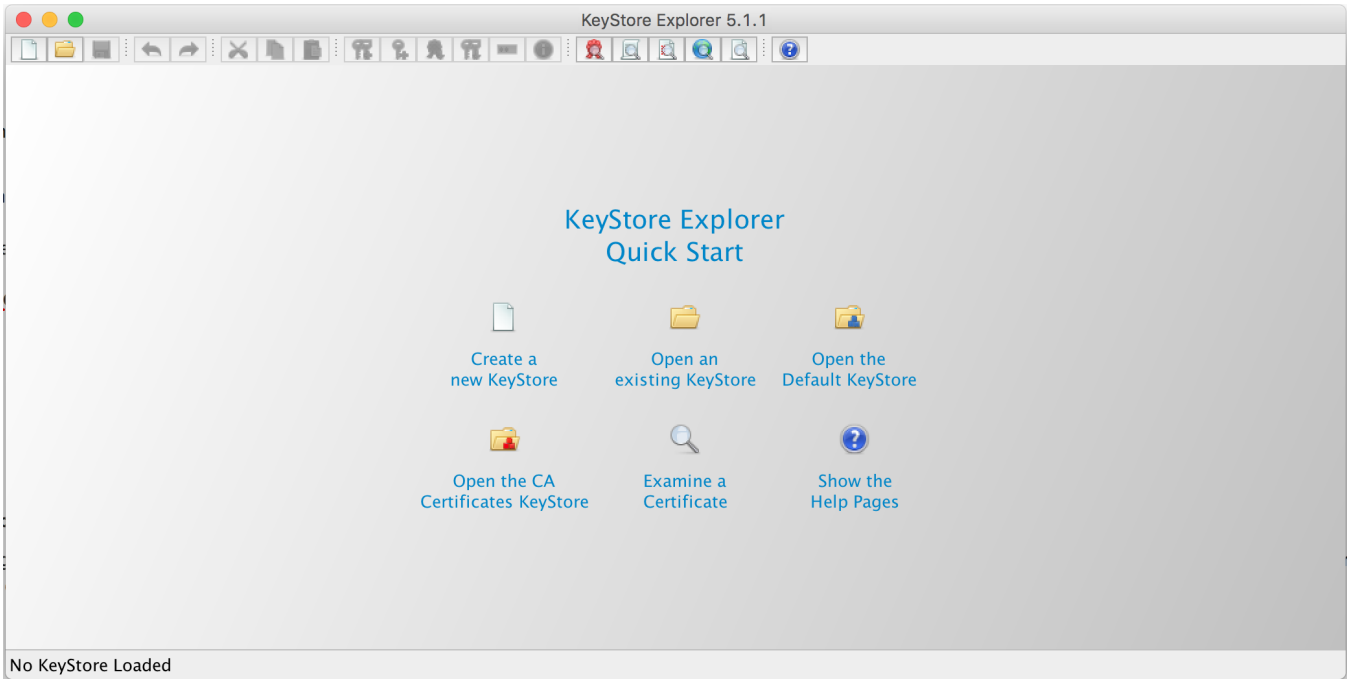
Create a Java Keystore (JKS) File

NOTE: The procedure below is only applicable when running pre-4.0.4 modules. Manually configuring MQTT Distributor to consume a Java Keystore is supported and will work properly when running pre-4.0.4 modules, but it is no longer the recommended process for encrypting MQTT communication. If possible, please upgrade to modules version 4.0.4 or higher and follow the default workflow to secure MQTT communication.

Whether you are using a certificate issued by a trusted CA (Certificate Authority) or a self-signed certificate, internally MQTT Distributor accesses these certificate(s) via the Java KeyStore file that it is configured to use. This KeyStore must contain the public certificate, the private key, and possibly an intermediate certificate if applicable.

