



NF
25

CONNECTING
THE WORLD



NIAGARA FORUM APAC | 9-10 SEPTEMBER 2025

Module Signing

Bo Wang

APAC Training Supervisor

Tridium

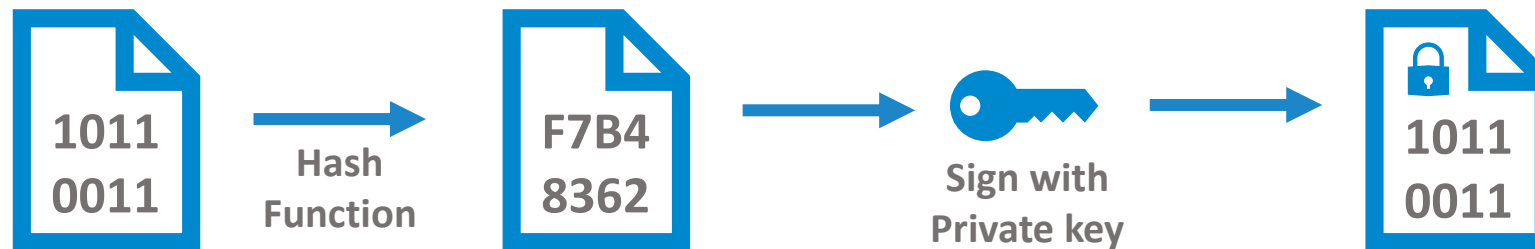


Agenda

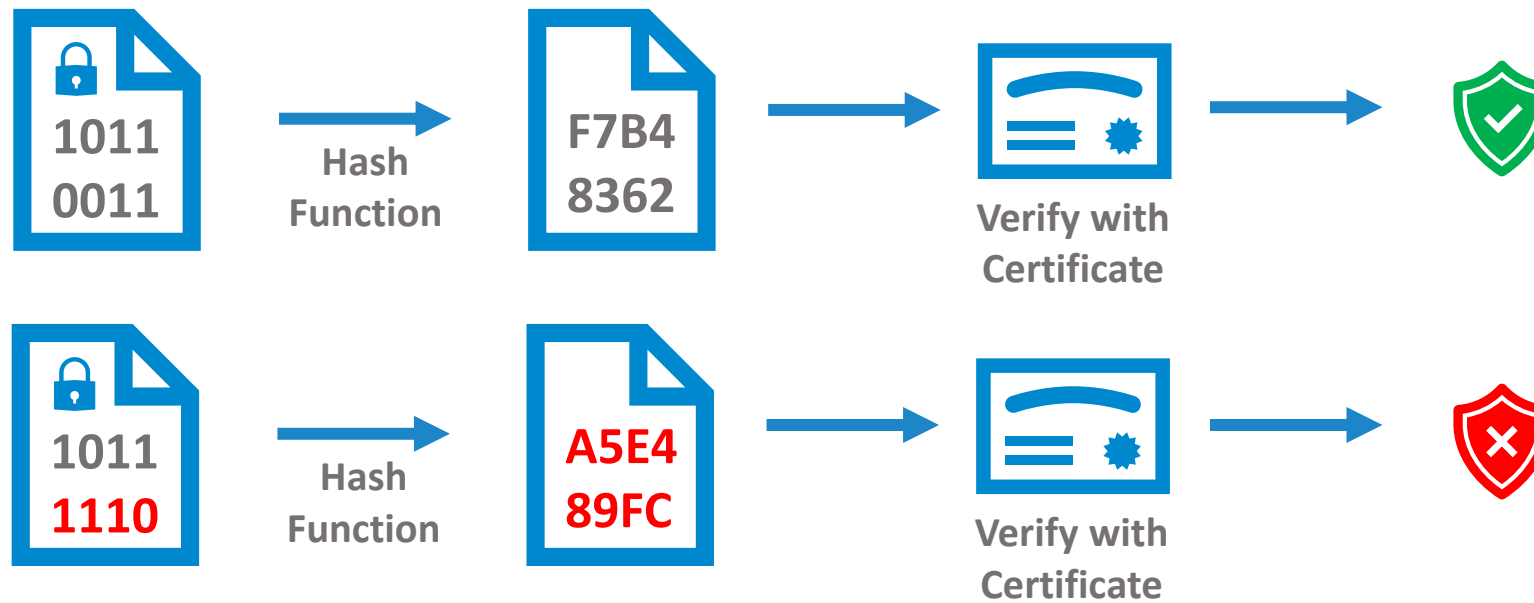
- What is code signing
- Niagara module verification mode
- Niagara Jar signer tool
- Code signing using certificate
- Code signing using HSMs

