Encryption in SAS® Viya® 3.3: Data in Motion

Overview

Encryption Coverage

SAS Viya provides encryption in two contexts:

- Data in motion is data that is being transmitted to another location. Data is most vulnerable while in transit. Sensitive data in transit should be encrypted. You can protect all traffic in transit between servers and clients. This document covers encrypting data in motion.

  Note: This document does not cover encryption for data in motion for Cloud Foundry. See *SAS Viya for Cloud Foundry: Operations* for encryption information relevant to Cloud Foundry.

- Data at rest is data stored in databases, file servers, endpoint devices, and various storage networks. This data can be on-premises, virtual, or in the cloud. This data is usually protected in conventional ways by firewalls. Numerous layers of defense are needed, and encrypting sensitive data is another layer. See *Encryption in SAS Viya: Data at Rest*.

Encryption in a SAS Viya Deployment

The SAS Viya deployment process provides a default level of encryption for data in motion. SAS Viya is deployed with Transport Layer Security (TLS) to secure network connections and is fully compliant with SAS security standards.

Data at rest can be secured with additional encryption options; by default, data at rest is presumed to be behind the firewall and is not encrypted.

In a full deployment of SAS Viya, all external communication paths are secured by default. In particular, the SAS Viya deployment provides the following default security and additional ways to increase the level of security.

- The default configuration of a full SAS Viya deployment assumes an Apache HTTP server that can be configured as a reverse proxy server. On the reverse proxy server, the module called mod_ssl provides a default level of TLS support. This module relies on OpenSSL to provide the cryptography engine. These settings are reasonably secure, but they are not compliant with SAS security standards. SAS recommends that you replace the default self-signed certificates with your own custom certificates and strengthen the default cryptography.
- SAS Viya uses HashiCorp Vault to generate and sign root and intermediate certificates used to secure communication between various SAS Viya processes. Vault provides a point of contact for SAS Viya services requiring certificates needed to maintain secured communication.

- SAS Viya supports TLS encryption between the data provider (Hadoop, Teradata) and the CAS server, and you can take steps to enable that encryption. If you are using a SAS Data Connect Accelerator, the data that is transferred between the data provider and the CAS server is not encrypted by default.

Security certificates are required to use TLS. SAS Viya provides default certificates at deployment that can be replaced with custom certificates that comply with the security policies at your enterprise.

- The Apache HTTP Server is installed with a localhost certificate and key that allow HTTPS access to SAS Studio, CAS Server Monitor, Visual Analytics, SAS Environment Manager, and SAS Theme Designer (depending on your order). However, SAS recommends that you replace these certificates with your own custom certificates pre-deployment to increase the level of security.
- Certificates used for connecting to CAS are signed by a root CA and intermediate certificate generated by Vault. These certificates are part of the CA bundle of trusted certificates (trustedcerts) provided by default. These certificates can be replaced with your own custom certificates post deployment.
- Ansible utilities are provided to easily update certificates and distribute them to the truststores in the deployment.

In a SAS Viya programming-only deployment, the basic framework for security is included by default, but is not enabled by default. In particular, the SAS Viya deployment provides the following default framework to secure data-in-motion.

- On the Apache HTTP Server (reverse proxy server) , the module called mod_ssl provides TLS support. This module relies on OpenSSL to provide the cryptography engine. In addition, SAS Viya includes customizations to support SAS internal standards for developing software that protects data-in-motion.
- The Apache HTTP Server (reverse proxy server), has a localhost certificate and key that allows HTTPS access to SAS Studio and CAS Server Monitor.
- Each machine in the deployment has a Mozilla bundle of trustedcerts CA certificates in the SASSecurityCertificateFramework that is used by SAS and Java processes if CAS Client TLS is turned on.
- The SASSecurityCertificateFramework also generates encrypted self-signed certificates for a CAS controller machine in the deployment that can be used to turn on CAS Client TLS. These self-signed certificates are part of the CA bundle of trusted certificates (trustedcerts).
- Customers can encrypt communication between the CAS controller machine in the deployment and implement TLS communications with CAS Clients. This requires generating and registering additional security certificates for the CAS controller machine.

Note: All discussion of TLS is also applicable to the predecessor protocol, Secure Sockets Layer (SSL).

Terminology

Various security strategies are used to maintain data usability and data confidentiality, as well as to validate the integrity of content. Various encryption, hashing, and encoding algorithms are used by

SAS to protect your data in motion and data at rest. SAS highly recommends using TLS for protecting data and credentials (user IDs and passwords) that are exchanged in a networked environment.

encoding
: Encoding transforms data into another format using a scheme that is publicly available so that it can easily be reversed. It does not require a key. The only thing required to decode it is the algorithm that was used to encode it. PROC PWENCODE, for example, encodes passwords.

encryption
: Encryption is a process of protecting data and credentials. Encryption transforms data into another format in such a way that only specific individuals can reverse the transformation. It uses a key that is kept secret, in conjunction with the plaintext and the algorithm, in order to perform the encryption operation. As such, the ciphertext, algorithm, and key are all required to return to the plaintext. Example encryption algorithms are AES and RSA. TLS is an encryption technology.

hashing
: Hashes are commonly used to store passwords to prevent them from being viewed. Hash algorithms are one-way functions. They turn any amount of data into a fixed-length "fingerprint" that cannot be reversed. If the input changes by even a tiny bit, the resulting hash is completely different. When passwords are hashed, only the hash is kept. To verify a password, you hash the password and check to see whether the password matches the stored hash. SHA-256 is a hashing algorithm.

salting
: Salt is data used as an additional input to the algorithm that encrypts data. The salt is randomly generated and is used to increase the difficulty of brute-force decryption attacks on the data.