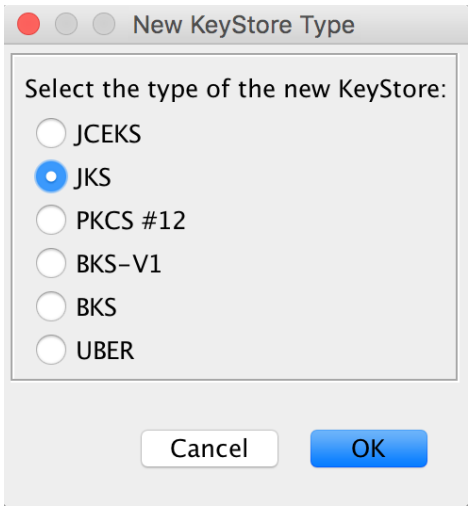
Creating a Keystore using Keystore Explorer

There are many ways to create a Java KeyStore. In this example, we'll show how it can be done using KeyStore Explorer. It can run on Windows, OSX, or any other OS that can run Java. It provides an easy to use graphical interface for creating and manipulating Java KeyStores. Keystore explorer can create a keystore from existing keypair (i.e., certificates) or can generate a private keypair if desired. After installing KeyStore Explorer, open it and you should see something similar to the following. It may ask you to modify some of your Java Security settings before starting. If so, follow the instructions it provides.



Using an Existing Keypair

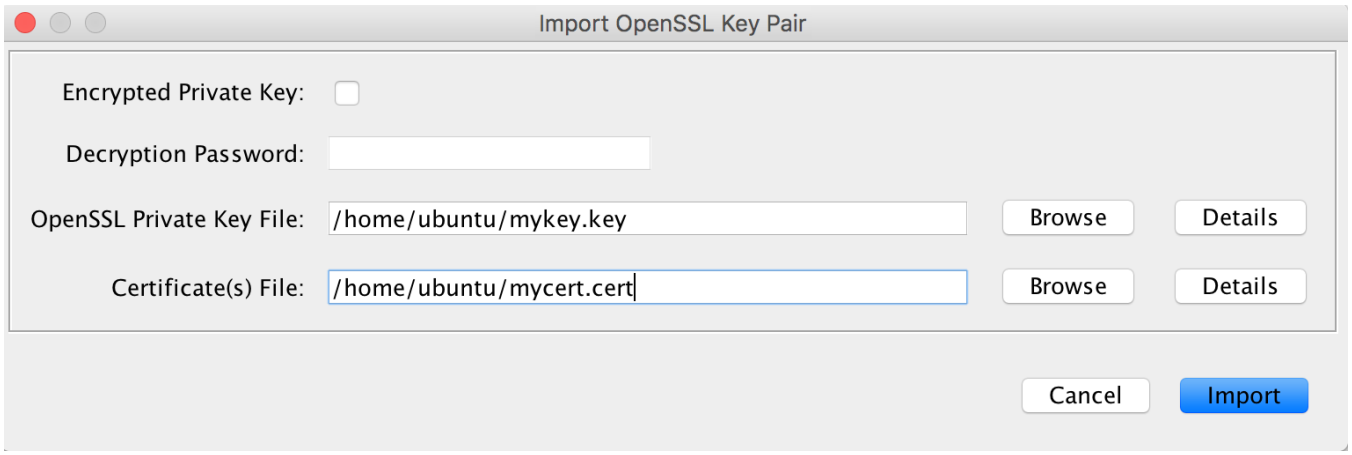
Select 'Create a new KeyStore'. Select a 'JKS' as the type as shown below.



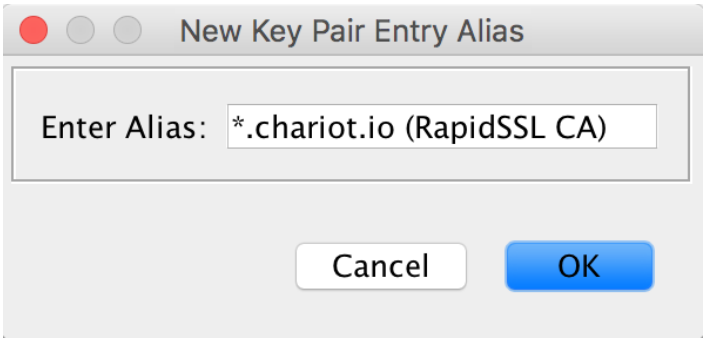
Pull the required components into the KeyStore starting with the public/private KeyPair. This is the public certificate and the private key that we originally generated. Click the 'Import Key Pair' icon from the KeyStore Explorer menu (the icon with two keys and a blue downward arrow).