What is Code Signing

- Code signing is the process of applying a digital signature to a piece of code so that it can be later verified to ensure that it has not been modified after it was signed.



What is Code Signing



Benefits of Code Signing

- Confirm code is from a trusted source
- Prevent installation of malicious code
- Satisfy security requirements



Verification Modes

Low	Medium	High
• Signing optional	• Modules must be signed by a trusted certificate • Self-signed certificates acceptable, but must be installed in user trust store	• Modules must be signed by a CA signed certificate • Internal CA is acceptable, but must be installed in user trust store

niagara.moduleVerificationMode=[**low, medium, high**] in *System.Properties* file

Module Information

Tools Window Help

Options

- AX Certificate Management
- Alarm Portal
- Bacnet EDE
- Certificate Management
- Certificate Signer Multiple Selection Tool
- Certificate Signer Tool
- Driver Upgrade Tool
- Embedded Device Font Tool
- Jar Signer Tool
- Kerberos Configuration Tool
- Lexicon Tool
- Local License Database
- Logger Configuration
- Lon Xml Tool
- Manage Credentials
- Module Info
- NDIO to NRIO Conversion Tool
- New ACE App
- New Driver

Modules

Name	Version
app-rt	Tridium 4.14.0.162
app-wb	Tridium 4.14.0.162
awsUtils-rt	Tridium 4.14.0.162
awsUtils-ux	Tridium 4.14.0.162
awsUtils-wb	Tridium 4.14.0.162
axCommunity-doc	Community 19.11.8.1.0
axCommunity-rt	Community 19.11.8.1.0
axCommunity-ux	Community 2019.9.9.0
axCommunity-wb	Community 17.2.17.0.0
axvelocity-rt	Tridium 4.14.0.162
axvelocity-wb	Tridium 4.14.0.162
azureUtils-rt	Tridium 4.14.0.162
backup-rt	Tridium 4.14.0.162

Module Details

File axCommunity-rt.jar

Size 377.7 KB

Module Name axCommunity-rt

Description Open source AX Community module

Version Community 19.11.8.1.0

Release Date 2019-11-08

Status Signature Warning

Signature Details

Status Unsigned

This module is unsigned. This platform requires modules to be signed with a valid trusted certificate.

