -R 755 <absolute path of install.app>`
   c. Find the installer's Info.plist file in the `install.app\Contents\Resources\install.app\Contents` folder, and open it in an editor.
   d. Under the SMAuthorizedClients key, replace the XXXXXXXXXX string with the user ID from your Developer ID Application certificate. This ID is displayed in the User ID field when you select your certificate in the Keychain Access utility.
   e. Find the authentication wrapper's Info.plist file in the install.app/Contents folder, and open it in an editor.
   f. Under the SMPrivilegedExecutables key, replace the XXXXXXXXXX string with the user ID from your Developer ID Application certificate. This ID is displayed in the User ID field when you select your certificate in the Keychain Access utility.

2. Code sign and check the installer (which is in a subfolder of the authentication wrapper; that is, the `install.app\Contents\Resources\install.app` folder):
   a. Use the following command line to code sign the installer:
      `codesign --force --deep --timestamp=none --sign <CERTIFICATE_ID> <install.app absolute path>/Contents/Resources/install.app`
      where <CERTIFICATE_ID> is the value without the *Developer ID Application:* string that is displayed in the Common Name field when you select your certificate in the Keychain Access utility. That is, if the Common Name field displays *Developer ID Application: ABC Software Inc*, use a certificate ID of *ABC Software Inc* in your command line.
   b. Verify that your signed installer has been properly built and signed. To learn how, see Verifying that Your Output Files Are Working as Expected.
3. Code sign and check the authentication wrapper (that is, the top-level install.app folder):
   a. Use the following command line to code sign the authentication wrapper:
      `codesign --force --deep --timestamp=none --sign <CERTIFICATE_ID> <install.app absolute path>`
      where <CERTIFICATE_ID> is the value without the *Developer ID Application:* string that is displayed in the Common Name field when you select your certificate in the Keychain Access utility. That is, if the Common Name field displays *Developer ID Application: ABC Software Inc*, use a certificate ID of *ABC Software Inc* in your command line.
   b. Verify that your signed authentication wrapper has been properly built and signed. To learn how, see Verifying that Your Output Files Are Working as Expected.
4. Compress the top-level install.app back into an install.zip file.

The resulting install.app file includes authentication support for the installer. Note that the only way to remove a product whose installer was code signed using this method is to drag it to the Trash.

Verifying that Your Output Files Are Working as Expected

Regardless of which method you use to prepare your code-signed installers, at various points in the process, it is important to verify whether the target file has been properly code signed before proceeding with the next step of the process.

It is also crucial to test the final output on a clean machine.

If you encounter issues while testing your output files, you can verify that the trust relationship between the helper tool and the authentication wrapper are intact.

Verifying that the Helper Tool, the Authentication Wrapper, and the Installer Have Been Successfully Signed

Regardless of which method you use to prepare your code-signed installers, at various points in the process, it is important to verify whether the target file has been properly code signed before proceeding with the next step of the process.

To verify that the helper tool, the authenticated wrapper, or the installer was signed properly, use the following commands in Terminal:

Command	Expected Result
`$> codesign -vvv <path_to_signed_file>`	This should specify that all designated requirements are satisfied.
`$> codesign -dvvv <path_to_signed_file>`	This should specify the code-signing information, including details about the certificate that was used.
`$> spctl -a <path_to_signed_file>`	If this command exits without any output, Gatekeeper will accept the file.

Testing Your Installer

Before you release your product, it is important to test the final output on a clean machine—one that does not have your Developer ID Application certificate available in the login or system keychains in the Keychain Access utility. Ensure that you are able to successfully install your product when you are logged on to the test machine as a standard user who is not a root user or an administrative user.

Verifying that the Trust Relationship Between the Helper Tool and the Authentication Wrapper Are Intact

In some cases, it is possible that the trust relationship between the helper tool and the authentication wrapper may be broken. Such cases are difficult to identify and debug. One easy way might be to first print out the code-signing requirements of both the helper tool and the authentication wrapper, and compare them. To do so, use the following command:

```
$> codesign -d -r- <path-to-app>
```

where <path-to-app> is the path to your helper tool or your authentication wrapper

Run this for both the helper and the authentication wrapper. This will produce output on Terminal. Compare the helper tool output with the authentication wrapper output, and look for any mismatches. The identifier portion can differ, but the other portions should match.

