Using a Certificate Authority for SHRINE

In order to reduce administrative overhead in larger hub-and-spoke SHRINE topologies, SHRINE will support using certificates signed by a central Certificate Authority (CA) starting with **SHRINE 1.18**. Using this approach, a SHRINE network will establish its own private CA (typically administered by the hub administrator), and sites generate certificate signing requests that will be signed by this CA. A SHRINE node then adds this new CA-signed certificate and the CA's own certificate to their keystore, and then adds a few values to shrine.conf. Now, instead of having to import the certificate of every other site on the network along with the hub's certificate, a site only needs to import the hub's CA certificate. If the site receives a query from another node that has been blessed by the same CA, SHRINE will trust that query. This moves the trust model from each individual site to the hub (and thus, the network as a whole), making it easier to add additional nodes to the network later on.

We **highly recommended** that SHRINE networks going forward adopt this approach whenever possible, especially for networks larger than 4 nodes. When using a CA in a hub-based topology, a node only needs to create firewall exceptions for one other host (the hub) and only needs to perform one certificate exchange, solving the N^2 problem that has traditionally afflicted larger SHRINE networks.

For Hub Administrators

The server that will function as the Certificate Authority will require a distribution of OpenSSL with the CA.sh/CA.pl/CA helper script. While it is possible to operate as a CA without the helper script, it is strongly recommended to use the helper script, as it makes life much easier. The exact location of this helper script varies from distribution to distribution, but three places to start are **/usr/local/ssl/misc, /etc/pki/tls/misc,** and **/System/Library/OpenSSL/misc**. The latter of these two is what CentOS uses, and the rest of the guide will be written assuming a CentOS environment.

Establish a Certificate Authority

Run the following command as root:

```
./CA -newca
```

The script will ask for a password, which you will be prompted for every time you wish to sign a certificate. Keep this password safe, and make sure it is different from any other password used for keystores and certificates! The script will also ask for the standard certificate information. When prompted for a challenge password and optional company name, leave these blank. Sample output from the script is provided below:

```
Enter PEM pass phrase: [pass]
Verifying - Enter PEM pass phrase: [pass]

Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:Example State
Locality Name (eg, city) [Default City]:Example City
Organization Name (eg, company) [Default Company Ltd]:University of Example
Organizational Unit Name (eg, section) []:SHRINE
Common Name (eg, your name or your server's hostname) []:shrine-hub.u-of-example.edu
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /etc/pki/tls/openssl.cnf
```

This will create files in **/etc/pki/CA** and **/etc/pki/CA/private**. In /etc/pki/CA will be **cacert.pem** and **careq.pem**. In /etc/pki/CA/private will be **cakey.pem**, the private key used by the CA. Keep this absolutely secure!

cacert.pem is the CA's certificate and should be copied and distributed to each site that requests a certificate. When a node trusts the CA certificate, it will trust all certificates signed by that CA.

Generate a Certificate Signing Request (CSR)

*This step is not necessary in most setups. In most cases, a CSR should be generated by the site remotely, but if the owner of a hub and the owner of a site overlap, it is possible to also generate the CSR via the CA helper script. This process **entirely optional** and involves additional conversion steps, so it is usually simpler to generate the CSR using the steps detailed in the "For Individual Sites" section anyway.*

Run the following command as root:

```
./CA -newreq
```

The script will ask for a password, which you will be prompted for whenever accessing the private key. Keep this password safe and secure. Additionally, in SHRINE setups that do not differentiate between HTTPS and SHRINE signing certificates, the password for this private key **must** be the same as the password used for the keystore ($KEYSTORE_PASSWORD).

The script will ask for standard certificate information. Of particular note is the **Common Name** (CN) value. Make sure that this matches the publicly-accessible hostname of the SHRINE node this certificate will be used on! When prompted for a challenge password and optional company name, leave these blank. Sample output from the script is provided below:

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:Example State
Locality Name (eg, city) [Default City]:Example City
Organization Name (eg, company) [Default Company Ltd]:University of Example
Organizational Unit Name (eg, section) []:SHRINE
Common Name (eg, your name or your server's hostname) []:shrine-client.u-of-example.edu
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

After the script completes, it will leave two files in the current working directory (which should be alongside the CA script): **newkey.pem** and **newreq.pem**. newkey.pem is the private key for this keypair, and should be kept secure. newreq.pem is the certificate that will need to be signed by the CA.

Create Keystore from Generated CSR Keypair

Sign the CSR certificate using ./CA -sign as detailed in the next section.

Bundle the private key and certificate into a single PKCS #12 format file with the following command:

```
openssl pkcs12 -export -in shrine-client.pem -inkey shrine-client.key -out shrine-client.p12 -name "shrine-client"
```

The value supplied for -name will be the alias of the key inside the keystore. The command will prompt for the password of the private key, and then prompt for an "export password". This export password must match the value of $KEYSTORE_PASSWORD.