How To

Configure TLS and HTTPS

Secure Apache HTTP Server

Overview

SAS Viya uses an Apache HTTP server to act as a reverse proxy server to secure your environment. At deployment, the Ansible playbook can install Apache httpd with mod_ssl automatically. This option uses default Apache security settings and self-signed certificates. These settings are reasonably secure, but they are not compliant with SAS security standards. SAS recommends that you replace the default self-signed certificates with your own custom certificates and strengthen the default cryptography.

By default, HTTPS access to SASHome is enabled in a SAS Viya full deployment. The URL to access SASHome after installing Apache httpd and installing SAS Viya is https://*reverse-proxy-server*/SASHome/.

Note: In a programming-only deployment, there is no SASHome. SAS Viya end users must use HTTP to connect to SAS Studio or CAS Server Monitor because the Apache HTTP Server does not support HTTPS.

SAS recommends strengthening the default cryptography using the sas-ssl.conf file. The mod_ssl module relies on OpenSSL to provide strong cryptography for the Apache server using TLS cryptographic protocols. You can read more about mod_ssl at User Manual for mod_ssl.

SAS also recommends that you replace the default certificates with your own custom certificates that comply with the security policies at your enterprise. These can be replaced pre-deployment or post-deployment of SAS Viya. SAS recommends replacing the Apache httpd certificates before deploying SAS Viya. Whether you replace the certificates pre-deployment or post-deployment, SAS recommends replacing the certificates before giving end users access to SAS Viya.

You can strengthen security on the Apache HTTP Server by performing the tasks in this section. These tasks can be performed at any time after your initial deployment. The task for replacing your certificates pre-deployment is the exception.

Block Port 80

When you block Port 80, the port is blocked internally and externally. Port 443 is now used for external communications. Refer to the Red Hat Enterprise Linux Reference and Security Guides for information about best practices for securing ports.

See "Enable Required Ports" in *SAS Viya for Linux: Deployment Guide* for more information.

To direct all access to use HTTPS and not HTTP, you can redirect port 80 to port 443 on the Apache HTTP server as follows.

1 Edit the ssl.conf file located in **/etc/httpd/conf.d/** directory.

2 Locate the *<VirtualHost>* code block for TLS. Before that block of code, add the following line of code:

Note: There needs to be a space after the / and before `https://<machine_name_where_HTTP_Server_installed>/` in the following code fragment.

```
<VirtualHost *:80>
ServerName <machine_name_where_HTTP_Server_installed>
Redirect / https://<machine_name_where_HTTP_Server_installed>/
</VirtualHost>
```

3 Restart the Apache HTTP Server by entering the following command:

```
sudo service httpd restart
```

Disable Consul Port (full deployment)

To disable Consul port 8500, set the following in vars.yml file:

```
DISABLE_CONSUL_HTTP_PORT: true
```

Note: See "Modify the vars.yml File" in *SAS Viya for Linux: Deployment Guide* for more details.

Replace Self-Signed Certificates with Custom Certificates (Pre-Deployment)

The SAS Viya deployment can install Apache httpd with mod_ssl and self-signed certificates. These settings are reasonably secure, but they are not compliant with SAS security standards. SAS recommends replacing these self-signed certificates with custom certificates that comply with the security policies at your enterprise.

Note: SAS recommends that you install Apache httpd and replace the self-signed certificates before you start the deployment process. When you perform this task before installing SAS Viya, the Ansible playbook used to deploy SAS Viya distributes your custom certificates and adds them to the truststore. This process avoids the brief outage necessary to replace the certificates after SAS Viya has been deployed.

For information about deploying Apache httpd see "Security Requirements" in *SAS Viya for Linux: Deployment Guide*

During the deployment, the playbook inspects any existing certificates and the CA chain to determine whether they comply with SAS security requirements.

- If compliant certificates are found (the custom certificates), the certificates are not changed.
- If certificates that do not meet SAS security standards are found, the playbook generates a SAS provided self-signed certificate and configures mod_ssl to use it.

If you do not add compliant certificates and instead keep the default security settings and certificates, end-users will see a standard web browser warning message. SAS recommends replacing the default certificates before giving end-users access to SAS Viya. Adding your own certificates post-deployment requires a brief outage. See "Replace Self-Signed Certificates with Custom Certificates (Post-Deployment)" on page 7.

The certificate filenames and locations of the certificates are set in the ssl.conf file at **`/etc/httpd/conf.d/ssl.conf`**.

- This is the server identity certificate. `SSLCertificateFile /etc/pki/tls/certs/localhost.crt`

  Note: localhost.crt is the name of the default Apache certificate.

