7.4.1. Creating the Truststore and Keystore

The SSL configuration works through two separate files that define the server and client side of the encryption configuration. Because individual hosts within a Tungsten Replication configuration are both servers (when acting as a master, or when providing status information), and clients (when reading remote THL and managing nodes remotely), both the server and client side of the configuration must be configured.

Configuration for all systems relies on two files, the **`truststore`**, which contains the server certificate information (the certificates it will accept from clients), and the **`keystore`** , which manages the client certificate information (the certificates that will be provided to servers). The truststore and keystore hold SSL certificate information, and are password protected.

The keystore and truststore operate by holding one or more certificates that will be used for encrypting communication. The following certificate options are available:

Create your own server and client certificates

Create your own server certificates, get the server certificate signed by a Certificate Authority (CA), and use a corresponding signed client certificate

Use a server and client certificate already signed by a CA. Care should be taken with these certificates, as they are associated with specific domains and/or hosts, and may cause problems in a dynamic environment.

In a multi-node environment such as Tungsten Replication, all the hosts in the dataservice can use the same keystore and truststore certificates. The **`tpm`** command will distribute these files along with the configuration when a new installation is deployed, or when updating an existing deployment.

7.4.1.1. Creating Your Own Client and Server Certificates

Because the client and server components of the Tungsten Replication configuration are the same, the same certificate can be used and add to both the keystore and truststore files.

The process is as follows:

Create the keystore and generate a certificate

Export the certificate

Import the certificate to the truststore

To start, use the supplied **`keytool`** to create a keystore and populate it with a certificate. The process asks for certain information. The alias is the name to use for the server and can be any identifier. When asked for the first and last name, use **`localhost`**, as this is used as the server identifier for the certificate. The other information should be entered accordingly.

Keystores (and truststores) also have their own passwords that are used to protect the store from updating the certificates. The password must be known as it is required in the configuration so that Tungsten Replication can open the keystore and read the contents.

```
shell> keytool -genkey -alias replserver -keyalg RSA -keystore keystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:  My OU
What is the name of your organization?
  [Unknown]:  Continuent
What is the name of your City or Locality?
  [Unknown]:  Mountain View
What is the name of your State or Province?
  [Unknown]:  CA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=My Name, OU=My OU, O=Continuent, L=Mountain View, ST=CA, C=US correct?
  [no]:  yes

Enter key password for <any>
        (RETURN if same as keystore password):
```

The above process has created the keystore and the 'server' certificate, stored in the file `keystore.jks`.

Alternatively, you can create a new certificate in a keystore non-interactively by specifying the passwords and certificate contents on the command-line:

```
shell> keytool -genkey -alias replserver \
    -keyalg RSA -keystore keystore.jks \
    -dname "cn=localhost, ou=IT, o=Continuent, c=US" \
    -storepass password -keypass password
```

Now you need to export the certificate so that it can be added to the truststore as the trusted certificate:

```
shell> keytool -export -alias replserver -file client.cer -keystore keystore.jks
Enter keystore password:
Certificate stored in file <client.cer>
```

This has created a certificate file in `client.cer` that can now be used to populate your truststore. When added the certificate to the truststore, it must be identified as a trusted certificate to be valid. The password for the truststore must be provided. It can be the same, or different, to the one for the keystore, but must be known so that it can be added to the Tungsten Replication configuration.

```
shell> keytool -import -v -trustcacerts -alias replserver -file client.cer -keystore
truststore.ts
Enter keystore password:
Re-enter new password:
Owner: CN=My Name, OU=My OU, O=Continuent, L=Mountain View, ST=CA, C=US
Issuer: CN=My Name, OU=My OU, O=Continuent, L=Mountain View, ST=CA, C=US
Serial number: 87db1e1
```

```
Valid from: Wed Jul 31 17:15:05 BST 2013 until: Tue Oct 29 16:15:05 GMT 2013
Certificate fingerprints:
         MD5:  8D:8B:F5:66:7E:34:08:5A:05:E7:A5:91:A7:FF:69:7E
         SHA1: 28:3B:E4:14:2C:80:6B:D5:50:9E:18:2A:22:B9:74:C5:C0:CF:C0:19
         SHA256:
1A:8D:83:BF:D3:00:55:58:DC:08:0C:F0:0C:4C:B8:8A:7D:9E:60:5E:C2:3D:6F:16:F1:B4:E8:C2:3C:87:38:26
         Signature algorithm name: SHA256withRSA
         Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: E7 D1 DB 0B 42 AC 61 84   D4 2E 9A F1 80 00 88 44  ....B.a........D
0010: E4 69 C6 C7                                        .i..
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
[Storing truststore.ts]
```

This has created the truststore file, **`truststore.ts`**.