After generating this .p12 file, generate a new keystore based on it with the following command:

```
keytool -importkeystore -srckeystore shrine-client.p12 -srcstoretype pkcs12 -destkeystore shrine.keystore -
deststoretype JKS
```

Import the CA's signing certificate with the following command:

```
keytool -import -v -alias shrine-hub-ca.pem -file shrine-hub-ca.pem -keystore shrine.keystore
```

Move the generated keystore to /opt/shrine and proceed with the shrine.conf changes detailed in the "For Individual Sites" section of this guide.

```
mv shrine.keystore /opt/shrine/shrine.keystore
```

Sign a Certificate Signing Request (CSR)

Ensure that the CSR to be signed is in the same directory as the CA script and is renamed to **newreq.pem**. The private key is not necessary and should **NOT** be included by the requesting client. Run the following command as root:

```
./CA -sign
```

The script will ask for the password of the CA. It will then verify the CSR, display information about it, and ask if this certificate should be signed. It will then ask if this certificate should be committed, which will store it in the CA's trusted cert database. Relevant snippets of script output are included below:

```
Enter pass phrase for /etc/pki/CA/private/cakey.pem:
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Signed certificate is in newcert.pem
```

As the script says, it will generate a file called **newcert.pem**. This is the final signed certificate that should be given back to the requestor. Alongside this signed certificate, the hub administrator should also send their **cacert.pem** (hopefully copied and renamed with a more descriptive name) file so the requestor can import the CA certificate into their keystore.

Signing Certificates with SubjectAlternativeName

If a certificate has X.509 extensions (typically SubjectAlternativeName), the CA script should not be used since it will ignore those extensions when signing the certificates. Instead, we must use the openssl tool manually and provide it a special .cnf file to match the certificate.

Make a copy of openssl.cnf (typically found in /etc/pki/tls on RedHat-based distributions) and place it alongside the CSR:

```
cp /etc/pki/tls/openssl.cnf /etc/pki/tls/misc/shrine-client.cnf
```

In the [req] block of the newly-created .cnf file, make sure the following line exists or is uncommented:

```
req_extensions = v3_req
```

In the [v3_req] block of the newly-created .cnf file, make sure the following line exists:

```
subjectAltName = @alt_names
```

Below the [v3_req] block, add the appropriate SubjectAlternativeName information like this:

```
[ alt_names ]
 IP.1 = CERTS.PUBLIC.IP.ADDRESS
```

Then sign the cert using the following command:

```
openssl ca -policy policy_anything -out shrine-client-signed.pem -config shrine-client.cnf -extensions v3_req -
infiles shrine-client.csr
```

Rename Files

This is mostly a warning, but when generating CSRs or signing certificates using the CA script, it will write to **newkey.pem**, **newreq.pem**, and **newcert.pem**. These files will be **overwritten** if generating or signing multiple certificates! Make sure to move or rename these files before moving on to the next certificate request! For example:

```
mv newcert.pem shrine-client-signed.pem
```

For All Other Nodes

The node administrator will require the Java keytool, which should already be included on a SHRINE-capable machine.

Generate Keystore

This step is likely unnecessary after a fresh install, since an empty keystore will already be in place. It would be wise, however, to start with a fresh keystore if a currently existing keystore is filled with old certs from other sites.

Ensure that all environment variables in shrine.rc are set properly, as these are the values that will be used to generate the certificate information. Most importantly, ensure that $KEYSTORE_ALIAS matches the publicly-accessible hostname of the machine that will be using this keystore. To generate a new keystore, run the following command:

```
keytool -genkeypair -keysize 2048 -alias $KEYSTORE_ALIAS -dname "CN=$KEYSTORE_ALIAS, OU=$KEYSTORE_HUMAN, O=SHRINE
Network, L=$KEYSTORE_CITY, S=$KEYSTORE_STATE, C=$KEYSTORE_COUNTRY" -keyalg RSA -keypass $KEYSTORE_PASSWORD -
storepass $KEYSTORE_PASSWORD -keystore $KEYSTORE_FILE -validity 7300
```

Or, for a more convenient method, copy the ssl_keytool.sh script from the SHRINE installer and run the following command instead:

```
./ssl_keytool.sh -generate
```

If using ssl_keytool.sh, discard the $KEYSTORE_ALIAS.cer file it generates, as it will not be needed for this configuration.

A new keystore should be created at **/opt/shrine/shrine.keystore**.