- RSA private key associated with certificate file server.crt
  `SSLCertificateKeyFile /etc/pki/tls/private/localhost.key`

  Note: localhost.key is the name of the default Apache key file.

- The file that contains the chain of trust. SAS recommends that this file contain the Root CA and all intermediate certificates. `SSLCertificateChainFile /etc/pki/tls/certs/custom-chain.crt`

1 Install httpd and mod_ssl on the desired machines.

```
sudo yum install -y httpd mod_ssl
```

2 Download your server identity certificate files.

3 Copy your new certificate file to **/etc/pki/tls/certs**. If you have both a root certificate file and a chain file that includes the root certificate and intermediate certificates, you only need to copy the chain file to this location.

Note: The certificate file needs to be a Base-64 PEM encoded file.

4 Copy your new key file to **/etc/pki/tls/private**.

Note: The key file needs to be a Base-64 PEM encoded file.

5 Change the permissions on your certificate file and your chain file to 644. Change permissions on the key file to 600. Use *chmod* or *sudo* commands to change the permissions.

```
chmod 600 custom.key
chmod 644 custom.crt
chmod 644 custom-chain.crt
```

When you list the files, you see the permissions are Read/Write only for the root account: -rw-r--r-- for the certificate files and -rw------- for the key file.

6 Update the certificate and key file directives in file **/etc/httpd/conf.d/ssl.conf** to point to your new certificates and key.

```
SSLCertificateFile /etc/pki/tls/certs/custom.crt
SSLCertificateKeyFile /etc/pki/tls/private/custom.key
SSLCertificateChainFile /etc/pki/tls/certs/custom-chain.crt
```

7 Update the value of HTTPD_CERT_PATH in vars.yml file to point to the CA root certificate file. If there are also intermediate CA certificates, point to the chain certificate file.

```
HTTPD_CERT_PATH: '/install/sas/sas_viya_playbook/certs/custom-chain.crt'
```

Note: See "Modify the vars.yml File" in *SAS Viya for Linux: Deployment Guide* for more details.

See "Modify the vars.yml File" in *SAS Viya for Linux: Deployment Guide*for details about setting the HTTPD_CERT_PATH variable.

8 Run the Ansible playbook to install the SAS Viya deployment. See "Installation" in *SAS Viya for Linux: Deployment Guide* for details.

Replace Self-Signed Certificates with Custom Certificates (Post-Deployment)

The SAS Viya deployment can install Apache httpd with mod_ssl and self-signed certificates. These settings are reasonably secure, but they are not compliant with SAS security standards. SAS recommends replacing these self-signed certificates with custom certificates that comply with the security policies at your enterprise.

Note: SAS recommends that you install Apache httpd and replace the self-signed certificates before you start the deployment process. When you perform this task before installing SAS Viya, the Ansible playbook used to deploy SAS Viya distributes your custom certificates and adds them to the

truststore. This process avoids the brief outage necessary to replace the certificates after SAS Viya has been deployed. See "Replace Self-Signed Certificates with Custom Certificates (Pre-Deployment)" on page 6.

During the deployment, the playbook inspects any existing certificates and the CA chain to determine whether they comply with SAS security requirements.

- If compliant certificates are found (custom certificates), the certificates are not changed.
- If no certificates are found or if certificates that do not meet SAS security standards are found, the playbook generates a SAS provided self-signed certificate and configures mod_ssl to use it. These are server identity certificates and can be found at **/etc/pki/tls/certs**.

If you do not add compliant certificates and instead keep the default security settings and self-signed certificates, end-users will see a standard web browser warning message. SAS recommends replacing the self-signed certificates before giving end-users access to SAS Viya. Adding your own certificates post-deployment requires a brief outage.

The certificates and key files that Apache specifies by default are set in the directives in the ssl.conf file at **/etc/httpd/conf.d/ssl.conf**. The SSLCertificateFile and SSLCertificateKeyFile are set by default. If you are replacing the Apache default certificates with your own custom site-signed certificates, you will modify the three directives shown below.

- This is your server identity certificate. `SSLCertificateFile /etc/pki/tls/certs/localhost.crt`.

  Note: Localhost.crt is the name of the certificates provided by Apache.

- RSA private key associated with certificate file `SSLCertificateKeyFile /etc/pki/tls/private/localhost.key`

  Note: Localhost.key is the name of the certificates provided by Apache.

- The file that contains the chain of trust. SAS recommends that this file contain the Root CA and all intermediate certificates. `SSLCertificateChainFile /etc/pki/tls/certs/custom-chain.crt`

To replace the default certificate files, add your custom certificates (site-signed or third-party-signed certificates) to the **/etc/pki/tls** directory structure and point to the new server certificate and key files.

1. Download your server identity certificate files.
2. Copy your new certificate file to **/etc/pki/tls/certs**. If you have both a root certificate file and a chain file that includes the root certificate and intermediate certificates, you only need to copy the chain file to this location.

   Note: The certificate file needs to be a Base-64 PEM encoded file.

3. Copy your new key file to **/etc/pki/tls/private**.

   Note: The key file needs to be a Base-64 PEM encoded file.

