LISTSERV Maestro Admin Tech Doc 9

Securing Access with HTTPS

May 17, 2019 | © L-Soft Sweden AB
lsoft.com

This document is a LISTSERV Maestro Admin Tech Doc. Each admin tech doc documents a certain facet of the LISTERV Maestro administration on a technical level. This document is number 9 of the collection of admin tech docs and explains the topic “Securing Access with HTTPS”.

Last updated for LISTSERV Maestro 8.2-7 on May 17, 2019. The information in this document also applies to later LISTSERV Maestro versions, unless a newer version of the document supersedes it.

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. L-Soft Sweden AB does not endorse or approve the use of any of the product names or trademarks appearing in this document.

Permission is granted to copy this document, at no charge and in its entirety, provided that the copies are not used for commercial advantage, that the source is cited, and that the present copyright notice is included in all copies so that the recipients of such copies are equally bound to abide by the present conditions. Prior written permission is required for any commercial use of this document, in whole or in part, and for any partial reproduction of the contents of this document exceeding 50 lines of up to 80 characters, or equivalent. The title page, table of contents and index, if any, are not considered part of the document for the purposes of this copyright notice, and can be freely removed if present.

Copyright © 2003-2019, L-Soft Sweden AB
All Rights Reserved Worldwide.

LISTSERV is a registered trademark licensed to L-Soft international, Inc.
L-SOFT and LMail are trademarks of L-Soft international, Inc.
CataList and EASE are service marks of L-Soft international, Inc.
All other trademarks, both marked and not marked, are the property of their respective owners.
Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/
This product includes code licensed from RSA Security, Inc.
This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

All of L-Soft's manuals for LISTSERV are available in ASCII-text format via LISTSERV and in popular word-processing formats via ftp.lsoft.com. They are also available on the World Wide Web at the following URL:

URL: http://www.lsoft.com/manuals.html

L-Soft invites comment on its manuals. Please feel free to send your comments by e-mail to: MANUALS@LSOFT.COM

Table of Contents

1 Secure Access with HTTPS

LISTSERV Maestro is accessed by the user with a web browser over the network, often even over unsecure networks like the internet.

And on the internet, network traffic as such is a public affair: Anyone with the right knowledge and access to certain nodes in the network may "listen in" on the communication between the user's browser and the LISTSERV Maestro server.

Such an attacker may gain knowledge about the data that is sent to the user's browser (for display) and sent back to the server (to trigger a certain action or to submit settings the user made). Even more dangerous, the attacker can find out the user name and password that the user or administrator uses for login, and can then log in himself with the same account.

In this day and age, running a web application like LISTSERV Maestro over an unsecure channel can no longer be considered state of the art. Instead, access should be encrypted using the HTTPS protocol.

The following sections describe how you can configure your LISTSERV Maestro server(s) to use HTTPS. Also, since the topic of encryption, server certificates, trusted authorities etc. is quite complex, an introduction is given that helps you understand the concepts involved, so that you can execute the required steps on an informed basis.

Note: Securing access with HTTPS as described in this tech doc is a separate issue in comparison to authenticating and encrypting communication between the components of LISTSERV Maestro, even though the two have many similarities and can even be combined. If what you are looking for is actually to authenticate and encrypt the communication between the separate components of LISTSERV Maestro (and not the communication between the user and the Maestro User Interface or Administration Hub) then please see "LMA Admin Tech Doc 5 – Multi Server Installation" instead.

2 Introduction to Secure Communication

Note: If you are only interested in the technical details of how to configure LISTSERV Maestro for HTTPS, then you can skip this general introduction section.

This section shall give you a short introduction about the basics of secure communication. Please see the many publications about this topic for more details.

Basically, for encrypted communication, the only thing that is required is, that both communication partners know the encryption key, so that the receiving partner may decrypt the data that was encrypted by the sending partner.

With online communication however, this is a bit more complex. After all, both partners (i.e. the browser and the server), which may even communicate with each other for the first time, do not have a common encryption key that is known only to them (or if you wanted to do this, you would have to give every server/browser combination in the world a unique encryption key, which is obviously impossible).

So when the connection is first established, the two partners decide "on the fly" on an encryption key that is used for the rest of the communication (this is a simplified view of the matter, but it explains the basics).

OK, let's assume that the two partners have decided what key to use, so they can now communicate in an encrypted manner.

There still is a problem: How can the two partners be sure that they are actually communicating with the partner they think they are communicating with?

An analogy to this problem can be found in real life: Let's say that two employees of two partner companies meet in a hotel to exchange confidential information. The two have never met each other, but they know each other's names and home addresses. How can each of them be sure that the other person they meet in the lobby of the hotel is actually the person they are supposed to meet, and not an impostor?

For example, the impostor could act as a "man in the middle": He meets with employee A of corporation A-Corp in the lobby and poses as employee B of corporation B-Corp. Thus he gains the confidential information from A and goes into the bar where he meets the real employee B. Here he poses as employee A from A-Corp, gives the confidential information from the "real" A to B and receives similar information back from B. Finally he goes back into the lobby and gives the information he received from B to the "real" employee A. Of course, on his way from the lobby to the bar and back, he made copies of the information he was carrying. At the end, both employee A and B are unaware that they did not talk to their "real" counterparts, but to an impostor that acted as a man in the middle, and the impostor goes back to his employer C-Corp with the confidential information he gained from their competitors.

On a network, this "man-in-the-middle" attack is even easier to mount: The only thing that server and client know of each other are their network addresses, which can be forged. So how is this solved?

In real live, the two employees of A-Corp and B-Corp would probably simply request to see some picture ID with name and home address of their communication partner. They would then compare the picture on the ID with the person they are talking to and verify, that the name and address on the ID matches the ones they have been told before. If the ID matches the person, they would know that they talk to whom they are supposed to talk to.

But in doing so, they actually implicitly trust a third party that has not been involved yet: This is the agency that issued the picture ID. With accepting the ID, they trust this agency, that it has created the ID in such a way, that it is hard to forge and they also trust this agency, that it in turn verified that the person they issued the ID to really is the person he claims to be. If employee A would have tried to use his golf club picture ID for identification, then employee B would probably have rejected it as improper identification, because he wouldn't trust either that the clerk in A's golf club responsible for issuing the ID really did a thorough check of A's identity, or he didn't trust the security features of the ID (these days, anyone can create a nice looking picture ID with the help of a color laser printer). So instead he would probably request a "proper" ID like a passport or something similar.

With online communication, the problem of identifying the communication partner is solved in a similar way: The role of a "picture ID" in real life is fulfilled by so called "certificates" in the online world. A certificate asserts, that the owner of the certificate really is the entity it claims to be. For example, a certificate could assert, that the server with the host name "`host.somecorp.com`" really is a server that belongs to SomeCorp, and that it is not a server of an impostor.

But how are the problems of credibility (has the identity really been checked thoroughly before the certificate was issued?) and falsification (can the certificate be falsified or a false certificate be created?) solved?

Simply falsifying a file that states "Yes, the server '`host.somecorp.com`' indeed is a server of the SomeCorp corporation" would not be too hard for an impostor.

So what happens is, that the certificate is digitally signed by a trustworthy certification authority (CA), so that it now reads: "Yes, the server '`host.somecorp.com`' is a server of the SomeCorp corporation, and we, the people from TrustCorp, have verified that this is indeed so."

The digital signature is very useful, because it prevents anyone from tampering with the certificate: If even a single letter (or byte) in the text of the certificate is changed, the signature no longer matches and the certificate is invalid.

But a last problem remains: To test the validity of the signature, you require the public key of the certification authority (this is the public counterpart of the private key that the CA used to sign the original certificate with).

Of course you cannot simply use any public key that you find anywhere (or are given by someone), because who would then guarantee that the public key you got really is a key from the entity it supposedly belongs to? You need to be very sure about the origin of the public key that you use to verify the digital signature of a certificate, otherwise, the "man in the middle" would still have a chance to spy on you: The intruder would create his own public/private keys with a forged name of "TrustCorp" and his own certificate with a forged host name of SomeCorp. Then he would use his own private key to sign the certificate and would give the public key to you claiming "this is the public key of TrustCorp". If you would then use this public key to check the validity of the forged certificate, you would get a match, e.g. you would believe the forged certificate that states that a certain host belongs to SomeCorp. While you believed that you communicated with a server of SomeCorp, you would instead be communicating with a server of the attacker.

So how do you verify the public key of the certification authority?

Luckily, when it comes to web-browser, in most cases you do not have to do this explicitly: Most web-browsers are already equipped with a list of trusted so called "root certificates". You do not have to verify that these certificates indeed come from the entity they claim they come from, because the browser vendor has already done this for you.

So the full trust-chain when you use your browser to access a secured server looks like this:

- The browser vendor received root certificates from the signing authorities, verifying their validity.
- You trust the browser vendor that it included only root certificates of entities whose identity he could confirm (and implicitly you also trust yourself, that you didn't install a browser from a 3rd party that only claims to be the browser vendor you think it is, but instead is a front-end for an attacker who wants to give you falsified root certificates this way).
- Your browser trusts any certificate that is signed with a certificate that can be traced to one of the trusted root certificates (this can be a very short chain like "a certificate signed with a root certificate" or a long chain like "a certificate signed with a certificate that was signed with a certificate … etc. … that was signed with a root certificate").
- Your browser trusts any server that has a certificate that the browser trusts.

In the real life example, we met the employees A and B who both needed picture IDs to verify each other. With online communication this verification is often only one-sided: For most purposes it is enough that the client is sure about the server it communicates with. It is usually not required that the server is sure about the client too.

Therefore, usually only the server has a certificate (which is, down the trust chain, signed by a trusted root certificate), but not the client.

There are real-word examples for this too: If you buy a car privately from its former owner, then you probably want to see a proper picture ID of that owner, to make sure that he matches the picture in the ID, and the name and address on the ID match the ones in the documents of the car. Otherwise you would risk buying a stolen car without knowing it. On the other hand, the seller does not necessarily have to see your ID, because he really doesn't care about who you are (as long as you pay cash, of course…).

