SSL/TLS Certificate Generation

Target: Lightstreamer Server v. 7.0 or greater
Last updated: 1/8/2022

Table of contents

1 Introduction

In order to configure Lightstreamer Server in HTTPS/WSS mode (see the various <keystore> blocks in the Server configuration file), an SSL/TLS certificate is required. This tutorial shows an example of the tasks needed to create an appropriate keystore in the Java “JKS” format, using the tools provided by the Java Development Kit.

The “JKS” format ensures the support by all versions of the Java runtime. The provided instructions, based on the JDK’s “keytool” utility, are also valid for all JDK versions since JDK 7.

Depending on the Java runtime used to run the Server, other keystore types may be supported. This regards in particular the PKCS12 keystore format, which is more powerful, and whose support is guaranteed since Java 9 and on late versions of Java 8.
Several other tools are available to create and handle PKCS12 keystores; the most widespread among them is “openssl”.

The demo keystore created through this example, named *myserver.keystore*, is already available in the *conf* directory of Lightstreamer distribution.

2 Procedures

2.1 Creation and Installation

➔ Generate a new keystore, called *myserver.keystore*, where the key pair is identified by the "LS" alias (you are free to change it). In this example the target hostname is "push.mycompany.com":

```
keytool -genkeypair -alias LS -keystore myserver.keystore -keypass
mypassword -storepass mypassword -storetype JKS -keyalg "RSA" -
keysize 2048 -validity 365 -dname "CN=push.mycompany.com, O=MYCOMPANY
INC., L=MyCity, ST=MyState, C=MyCountry" -ext
SAN=dns:push.mycompany.com
```

NOTE: "keypass" is the password of the key pair; "storepass" is the password of the keystore. The two passwords must be the same[1].

Of course you are free to change any parameters to meet your requirements in terms of algorithms, validity, and contact details.
For instance, by leveraging the SAN (Subject Alternative Name) field, it is possible to create a Multi-domain and/or Wildcard Certificate.

NOTE: The "keytool" command line may turn out to be limited as for support for the SAN field. In this case, you may try other tools. For instance, you can resort to the PKCS12 keystore format and use "openssl", which doesn't suffer from such limitation.

At this point, the *myserver.keystore* file can already be used with Lightstreamer Server for test purpose. But since it is a self-signed keystore, it will raise a security alert in any browser. So, for production environment it is necessary to have the certificate signed by a Certification Authority. The following steps show how to accomplish this goal.

➔ Create the Certificate Signing Request (CSR) to be delivered to a Certification Authority (CA):

```
keytool -certreq -alias LS -keystore myserver.keystore -keypass
mypassword -storepass mypassword -ext SAN=dns:push.mycompany.com -
file myserver.req
```

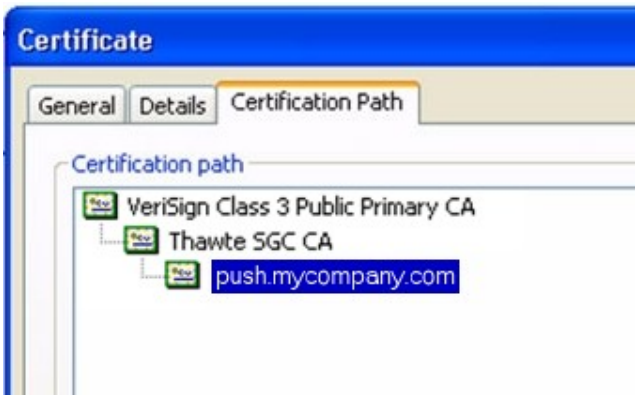
Note the replication, in particular, of the -ext parameter.
The *myserver.req* file is created. It must be sent to a CA to be signed.

NOTE: Some CA may require that in the previous step (i.e. the keystore generation) you have set the "CN" attribute to a proper personal identification key, instead of the target hostname, as a condition for them to accept the CSR. Obviously, you have to supply the target hostname to the CA in some other part of the interaction, as determined by the CA. The CA will take care of providing a signed certificate with the "CN" attribute set to the correct hostname.

➔ When the CA sends the certificate back, it is necessary to import the certificate chain provided by the CA (not including the final certificate) in *myserver.keystore*.

[1] For example see https://search.thawte.com/support/ssl-digital-certificates/index?page=content&id=SO832

In this example, the certificate provided by the CA is called *push.mycompany.com.crt*, which is validated through the following certification chain: “Thawte SGC CA” and “VeriSign Class 3 Public Primary CA”.