4 Change the permissions on the certificate files to 644. Change the permissions on the key file to 600. Use *chmod* or *sudo* commands to change the permissions.

```
chmod 600 custom.key
chmod 644 custom.crt
chmod 644 custom-chain.crt
```

When you list the files, you see the permissions are Read/Write only for the root account: -rw-r--r-- for the certificate files and -rw------- for the key file.

5 Update the certificate and key file directives in file **/etc/httpd/conf.d/ssl.conf** to point to your new certificates and key.

```
SSLCertificateFile /etc/pki/tls/certs/custom.crt
SSLCertificateKeyFile /etc/pki/tls/private/custom.key
SSLCertificateChainFile /etc/pki/tls/certs/custom-chain.crt
```

6 Restart the httpproxy service.

```
sudo sas-viya-httpproxy-default restart
```

7 Restart httpd.

```
sudo service httpd restart
```

8 Update the value of HTTPD_CERT_PATH in vars.yml file to point to the CA root certificate file. If there are also intermediate CA certificates, point to the chain certificate file.

```
HTTPD_CERT_PATH: '/etc/pki/tls/certs/custom_chain.cer'
```

See "Modify the vars.yml File" in *SAS Viya for Linux: Deployment Guide* for details about setting the HTTPD_CERT_PATH variable.Also see "Modify the vars.yml File" in *SAS Viya for Linux: Deployment Guide* for more details.

9 Distribute the certificate chain file to the CA Certificate directory and rebuild the truststores. You can run the following Ansible play to perform this function. On the Ansible controller machine, from the **/viya/sas_viya_playbook** directory, run the distribute-httpd-certs.yml play.

```
ansible-playbook -i inventory.ini utility/distribute-httpd-certs.yml
```

This play adds your new custom certificate to **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts**. The play distributes copies of the certificate chain file to all machines with a name of httpproxy-**inventory name**-ca.crt. The play then rebuilds the trustedcerts.pem and trustedcerts.jks files and includes the CA certificates from custom-chain.cer in the trustedcerts.pem and trustedcerts.jks file on every machine in the deployment.

10 Restart all services on all machines.

```
sas-viya-all-services stop
sas-viya-all-services start
```

Improve TLS Security for the Apache HTTP Server

In the SAS Viya deployment, the Apache HTTP Server is configured with the mod_ssl security module enabled. The mod_ssl module provides strong cryptography for the Apache server using SSL and TLS cryptographic protocols. You can read more about what mod_ssl does at User Manual for mod_ssl.

SAS recommends that you update your Apache HTTP Server to not only use your own custom certificates, but to also upgrade the security protocol and ciphers being used by default with the

ones contained in the sas-ssl.conf file. The TLS protocols and ciphers are recommended by SAS to meet the highest data-in-motion standard for cryptography.

The sas-ssl.conf is typically located at **/opt/sas/viya/config/etc/httpd/conf.d/**. If you do not find the sas-ssl.conf file at that location, create your own sas-ssl.conf file using the following example code:

Note: The following code is shown on more than one line for display purposes only. The SSLCipherSuite variable plus the ciphers must be on one line and must not contain line breaks.

```
Header set Strict-Transport-Security "max-age=31536000"
SSLProtocol TLSv1.2
SSLHonorCipherOrder On
# The line containing variable SSLCipherSuite and values
# must not include line breaks
SSLCipherSuite
ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:
ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:
ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:
ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:
AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:
AES128-SHA256
```

After creating or locating the sas-ssl.conf file, edit the ssl.conf file to include the sas-ssl.conf file.

1 From **/etc/httpd/conf.d/**, edit the ssl.conf file. In the file, locate the `<VirtualHost_default:443>` block of code. Just above that line that code, add the following line of code:

   ```
   Include /opt/sas/viya/config/etc/httpd/conf.d/sas-ssl.conf
   ```

2 Restart the Apache HTTP Server by entering the following command:

   ```
   sudo service httpd restart
   ```

Configure CAS TLS to Use Custom Certificates (full deployment)

Note: The following instructions are for adding custom certificates to a SAS Viya full deployment.

By default, in a full deployment of SAS Viya, Hashicorp Vault issues certificates and keys that are used to secure the deployment. These certificates issued by Vault are provided for each CAS machine and are added to the Mozilla bundle of trusted CA certificates by default.

Table 1 *Security Certificates and Keys Provided for CAS in a SAS Viya Full Deployment*

Security Artifact	Default Certificate and Key Files	Location	Description
trusted CA certificates	trustedcerts.pem	/opt/sas/viya/config/etc/	CA certificates issued by Vault. The trusted list of CA

Security Artifact	Default Certificate and Key Files	Location	Description
	trustedcerts.jks	SASSecurityCertificate Framework/cacerts	certificates includes the Mozilla Bundle of trusted CA certificates, the Root CA certificates issued by Vault, the Apache httpd certificates, and the chain of trust certificates.
Certificate File	sas_encrypted.crt	/opt/sas/viya/config/etc/SASSecurityCertificate Framework/tls/certs/cas/shared/default/sas_encrypted.crt (full deployment)	Vault issued certificates. This file contains the SAS Viya root CA and the Intermediate CA chain.
Private Key File	sas_encrypted.key	/opt/sas/viya/config/etc/SASSecurityCertificate Framework/private/cas/shared/default	Vault issued key file that is encrypted.
Certificate Private Key Passphrase File	*customerName*.key (optional)	/opt/sas/viya/config/etc/SASSecurityCertificate Framework/private/cas/shared/default	It is highly recommended that you protect the sas_encrypted.key file with a passphrase (key). When you create your passphrase and create a file to hold the passphrase, you will need to encrypt this new file also.