Module Details

File bacnet-rt.jar

Size 2,486.7 KB

Module Name bacnet-rt

Description Niagara BACnet Driver

Version Tridium 4.14.0.162

Release Date 2024-05-28

Status OK

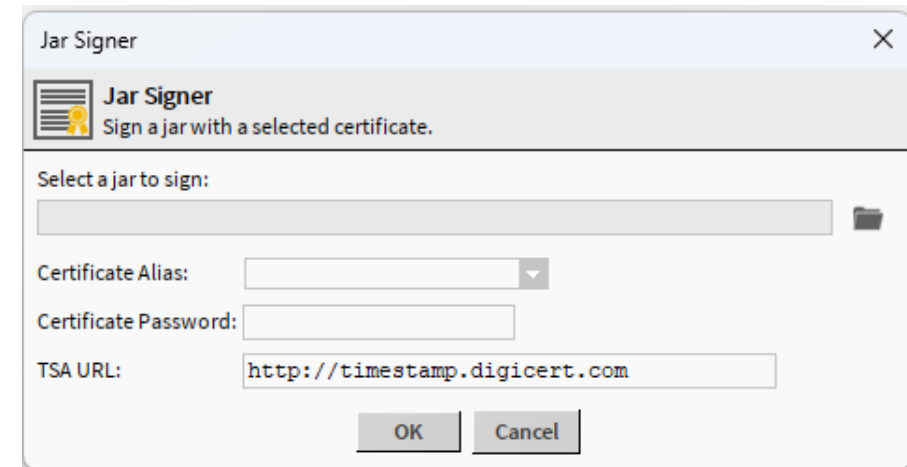
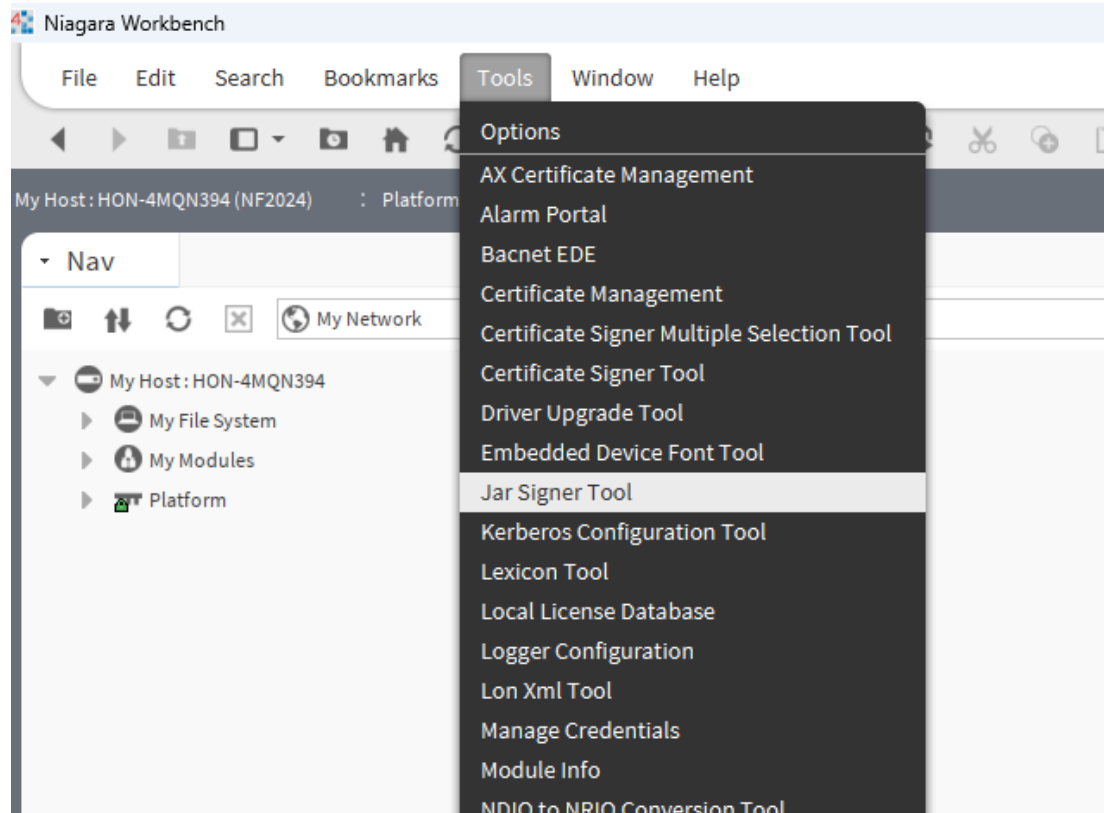
Signature Details