Finally, lets summarize the concepts we have introduced:

- **Server Certificate**

  This certificate asserts, that a certain server (with the given host name) really belongs to a certain organization, so that you can trust the server and safely communicate confidential data to it. This certificate is digitally signed to prevent tampering and falsification.

- **Trusted Root Certificate**

  The trusted root certificate was used to sign the actual server certificate (or another certificate down the trust-chain that was used to sign the actual server certificate, etc.).

  Usually the fact that a root certificate is installed together with a trusted software (like the browser) already makes it a trusted root certificate. Sometimes you would also receive a root certificate via other means (for example via e-mail). In that case you would first have to verify it before you can rate it as "trusted". For example, you could call the person that sent you the certificate and let her read the certificate's fingerprint to you, which you could compare to the fingerprint of the certificate you received.

- **Encrypted Communication**

  Happens with the help of an encryption key which is generated "on the fly" when the communication begins. The verification that you are not talking to some "man-in-the-middle" while negotiating the encryption key happens through verifying the communication partner's certificate and matching its digital signature (down the trust-chain) to one of the trusted root certificates.

3 The Trusted Root Certificate Keystore

LISTSERV Maestro is shipped with a default keystore that contains the trusted root certificates of many CAs (certification authorities). This keystore is called the trusted root certificate keystore and it plays an important role when using HTTPS.

3.1 Securing the Trusted Root Certificate Keystore

As a first step when starting to use certificates, you should secure the trusted root certificate keystore. The keystore initially has the well-known default password "`changeit`" and is therefore unprotected by default.

To change the password of the trusted root certificate keystore, execute the following command:

```
[maestro_install_folder]/java/bin/keytool -storepasswd
-keystore [maestro_install_folder]/java/lib/security/cacerts
```

You are first queried for the old password (which is "`changeit`", if it has not been changed since installation of LISTSERV Maestro), and then twice for the new password. You need to enter a new password with at least six characters, but longer and more complex passwords are safer!

4 Let LISTSERV Maestro Manage the Certificates

While it is possible to create and manage certificates manually (see section 4.2.2), it is much easier to let LISTSERV Maestro do this for you automatically.

For this, LISTSERV Maestro makes use of the free Let's Encrypt service. Let's Encrypt is a free and well established CA that offers an automation interface through which LISTSERV Maestro can obtain and refresh your certificates for you. The only thing you need to do to use this feature free of charge is to accept the Let's Encrypt subscriber agreement (please see the Let's Encrypt website for details about this agreement).

Using the automatic certificate handling has several advantages:

- It is much easier to configure in LISTSERV Maestro.
- You do not have to use a command line tool to create and manage the certificates.
- You do not have to manually refresh the certificates before they expire, as LISTSERV Maestro refreshes them automatically.
- The certificates that are issued by Let's Encrypt are free to use. You do not have to pay a CA to sign the certificates for you.

4.1 Preconditions for Using Automatic Let's Encrypt Certificates

For the automated Let's Encrypt certification process to work, the following preconditions must be fulfilled:

- **LISTSERV Maestro must use the standard HTTP port 80 for unencrypted HTTP access**

  This applies both to a LISTSERV Maestro server where HTTPS is not yet enabled at all and to a server that is already secured with HTTPS:

  - On a server where HTTPS is not yet enabled: If you now want to create your first Let's Encrypt certificate, then LISTSERV Maestro on that server must currently be using HTTP with the standard HTTP port 80.
  - On a server where HTTPS is already enabled (either manually or with automatic Let's Encrypt certificates): LISTSERV Maestro on that server must be configured to do an automatic HTTP to HTTPS redirect from the standard HTTP port 80 to the HTTPS port.

  This is important because otherwise LISTSERV Maestro will either be unable to obtain the certificates from Let's Encrypt in the first place, or will be unable to automatically refresh the certificates before they expire. See section "4.2.1 Troubleshooting Port 80 Problems" for options in case that your LISTSERV Maestro uses a non-standard port for unencrypted HTTP or if it has HTTPS enabled without a HTTP to HTTPS redirect or if the redirect uses a non-standard HTTP port.

- **Certificate host names and secured IP addresses must match**

  Certificates are issued for certain host names. A given certificate can be issued for a single host name or for a set of host names. Such a certificate is then used to secure a given IP address for HTTPS.

  For this to work, the host names in the certificate must match the host names that are assigned to the IP address. Specifically: All host names that are assigned to an IP address must also be included in the certificate that is to be used to secure this IP address.

For example, for an IP address with only one host name, the certificate should contain only this one host name. For an IP address with three different host names, the certificate should contain exactly these three host names.

This is important, because otherwise LISTSERV Maestro may serve the wrong certificate to the user's browser, which the browser will then show as an invalid and insecure connection.

4.2 Securing an IP Address With an Automatic Let's Encrypt Certificate

To secure an IP address with an automatic Let's Encrypt certificate, follow this procedure:

1. **Decide which IP address you want to secure**

   If you do not know the IP address, but only the server's host name, for example from the LISTSERV Maestro access URL, then you need to find out the IP address that this host name is assigned to (for example with the command line tool "`nslookup`" or something similar).

2. **Determine all public host names that are assigned to this IP address**

   An IP address may have several public host names assigned. You need to find out all of these host names for the certificate.

   It is important that the server in question is accessible from the public internet via all of these host names (this usually means that there must be a registered DNS A record for each host name that points to the selected IP address).

3. **In LISTSERV Maestro, create a new certificate that contains all of these host names**

   To do so, login into the Administration Hub (HUB) as the administrator and select "Global Settings" → "HTTPS Certificates" from the menu. Then on the certificates page, if still necessary, subscribe your LISTSERV Maestro at Let's Encrypt.

   To create a certificate, in the section that lists the servers on which the LISTSERV Maestro components are installed (can be one, two or three servers), locate the server to which the desired IP address is assigned and click the associated "Create new certificate manage by this server" link. Then proceed as described on screen (for more information, refer to the online help via the **[?]** icon in the top right corner).

4. **Use the new certificate to secure the IP address that you decided on**

   The certificate that has been created in the previous step is *not* automatically used for HTTPS connections. To actually make use of the certificate, you need to add a specific entry to the "`tomcat.ini`" file of the corresponding server (i.e. the server for which you created the certificate in the previous step):

   ```
   [maestro_install_folder]/conf/tomcat.ini
   ```

   The entry that you need to add is different for each certificate. To find out the correct entry for a given certificate, click the "How to Apply This Certificate" link that is associated with the certificate.

   On the following page, you can enter the IP address that you want to secure, plus the desired HTTPS port. This will construct the correct INI entry for you, which you can then simply copy & paste to the desired "`tomcat.ini`" file.

   Again, see the online help of that page (via the **[?]** icon in the top right corner) for more information.

With the above procedure complete and the necessary entry added to the tomcat.ini, LISTSERV Maestro on that server will then bind **two** ports on the given IP address:

- The specified HTTPS port for HTTPS connections.
- The default HTTP port 80 for automatic HTTP-to-HTTPS redirects and to enable the server to automatically request and refresh certificates from the Let's Encrypt CA.

Important: It is necessary for LISTSERV Maestro to not only bind the specified HTTPS port, but also the standard HTTP port 80, for the automatic certificate mechanism to work. For the same reason, both ports must be accessible from the public internet. See section "4.2.1 Troubleshooting Port 80 Problems" for options if binding port 80 is not possible.

Remember that, as always when you make changes to the "`tomcat.ini`", you need to restart LISTSERV Maestro on that server to make the changes effective.

Reminder: You can configure which other addresses LISTSERV Maestro shall bind to (in addition to the addresses that are secured as explained above) via the "`BindAddress`" entry. See "LMA Admin Tech Doc 6 – IP Addresses and Ports" for details.

If you are enabling HTTPS for the first time, then most likely you will also have to configure the applicable access URLs and/or tracking URLs to use the "https://" protocol. This is done on the **Default URL Settings** page (menu "Global Settings" → "Maestro User Interface" → "Default URL Settings") and these defaults can also be overridden on account or group level.

4.2.1 Troubleshooting Port 80 Problems

As was explained above, for the automatic certificate mechanism to work, LISTSERV Maestro must be able to bind the standard HTTP port 80 on the IP address that you want to secure (in addition to the desired HTTPS port).

However , in some situations this is not possible. Usually because some other server software (like a web server) is running on the same server and already uses port 80 on the given IP address, which forces the administrator to configure LISTSERV Maestro to use a different port for HTTP connections (for example port 8080).

In such a situation, where the standard HTTP port is not free to be used, it is normally not possible for LISTSERV Maestro to automatically obtain and refresh the Let's Encrypt certificates, which is why the automatic certificate management feature normally cannot be used in such a case.

Unless the other server software supports HTTP redirecting or forwarding. If it does, then the following workaround can be implemented to again enable LISTSERV Maestro to use the automatic certificate management feature:

- The other server software (the one that uses port 80) must be configured to redirect or forward all HTTP requests on the following group of URLs to LISTSERV Maestro.

  The software must be configured to redirect/forward any URL of this pattern:

  `http://`*`YOUR_SERVER`*`/.well-known/acme-challenge/*`

  to this modified URL:

  `http://`*`YOUR_SERVER`*`:`*`MAESTRO_HTTP_PORT`*`/.well-known/acme-challenge/*`

  without otherwise changing the URL (where "`MAESTRO_HTTP_PORT`" is the port as defined in the next step).

- The entry for the "`tomcat.ini`" that was added in step 4 in section 4.2 above, must be augmented with a second port parameter, which specifies the same "`MAESTRO_HTTP_PORT`" as was used above for the redirect/forward.

  This second port parameter is appended to the INI file entry separated by a second colon "`:`". Obviously, this port must be a free port that is not currently in use (most importantly, it must not be the HTTPS port or the default port 80 that is already in use by the other server software).

  For example, if you want to use port 8080 and the INI entry originally looks like this:

  ```
  Secure-192.168.1.1=5O7THERW2NKE6VQLW3UN:443
  ```

  then you must modify this entry to look like this:

  ```
  Secure-192.168.1.1=5O7THERW2NKE6VQLW3UN:443:8080
  ```

  With the original entry, LISTSERV Maestro would attempt to bind both port 443 and port 80 on the given IP address. With the modified entry, LISTSERV Maestro will instead bind port 443 and port 8080.