Let the intermediate certificate in the chain be the *SuperCertIntermediateCA.crt* file, and the root certificate be the *VeriSign.Class3.Public.Primary.CA.cer* file. Then issue:

```
keytool -importcert -alias int1 -keystore myserver.keystore -keypass
mypassword -storepass mypassword -file SuperCertIntermediateCA.crt

keytool -importcert -alias int2 -keystore myserver.keystore -keypass
mypassword -storepass mypassword -file
VeriSign.Class3.Public.Primary.CA.cer
```

Make sure that all the needed certificates were either supplied by the CA or included in the returned .crt file (*push.mycompany.com.crt*). Otherwise you may need to acquire them in another way.
To extract any certificate from a chain, open the main .crt file (*push.mycompany.com.crt*) in Windows by double clicking, select an intermediate certificate under the “Certification Path” tab, display its details, then copy it on file. An export wizard will open. Choose the binary X.509 format coded with DER and the file name (e.g. *VeriSign.Class3.Public.Primary.CA.cer*).
In fact, all intermediate certificates are public and should be made available by the CA also directly.

➔ After importing the intermediate certificates of the chain, the final certificate can be imported too:

```
keytool -importcert -alias LS -keystore myserver.keystore -keypass
mypassword -storepass mypassword -file push.mycompany.com.crt
-trustcacerts
```

Make sure you use the same alias you used with the “-genkeypair” command.

The *myserver.keystore* file is now ready to be used with Lightstreamer Server.

2.2 Conversion of an Existing Certificate Chain Available in a Different Format

The "keytool" utility supports conversions from different keystore formats into the Java format (JKS), through the "-importkeystore" command.

This enables, for instance, the reuse of existing certificate chains stored in the PKCS12 format, in case the format is not supported by the Java installation running Lightstreamer Server. Other format conversions might be supported by keytool as well.

➔ Case 1: If *mycerts.p12* is a container file in PKCS12 format (the .pfx extension is also commonly used for such files), an equivalent JKS keystore can be created through:

```
keytool -importkeystore -destkeystore myserver.keystore -
deststorepass mypassword -deststoretype JKS -srckeystore mycerts.p12
-srcstoretype PKCS12
```

The command will prompt you for the password associated to *mycerts.p12*.
In the above command, "mypassword" is the password to be assigned to the new keystore; it should be the same as the password associated to *mycerts.p12*.

The *myserver.keystore* file is now ready to be used with Lightstreamer Server.

➔ Case 2: openssl was used to generate a key pair and the related certification request. So a *MYKEY.key* in PEM format has been produced and a *MYCERT.crt* public key certificate has been received back from the Certification Authority. Furthermore, the CA should have supplied a root and one or more intermediate certificates (in PEM format).
Make sure that all the needed intermediate certificates have been received. Otherwise you may need to acquire them in PEM format, in another way, similarly to how discussed above.

Now you could import all of them into a PKCS12 keystore through "openssl".
The openssl command has the form:

```
openssl pkcs12 -export -in MYCERT.crt -inkey MYKEY.key -out
mycerts.p12 -name LS -CAfile MYCACERT.crt -caname myCAname -chain
```

The command will prompt you for a password to be associated to *mycerts.p12*.
Note that *MYCACERT.crt* file must be the chain of the root and all the intermediate certificates of your CA. If you have these in different PEM files you can create the *MYCACERT.crt* file as:

MYCACERT.crt = <concatenation of the encoded public key certificates>

you must order the certificates such that the root certificate is the last certificate in the chain.

Now, depending on the Java installation running Lightstreamer Server, the generated *mycerts.p12* file may or may not be supported. If not supported, you can now convert the keystore to JKS format by just falling into the previous case 1.

2.3 Storing multiple certificate entries in the keystore

TLS certificates can be of various types. In particular, two main options are available for the type of keypair: RSA and ECDSA. The choice can affect the client support, as RSA is weaker and may not be accepted by all clients, whereas ECDSA is more recent and may not be supported by old clients.

The solution is to create both types of certificate for the same hostnames and pack them in the same keystore. In this way, the Server will be able to supply to each client the best certificate based on the client constraints. Note that this mechanism is only supported for PKCS12 keystores.

This can be achieved, for instance, through the "keytool" utility. Suppose, for simplicity, that the two certificates, after being signed separately, have been stored in two PKCS12 keystores, named *myRSA.p12* and *myECDSA.p12,* both with alias LS. Then a new *myserver.p12* keystore that contains both certificates can be created by issuing the following two commands:

```
keytool -importkeystore \
   -srckeystore myRSA.p12 -srcstoretype PKCS12 -srcalias LS \
   -destkeystore myserver.p12 -deststorepass mypassword \
   -deststoretype PKCS12 -destkeypass mypassword -destalias LS_RSA

keytool -importkeystore \
   -srckeystore myECDSA.p12 -srcstoretype PKCS12 -srcalias LS \
   -destkeystore myserver.p12 -deststorepass mypassword \
   -deststoretype PKCS12 -destkeypass mypassword -destalias LS_ECDSA
```

Note that the two entries should have two different aliases and the same password, also equal to the keystore password.

2.4 Utilities

➔ The certificates included in a keystore can be listed with the following command:

```
keytool -list -keystore myserver.keystore -storepass mypassword
```

➔ Each certificate in the keystore can be extracted in a readable format with the following command:

```
keytool -exportcert -alias int1 -keystore myserver.keystore -
storepass mypassword -file int1.cer
```