Status Ok

Signers

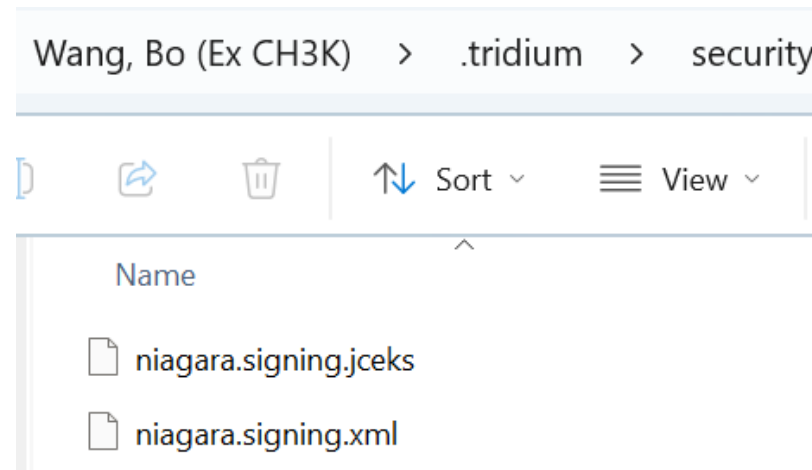
[Niagara4Modules Code Signing](#)