Select OpenSSL as the type and click OK:

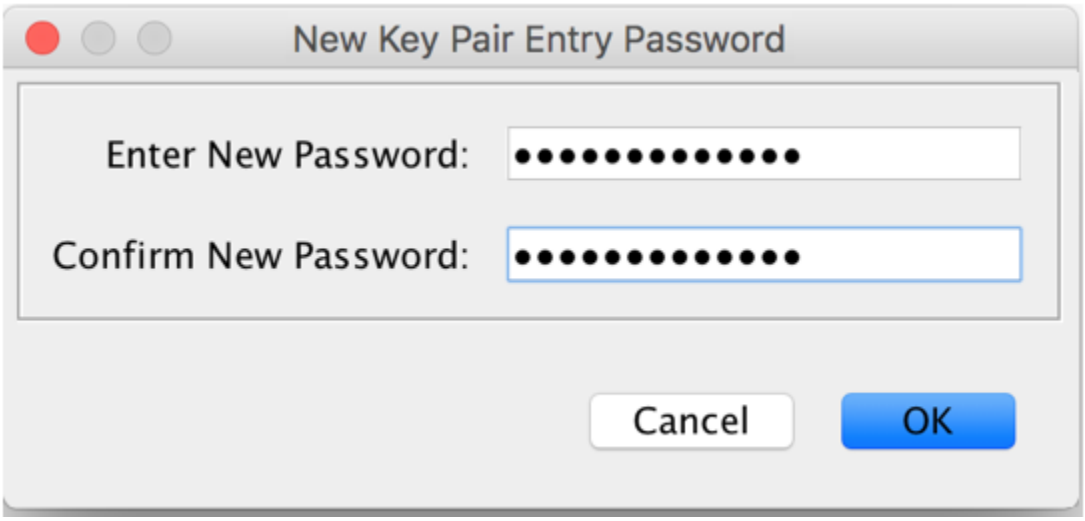
Browse to the key and certificate files as shown below and click import:



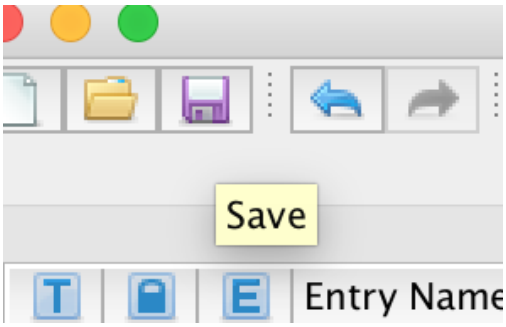
Now you will be asked to specify the alias. You can leave this as the default. It will reflect the Common Name that was specified during the CSR generation and the CA:



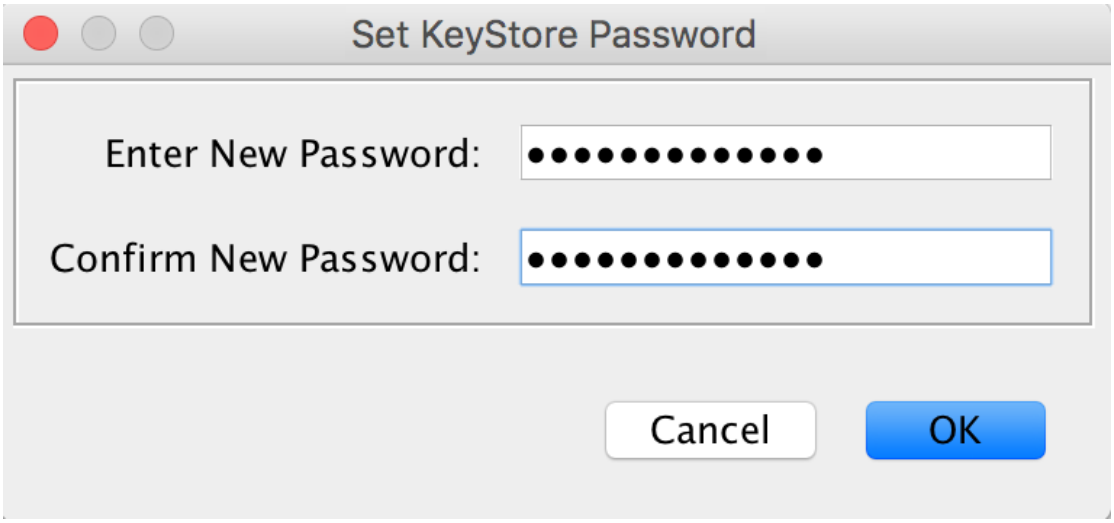
You will now be asked to specify a password for the KeyPair. At this point MQTT Distributor requires that the Key Pair passwords match the overall KeyStore password. So, make sure you note this password because we'll need to use it as the overall KeyStore password as well. **Note: Use of a Key Pair password is a constraint of the JKS file and therefore a requirement in the configuration of TLS**.