The following outputs demonstrate an example of a working relationship.

Helper Tool Output:

```
iadevs-Mac-mini:Build iadev$ codesign -d -r- com.flexera.ia.helper
designated => anchor apple generic and
identifier "com.flexera.ia.helper" and
(certificate leaf[field.1.2.840.113635.100.6.1.9] /* exists */ or
certificate 1[field.1.2.840.113635.100.6.2.6] /* exists */ and
certificate leaf[field.1.2.840.113635.100.6.1.13] /* exists */ and
certificate leaf[subject.OU] = XXXXXXXXXX)
```

Authentication Wrapper Output:

```
iadevs-Mac-mini:NoVM iadev$ codesign -d -r- install.app
designated => identifier "com.flexera.authenticator" and
anchor apple generic and
certificate 1[field.1.2.840.113635.100.6.2.6] /* exists */ and
certificate leaf[field.1.2.840.113635.100.6.1.13] /* exists */
and certificate leaf[subject.OU] = XXXXXXXXXX
```

You can also optionally use the following command to obtain additional debugging information:

```
$> otool -X -s __TEXT __info_plist com.flexera.ia.helper | xxd -r
```

This assumes that the helper tool (com.flexera.ia.helper) is present in the current directory, and that it can be used to verify that the info.plist file that is injected into the helper tool was customized properly.

Troubleshooting Tips

The following table provides tips for various issues that you may encounter when preparing or testing code-signed installers with authentication support.

Issue	Tip
While you are code signing one of your files, you encounter an error about the code-signing identity not being found.	This error indicates that the certificate is missing from the login keychain, or that the spelling of the specified common name is incorrect.
When you are building a code signed installer with authentication support from within InstallAnywhere (option 1), the stderr/stdout messages indicate that your code-sign command exited with a non-zero exit code.	Check to ensure that the certificate is found in the login keychain on your InstallAnywhere build machine. Also verify that you are not logged in as the root user on the machine when you are building and signing; you should be logged in as the same user that was logged in when the certificate was added to the login keychain.
After performing all of the building and code-signing steps, your installer does not launch when you try to run it.	This issue may occur in the following scenarios: • One or more of the code-signing requirements are not met on the machine that you are using for code signing (either your InstallAnywhere build machine or a designated signing machine). To learn more, see Requirements for Code Signing. • The helper tool, the authentication wrapper, or the installer was not properly signed. Ensure that you perform the verification steps after signing each file. • The code signature identities of the helper tool and the authentication wrapper do not match, or the trust relationship between them is not intact. • Your login keychain is locked. Ensure that it is not locked. To learn more, see Verifying that Your Output Files Are Working as Expected. The Console utility that is built into OS X may provide relevant clues. (This is different from Terminal; the Console utility is the OS-level log-reporting application.) When you try to launch the installer but it fails to launch, check the Console for errors. If you still cannot identify the problem, try removing the certificate and corresponding private key from your keychain, re-adding them, and then rebuilding and re-signing as needed.

Issue	Tip
When the code-signing step occurs, a dialog box opens, prompting for administrative credentials.	On some machines, when the certificate is added to the login keychain, only certain applications have access to it. If you try to code sign in this case, the prompt for credentials is displayed. To avoid this prompt, ensure that the certificate has been granted access to be used by all applications: 1. In **Keychain Access**, right-click the certificate and then click **Get Info**. 2. On the **Access Control** tab, click the **Allow all applications to access this item** option.
Your installer successfully installs your product, but your uninstaller does not uninstall it.	Being unable to uninstall the product is a limitation of the option 2 method of code signing, where signing is performed on a separate designated code-signing server. The only way to remove a product whose installer was code signed using this method is to drag it to the Trash.

Legal Information

Copyright Notice

Intellectual Property

Restricted Rights Legend