Niagara Jar Signer Tool



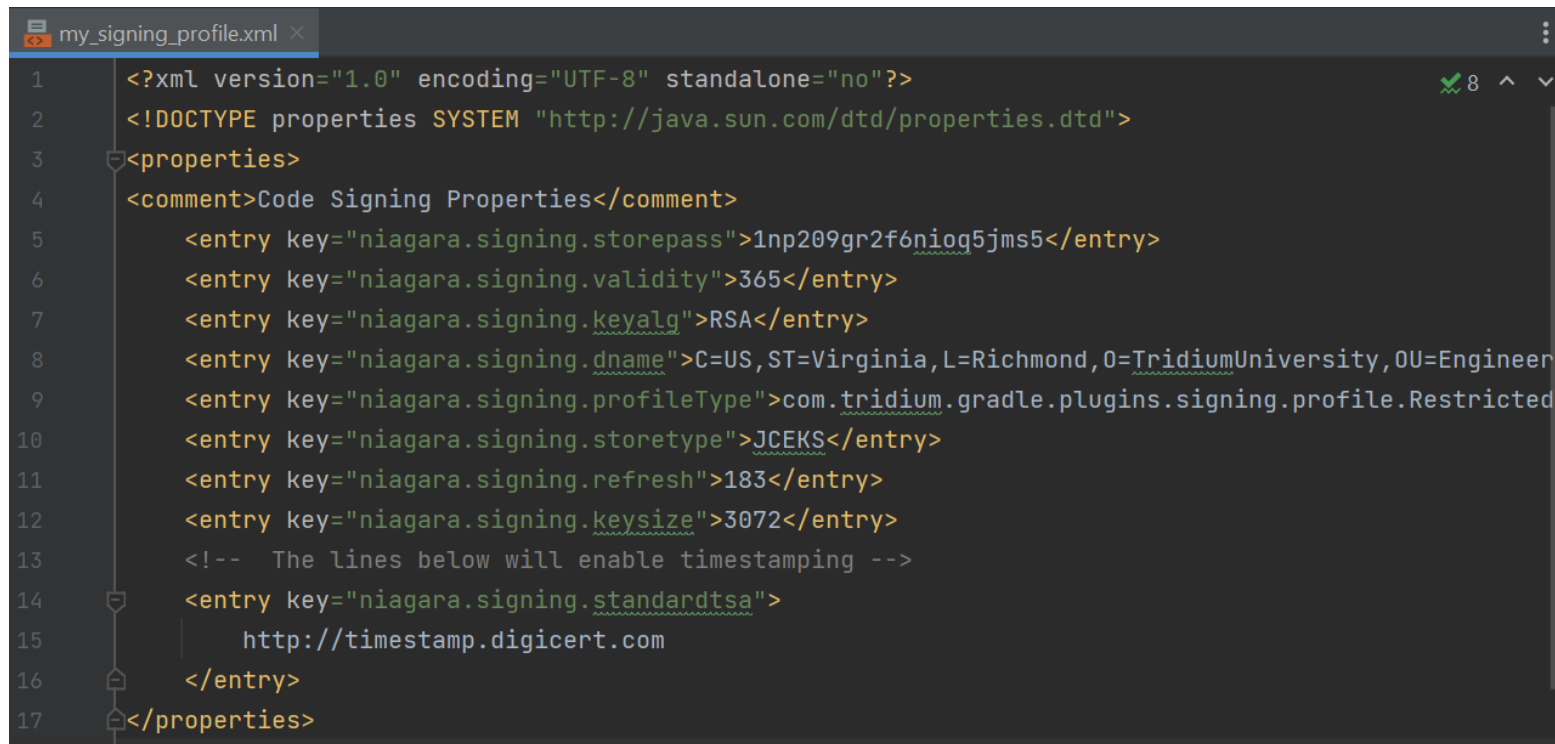
Default Signing Profile

- Starting in 4.6 any module you build will automatically be signed with a generic, auto-generated, self-signed certificate.
- Default Signing Profile is in USER_HOME/.tridium/Security



Creating a Signing Profile

```
gradlew :createProfile --profile-path path\to\my_signing_profile.xml
```



```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3 <properties>
4   <comment>Code Signing Properties</comment>
5   <entry key="niagara.signing.storepass">1np209gr2f6niog5jms5</entry>
6   <entry key="niagara.signing.validity">365</entry>
7   <entry key="niagara.signing.keyalg">RSA</entry>
8   <entry key="niagara.signing.dname">C=US,ST=Virginia,L=Richmond,O=TridiumUniversity,OU=Engineer
9   <entry key="niagara.signing.profileType">com.tridium.gradle.plugins.signing.profile.Restricted
10  <entry key="niagara.signing.storetype">JCEKS</entry>
11  <entry key="niagara.signing.refresh">183</entry>
12  <entry key="niagara.signing.keysize">3072</entry>
13  <!-- The lines below will enable timestamping -->
14  <entry key="niagara.signing.standardtsa">
15    http://timestamp.digicert.com
16  </entry>
17 </properties>
```

Generate Self-Signed Certificate

```
gradlew :generateCertificate --profile-path path\to\my_signing_profile.xml --alias MyCertificate
```



The screenshot shows an IDE window titled 'build.gradle.kts (helloNiagara)'. The code is a Kotlin Gradle build script. It defines two signing services: 'signingServices' and 'niagaraSigning'. The 'signingServices' block disables the default profile and sets 'allowDefaultProfile' to false. The 'niagaraSigning' block sets the alias to 'MyCertificate' and the signing profile file to 'path/to/my_signing_profile.xml'.

```
45  
46  
47 signingServices { this: SigningServices  
48     // Disable the use of the default profile; this will cause build failures instead  
49     // of silently falling back to the default  
50     signingProfileFactory { this: SigningProfileFactorySpec  
51         allowDefaultProfile.set(false)  
52     }  
53 }  
54  
55 niagaraSigning { this: SigningExtension  
56     aliases.set(listOf("MyCertificate"))  
57     signingProfileFile.set(project.layout.projectDirectory.file(path: "path/to/my_signing_profile.xml"))  
58 }
```

Get Certificate Signed by CA

- Create Certificate Signing Request

```
gradlew :exportCertificate --profile-path path\to\my_signing_profile.xml --alias MyCertificate --csr-only --pem-file my-cert.csr
```

- Import Signed Certificate

```
gradlew :importCertificate --profile-path path\to\my_signing_profile.xml --alias MyCertificate --pem-file my-cert.pem
```

Establishing Trust

```
gradlew :exportCertificate --profile-path path\to\my_signing_profile.xml --alias MyCertificate --pem-file my-cert.pem
```