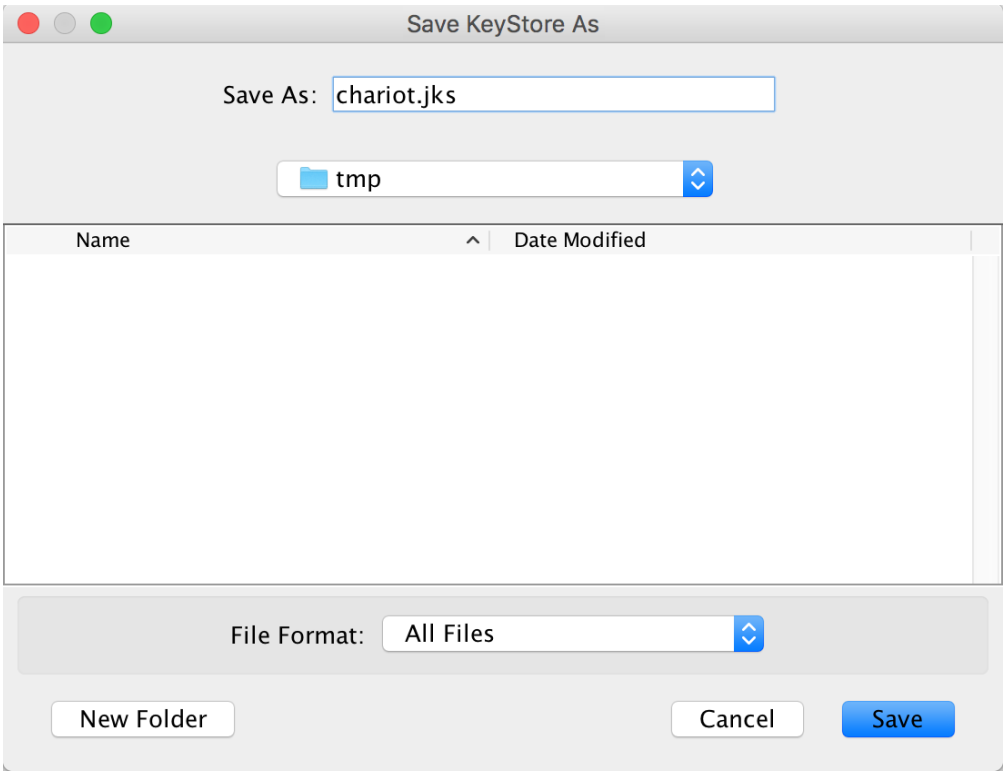
At this point, you can save your KeyStore and specify a KeyStore password. Do so by clicking the save icon in the upper left menu:



You will now be prompted for a password. Provide the same secure password you used for the public/private KeyPair earlier. **Note: Use of a Key Pair /KeyStore password is a constraint of the JKS file and therefore a requirement in the configuration of TLS**.