A non-interactive version is available by using the **`-noprompt`** option and supplying the truststore name:

```
shell> keytool -import -trustcacerts -alias replserver -file client.cer \
    -keystore truststore.ts -storepass password -noprompt
```

The two files, the keystore (**`keystore.jks`**), and truststore (**`truststore.ts`**), along with their corresponding passwords can be now be used with **`tpm`** to configure the cluster. See **Section 7.4.3, "Configuring the Secure Service through `tpm`"**.

Show Copy-friendly Text

7.4.1.2. Creating a Custom Certificate and Getting it Signed

You can create your own certificate and get it signed by an authority such as VeriSign or Thawte. To do this, the certificate must be created first, then you create a certificate signing request, send this to your signing authority, and then import the signed certificate and the certificate authority certificate into your keystore and truststore.

Create the certificate:

```
shell> keytool -genkey -alias replserver -keyalg RSA -keystore keystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:  My OU
What is the name of your organization?
```

```
  [Unknown]:  Continuent
What is the name of your City or Locality?
  [Unknown]:  Mountain View
What is the name of your State or Province?
  [Unknown]:  CA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=My Name, OU=My OU, O=Continuent, L=Mountain View, ST=CA, C=US correct?
  [no]:  yes

Enter key password for <any>
        (RETURN if same as keystore password):
```

Create a new signing request the certificate:

```
shell> keytool -certreq -alias replserver -file certrequest.pem \
    -keypass  password -keystore keystore.jks -storepass password
```

This creates a certificate request, **`certrequest.pem`**. This must be sent the to the signing authority to be signed.

Official Signing

Send the certificate file to your signing authority. They will send a signed certificate back, and also include a root CA and/or intermediary CA certificate. Both these and the signed certificate must be included in the keystore and truststore files.

First, import the returned signed certificate:

```
shell> keytool -import -alias replserver -file signedcert.pem -keypass password \
    -keystore keystore.jks -storepass password
```

Now install the root CA certificate:

```
shell> keytool -import -alias careplserver -file cacert.pem -keypass password \
    -keystore keystore.jks -storepass password
```

> **Note**
>
> If the import of your certificate with **`keytool`** fails, it may be due to an incompatibility with some versions of OpenSSL, which fail to create suitable certificates for third-party tools. In this case, see **Section 7.4.1.4, "Converting SSL Certificates for `keytool`"** for more information.

And an intermediary certificate if you were sent one:

```
shell> keytool -import -alias interreplserver -file intercert.pem -keypass password \
    -keystore keystore.jks -storepass password
```

Now export the signed certificate so that it can be added to the truststore. Although you can import the certificate supplied, by exporting the certificate in your keystore for inclusion into your truststore you can ensure that the two certificates will match:

```
shell> keytool -export -alias replserver -file client.cer -keystore keystore.jks
Enter keystore password:
Certificate stored in file <client.cer>
```

The exported certificate and CA root and/or intermediary certificates must now be imported to the truststore:

```
shell> keytool -import -trustcacerts -alias replserver -file client.cer \
    -keystore truststore.ts -storepass password -noprompt
shell> keytool -import -trustcacerts -alias careplserver -file cacert.pem \
    -keystore truststore.ts -storepass password -noprompt
shell> keytool -import -trustcacerts -alias interreplserver -file intercert.pem \
    -keystore truststore.ts -storepass password -noprompt
```

Self-Signing

If you have setup your own certificate authority, you can self-sign the request using **`openssl`**:

```
shell> openssl ca -in certrequest.pem -out certificate.pem
```

Convert the certificate to a plain PEM certificate:

```
shell> openssl x509 -in certificate.pem -out certificate.pem -outform PEM
```

Finally, for a self-signed certificate, you must combine the signed certificate with the CA certificate:

```
shell> cat certificate.pem cacert.pem > certfull.pem
```

This certificate can be imported into your keystore and truststore.