AX Certificate Management

Certificate Management for Niagara Workbench

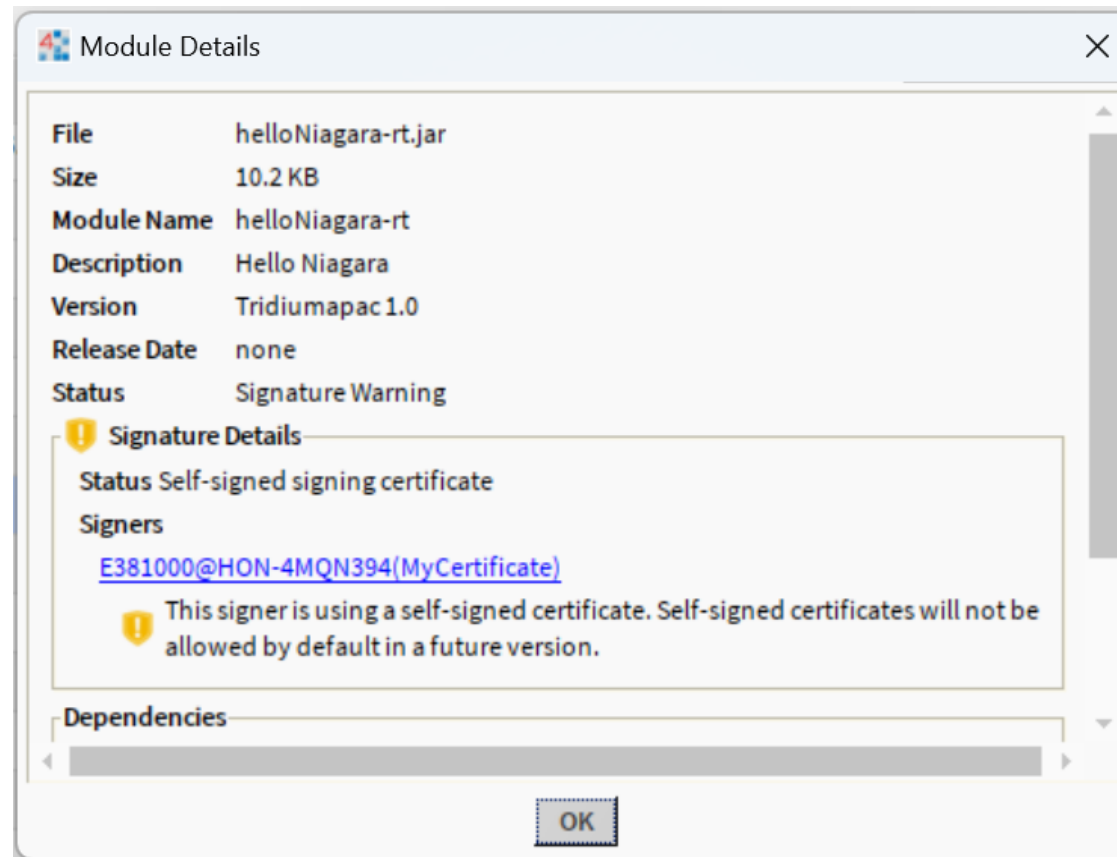
User Key Store System Trust Store User Trust Store Allowed Hosts

You have user certificates that identify these certificate authorities:

User Trust Store

Alias	Subject	Not After	Key Algorithm	Key Size	Valid
✓ my-cert	E381000@HON-4MQN394(MyCertificate)	Tue Jun 30 15:50:14 CST 2026	RSA	3072	true

Module Verification



Hardware Security Module (HSM)

- HSMs are hardened, tamper-resistant hardware devices used to secure cryptographic processes
- HSMs could be a card inside of a PC, a rack mounted device, a USB device or part of a cloud service
- As of mid-2023, all commercial certificate authorities require the use of a HSM to protect the private key of any code signing certificate
- Starting in 4.14 version, the gradle scripts are updated to support code signing compiled modules using a physical HSM device that supports the PKCS11 standard