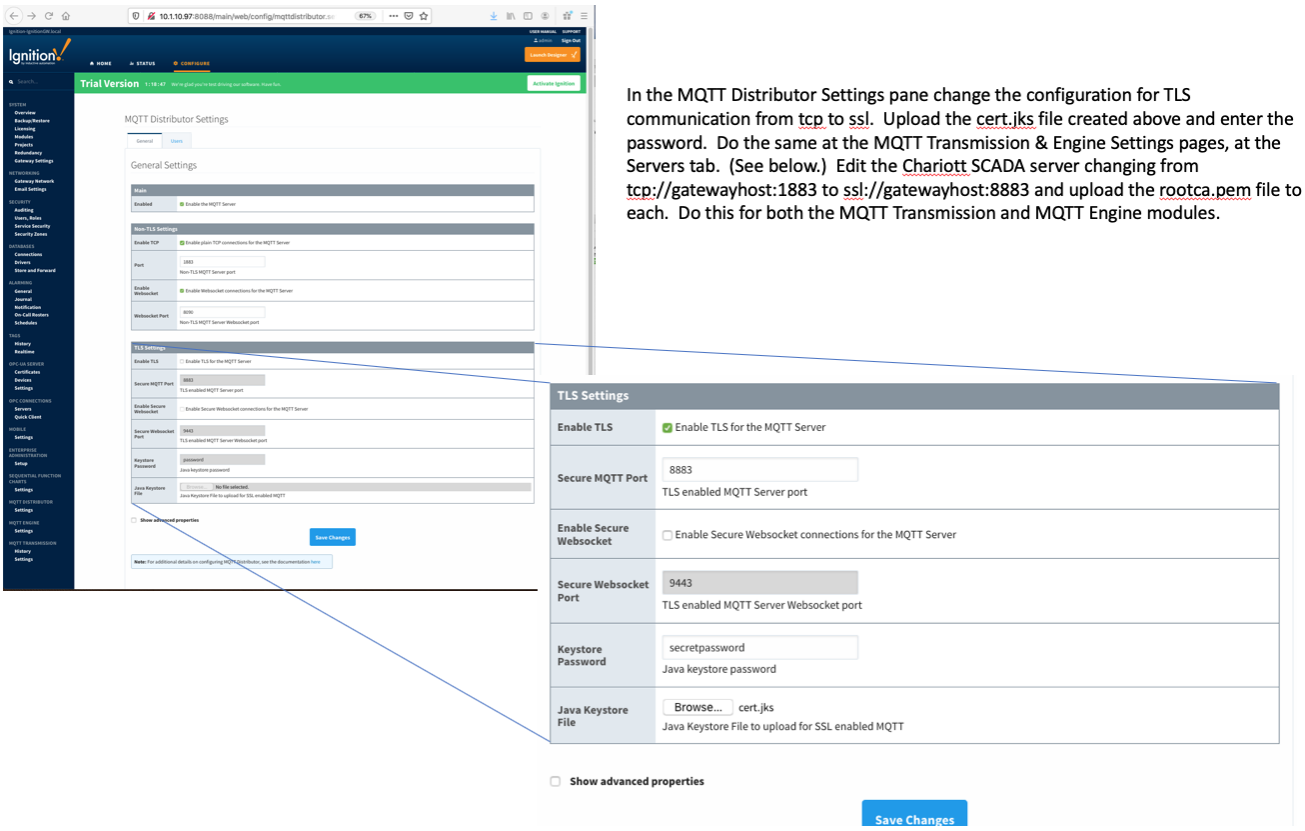
Finally, give it a name and location on the filesystem and click Save:



Configuring MQTT Distributor to use a Keystore

Use your browser and login to your Central Gateway (Distributor). Under Config MQTT Distributor Settings page under the General Tab upload the keystore file. Uncheck the box to Enable the plain TCP connection and check the box under TLS Settings to Enable the TLS port(s). Don't forget to enter the Password in the box just above the Java KeyStore File portion.