You can use your own custom certificates instead of the certificates provided by SAS. Best practices for managing certificates and securing your private keys should be followed.

The following instructions are provided to configure TLS for the CAS Client with your own custom certificates. In a full deployment, the SAS Configuration Server (Consul) handles most configuration tasks.

1 On the main Consul machine (the machine listed within the [consul] host group in inventory.ini file), edit the sitedefault.yml file located at **/opt/sas/viya/config/etc/consul.d/default/** and place the certificate strings for your custom CA certificates (in PEM format) in the file.

```
sudo vi /opt/sas/viya/config/etc/consul.d/default/sitedefault.yml
```

Here is an example of what the sitedefault.yml file might look like after you add your custom CA bundle to this file. First add the identifier (node) named cacerts. Beneath that node, add nodes that identify each of the CA root certificates that are being used. In this example, certificate

identifiers were added named sascaroot, sassh2rootca, and digicertrootca certificates. Another node was also added to include the custom CA root chain of certificates (custom_ca_chain).

Note: Keep the indentation of this file.

```
cacerts:
    sascaroot: |
        -----BEGIN CERTIFICATE-----
        certificate string
        -----END CERTIFICATE-----
    sassha2rootca: |
        -----BEGIN CERTIFICATE-----
         certificate string
        -----END CERTIFICATE-----
    digicertrootca: |
        -----BEGIN CERTIFICATE-----
        certificate string
        -----END CERTIFICATE-----
    custom_ca_chain: |
        -----BEGIN CERTIFICATE-----
        certificate string
        -----END CERTIFICATE-----
        -----BEGIN CERTIFICATE-----
        certificate string
        -----END CERTIFICATE-----
```

2 On the main Consul machine, restart Consul. This act copies the customer CA bundle from the sitedefault.yml file to Consul's key-value (KV) store.

```
sudo service sas-viya-consul-default restart
```

3 Rebuild the truststores. On an Ansible controller machine, run the **./utility/rebuild-trust-stores.yml** Ansible play. This act incorporates the customer CA bundle of trusted certificates (from Consul configuration) into the various truststore files (trustedcerts.pem and trustedcerts.jks) on each machine in the Viya deployment.

```
sudo ansible-playbook -i inventory.ini
./utility/rebuild-trust-stores.yml
```

Note: Use the same admin user that you used during the initial SAS Viya deployment.

For more information, see "Deploy the Software" in *SAS Viya for Linux: Deployment Guide*.

To configure TLS between the CAS client and server:

1 Log on to the CAS controller machine as a user with root or sudo privileges.

2 If you have a CAS session running, cancel it.

3 Place your custom certificate in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs/cas/shared/default**. The certificate file provided by SAS Viya is named sas_encrypted.crt.

Note: Intermediate certificates need to be added to the server identity certificate in a certificate chain. The file needs to include the server identity certificate first, and then the signing intermediate CA certificates in the order in which they were signed. The root CA does not need to be included in this chain file.

Note: Ensure that your files have file system permissions 644: -rw-r--r--. Also, ensure that the file has appropriate file system ownership and permissions for CAS ADMIN user. See "Set Up the CAS Administrator" in *SAS Viya for Linux: Deployment Guide*.

4 Place your private key in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/cas/shared/default**. The certificate file provided by SAS Viya is named sas_encrypted.key.

Note: Ensure that your files have file system permissions 644: -rw-r--r--. Also, ensure that the file has appropriate file system ownership and permissions for CAS ADMIN user.

5 Protect your certificate private key file with a passphrase. In this example, the key file is named custom.key and the encrypted key file is custom_encrypted.key.

a Use the following OpenSSL command to password protect the file.

```
openssl rsa -aes128 -in /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/cas/shared/default/custom.key
-out /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
private/cas/shared/default/custom_encyrpted.key -passout pass:password
```

b Ensure that the customer's encrypted private key file has permissions set to 644. Use chmod to change the permissions:

```
chmod 644 custom_encrypted.key
```

6 Create a customer-supplied certificate private key passphrase file. Use the echo command to create the private key passphrase file. In this example, the private key passphrase filename is customer_encrypted.encryption.key.

```
sudo bash -c "echo -n 'password' > custom_encrypted.encryption.key"
sudo chown cas:sas custom_encrypted.encryption.key
sudo chmod 0640 custom_encrypted.encryption.key
sudo cat custom_encrypted.encryption.key ;
echo password
```

7 You can remove the original custom.key file as you have now secured the custom_encyrpted.key (encrypted key file) and custom_encrypted.encryption.key (passphrase protected encrypted key file) files.