Create a JarSignerSigningProfile

- A Java properties file
- Configure the command and arguments used to run *jarsigner*

```
# TODO Update with the TSA recommended by your certificate vendor and uncomment
# these lines
# jarsigner.args+=-tsa
# jarsigner.args+=http://timestamp.digicert.com

# jarsigner.args+=-tsadigestalg
# jarsigner.args+=SHA256

jarsigner.args+=-keystore
jarsigner.args+=NONE

jarsigner.args+=-storetype
jarsigner.args+=PKCS11

jarsigner.args+=-storepass
# TODO Update this if your credentials file specifies a different variable
jarsigner.args+=${jarsigner.storepass}

jarsigner.args+=-providerClass
jarsigner.args+=sun.security.pkcs11.SunPKCS11

# TODO Newer jarsigner versions may require the use of 'addProvider' instead of
# 'providerClass'; this may be the case if you get odd errors like
# 'java.security.NoSuchAlgorithmException: SHA256 MessageDigest not available'
# jarsigner.args+=-addProvider
# jarsigner.args+=SunPKCS11
jarsigner.args+=-providerArg

# TODO Update with the absolute path to your HSM .cfg file
```

Signing with JarSignerSigningProfile

- Set the configured Profile file as the default in *build.gradle.kts*

```
// Load the correct signing profile
niagaraSigning {
    aliases.set(listOf(TODO("Set the correct alias as reported by your HSM")))
    signingProfileFile.set(
        project.rootProject.layout.projectDirectory.file("security/niagara.signing.properties")
    ) ^niagaraSigning
}
```

Reference

- Niagara help doc

local:|module://docDeveloper/doc/security/codeSigning.html

- Niagara community article

<https://www.niagara-community.com/s/article/Code-Signing-Tips-and-Troubleshooting>

<https://www.niagara-community.com/s/article/Code-Signing-using-Hardware-Security-Modules>

NIAGARA FORUM
CONNECTING
THE WORLD

niagara
community

[HOME](#)[TOPICS](#) ▾[GROUP](#)[BOOKMARK FEED](#)[ARTICLES](#)[INNOVATION HUB](#)

Code Signing Tips and Troubleshooting

This article is intended to help Niagara developers resolve problems with the module-signing process.

🕒 Sep 13, 2021 Knowledge

TITLE

Code Signing Tips and Troubleshooting

URL NAME

Code-Signing-Tips-and-Troubleshooting

DESCRIPTION

Notice: This Tech Tip is superseded by the following documents that can be referenced online, and in Niagara Workbench:

[Niagara Third Party Module Signing](#)

[Code Signing Options](#) in Getting Started with Niagara

Note that the developer module signing documentation is available within Workbench at <module://docDeveloper/doc/security/codeSigning.html>.

Jarsigner

It is recommended to follow the documented process for integrating signing into the Niagara build process instead of using Java's jarsigner utility. Some developers have reported problems with modules signed by CA issued certificates with jarsigner failing certificate path validation in Niagara. This is caused by jarsigner embedding the certificate chain in the module in the incorrect order. If you encounter this issue switch to using Niagara build process or the Workbench Jar Signer Tool to sign your module.

Generating CSR

Depending on how your CSR is signed, the instructions for generating a CSR in the code signing documentation may cause the signed certificate to be missing the "Extended Key Usage" extension. This may result in the gradle build failing with "Could not sign module". Below is the correct command for generating a CSR with the correct Extended Key Usage.

```
keytool -certreq -alias my-cert -ext EKU="codeSigning" -keypass K3yP@ss -storepass St0reP@ss -keystore my_signing_profile.jks -file my-cert.csr
```

Importing Signed Certificate

When you receive a signed certificate from your CA, it's important to import the entire certificate chain into the keystore. Since CAs provide certificates in various formats, there is no one size fits all solution, but below are some of the most common scenarios and how to handle them. When importing your signed certificate chain, be sure to import with the same alias that you generated your key with.

Summary

- Module signing is ***not optional***, it is a critical cybersecurity practice
- It verifies the ***who*** (publisher) and the ***what*** (integrity) of a software module
- It protects Niagara stations from malware and unauthorized code execution
- ***Always use signed modules*** from trusted sources to ensure system safety and reliability

THANK YOU