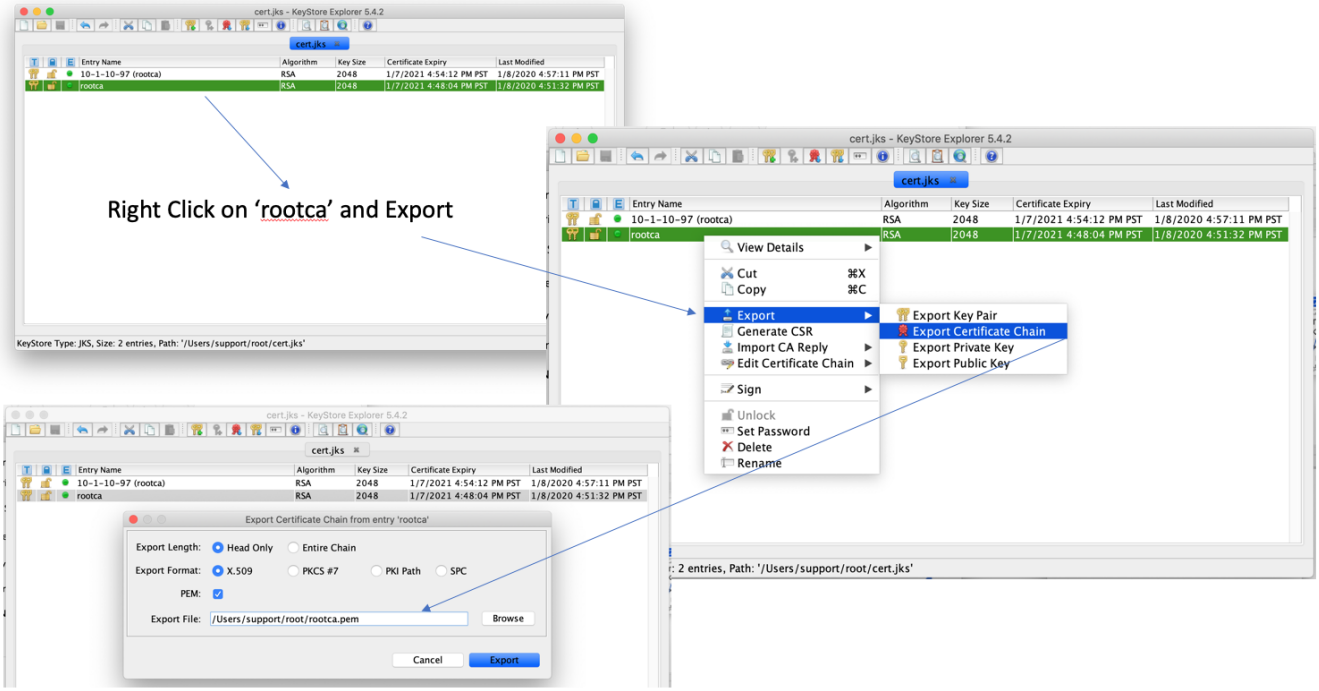
In the MQTT Distributor Settings, change the configuration for TLS communication from TCP to SSL. Upload the keystore file created and enter the password.



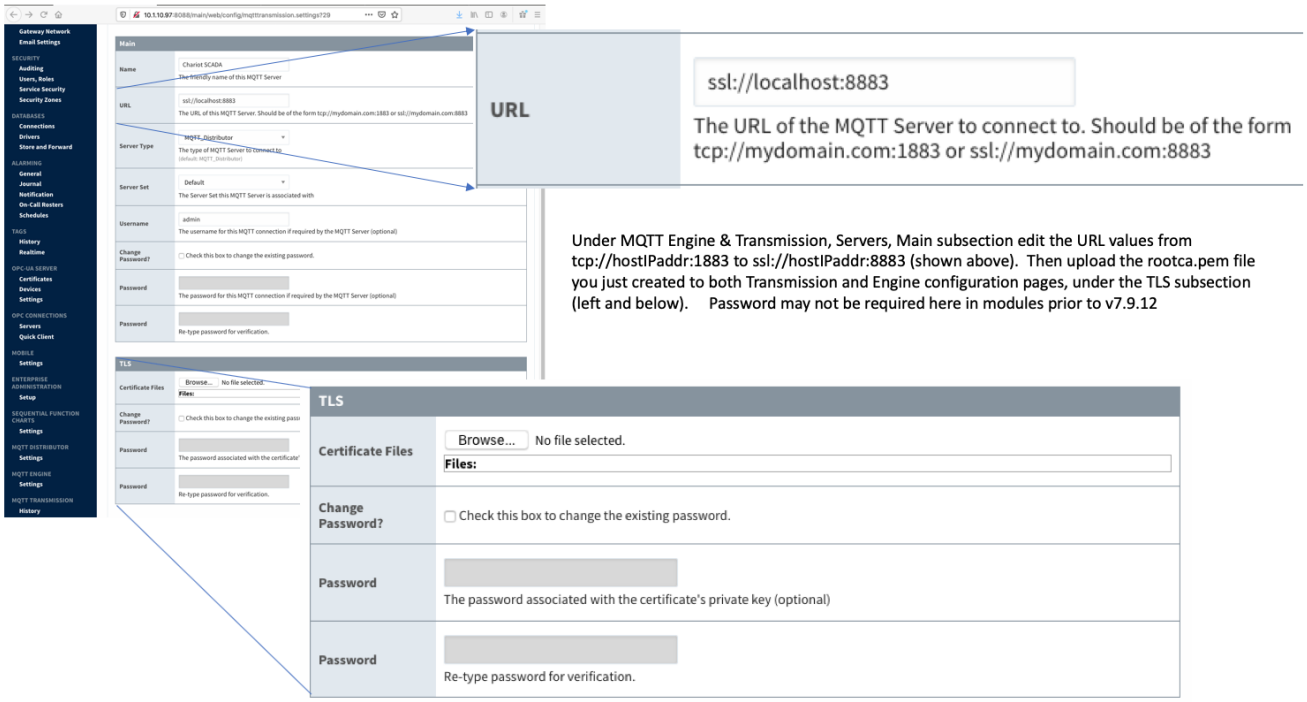
In the MQTT Distributor Settings pane change the configuration for TLS communication from tcp to ssl. Upload the cert.jks file created above and enter the password. Do the same at the MQTT Transmission & Engine Settings pages, at the Servers tab. (See below.) Edit the Chariott SCADA server changing from tcp://gatewayhost:1883 to ssl://gatewayhost:8883 and upload the rootca.pem file to each. Do this for both the MQTT Transmission and MQTT Engine modules.