8 Configure CAS to use the customer-supplied certificates and key. Edit casconfig_usermods.lua file located at **/opt/sas/viya/config/etc/cas/default** on the CAS Controller. Change the required CAS_CLIENT_SSL environment variables. Specify the names of your custom certificate and the custom encrypted certificate private key file (custom_encrypted.key), and the customer-supplied certificate private key passphrase file (custom_encrypted.encryption.key).

Note: This file has 600 permissions: -rw-r--r--

```
env.CAS_CLIENT_SSL_REQUIRED=true
env.CAS_CLIENT_SSL_CERT="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/cas/shared/default/custom.crt"
env.CAS_CLIENT_SSL_KEY="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/cas/shared/default/custom_encrypted.key"
env.CAS_CLIENT_SSL_KEYPWLOC = '/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/cas/shared/default/
custom_encrypted.encryption.key'
env.CAS_CLIENT_SSL_KEYPW    = nil
```

When setting the CAS Client environment variables, consider the following information.

Note: See "Modify the vars.yml File" in *SAS Viya for Linux: Deployment Guide* for more details.

- If you are using an intermediate CA certificate, then a certificate chain file needs to be specified for the CAS_CLIENT_SSL_CERT= environment variable. The file needs to include the server identity certificate first, and then the signing intermediate CA certificates in the order in which they were signed. The root CA does not need to be included in this chain file.
- If you are using your own custom certificate and key, you should copy the changes made to CAS_CLIENT_SSL_CERT= and CAS_CLIENT_SSL_KEY= environment variables to the vars.yml file. This change ensures that your settings are not changed when upgrades are made to the deployment.
- If you are setting the CAS_CLIENT_SSL_REQUIRED= environment variable to **true**, you should copy the change made to this environment variable to the vars.yml file. This change ensures that your settings are not changed when upgrades are made to the deployment.

9 On the CAS controller, restart the cascontroller service.

```
sudo service sas-viya-cascontroller-default restart
```

10 For other client-side connections (Lua, Python) you need the root CA certificate on the client specified on Linux.

- The workspace server exports the trustedcerts.pem file by default.
- On Linux, if the root CA is already in the OpenSSL trusted certificate store, Lua or Python clients should work without having to set the CAS_CLIENT_SSL_CA_LIST= environment variable.
- Otherwise, set the CAS_CLIENT_SSL_CA_LIST= environment variable to point to the location of your certificate chain. Root CA certificates at a minimum are needed in the certificate chain.

  ```
  export CAS_CLIENT_SSL_CA_LIST="/opt/sas/viya/config/etc/
  SASSecurityCertificateFramework/cacerts/trustedcerts.pem"
  ```

- For SAS client-side connections, SAS should automatically find the trustedcerts.pem file that is located in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts** either through the workspace server export statement shown previously or the SSLCALISTLOC= system option that is set during installation.

Configure CAS TLS to Use Custom Certificates (programming-only deployment)

Note: The following instructions are for adding custom certificates to a SAS Viya progamming-only deployment.

CAS supports encrypted connections between the server and the clients. Use TLS to secure communications between the server and clients. The certificate used for client server communication needs to be signed by a certificate authority (CA) that is trusted by all potential clients.

The Viya deployment provides certificates and keys at installation that secures the deployment. SAS also adds self-signed certificates created for each CAS machine in the deployment to the Mozilla bundle of trusted certificates.

Table 2 *Security Certificates and Keys Provided for CAS in a SAS Viya Programming-only Deployment*

Security Artifact	Deployment File name	Location	Description
trusted CA certificates	trustedcerts.pem trustedcerts.jks	/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts	Contains the trusted list of CA certificates. The trusted list of CA certificates includes the Mozilla Bundle of trusted CA certificates, the SAS-issued Root CA certificates, the Apache httpd certificates, the chain of trust certificates
Certificate File	sas_encrypted.crt	/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs/sas_encrypted.crt	SAS issued certificates. This file contains the SAS self-signed certificates.
Certificate Private Key File	sas_encrypted.key	/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private	SAS-issued key file. Contains the private key generated by SAS.
Certificate Private Key Passphrase File	*customerName*.key (optional)	/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private	Contains the encrypted passphrase. It is highly recommended that you protect the sas_encrypted.key file with a passphrase (key).

When you create your passphrase and create a file to hold the passphrase, you will need to encrypt this new file also.

You can use your own custom certificates instead of the certificates provided by default by SAS. See "Use Best Practices to Manage Certificates" on page 41.

The following instructions are provided to configure TLS for CAS using your own custom certificates.

Note: If you plan to use the SAS provided self-signed certificates to configure TLS for CAS , see "Configure CAS TLS to Use SAS Default Certificates (programming-only deployment)" on page 18.

You can use your own custom certificates or generate your own custom certificates.

- Generate your own site-signed or third-party-signed certificates. To generate site-signed or third-party-signed certificates in PEM format using OpenSSL, see "Generate Site-Signed or Third-Party-Signed Certificates in PEM Format" on page 45.
- Generate your own self-signed or root CA certificates. To generate self-signed or root CA certificates, see "Generate Self-Signed Certificates" on page 49.
- If your custom root CA is not already included in the trusted CA bundle of certificates, you can add those certificates to the trustedcerts files. For information, see "Add Your Certificates to the Trust List or to a Certificate Chain" on page 42.