Generate a Certificate Signing Request (CSR)

Run the following command:

```
keytool -certreq -alias $KEYSTORE_ALIAS -keyalg RSA -file shrine-client.csr -keypass $KEYSTORE_PASSWORD -
storepass $KEYSTORE_PASSWORD -keystore $KEYSTORE_FILE
```

This should output a file called **shrine-client.csr** (feel free to pick another more descriptive name instead), which should then be sent off to the hub administrator. The hub administrator will review the CSR and respond with either approval or rejection. The most common reason for rejection of a CSR is an invalid CN value. The CN of a certificate should match the publicly-accessible hostname of the machine that will use the certificate. Using other values can cause problems with verifying the identity of that host. If the CSR is approved, follow through with the rest of this guide.

Import CA Certificate and Signed Certificate

After the hub administrator (or CA administrator, if the two entities are separate) approves and signs the certificate, they will send back a signed version of your cert along with the hub's CA cert, as well as the cert used for normal HTTPS communication with the hub. Import all of these into your keystore with the following commands in order:

```
keytool -import -v -alias shrine-hub-ca -file shrine-hub-ca.crt -keystore $KEYSTORE_FILE -storepass
$KEYSTORE_PASSWORD
keytool -import -v -alias shrine-hub-https -file shrine-hub-https.crt -keystore $KEYSTORE_FILE -storepass
$KEYSTORE_PASSWORD
keytool -import -v -alias $KEYSTORE_ALIAS -file shrine-client-signed.crt -keystore $KEYSTORE_FILE -storepass
$KEYSTORE_PASSWORD -keypass $KEYSTORE_PASSWORD -trustcacerts
```

Make absolutely sure the -alias value in the **third and final** command **exactly** matches the alias used for your original certificate! (the one marked as **PrivateKeyEntry** in the keystore) Otherwise, queries will fail with signature verification errors, since the CA's signature will not be on the exact same keystore entry that SHRINE uses. Upon importing your signed certificate, the following message should appear:

```
Certificate reply was installed in keystore
```

If it does not, verify that $KEYSTORE_ALIAS is set appropriately and that the keytool command is set to import to the correct alias. Also make sure that the CA's signing certificate was imported first. This must be present before the last command will work.

After a successful round of imports, verify the contents of the keystore with keytool -list -v -keystore $KEYSTORE_FILE. There should be at least 3 entries in the keystore:

1. Your own PrivateKeyEntry, with an additional certificate chained to it. The "Issuer:" line should reflect information from the hub's CA, not your own. This is the signed certificate from the Hub (signed from the CSR).
2. The hub's HTTPS certificate.
3. The hub's CA signature certificate. The "Owner:" line on this should match the "Issuer:" line on your PrivateKeyEntry's certificate.

Serve https

It is often fine to serve https using your own private key entry. However, if the CA is not signed by a certificate authority recognized by your users' browsers then they will have to click through warnings to use SHRINE. Also, some institutions may prefer SHRINE nodes to use different certificates to serve https. It is perfectly fine to import an extra cert into the keystore to serve https, but you should inform the hub admin of changes in advance. Your adapter will not run queries until the hub trusts your new cert, but the hub admin can harvest the public side of the cert using openssl.

Configure tomcat to serve https via its server.xml file:

```
<Connector port="{{ shrine_ports.https }}" protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"
               keystoreFile="{{ tomcat_keystore_path }}"
               keystorePass="{{ tomcat_keystore_password }}"
               keyAlias="{{ tomcat_keystore_alias }}" />
```

Update shrine.conf and Restart SHRINE

SHRINE must be configured to understand this type of setup, as it does not assume the presence of a CA by default. For more information on these values, consult the SHRINE Configuration File article and look up **attachSigningCert** and **caCertAliases**.

If the node is a Query Entry Point, add the **attachSigningCert** option into the queryEntryPoint block, and make sure the broadcasterServiceEndpoint.url value points at the appropriate hub.

```
queryEntryPoint {
  [...]
  attachSigningCert = true
  [...]
  broadcasterServiceEndpoint {
    url = "https://shrine-hub.u-of-example.edu:6443/shrine/rest/broadcaster/broadcast"
  }
}
```

In all cases, add the **caCertAliases** option into the keystore block, and make sure that the values for privateKeyAlias and password are also set appropriately. privateKeyAlias should be the same as $KEYSTORE_ALIAS. Also note that caCertAliases is a json array of values. Do NOT surround it in only quotation marks, square brackets must be used. The value should match the alias of the -ca cert imported earlier. If using the ssl_keytool.sh script to import, the alias will automatically be set to the filename of the cert file.

```
keystore {
  [...]
  caCertAliases = ["shrine-hub-ca"]
  [...]
}
```

Save these changes and restart SHRINE. Assuming all passwords and other values are set appropriately, SHRINE should start up successfully, and it will now trust incoming queries from any node that attaches the CA certificate to their query signature.

```
/opt/shrine/tomcat/bin/shutdown.sh && /opt/shrine/tomcat/bin/startup.sh
```