  (Remember that you need to restart LISTSERV Maestro to make the change effective.)

Advanced Feature 1: It is possible to configure LISTSERV Maestro to bind to more than one HTTP-to-HTTPS redirect port. Any incoming HTTP requests on these ports are then forwarded as a HTTPS requests to the HTTPS port. If you want to have several such redirect ports, simply append them with additional colons ":" as separators. For example, if you want to bind the standard port 80 and the non-standard ports 8080 and 8081 to all redirect from HTTP to HTTPS (on port 443), then the entry would look like this:
`Secure-192.168.1.1=5O7THERW2NKE6VQLW3UN:443:80:8080:8081`

Advanced Feature 2: The HTTPS port as specified in the `Secure-xxx` entry is used for two purposes: It is the port that Maestro binds to (for the HTTPS protocol) and it is the target HTTPS port that is used for the automatic HTTP-to-HTTPS redirect.
In some situations however, it can be necessary to use two different ports for these two purposes. For example if Maestro is installed behind a proxy and the proxy exposes Maestro on a different port than what Maestro uses to bind to on the local machine. E.g. if the proxy exposes Maestro on the default HTTPS port 443, but on the local machine Maestro is instead supposed to bind on the non-standard port 7701. In that case, it is necessary to specify in the `tomcat.ini`, that Maestro is supposed to bind on one port (7701) but is also supposed to send the HTTP-to-HTTPS redirect to a different port (443).
This can be done by specifying the two HTTPS ports as a comma separated pair, like this:
`Secure-192.168.1.1=5O7THERW2NKE6VQLW3UN:7701,443`
Of this pair, the first port is the port that Maestro shall bind to (for HTTPS) and the second port is the port that is to be used as the target HTTPS port when doing HTTP-to-HTTPS redirects.
In the example above, no special HTTP port is given, so the HTTP port used will be the default of 80. Of course this can also be combined with specifying a different HTTP port (after a colon):
`Secure-192.168.1.1=5O7THERW2NKE6VQLW3UN:7701,443:8080`
Or even with specifying a list of different HTTP ports:
`Secure-192.168.1.1=5O7THERW2NKE6VQLW3UN:7701,443:80:8080:8081`

Advanced Feature 3: For very special use cases, it is even possible to tell LISTSERV Maestro to only bind the HTTPS port, and not bind any HTTP redirect port at all. To do so, modify the INI entry with a second colon "`:`" after the HTTPS port, but *not* followed by a HTTP port. For example like this (note the colon at the very end):
`Secure-192.168.1.1=5O7THERW2NKE6VQLW3UN:443:`

4.2.2 Modifying an Existing Let's Encrypt Certificate

In some situations, it may be necessary to add a new host name to a certificate, or to remove one of the names from a certificate, for example if a new host name is assigned to the IP-address (or removed from it).

In fact, you cannot actually change the host names of an existing certificate. Instead, you create a new certificate in addition to the old certificate, only with more (or less) host names than the old certificate. This new automatic Let's Encrypt certificate is created in the same fashion as the original one (see above).

So for a time, two certificates exist, with an overlapping set of host names. Of these two certificates, initially only the old certificate is in use.

Once the new certificate is created, you then modify the `tomcat.ini` to use the new certificate instead of the old one and restart LISTSERV Maestro. After the restart, only the new certificate will be in use, and the now superfluous old certificate can be revoked in the HUB.

5 Manage Certificates Manually

In addition to handling certificates automatically for you, as described in section 4, LISTSERV Maestro also allows you to create, manage and refresh your certificates manually. This section and its sub-sections describe the procedure for doing this.

Note: Managing certificates manually is much more complicated (as evidenced by the lengthy description in the following sub-sections) and is not recommended, unless you have good reasons why you cannot use LISTSERV Maestro's automatic certificate management as described in the previous section.

5.1 The keytool Tool

To manually manage certificates for LISTSERV Maestro, you use the command line tool called "`keytool`", that is installed together with LISTSERV Maestro and that is part of the Java installation that comes with LISTSERV Maestro. For more information about this tool, see the documentation of this tool at Oracle's website:

http://docs.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html

or

http://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

5.2 Install a Server Certificate

To enable HTTPS in LISTSERV Maestro, you need to obtain a signed server certificate for the server you want to secure.

You cannot simply obtain just any server certificate and use it on whatever server you want: The certificate is always bound to the explicit server name that you chose when you created the certificate. Should you want to move your LISTSERV Maestro component to a different server (with a different name) or want to rename your server, then you would have to obtain a new certificate for the new name.

You can either create a new server certificate from scratch, using the `keytool` that is included with LISTSERV Maestro, as described in section "5.2.1 Create and Install a New Server Certificate", or you

can use an existing certificate (for the correct host name) that you have already obtained, as described in section "5.2.2 Install an Existing Server Certificate".

5.2.1 Create and Install a New Server Certificate

If you do not already have a signed certificate for your server, then you will need to create and install a new server certificate. This basically involves three steps, which are explained in more detail in the following sub-sections:

1. Create an unsigned certificate with the name of the server you want to secure.
2. Create a certificate signing request (CSR) from that certificate and send it to a certification authority (CA). The CA first verifies that you really are who you claim to be and then returns a signed version of your certificate to you.
3. Replace the unsigned certificate with the signed certificate you get back from the CA.

5.2.1.1 Create an Unsigned Server Certificate

In Java, all certificates are stored in a so called "keystore", which is a special file that is protected with a password.

Important: Be **very** careful with the keystore file into which you create the certificate. You need to protect this file in several respects:

- **Do not "lose" or accidentally delete this file**, since it contains your certificates and you would have to purchase new certificates if you lose this file. **You should always keep a backup at a safe location**. This backup location should be outside of your LISTSERV Maestro installation folder, or otherwise the files could be deleted during an uninstallation of LISTSERV Maestro.
- **Protect the file against unauthorized access.** Even though the file is password protected, passwords can always be cracked, and an attacker could thus gain access to your certificates.

To create a new certificate in a keystore file, execute the following command:

```
[maestro_install_folder]/java/bin/keytool -genkeypair -alias NAME -validity
DAYS -keystore KEYFILE -keyalg RSA -keysize 2048
```

with the following replacements:

NAME: The name of the certificate. Can be any name that is not already in use in the keystore file you specify (see below), but you should choose an informative name that helps you recognize the certificate at a later time.

DAYS: Limits the validity of the certificate. The certificate will expire so many days after the day when you created it. Can be any number of days. Usually, when you purchase the signing service from the CA, you pay for only a limited period during which the certificate shall be valid. Choose a number of days for this parameter, that is no shorter than the period you purchase from the CA (a little padding here is probably a good idea, to be on the safe side).

KEYFILE: The keystore file to which the certificate shall be added. If the file doesn't exist, it is created. If it already exists, a certificate with the given "*NAME*" is added to it. Note: This is **not**

the same keystore file as the trusted root certificate keystore file that was used in section 3, but is a different keystore, in which you store your server certificate.

Choose a suitable location and filename for your keystore file, that takes into account the special security considerations for this file (see below).

Note about the `-keysize` parameter: The command line as shown above specifies a key size of 2048 bits, which conforms to the current recommended standard. You should only specify a shorter key if there is a good reason for this.

The tool first prompts you to enter the password with which the keystore is protected. If you are using an existing keystore, you need to enter its password. If you gave a filename of a keystore that does not yet exist, then a new keystore is created and it is protected with the password you enter at the first prompt (you must choose a password with at least six characters, but longer and more complex passwords are safer).

Next, the tool prompts you for the following information values. Each time you may press ENTER to simply accept the default value "Unknown". However, some values you must enter for the certificate to work and some CAs require that you fill out other values. So it is generally a good idea to fill out all values with whatever fits bets in your case (see below):

- What is your first and last name?

  Here you **must** enter the **host name** of the server that you want to secure with the certificate you are creating. Yes, even though the question asks after your "first and last name", you have to enter the host name of the computer instead!

  This should be the same host name that is used in the URLs to access the server. For example, if the URL is "`http://maestro.mycorp.com/lui`", then you need to enter the host name "`maestro.mycorp.com`" (without the quotes, of course).

  If the certificate is supposed to be a wild card certificate, then you enter the appropriate wild card host name or domain name here.

  If the certificate is supposed to contain several host names, then pick one of the host names as you main host name and enter it here. The remaining host names will be added in a later step, when you generate the certificate signing request (CSR) for the certificate (see below).

- What is the name of your organizational unit?
- What is the name of your organization?
- What is the name of your City or Locality?
- What is the name of your State or Province?
- What is the two-letter country code for this unit?

  Here you should use the two-letter uppercase code that fits the country where the server is deployed, like US, DE, SE, CH, etc.

When you have answered the last question, you see a summary of your input and are requested to confirm it. Type "yes" and ENTER to accept your input, or "no" and ENTER (or simply ENTER) to reject it (in this case you can enter the values again, until you are satisfied).

After you have confirmed your input, the tool generates the certificate. When it is done, it asks you for a password to protected the certificate. Although generally you are able to choose any password, for the certificate to be usable with LISTSERV Maestro, you **must** enter the same password as you

chose for the keystore itself. To do so, simply press ENTER without entering anything, so that you accept the default.

At this point, the certificate is created (as a public/private key pair), but it is still unsigned and can therefore not yet be used for a secure HTTPS connection.

5.2.1.2 Perform a Certificate Signing Request (CSR)

Once you have created an unsigned certificate, you can generate a certificate signing request (CSR) from it, that you can submit to a certification authority (CA) of your choice, for example VeriSign.

To generate a CSR for a certificate in the keystore for the single host name that is already stored in the certificate, execute the following command:

```
[maestro_install_folder]/java/bin/keytool -certreq -alias NAME -file
OUTFILE -keystore KEYFILE
```

If you want to generate a CSR for a certificate that shall include additional host names (in addition to the main host name that is already stored in the certificate), execute the following command instead:

```
[maestro_install_folder]/java/bin/keytool -certreq -alias NAME -file
OUTFILE -keystore KEYFILE -ext san=ADDITIONAL_HOST_NAMES
```

with the following replacements:

NAME: The name of the certificate. This must be the name of the certificate that you want to create the CSR for (the same name that you specified when you created the certificate).