To import your signed certificate into your keystore:

```
shell> keytool -import -alias replserver -file certfull.pem -keypass password \
    -keystore keystore.jks -storepass password
```

Then export the certificate for use in your truststore:

```
shell> keytool -export -alias replserver -file client.cer -keystore keystore.jks
Enter keystore password:
Certificate stored in file <client.cer>
```

The same certificate must also be exported and added to the truststore:

```
shell> keytool -import -trustcacerts -alias replserver -file client.cer \
    -keystore truststore.ts -storepass password -noprompt
```

This completes the setup of your truststore and keystore. The files created can be used in your **`tpm`** configuration. See **Section 7.4.3, “Configuring the Secure Service through `tpm`”**.

Show Copy-friendly Text

7.4.1.3. Using an existing Certificate

If you have an existing certificate (for example with your MySQL, HTTP server or other configuration) that you want to use, you can import that certificate into your truststore and keystore. When using this method, you must import the signed certificate, and the certificate for the signing authority.

When importing the certificate into your keystore and truststore, the certificate supplied by the certificate authority can be used directly, but must be imported alongside the certificate authorities root and/or intermediary certificates. All the certificates must be imported for the SSL configuration to work.

The certificate should be in the PEM format if it is not already. You can convert to the PEM format by using the **`openssl`** tool:

```
shell> openssl x509 -in signedcert.crt -out certificate.pem -outform PEM
```

First, import the returned signed certificate:

```
shell> keytool -import -file certificate.pem -keypass password \
    -keystore keystore.jks -storepass password
```

> **Note**
>
> If the import of your certificate with **`keytool`** fails, it may be due to an incompatibility with some versions of OpenSSL, which fail to create suitable certificates for third-party tools. In this case, see **Section 7.4.1.4, "Converting SSL Certificates for `keytool`"** for more information.

Now install the root CA certificate:

```
shell> keytool -import -file cacert.pem -keypass password \
    -keystore keystore.jks -storepass password
```

And an intermediary certificate if you were sent one:

```
shell> keytool -import -file intercert.pem -keypass password \
    -keystore keystore.jks -storepass password
```

Now export the signed certificate so that it can be added to the truststore:

```
shell> keytool -export -alias replserver -file client.cer -keystore keystore.jks
Enter keystore password:
Certificate stored in file <client.cer>
```

The exported certificate and CA root and/or intermediary certificates must now be imported to the truststore:

```
shell> keytool -import -trustcacerts -alias replserver -file client.cer \
    -keystore truststore.ts -storepass password -noprompt
shell> keytool -import -trustcacerts -alias replserver -file cacert.pem \
    -keystore truststore.ts -storepass password -noprompt
shell> keytool -import -trustcacerts -alias replserver -file intercert.pem \
    -keystore truststore.ts -storepass password -noprompt
```

Show Copy-friendly Text

7.4.1.4. Converting SSL Certificates for `keytool`

Some versions of the **`openssl`** toolkit generate certificates which are incompatible with the certificate mechanisms of third-party tools, even though the certificates themselves work fine with OpenSSL tools and libraries. This is due to a bug which affected certain releases of **`openssl`** 1.0.0 and later and the X.509 certificates that are created.

This problem only affects self-generated and/or self-signed certificates generated using the **`openssl`** command. Officially signed certificates from Thawte, VeriSign, or others should be compatible with **`keytool`** without conversion.

To get round this issue, the keys can be converted to a different format, and then imported into a keystore and truststore for use with Tungsten Replication.

To convert a certificate, use **`openssl`** to convert the X.509 into PKCS12 format. You will be prompted to enter a password for the generated file which is required in the next step:

```
shell> openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem >client.p12
Enter Export Password:
Verifying - Enter Export Password:
```

To import the converted certificate into a keystore, specifying the destination keystore name, as well as the source PKCS12 password used in the previous step:

```
shell> keytool -importkeystore -srckeystore client.p12 -destkeystore keystore.jks -srcstoretype
pkcs12
Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias 1 successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled
```

The same process can be used to import server certificates into truststore, by converting the server certificate and private key:

```
shell> openssl pkcs12 -export -in server-cert.pem -inkey server-key.pem >server.p12
Enter Export Password:
Verifying - Enter Export Password:
```

Then importing that into a truststore

```
shell> keytool -importkeystore -srckeystore server.p12 -destkeystore truststore.ts -
srcstoretype pkcs12
Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias 1 successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled
```

For official CA certificates, the generated certificate information should be valid for importing using **`keytool`**, and this file should not need conversion.