To configure TLS between the CAS client and server:

1. Log on to the CAS controller machine as a user with root or sudo privileges.
2. If you have a CAS session running, cancel it.
3. Place your custom certificates and key in the following locations:
   - The server identity certificate chained to any intermediate certificates should be placed in `/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs`.

     Note: Intermediate certificates need to be added to the server identity certificate in a certificate chain. The file needs to include the server identity certificate first, and then the signing intermediate CA certificates in the order in which they were signed. The root CA does not need to be included in this chain file.

     The private server key is placed in `/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private`.
   - If your CA certificate is not already included in the Mozilla bundle of CA certificates, append the root certificate to the trustedcerts files in the `/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/` directory.
     - To add the root certificate to the trustedcerts.pem file, just include the root certificate at the end of the trustedcerts.pem file.

- To add the root certificate to the trustedcerts.jks file, you need to import the file using a keytool command.

See "Add Certificates to the Trustedcerts File" on page 42 for information about adding your certificates to the truststore.

Note: Do not delete the trustedcerts files.

Note: Add your root certificate to the trustedcerts.pem and trustedcerts.jks files on every machine in the deployment.

4 Protect your certificate private key file with a passphrase. In this example, the key file is named custom.key and the encrypted key file is custom_encrypted.key.

a Use the following OpenSSL command to password protect the file.

```
openssl rsa -aes128 -in /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/cas/shared/default/custom.key
-out /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
private/cas/shared/default/custom_encyrpted.key -passout pass:password
```

b Ensure that the customers encrypted private key file has permissions set to 644. Use chmod to change the permissions:

```
chmod 644 custom_encrypted.key
```

5 Create a customer-supplied certificate private key passphrase file. Use the echo command to create the private key passphrase file. In this example, the private key passphrase filename is customer_encrypted.encryption.key.

```
sudo bash -c "echo -n 'password' > custom_encrypted.encryption.key"
sudo chown cas:sas custom_encrypted.encryption.key
sudo chmod 0640 custom_encrypted.encryption.key
sudo cat custom_encrypted.encryption.key ;
echo password
```

6 You can remove the original custom.key file as you have now secured the custom_encyrpted.key (encrypted key file) and custom_encrypted.encryption.key (passphrase protected encrypted key file) files.

7 Edit the casconfig_usermods.lua file located at **/opt/sas/viya/config/etc/cas/default**on the CAS Controller. Change the required CAS_CLIENT_SSL environment variables. Specify the names of your custom certificate and the custom encrypted certificate private key file (custom_encrypted.key), and the customer-supplied certificate private key passphrase file (custom_encrypted.encryption.key).

Note: This file has 0600 permissions: -rw-r--r--

```
env.CAS_CLIENT_SSL_REQUIRED=true
env.CAS_CLIENT_SSL_CERT="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/custom.crt"
env.CAS_CLIENT_SSL_KEY="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/custom_encrypted.key"
env.CAS_CLIENT_SSL_KEYPWLOC = '/opt/sas/viya/config/etc/
```

```
SASSecurityCertificateFramework/private/cas/shared/default/
custom_encrypted.encryption.key'
env.CAS_CLIENT_SSL_KEYPW    = nil
```

When setting the CAS Client environment variables, consider the following information.

- If you are using an intermediate CA certificate, then a certificate chain file needs to be specified for the CAS_CLIENT_SSL_CERT= environment variable. The file needs to include the server identity certificate first, and then the signing intermediate CA certificates in the order in which they were signed. The root CA does not need to be included in this chain file.
- If you are using your own custom certificate and key, you should copy the changes made to CAS_CLIENT_SSL_CERT= and CAS_CLIENT_SSL_KEY= environment variables to the vars.yml file. This change ensures that your settings are not changed when upgrades are made to the deployment.

8 On the CAS controller, restart the cascontroller service.

```
sudo service sas-viya-cascontroller-default restart
```

9 For other client-side connections (Lua, Python) you need the root CA certificate on the client specified on Linux.

- The workspace server exports the trustedcerts.pem file by default.
- On Linux, if the root CA is already in the OpenSSL trusted certificate store, Lua or Python clients should work without having to set the CAS_CLIENT_SSL_CA_LIST= environment variable.
- Otherwise, set the CAS_CLIENT_SSL_CA_LIST= environment variable to point to the location of your certificate chain. Root CA certificates at a minimum are needed in the certificate chain.

  ```
  export CAS_CLIENT_SSL_CA_LIST="/opt/sas/viya/config/etc/
  SASSecurityCertificateFramework/cacerts/trustedcerts.pem"
  ```

- For SAS client-side connections, SAS should automatically find the trustedcerts.pem file that is located in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts** either through the workspace server export statement shown previously or the SSLCALISTLOC= system option that is set during installation.

Configure CAS TLS to Use SAS Default Certificates (programming-only deployment)

Use TLS to secure communications between the CAS server and clients. The certificate used for client and server communication needs to be signed by a certificate authority (CA) that is trusted by all potential clients.

At installation, SAS provides certificates that can be used to secure the deployment. Here are the certificates that are added to the CAS machines in a SAS Viya programming-only deployment..