Export the Certificate Chain for Client-side Use (self-signed certs only)

If using self-signed certificates, the required CA certificates are not known to MQTT clients by default as they would be if the certificate was generated by a real CA. This requires one to acquire and upload the CA certificates that make up the certificate chain (aka. "chain-of-trust"). The certificate chain can be exported from an existing keystore (like the one created here) using the steps below. Return to the KeyStore Explorer application and generate the necessary root.ca.pem file. Save this file in same location (by default) as your cert.jks file. Use this template below to upload this root.ca.pem file to Transmission and Engine. (Password not required on these pages.)



Save this rootca.pem key file. This will be installed on both Engine and Transmission Modules to allow and connect securely via SSL protocol to your Distributor (Ignition Server).



Under MQTT Engine & Transmission, Servers, Main subsection edit the URL values from tcp://hostIPaddr:1883 to ssl://hostIPaddr:8883 (shown above). Then upload the rootca.pem file you just created to both Transmission and Engine configuration pages, under the TLS subsection (left and below). Password may not be required here in modules prior to v7.9.12

At this point, all MQTT clients can now connect over TLS enabled connections. Note the new port of 8883. If using a certificate signed by a publicly trusted CA and the OS with the MQTT client supports that specific CA, the clients don't have to make any modifications to their list of trusted root certificates. If using a self-signed certificate there are a couple options:

- The root CA cert can be added to the Operation System's list of trusted root certificates
    - This means the application doesn't need to handle special cases (i.e. modifications to the Java Truststore)

- The client side application can be modified to load the root CA certificate to validate the server certificate against
    - This doesn't require OS changes