OUTFILE: The file into which the CSR will be written. If the file already exists, it will be replaced with the new file.

KEYFILE: The keystore file in which the certificate is stored.

ADDITIONAL_HOST_NAMES: Only necessary if you want to add additional host names to the CSR. Specify all additional host names (but not the main host name that is already stored in the certificate) as a comma-separated list (without separating spaces), where each host name is prefixed with the token "`dns:`". For example:

```
dns:www.mycorp.com,dns:maestro.mycorp.com,dns:sample.mycorp.com
```

The command queries you for the password of the keystore. After you have entered it, the file you specified as "*OUTFILE*" will be written.

This file is a text file that contains the CSR in Base64-encoded form (PEM). It will look similar to this:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
tPnJhsLOuocsBYAmyM11qiZ5BEVWAnJfZ6kyN/XfT5NFxGIy9Uynz5kODfBwFUgiu98iQKWyMKC/
bGFuZ2VuMQ8wDQYDVQQKEwZMLVNvZnQxEDAOBgNVBAsTB1Vua25vd24xDzANBgNVBAMTBnRlcHBp
6E7Zyl9wkPyVpn1qbnbtXQGAablJInE9/LruaJ1NX1f/NVJgL4vPiDKsU4laGvJHBNhdj+F0uVb3
SIb3DQEBBAUAA4GBAB6XqdfJvhy7dTHijsHjw+c4ELQFI/TkHBvgp5NaCccQoNwwW9lnIeOikDb2
lwWg56G6LiYfpVBss5+OOW2jXlq9CdNw1KLSDQ+kMtZjdVr8+iQ9gsqxvskCAwEAAaAAMA0GCSqG
MIIBpjCCAQ8CAQAwZjELMAkGA1UEBhMCREUxEDAOBgNVBAgTB0dlcm1hbnkxETAPBgNVBAcTCEVy
YzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAz+hQRsqDWRLvmV4YD5+JaQEXn5qqJeyzkfg2
PQoU2VPgHID0VnyTPt8r/t4uFk8p1NxjYkC4
-----END NEW CERTIFICATE REQUEST-----
```

You now have to submit this CSR to the CA of your choice which is usually possible online.

After the CA has received your CSR, they will first verify that you (or your corporation) indeed are who you claim to be, i.e. if the content of the certificate can be trusted or not. This usually happens via conventional means (making phone calls, checking company registrations, etc.) and may take a few days.

Once they have verified the validity of the certificate, they will either return the signed certificate to you, or will supply you with further instructions on how to obtain it.

5.2.1.3 Install the Signed Server Certificate

The signed certificate that you receive back from your CA must be in X.509 format (v1, v2 or v3), either in binary format (DER encoded) or Base64 encoded text format (PEM). The reply can be a single certificate or a certificate chain (with intermediate certificates), where the latter must be a PKCS#7 formatted reply or a sequence of X.509 certificates in a single file.

Once you have received the signed certificate (or certificate chain), store it into a file. Then execute the following command:

```
[maestro_install_folder]/java/bin/keytool –importcert –alias NAME –file
INFILE -keystore KEYFILE –trustcacerts
```

with the following replacements:

NAME: The name of the certificate. This must be the name of the certificate that you made the CSR for (the same name that you specified when you created the certificate and the CSR).

INFILE: The file in which you have stored the reply from the CA that contains your signed certificate or certificate chain.

KEYFILE: The keystore file in which the certificate is stored.

The import command loads the certificate from the given file and tries to validate it using the other certificates in the given file (if the file contained a certificate chain) and the trusted certificates in LISTSERV Maestro's trusted root certificate keystore.

If a trust chain can be established, starting at the imported certificate and ending at a self-signed trusted root certificate, then the signed certificate is imported into the keystore, and thus replaces the unsigned version of the certificate that was in the keystore before. You now have a keystore that you can use to enable HTTPS in LISTSERV Maestro.

If the trust chain cannot be established, an error message is printed out and the signed certificate is not imported. See section "5.2.1.3.1 Troubleshooting Certificate Import Problems" for details on how to proceed in this case.

But if the certificate *was* imported successfully, continue with section "5.3 Make LISTSERV Maestro aware of the Server Certificate". Optionally, you can first verify the keystore, as described in section "5.2.4 Verify the Certificate Keystore".

5.2.1.3.1 Troubleshooting Certificate Import Problems

If the certificate import command that was described above did not successfully import the signed certificate into the keystore, but instead you got an error message that no trust chain could be established, then you are either missing one or more intermediate certificates in the certificate

chain, and/or LISTSERV Maestro does not have the CA's trusted root certificate in its trusted root certificate keystore. The following two items describe how to proceed in these situations.

1. **Missing Intermediate Certificates**

   Situation: The certificate was not signed directly with a trusted root certificate, but was signed with an intermediate certificate, and this intermediate certificate was missing from the imported certificate file.

   In this case, it is not possible to build a trust chain from the signed certificate to the trusted root certificate, via the intermediate certificate, because the intermediate certificate is missing.

   Solution: Make sure that the *INFILE* that you used for the command above contains both the signed server certificate and also the necessary intermediate certificate (or even several intermediate certificates, if several intermediates were used between the server certificate and the root certificate).

   For example, if the *INFILE* is in Base64 encoded PEM format, then a certificate begins with a `-----BEGIN CERTIFICATE-----` line and ends with an `-----END CERTIFICATE-----` line. If you need to import the server certificate plus one intermediate certificate, then the *INFILE* should therefore contain two such blocks (of which the first usually is the server certificate and the second the intermediate certificate), similar to this:

```
-----BEGIN CERTIFICATE-----
MIIJhsLOuocsBYAmyM11qiZ5BEVWAnJfZ6kyN/XfT5NFxGIy9Uynz5kODfBwFUgiu98iQKWyMKCa
6E7Zyl9wkPyVpn1qbnbtXQGAablJInE9/LruaJ1NX1f/NVJgL4vPiDKsU4laGvJHBNhdj+F0uVb3
BpjCCAQ8CAQAwZjELMAkGA1UEBhMCREUxEDAOBgNVBAgTB0dlcm1hbnkxETAPBgNVBAcTCEVytPn
SIb3DQEBBAUAA4GBAB6XqdfJvhy7dTHijsHjw+c4ELQFI/TkHBvgp5NaCccQoNwwW9lnIeOikDb2
lwWg56G6LiYfpVBss5+OOW2jXlq9CdNw1KLSDQ+kMtZjdVr8+iQ9gsqxvskCAwEAAaAAMA0GCSqG
YzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAz+hQRsqDWRLvmV4YD5+JaQEXn5qqJeyzkfg2
BgNVBAsTB1Vua25vd24xDzANBgNVBAMTBnRlcHBpPQoU2VPgHID0VnyTPt8r/t4uFk8p1NxjYkC4
bGFuZ2VuMQ8wDQYDVQQKEwZMLVNvZnQxEDAO
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIBpjCCAQ8CAQAwZjELMAkGA1UEBhMCREUxEDAOBgNVBAgTB0dlcm1hbnkxETAPBgNVBAcTCEVy
tPnJhsLOuocsBYAmyM11qiZ5BEVWAnJfZ6kyN/XfT5NFxGIy9Uynz5kODfBwFUgiu98iQKWyMKCa
bGFuZ2VuMQ8wDQYDVQQKEwZMLVNvZnQxEDAOBgNVBAsTB1Vua25vd24xDzANBgNVBAMTBnRlcHBp
6E7Zyl9wkPyVpn1qbnbtXQGAablJInE9/LruaJ1NX1f/NVJgL4vPiDKsU4laGvJHBNhdj+F0uVb3
SIb3DQEBBAUAA4GBAB6XqdfJvhy7dTHijsHjw+c4ELQFI/TkHBvgp5NaCccQoNwwW9lnIeOikDb2
lwWg56G6LiYfpVBss5+OOW2jXlq9CdNw1KLSDQ+kMtZjdVr8+iQ9gsqxvskCAwEAAaAAMA0GCSqG
YzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAz+hQRsqDWRLvmV4YD5+JaQEXn5qqJeyzkfg2
PQoU2VPgHID0VnyTPt8r/t4uFk8p1NxjYkC4
-----END CERTIFICATE-----
```

   Note: Do *not* include the actual trusted root certificate of the root CA in this file, but only the (signed) server certificate that you received back from the CA, plus any necessary intermediate certificates. In many cases, if intermediate certificates are required, they will already be included with the server certificate in the fashion described above, when the signed server certificate is returned to you by the CA.

   If all the necessary intermediate certificates are included correctly (or if none are necessary in the first place), then the import as described above should be successful. But if you *still* get the trust chain error during import, even with all intermediate certificates present, then most likely you have encountered the "Missing Trusted Root Certificate" situation as described in the following item. In this case, proceed as described below.

2. **Missing Trusted Root Certificate**

   Situation: The certificate was signed by a CA for which LISTSERV Maestro does not already have a trusted root certificate in the trusted root certificate keystore.

   In this case, it is not possible to build a trust chain from the signed certificate to the trusted root certificate (possibly via intermediate certificates), because the trusted root certificate itself is missing.

   Solution: Install the root certificate of the CA (the one that was used to sign your server certificate) into the trusted root certificate keystore of LISTSERV Maestro, as described in section "5.2.3 Install a New Trusted Root Certificate". Then return to the section 5.2.1.3 above and try to import the signed server certificate again.

   Remember: Only install the actual root certificate into LISTSERV Maestro's trusted root certificate keystore, never the intermediate certificates.

5.2.2 Install an Existing Server Certificate

If you already have a signed server certificate for the host name that you use with LISTSERV Maestro, then it is possible to use this existing certificate to enable HTTPS in LISTSERV Maestro.

To do so, you must store the certificate (and its private key), plus all necessary intermediate certificates, as a single entry in a Java keystore file. You may also have to add the root certificate of your CA to the trusted root certificate keystore of LISTSERV Maestro.

The procedure for how you do this depends on the format that the certificate is currently stored in and can therefore differ from case to case. The following subsections describe the most common scenarios.

5.2.2.1 Existing Certificate in a Java Keystore

Situation: The server certificate plus the necessary intermediate certificates (if any) are already stored in a Java keystore file as a single signed `PrivateKeyEntry` (with a certificate chain length that reflects the number of intermediate certificates).

Solution: If the certificate entry in the keystore conforms to the rules as described in section "5.2.4 Verify the Certificate Keystore", then you can use this keystore directly, as described in section "5.3 Make LISTSERV Maestro aware of the Server Certificate", without the need to further convert it into a LISTSERV Maestro compatible format. When using this keystore, you should also check for any root certificate problems as described in section "5.2.2.3 Check the Root CA Certificate".

If the certificate does not conform to these rules, then the keystore entry must be changed so that it fulfills these rules. Exactly which changes are necessary for this is beyond the scope of this document. In such a case, please contact L-Soft support for help.

5.2.2.2 Existing Certificate in X.509 Certificate Files or PKCS12 Format

An existing server certificate is often not already available in a Java keystore format, but rather as standard X.509 certificate files. Sometimes as separate files for the server certificate and intermediate certificates, sometimes as a combined file, sometimes even already in PKCS12 format.

Each of the following subsections starts off with a different situation regarding the initial structure and format of the certificate file(s) and describes how to convert the certificate into the Java keystore format that can be used with LISTSERV Maestro.

Please proceed to the subsection that comes closest to your current situation and continue from there.

5.2.2.2.1 Existing Certificate in Separate X.509 Certificate Files, Plus Private Key File

Situation: The server certificate is stored in a X.509 certificate file and the private key for the certificate is stored in another separate file. There is also at least one intermediate certificate (excluding the CA's root certificate) and the intermediate certificates are also stored in their own separate X.509 certificate files (i.e. there are individual files for all certificates and the private key).

Solution: You need to combine the server certificate and the intermediate certificates into a single certificate file, forming a sequence of X.509 certificates. If the certificates are stored in binary (DER encoded) format, you will need to use a tool like openssl to do this (see https://www.openssl.org/). If they are stored in Base64 encoded text format (PEM), you can simply do so with a text editor:

In Base64 encoded PEM format, a certificate begins with a `-----BEGIN CERTIFICATE-----` line and ends with an `-----END CERTIFICATE-----` line. To combine the server certificate and the intermediate certificates into a single file, simply put them into that file one after the other, with a text editor. For example, if you have the server certificate plus one intermediate certificate, then the resulting file would be similar to this:

```
-----BEGIN CERTIFICATE-----
MIIJhsLOuocsBYAmyM11qiZ5BEVWAnJfZ6kyN/XfT5NFxGIy9Uynz5kODfBwFUgiu98iQKWyMKCa
6E7Zyl9wkPyVpn1qbnbtXQGAablJInE9/LruaJ1NX1f/NVJgL4vPiDKsU4laGvJHBNhdj+F0uVb3
BpjCCAQ8CAQAwZjELMAkGA1UEBhMCREUxEDAOBgNVBAgTB0dlcm1hbnkxETAPBgNVBAcTCEVytPn
SIb3DQEBBAUAA4GBAB6XqdfJvhy7dTHijsHjw+c4ELQFI/TkHBvgp5NaCccQoNwwW9lnIeOikDb2
lwWg56G6LiYfpVBss5+OOW2jXlq9CdNw1KLSDQ+kMtZjdVr8+iQ9gsqxvskCAwEAAaAAMA0GCSqG
YzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAz+hQRsqDWRLvmV4YD5+JaQEXn5qqJeyzkfg2
BgNVBAsTB1Vua25vd24xDzANBgNVBAMTBnRlcHBpPQoU2VPgHID0VnyTPt8r/t4uFk8p1NxjYkC4
bGFuZ2VuMQ8wDQYDVQQKEwZMLVNvZnQxEDAO
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIBpjCCAQ8CAQAwZjELMAkGA1UEBhMCREUxEDAOBgNVBAgTB0dlcm1hbnkxETAPBgNVBAcTCEVy
tPnJhsLOuocsBYAmyM11qiZ5BEVWAnJfZ6kyN/XfT5NFxGIy9Uynz5kODfBwFUgiu98iQKWyMKCa
bGFuZ2VuMQ8wDQYDVQQKEwZMLVNvZnQxEDAOBgNVBAsTB1Vua25vd24xDzANBgNVBAMTBnRlcHBp
6E7Zyl9wkPyVpn1qbnbtXQGAablJInE9/LruaJ1NX1f/NVJgL4vPiDKsU4laGvJHBNhdj+F0uVb3
SIb3DQEBBAUAA4GBAB6XqdfJvhy7dTHijsHjw+c4ELQFI/TkHBvgp5NaCccQoNwwW9lnIeOikDb2
lwWg56G6LiYfpVBss5+OOW2jXlq9CdNw1KLSDQ+kMtZjdVr8+iQ9gsqxvskCAwEAAaAAMA0GCSqG
YzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAz+hQRsqDWRLvmV4YD5+JaQEXn5qqJeyzkfg2
PQoU2VPgHID0VnyTPt8r/t4uFk8p1NxjYkC4
-----END CERTIFICATE-----
```

Note: Do *not* include the actual trusted root certificate of the root CA in this file, but only the (signed) server certificate, plus any necessary intermediate certificates.

Once you have combined the server certificate and the intermediate certificates into a single file, proceed as described in the next subsection.

5.2.2.2.2 Existing Certificate in Single X.509 Certificate File, Plus Private Key File

Situation: The server certificate is stored in a X.509 certificate file and there are no necessary intermediate certificates, or all intermediate certificates are stored in the same certificate file as the server certificate, forming a sequence of X.509 certificates in the file. Also, the private key for the server certificate is stored in second file.

Solution: You need to convert the combined certificate file and the private key file into a single PKCS12 file. This can for example be done with the following openssl command (see https://www.openssl.org/):

```
openssl pkcs12 -export -in CERTFILE -inkey PRIVKEYFILE -out PKCS12FILE
-name NAME
```

with the following replacements:

CERTFILE: The file that contains the server certificate and the intermediate certificates (if any).

PRIVKEYFILE: The file that contains the private key for the server certificate.

PKCS12FILE: The file to which the resulting PKCS12 file shall be written. Usually with a `*.p12` extension.

NAME: The name to identify the certificate. Can be any name, but you should choose an informative name that helps you recognize the certificate at a later time.

Once you have combined the certificate file and private key file into a single PKCS12 file, proceed as described in the next subsection.

5.2.2.2.3 Existing Certificate in Single PKCS12 File

Situation: The server certificate and all intermediate certificates (if any) and the private key for the certificate are stored in a single PKCS12 file.

Solution: You need to convert this PKCS12 file into a Java keystore that can be used by LISTSERV Maestro. This is done with the following command:

```
keytool –importkeystore –srckeystore PKCS12FILE -srcstoretype PKCS12
-destkeystore KEYFILE
```

with the following replacements:

PKCS12FILE: The PKCS12 file that contains the certificate and private key.

KEYFILE: The Java keystore file into which the PKCS12 file shall be converted. Specify the name of a file that does not yet exist, so that a new file is created.

Once you have converted the PKCS12 file into a Java keystore, you now have a keystore that you can use to enable HTTPS in LISTSERV Maestro. To verify that the keystore is indeed compatible with LISTSERV Maestro, please check it as described in section “5.2.4 Verify the Certificate Keystore”.

If the keystore is compatible, see section “5.3 Make LISTSERV Maestro aware of the Server Certificate” on how to use it with LISTSERV Maestro. When using this keystore, you should also check for any root certificate problems as described in section “5.2.2.3 Check the Root CA Certificate”.

5.2.2.3 Check the Root CA Certificate

If you use an existing server certificate with LISTSERV Maestro as described here in section 5.2.2, no matter if the certificate was already stored in a Java keystore or was converted into a keystore from a different format, then this bypasses the automatic validation of the certificate against the CA’s root certificate (from LISTSERV Maestro’s trusted root certificate keystore) that would happen if a new certificate were created as described in section 5.2.1.

Or in other words: It is possible to use such an existing certificate without having the CA’s root certificate in the trusted root certificate keystore of LISTSERV Maestro.

This is fine for the server side, i.e. when LISTSERV Maestro acts as the server for a HTTPS connection. But it creates problems when LISTSERV Maestro acts both as a HTTPS client and server while accessing itself (which is necessary for some LISTSERV Maestro features).

As a result, if LISTSERV Maestro does not have the CA's root certificate in its trusted root certificate keystore, then these specific LISTSERV Maestro features will fail, sometimes in ways that are not immediately recognizable as being related to the HTTPS setup.

Therefore, if you are using an existing server certificate (and not creating a new one), then after enabling the HTTPS access you should check, if there are such root certificate related problems.

In LMA 7.0 and later, with HTTPS enabled, you can check this by logging in as a normal LISTSERV Maestro user and then selecting the **Mail Jobs** node in the Maestro explorer. With the node selected, if the main window pane correctly shows the job list (with "Jobs In Folder" tab, etc.), then there are no root certificate problems on your system. But if the main window pane remains empty and you see SSL/encryption related errors in the LUI log file, then this means that your CA's root certificate is indeed missing in LISTSERV Maestro.

To fix this, add the root certificate as a new trusted root certificate as described in section 5.2.3. Then restart LISTSERV Maestro and try again.

5.2.3 Install a New Trusted Root Certificate

IMPORTANT: This step is **only** required if during the import of a certificate, or when checking the root CA certificate, it turns out that the root certificate for your CA is not already included in LISTSERV Maestro's trusted root certificate keystore.

In this case, install the trusted root certificate as follows:

You should be able to get the root certificate you require from your CA. The certificate must be stored in a file, either in "DER encoded binary X.509" or "Base-64 encoded X.509 (PEM)" format.

If you have access to such a certificate file, you can import it into the trusted root certificate keystore by executing the following command:

```
[maestro_install_folder]/java/bin/keytool –importcert –alias NAME –file
INFILE -keystore [maestro_install_folder]/java/lib/security/cacerts
```

with the following replacements:

NAME: The name that you want to give the certificate in your keystore. This name is not really important for anything, except for you to be able to recognize the certificate as what it is at a later time. Also this name must not already be in use in the keystore.

INFILE: The file in which you have stored the X.509 certificate from your CA.

You will be queried for the password of the default keystore file, which you should have set to something else than its default "`changeit`" earlier (see section "3.1 Securing the Trusted Root Certificate Keystore").

The command will present the details of the certificate you want to import in a way similar to this:

```
Owner: OU=For VeriSign authorized testing only. No assurances (C)VS1997,
OU=www.verisign.com/repository/TestCPS Incorp. By Ref. Liab. LTD., O="VeriSign, Inc"
Issuer: OU=For VeriSign authorized testing only. No assurances (C)VS1997,
OU=www.verisign.com/repository/TestCPS Incorp. By Ref. Liab. LTD., O="VeriSign, Inc"
Serial number: 52a9f424da674c9daf4f537852abef6e
Valid from: Sun Jun 07 02:00:00 GMT+02:00 1998 until: Wed Jun 07 01:59:59
GMT+02:00 2006
Certificate fingerprints:
         MD5:  40:06:53:11:FD:B3:3E:88:0A:6F:7D:D1:4E:22:91:87
         SHA1: 93:71:C9:EE:57:09:92:5D:0A:8E:FA:02:0B:E2:F5:E6:98:6C:60:DE
Trust this certificate? [no]:
```

The presentation contains details about the certificate, but those could have been forged. It also contains the certificate's fingerprints, which you can use to verify that the certificate has indeed not been falsified. For example, if the certificate was mailed to you (thus giving a potential attacker the possibility to "catch" the mail to you before it reaches you and replacing the certificate therein with his own certificate for a future "man-in-the-middle" attack), then you might want to call the responsible person of your CA, to verify the fingerprint of the certificate over the phone. Or check the CA's website for the fingerprint of their root certificate.

Once you are sure that the certificate is indeed genuine, you should answer "yes" and ENTER to the question "Trust this certificate?".

After you have done this, the certificate is installed as a new trusted root certificate in your trusted root certificate keystore.

5.2.4 Verify the Certificate Keystore

The certificate keystore file that holds the signed server certificate must conform to certain rules to be compatible with LISTSERV Maestro:

- The password for the keystore itself and the password for the certificate entry in the keystore **must** be the same, otherwise the certificate is not usable by LISTSERV Maestro.
- If the server certificate was not signed directly by the CA's root certificate, but if it was instead signed with an intermedia certificate (or possibly even a chain of several intermediate certificates), then all these intermediate certificates must be included in the same certificate entry in the keystore:
    - There must still be only a single entry for the certificate, but this entry must contain several certificates: The server certificate itself as the first certificate, followed by all the intermediate certificates.
    - It is not possible to have the intermediate certificates as separate entries in the keystore, as some features of LISTSERV Maestro will then not work.

    Essentially, if you import the server certificate and all intermediate certificates together with a single import command, from a single file that contains all certificates at once, then this correctly creates a single entry in the keystore, which contains several certificates. This is how it is supposed to be.

    But if you import the server certificate and the intermediate certificates separately, with multiple import commands and from multiple files, then you end up with multiple entries in the keystore file (one for each imported certificate), which is not compatible with LISTSERV Maestro.

- The actual root certificate of your CA should never be imported explicitly into your keystore (the one that also holds the server certificate) as a separate keystore entry, but only into LISTSERV Maestro's trusted root certificate keystore, if it is not already included in that keystore. See section "5.2.3 Install a New Trusted Root Certificate" for details.

  However, when you import a certificate chain (consisting of your server certificate and the necessary intermediate certificates, see above) into a keystore entry, then the certificate chain that is contained in that entry sometimes also contains the CA's root certificate as the very last certificate of the chain (even if your imported certificate file did not contain it explicitly). This is fine and has no negative impact on LISTSERV Maestro.

To display the details of the certificate entry in your keystore, use the following command:

```
[maestro_install_folder]/java/bin/keytool –list –v –alias NAME -keystore
KEYFILE
```

with the following replacements:

NAME: The name of the certificate.

KEYFILE: The keystore file.

If you do not remember the name, leave out the "`–alias NAME`" part. The command will then list *all* entries from the keystore (so if your certificate entry is supposed to be the only entry, the result should actually be identical to what you get when you specify the alias – you can also use this to verify that there are indeed no unwanted other entries in the keystore).

The output of this command lists all details of the specified certificate entry (or of all entries, if you do not specify an alias).

For a proper "signed server certificate" entry that is usable by LISTSERV Maestro, the following must apply:

- The entry type must be `PrivateKeyEntry`.
- If your certificate was signed directly by the CA's root certificate (which is actually uncommon), then the entry should have a certificate chain length of 1 and should contain only one certificate: Your server certificate.
- If your certificate was signed with intermediate certificates, then the certificate chain length should be > 1 and the entry should contain all of these certificates: The first certificate should by your server certificate, followed by the first intermediate certificate, and so on.
- For each certificate, the output shows the certificate fingerprint. If in doubt, you can use this to validate the authenticity of the certificates, by comparing the fingerprint from the output to the official fingerprint of the certificate in question (which you would have to obtain from a reliable and secure separate source).

5.3 Make LISTSERV Maestro aware of the Server Certificate

Once you have the signed server certificate in your keystore file, you now need to make the LISTSERV Maestro server aware of this certificate, as a last step of securing your server.

You can either secure all bound IP addresses with HTTPS, or only some of them. Each option is described in its own sub-section below.

In both cases, you need to edit the file "tomcat.ini" on the server that you want to secure with HTTPS:

```
[maestro_install_folder]/conf/tomcat.ini
```

5.3.1 Secure All IP Addresses with HTTPS

To enable HTTPS on all bound IP addresses, you need to add the following three (or four) entries to the "tomcat.ini".

Three required entries:

```
SecureServer=true
KeystoreFile=KEYSTORE_PATH
KeystorePassword=PASSWORD
```

And one optional entry:

```
CertificateAlias=ALIAS_INFO
```

Specifying these entries has the effect that HTTPS is enabled on all addresses that LISTSERV Maestro binds to (=all addresses are secured), using the certificate(s) from the specified keystore.

Reminder: You can configure which addresses LISTSERV Maestro shall bind to (either all addresses or only a subset) via the "BindAddress" entry. See "LMA Admin Tech Doc 6 – IP Addresses and Ports" for details.

In the entries above, make the following replacements:

KEYSTORE_PATH: Replace with the absolute path to the keystore file (on Windows including drive letter) in which the signed certificate can be found. You cannot use a relative path name but must supply the full path to the file.

You can store the keystore file itself in any place that seems appropriate, but the "[maestro_install_folder]/conf" folder seems like a good choice.

PASSWORD: Replace with the password that you used for the keystore (as explained earlier, you must have used the same password for the certificate itself too).

Regarding the optional "CertificateAlias" entry: If not present, LISTSERV Maestro automatically uses the first applicable certificate that it finds in the specified keystore. If you have several applicable certificates in the keystore, for example old expired versions and a new current version of the same certificate, and you want to make sure that LISTSERV Maestro picks the correct certificate, you can specify the certificate that shall be used by including the "CertificateAlias" entry and replacing the "ALIAS_INFO" part as follows:

ALIAS_INFO: If no "BindAddress" entry is specified, i.e. if LISTSERV Maestro currently binds to all addresses (all of which are now to be secured), then "ALIAS_INFO" must be replaced with a single certificate alias. The certificate from the keystore that has this alias will be used for all HTTPS connections on all bound addresses. Make sure that this certificate contains all host names that are assigned to all IP addresses.

If the "BindAddress" entry is actually specified, i.e. if LISTSERV Maestro currently binds only to the specified list of addresses (all of which are now to be secured), then "ALIAS_INFO" must be replaced with a comma separated list of certificate aliases in the format specified below. For each address, the certificate from the keystore that has the specified alias will be used for all

HTTP connections on that address. Make sure that each certificate contains all host names that are assigned to the IP address that the certificate is used for.

The replacement for the "`ALIAS_INFO`" must be specified as a comma-separated list of IP-address/alias pairs. In each pair, the IP address comes first and the alias second, separated by a "`#`" (and the alias name must not contain a comma):

```
IP_ADDRESS_1#ALIAS_1,IP_ADDRESS_2#ALIAS2,IP_ADDRESS_3#ALIAS_3,...
```

For each IP address that is listed in the "`BindAddress`" entry, LISTSERV Maestro looks up the same address in the "`CertificateAlias`" entry. If an IP-address/alias pair is found for a given address, then the alias from this pair defines the certificate that is used for this IP address. If an address is not found in the list, then for this address, LISTSERV Maestro simply picks the first applicable certificate that it finds in the keystore. In order that this matching between the IP addresses in the "`BindAddress`" and "`CertificateAlias`" entries works correctly, make sure to type the addresses in exactly the same way, in both entries.

5.3.2 Secure Only Some IP Addresses with HTTPS

To enable HTTPS on only some of the bound IP addresses, you need to add the following entries to the "`tomcat.ini`".

Single required entry:

`SecureServer=`*`SECURED_ADDRESSES`*

Two required entries *for each* secured IP address:

`KeystoreFile-`*`IP_ADDRESS`*`=`*`KEYSTORE_PATH`*

`KeystorePassword-`*`IP_ADDRESS`*`=`*`PASSWORD`*

One optional entry *for each* secured IP address:

`CertificateAlias-`*`IP_ADDRESS`*`=`*`ALIAS`*

Note, that the first entry "`SecureServer`" must appear only once, with the following replacement:

`SECURED_ADDRESSES`: Replace with a comma-separated list of all IP addresses that are to be secured, for example:

```
192.168.0.15,192.168.0.16,192.168.0.43
```

You can specify both IPv4 and IPv6 addresses (also mixed), but IPv6 addresses must be enclosed in brackets "`[`" and "`]`". Each address can also specify an individual port. An IPv4 address with port is specified as `address:port`, an IPv6 address with port is specified as `[address]:port`.

Of all the addresses that LISTSERV Maestro binds to, all addresses that appear in this "`SecureServer`" entry are secured, all other addresses remain unsecure. If the "`SecureServer`" entry contains an address that LISTSERV Maestro does not actually bind to, then this address is ignored. Or in other words: If an address is not bound at all, it does not make sense to secure it either.

Reminder: You can configure which addresses LISTSERV Maestro shall bind to (either all addresses or only a subset) via the "`BindAddress`" entry. See "Admin Tech Doc 6 – IP Addresses and Ports" for details.

The other two (or three) entries must appear once for each IP address that is listed in the "SECURED_ADDRESSES" list (see above), with the following replacement:

IP_ADDRESS: This replacement happens in the key of all two (or three) entries. This tells LISTSERV Maestro which IP address the entry references. Replace this part with the IP address in question (must be one of the addresses from the "SECURED_ADDRESSES" list). Can be an IPv4 or IPv6 address. IPv6 addresses must *not* be enclosed in brackets.

KEYSTORE_PATH: Replace with the absolute path to the keystore file (on Windows including drive letter) in which the signed certificate for this IP address can be found. You cannot use a relative path name but must supply the full path to the file.

You can store the keystore file itself in any place that seems appropriate, but the "[maestro_install_folder]/conf" folder seems like a good choice.

PASSWORD: Replace with the password that you used for the keystore (as explained earlier, you must have used the same password for the certificate itself too).

Regarding the optional "CertificateAlias-*IP_ADDRESS*" entry: If not present for a secured IP address, LISTSERV Maestro automatically uses the first applicable certificate for this IP address that it finds in the specified keystore. If you have several applicable certificates in the keystore, for example old expired versions and a new current version of the same certificate, and you want to make sure that LISTSERV Maestro picks the correct certificate, you can specify the certificate that shall be used by including the "CertificateAlias-*IP_ADDRESS*" entry and replacing the "ALIAS" part as follows:

ALIAS: Replace with the alias of the certificate that shall be used for all HTTPS connections on the given IP address. Make sure that this certificate contains all host names that are assigned to the IP address.

5.3.3 Define the HTTPS Port

The HTTPS port that is used for a secured IP address is defined in the following hierarchical order:

1. Any address that is listed in the "SecureServer" entry together with its own specific port uses this port as its HTTPS port. For all other addresses, the next rule applies:
2. Any address that does not yet have its HTTPS port defined by the previous rule and that is listed in the "BindAddress" entry together with its own specific port, uses this port as its HTTPS port. For all other addresses, the next rule applies:
3. Any address that does not yet have its HTTPS port defined by the previous rules, uses the port that is defined by the "SecurePort" entry, unless that entry is not present, in which case the default port 443 is used.

5.3.4 Configure the Access URLs

Once all configuration entries are specified in the "tomcat.ini", LISTSERV Maestro is prepared for HTTPS access. Start (or restart) LISTSERV Maestro to make the changes effective.

You can access LISTSERV Maestro normally, only now you need to use "**https://**" URLs instead of the standard "http://", if the access URL uses a host name that is assigned to one of the secured IP addresses.

IMPORTANT: After HTTPS access is enabled, you will also have to adjust the LISTSERV Maestro access URLs in the HUB, so that these URLs now specify the "https://" protocol, and no longer the default "http://" protocol.

If you do not do this, then if you use the new "https://" protocol to access the LUI login page, even though the login page is displayed fine, you will not be able to login (neither with the admin account nor with any of the user accounts).

To change the access URLs, login to the HUB using the non-standard login URL "`https://YOURSERVER/hub/?loginOverride`" and login with the admin account. Then change all access URLs that are now affected by HTTPS.

This can apply to any of the following: "LUI Access URL for Users", "HUB Access URL for Users", "LUI Access URL for Admin", "HUB Access URL for Admin", "URL for Recipients, Subscribers and Others", and even to the "Tracking URL", depending on which of the components you have secured with HTTPS and which not.

The defaults for these URLs are defined on the **Default URL Settings** page (menu "Global Settings" → "Maestro User Interface" → "Default URL Settings") but these defaults can also be overridden on account or group level. So if after the change to HTTPS a given account is no longer able to login, check the access URL settings of that account to make sure that HTTPS is specified in all applicable access URLs.

5.3.5 Redirect from HTTP to HTTPs

No matter if you secure all addresses that LISTSERV Maestro binds to (with "`SecureServer=true`") or only some of the addresses (by specifying a list of addresses in the "`SecureServer`" entry), the result is always, that on *all* secured addresses, LISTSERV Maestro will now only accept access with the secure HTTPS protocol, on the specified secure HTTPS port(s).

This means that if a user mistakenly tries to access LISTSERV Maestro with the normal HTTP protocol in the URL, then this user will not be able to make a connection.

This may be the desired behavior, but in many situations this is not desired, because users are prone to forget the additional "s" in the protocol when they enter the access URL. And then they get confused that they are unable to access LISTSERV Maestro, as they do not realize their small mistake. This is true especially in situations where LISTSERV Maestro *used* to be accessed with the normal HTTP protocol and was only recently switched to the secure HTTPS protocol, and now all old HTTP bookmarks that the users may still have suddenly cease to function.

To avoid this kind of confusion, you can configure LISTERV Maestro to also accept normal HTTP requests, on the normal HTTP port(s), and automatically redirect these requests to use the HTTPS protocol instead.

You do this by adding the following entry to the tomcat.ini file:

```
HttpToHttpsAutoRedirect=IP_ADDRESS_LIST
```

where `IP_ADDRESS_LIST` must be replaced with a comma separated list of IP addresses. You can specify both IPv4 and/or IPv6 addresses. Each IP address can optionally also specify a port. An IPv4 address with port is specified as `address:port`, an IPv6 address with port is specified as `[address]:port`. If no port is specified for an address, the default HTTP port as defined by the "`Port`" entry will be assumed.

LISTSERV Maestro then accepts normal HTTP connections on each address/port combination in this list but redirects each HTTP request that it receives on them and converts it into a HTTPS request that goes to the same IP address but with the secure HTTPS port that was configured for this address.

In this list of addresses, you can also use the keyword "`all`" instead of an IP address, either with or without a specific port (the latter separated with a colon "`:`"). This has the effect that LISTSERV Maestro accepts normal HTTP connections on *all* secured addresses (on the given port). I.e. the keyword is a shortcut for spelling out all secured addresses explicitly.

Examples:

1. `HttpToHttpsAutoRedirect=all`

   In addition to accepting secure HTTPS requests, LISTSERV Maestro will now also accept normal HTTP requests on *all* secured IP addresses, using the default HTTP port as configured by the "`Port`" entry, and will redirect these HTTP requests to HTTPS.

2. `HttpToHttpsAutoRedirect=all,all:8080`

   Same as before, but HTTP requests on all secured IP addresses will now be accepted *both* on the default HTTP port as configured by the "`Port`" entry *and* on the specific port "`8080`".

3. `HttpToHttpsAutoRedirect=192.168.1.1,192.168.15.3`

   Same as the first example, but HTTP requests (on the default HTTP port) will not be accepted on all secured IP addresses, but only on the two addresses specified.

4. `HttpToHttpsAutoRedirect=192.168.1.1,192.168.1.1:8080,192.168.15.3`

   Same as the previous example, but now the IP address `192.168.1.1` will accept HTTP requests both on the default HTTP port and the explicitly specified port `8080`.

5. `HttpToHttpsAutoRedirect=all,192.168.1.1:8081,192.168.15.3:8082`

   HTTP request are now accepted on all secured IP addresses with the default HTTP port and on address `192.168.1.1` additionally with port `8081` and on address `192.168.15.3` additionally with port `8082`.

Note: If you specify an IP address that is not actually secured with HTTPS, or if you specify an address/port combination that is not bound at all, then this address (or address/port combination) is ignored. Similarly, if you specify an address with a port where the same port is already in use as the secure HTTPS port on this IP address, then this address/port combination is also ignored.

5.4 Refreshing an Expired Certificate

When you purchase the signed certificate from your CA, this certificate will be valid for a limited period only (usually a year, or several years). Once the certificate expires, users will no longer be able to access LISTSERV Maestro, as the users' browsers will reject the connection because of the expired certificate.

It is therefore important that you keep the expiration date of your certificate in mind and install a freshly signed new version of the certificate well in time before the old certificate expires.

Note: This is one of the reasons why it is recommended to let LISTSERV Maestro manage the certificates automatically (see section "4 Let LISTSERV Maestro Manage the Certificates"). Because then LISTSERV Maestro will refresh the certificate for you, automatically.

To get a fresh certificate, you will need to send a new certificate signing request (CSR) to your CA and then install the freshly signed version of the certificate in LISTSERV Maestro.

For this you have two options: Create a new certificate and submit a new CSR for this new certificate, or re-use the existing certificate, and simply submit a new CSR for it. Both options are described in more detail in the following sub-sections.

5.4.1 Creating a New Certificate

To create a new certificate that is supposed to take the place of the soon-to-be-expired old certificate, you need to go through the same procedure as you did when you created the original

certificate: First create a new unsigned certificate, then create a CSR for this certificate and submit it to your CA, then import the signed certificate that you get back from the CA.

For this, proceed as described in these sections:

1. See section “5.2.1.1 Create an Unsigned Server Certificate”

   Important: Either create the new certificate in a new keystore file, or create it in the same keystore file, but with a different alias than the existing certificate.

2. See section “5.2.1.2 Perform a Certificate Signing Request (CSR)”

   Important: Remember to specify the correct keystore file and alias for the new certificate!

3. See section “5.2.1.3 Install the Signed Server Certificate”

   Important: Again, remember to specify the correct keystore file and alias for the new certificate!

Once you have installed the new signed certificate in your keystore file, you will need to make some changes to the `tomcat.ini` to make LISTSERV Maestro aware of the new certificate:

- If you created the new certificate in a new keystore file, then it is important that you adjust the entries in the `tomcat.ini` to point to this new keystore file, with the correct password for that file.
- If you created the new certificate in the same keystore file, but with a new alias, then it is important that you adjust the `CertificateAlias` entry in the `tomcat.ini` (or add such an entry if it does not already exist), so that it points to the correct new alias.

See section “5.3 Make LISTSERV Maestro aware of the Server Certificate” for more details about the `tomcat.ini`.

After you have adjusted the `tomcat.ini`, restart LISTSERV Maestro to make it aware of the changes.

5.4.2 Re-using the Existing Certificate

Security Issue: By re-using the existing certificate, you also re-use the certificate’s private key. The main benefit of having a certificate with limited duration is to reduce the damage if your private key is leaked. If you switch to a new private key every now and then (i.e. when the certificate expires), then the old private key will become useless to any attackers who somehow got hold of it in the past, i.e. any further attack attempts with this old private key will no longer work.

It is therefore often recommended, **not** to re-use an existing certificate when it expires, but instead create a new one to take its place, as described in section “5.4.1 Creating a New Certificate” above.

In certain cases it may however be fine to re-use a certificate (maybe for a limited time) and this sub-section describes how to do this in LISTSERV Maestro.

To re-use an existing certificate, you simply submit a CSR for this certificate to your CA:

- Some CAs store the old CSRs and can re-use them to freshly sign the certificate. If that is the case, then you do not even need to submit the CSR to the CA, but you can simply ask the CA to use the CSR they already have to freshly sign the certificate.
- If you need to submit a new CSR to the CA but you yourself still have a copy of the CSR file that you submitted last time, then you can usually simply submit this same CSR again to get the freshly signed certificate.

- If you no longer have the CSR or need a new CSR, create one for the existing certificate (in the existing keystore) as described in section "5.2.1.2 Perform a Certificate Signing Request (CSR)".

Once you get the freshly signed certificate back from your CA, you need to install it in LISTSERV Maestro as described in section "5.2.1.3 Install the Signed Server Certificate".

When doing so, use the existing keystore file (make a backup copy of the file before changing it), and supply the same certificate alias as for the old version of the certificate, to overwrite it in the keystore.

Then restart LISTSERV Maestro to make it aware of the updated certificate.

6 Support for HTTP Strict Transport Security (HSTS)

It is outside of the scope of this document to describe what HSTS is and how it works. Please inform yourself via other sources about HSTS. This section is only meant to describe *how* to enable HSTS in LISTSERV Maestro, in case that you decide that you want to use this feature.

Some things to keep in mind:

- **Be sure that you understand HSTS and know what you are doing before enabling HSTS!**
- In LISTSERV Maestro, HSTS can only be enabled for the whole LISTSERV Maestro server.

  That means that it is not possible to enable HSTS only for some host names of the server and not for others. Once it is enabled, it applies to *all* HTTPS connections that are served by this LISTSERV Maestro instance.

  Consequently that also means, that if all LISTSERV Maestro components are installed on the same server, then HSTS is either enabled or disabled for *all* of these components (and all their host names). But if the LISTSERV Maestro components are distributed over several servers, then HSTS must be enabled/disabled individually for each of these servers.

To enable HSTS for a LISTSERV Maestro instance, include the following setting in the `tomcat.ini`:

```
HSTSEnabled=true
```

If this setting is not present in the `tomcat.ini` (or has a value other than "`true`"), then HSTS is disabled. With HSTS enabled, the following optional `tomcat.ini` settings can also be used (they are ignored if HSTS is disabled):

```
HSTSMaxAge=MAX_AGE_IN_SECONDS
HSTSIncludeSubDomains=true
HSTSPreload=true
```

where you need to replace *`MAX_AGE_IN_SECONDS`* with the desired max age, expressed as the number of seconds. The default is 31536000, i.e. 1 year. For the other two optional settings, the default is "`false`", i.e. if you do not specify one of these settings, or set it to a value other than "`true`", then the corresponding option will not be enabled in the HSTS header.

As usual with the `tomcat.ini`, any changes to these settings only become effective once you restart the corresponding LISTSERV Maestro instance.

7 Supported SSL/TLS Ciphers

The `SSLCiphers` entry in the `tomcat.ini` file defines the list of ciphers that the LISTSERV Maestro server supports for HTTPS connections. Connecting clients must use one of these ciphers to be able to connect via HTTPS.

The value for this entry is a comma separated list that can contain the alias `DefaultStandard` and/or the alias `DefaultStrong` and/or any of the individual cipher names as listed in the two tables below, in any combination. I.e. the two aliases can also be combined with each other or with any of the cipher names.

The alias `DefaultStandard` stands for all ciphers that are shown as "included in alias `DefaultStandard`" in the first table below. Similarly, the alias `DefaultStrong` stands for all ciphers that are shown as "included in alias `DefaultStrong`" in the second table below.

If left out, the `SSLCiphers` entry defaults to `SSLCiphers=DefaultStrong,DefaultStandard`.

The following low and standard strength ciphers (up to 168-bit) are supported by LISTSERV Maestro:

Cipher Name:	Included in alias `DefaultStandard`
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	Yes
SSL_DHE_DSS_WITH_DES_CBC_SHA	
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	
SSL_DHE_RSA_WITH_DES_CBC_SHA	
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	
SSL_DH_anon_WITH_DES_CBC_SHA	
SSL_DH_anon_WITH_RC4_128_MD5	
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	
SSL_RSA_EXPORT_WITH_RC4_40_MD5	
SSL_RSA_WITH_3DES_EDE_CBC_SHA	Yes
SSL_RSA_WITH_DES_CBC_SHA	
SSL_RSA_WITH_NULL_MD5	
SSL_RSA_WITH_NULL_SHA	
SSL_RSA_WITH_RC4_128_MD5	
SSL_RSA_WITH_RC4_128_SHA	
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	Yes
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	Yes
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	Yes
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	
TLS_DH_anon_WITH_AES_128_CBC_SHA	
TLS_DH_anon_WITH_AES_128_CBC_SHA256	
TLS_DH_anon_WITH_AES_128_GCM_SHA256	
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Yes
TLS_ECDHE_ECDSA_WITH_NULL_SHA	
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	Yes

TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Yes
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Yes
TLS_ECDHE_RSA_WITH_NULL_SHA	
TLS_ECDHE_RSA_WITH_RC4_128_SHA	
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA	Yes
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	Yes
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	Yes
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	Yes
TLS_ECDH_ECDSA_WITH_NULL_SHA	
TLS_ECDH_ECDSA_WITH_RC4_128_SHA	
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA	Yes
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	Yes
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	Yes
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	Yes
TLS_ECDH_RSA_WITH_NULL_SHA	
TLS_ECDH_RSA_WITH_RC4_128_SHA	
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA	
TLS_ECDH_anon_WITH_AES_128_CBC_SHA	
TLS_ECDH_anon_WITH_NULL_SHA	
TLS_ECDH_anon_WITH_RC4_128_SHA	
TLS_EMPTY_RENEGOTIATION_INFO_SCSV	
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5	
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA	
TLS_KRB5_EXPORT_WITH_RC4_40_MD5	
TLS_KRB5_EXPORT_WITH_RC4_40_SHA	
TLS_KRB5_WITH_3DES_EDE_CBC_MD5	
TLS_KRB5_WITH_3DES_EDE_CBC_SHA	
TLS_KRB5_WITH_DES_CBC_MD5	
TLS_KRB5_WITH_DES_CBC_SHA	
TLS_KRB5_WITH_RC4_128_MD5	
TLS_KRB5_WITH_RC4_128_SHA	
TLS_RSA_WITH_AES_128_CBC_SHA	Yes
TLS_RSA_WITH_AES_128_CBC_SHA256	Yes
TLS_RSA_WITH_AES_128_GCM_SHA256	Yes
TLS_RSA_WITH_NULL_SHA256	

In addition to the above “weaker” ciphers, the following stronger 256-bit encryption ciphers are also supported by LISTSERV Maestro.

Note: These strong ciphers are appropriate for most countries, but some countries restrict the usage of strong encryption. Make sure your country allows for strong encryption before using these ciphers. Remember that these ciphers are enabled by default if the `SSLCiphers` entry is not present in `tomcat.ini`. So if these ciphers are not allowed in your country, make sure to provide an `SSLCiphers` entry that does not include them.

Cipher Name:	Included in alias `DefaultStrong`
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	Yes
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	Yes
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	Yes
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	
TLS_DH_anon_WITH_AES_256_CBC_SHA	
TLS_DH_anon_WITH_AES_256_CBC_SHA256	
TLS_DH_anon_WITH_AES_256_GCM_SHA384	
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	Yes
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Yes
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	Yes
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	Yes
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	Yes
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	Yes
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	Yes
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	Yes
TLS_ECDH_anon_WITH_AES_256_CBC_SHA	
TLS_RSA_WITH_AES_256_CBC_SHA	Yes
TLS_RSA_WITH_AES_256_CBC_SHA256	Yes
TLS_RSA_WITH_AES_256_GCM_SHA384	Yes

8 Minimum TLS Protocol Version

The `SSLMinimumTLSVersion` entry in the `tomcat.ini` file defines the lowest TLS (or SSL) protocol version that the LISTSERV Maestro server supports for HTTPS connections. Connecting clients must use at least the specified protocol version, or any of the higher (later) supported versions, to be able to connect via HTTPS.

The lowest supported protocol version is "SSL version 3", the highest supported version is "TLS version 1.2", so the following values are allowed for the `SSLMinimumTLSVersion` entry (in ascending version order):

```
SSLv3
TLSv1
TLSv1.1
TLSv1.2
```

It should normally not be necessary to support the outdated `SSLv3` protocol, because all browsers that are not capable of at least `TLSv1` are not compatible with LISTSERV Maestro anyway. So unless you have a very good reason, you should not set this entry to `SSLv3`.

As of May 2014, there are still quite a few browsers in common use that do not yet support the newest `TLSv1.2` (or even `TLSv1.1`) out of the box, so currently it is advisable to set the minimum to `TLSv1`, unless you know that all your users are using a modern browser that is capable of `TLSv1.2`. In such a case, you can increase security by setting the minimum to `TLSv1.2`, to keep out all browsers that do not support it.

If left out, the `SSLMinimumTLSVersion` entry defaults to `SSLMinimumTLSVersion=TLSv1`.