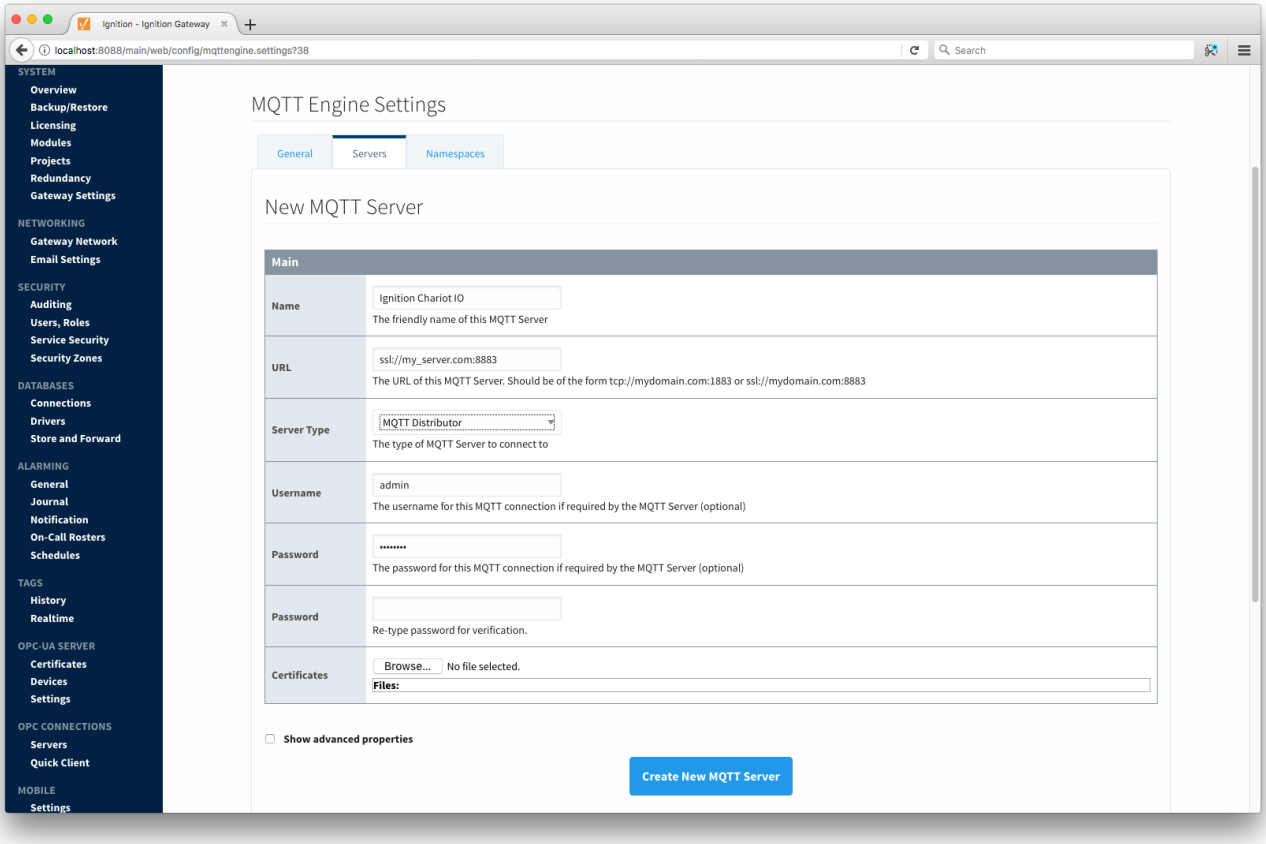
Note if your certificate also requires an intermediate certificate, this must also be added to the MQTT client so the full chain of trust can be established.

Using the Certificate to Secure Communication with MQTT Engine or MQTT Transmission:

In MQTT Engine or Transmission, there may be a need to specify the TLS components for the client configuration. If using certificates signed by a trusted CA, no additional configuration is required beyond changing the form of the URL. The form should be as follows:

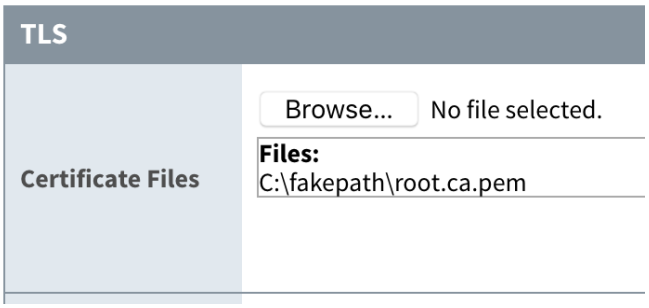
- ssl://[sever_url]:8883

An example is here:



If the trusted CA you purchased your certificate from requires an intermediate certificate or if you created a self signed certificate, you will need to specify the CA certificate chain in the configuration. If you received your certificate from a trusted CA and they require an intermediate certificate, it will be provided by the CA. If you followed the tutorial above for a self-signed certificate and also created an intermediate CA, it will be the file called 'ca-chain.cert.pem'. If you simply created a CA without an intermediate cert, it will be the public CA certificate. Once you've identified the CA certificate chain based on these descriptions, copy it to a file called 'root.ca.pem' on your development system. Note this filename change is important and required.

Upload the file via the configuration as shown here by clicking Save Changes:



Once the settings are saved, the MQTT client associated with MQTT Engine or MQTT Transmission will connect using TLS.

Additional Resources

- Inductive Automation's Ignition download with free trial
  - https://inductiveautomation.com/downloads/
- Azure Injector download with free trial
  - https://inductiveautomation.com/downloads/third-party-modules
- Questions about this tutorial?
  - Check out the Cirrus Link Forum: https://forum.cirrus-link.com/
  - Contact support: support@cirrus-link.com
- Sales questions
  - Email: sales@cirrus-link.com
  - Phone: +1 (844) 924-7787
- About Cirrus Link
  - https://www.cirrus-link.com/about-us/