Table 3 *Security Artifacts Provided at Installation for Programming-only Viya deployment*

Security Artificact	Deployment File Name	Location	Description

Certificate truststore	trustedcerts.pem trustedcerts.jks	/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/	Contains the trusted list of CA certificates. These include the Mozilla bundle of CA certificates, the SAS self-signed certificate, and the Apache Server certificates?
Certificate File	sas_encrypted.crt	/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs/	Contains the certificate generated by SAS. These are self-signed certificates.
Certificate Private Key File	sas_encrypted.key	/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private	Contains the private key generated by SAS.
Certificate Private Key Passphrase File	your_EncryptedPassphrase.key	/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/	Contains the encrypted passphrase. You generate your own passphrase and then encrypt the file that contains it.

To use the certificates shown above, configure TLS between CAS servers and the CAS Client If you plan to use the SAS certificates that are provided at installation, perform the following tasks:

Note: If you plan to use your own custom certificates to configure CAS Client TLS, see "Configure CAS TLS to Use Custom Certificates (programming-only deployment)" on page 15.

1 Log on to the CAS controller machine as a user with root or sudo privileges.

2 If you have a CAS session running, cancel it.

3 Add the following environment variables to casconfig_usermods.lua file located at **/opt/sas/viya/config/etc/default** for the CAS Client on port 5570. Set the CAS_CLIENT_SSL_REQUIRED= environment variable to **true**. The other environment variables are already set to point to the self-signed certificates and keys that were provided at installation.

```
env.CAS_CLIENT_SSL_REQUIRED  = 'true'
env.CAS_CLIENT_SSL_CA_LIST = /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/cacerts/trustedcerts.pem'
env.CAS_CLIENT_SSL_CERT = 'opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/sas_encrypted.crt'
env.CAS_CLIENT_SSL_KEY = '/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/sas_encryption.key'
env.CAS_CLIENT_SSL_KEYPWLOC = '/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/encryption.key'
```

Note: By default, SAS self-signed certificates are generated using the fully qualified domain name for the Common Name. Make sure that the CAS hostname in programs submitted to CAS matche the Common Name used in the SAS self-signed certificates.

Note: If you are setting the CAS_CLIENT_SSL_REQUIRED= environment variable to **true**, you should copy the change made to this environment variable to the vars.yml file. This change ensures that your settings are not changed when upgrades are made to the deployment. See "Modify the vars.yml File" in *SAS Viya for Linux: Deployment Guide* for more details.

4 Restart the cascontroller service.

```
sudo service sas-viya-cascontroller-default restart
```

5 For other client-side connections (Lua, Python), you need the root CA certificate on the client specified on Linux.

In SAS Viya, the workspace server exports the trustedcerts.pem file by default.

On LAX, if the root CA is already in the OpenSSL trusted certificate store, Lua or Python clients should work without having to set the CAS_CLIENT_SSL_CA_LIST= environment variable.

Otherwise, set the CAS_CLIENT_SSL_CA_LIST= environment variable to point to the location of your certificate chain. Root CA certificates at a minimum are needed in the certificate chain.

```
export CAS_CLIENT_SSL_CA_LIST="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/cacerts/trustedcerts.pem"
```

Note: For SAS client-side connections, SAS should automatically find the trustedcerts.pem file that is located in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts** either through the workspace server export statement shown previously or the SSLCALISTLOC= system option that is set during installation.

CAS Server Monitor HTTP and HTTPS Access

Overview

CAS Server Monitor is set up for HTTP and HTTPS during initial deployment. Open a web browser and enter one of the URLs in the address field in the following format:

- In a full deployment where CAS is secured by default, enter https://*host1.sas.com*/cas-shared-default-http/ (Encrypted Communications)
- In a programming-only deployment, enter http://*reverse-proxy-server*/cas-shared-default-http/ (Insecure Communication)

Note: To access CAS Server Monitor, the password must be set for the CAS user ID or other administrative account. See "Access CAS Server Monitor" in *SAS Viya for Linux: Deployment Guide* for additional information. for information about password access set up during deployment.

If you did not add compliant certificates and instead kept the default security settings and certificates, you will see the `Your connection is not private` message. SAS recommends replacing the certificates before giving end-users access to SAS Viya. In a full deployment, dual authentication occurs for logon to CAS Server Monitor and access to CAS from SAS Studio.

Block External Connections to Port 8777

You can also access CAS Server Monitor directly using http://*controller-machine*:8777/.

However, to secure web access to your SAS Viya software, you should block external communications to port 8777. Refer to the Red Hat Enterprise Linux Reference and Security Guides at https://access.redhat.com/documentation/en/red-hat-enterprise-linux/ for information about best practices for securing ports.

See "Enable Required Ports" in *SAS Viya for Linux: Deployment Guide* for more information.

Access to CAS Server Monitor from SAS Studio

If you access CAS Server Monitor from SAS Studio, CAS Server Monitor is accessed using the HTTPS protocol by default. If you receive a "Connection Not Secure" message because of HTTPS access, you need to take one of the following actions:

- Import the Certificate Authority certificates used by the Apache HTTP Server into your browser.
- Change the HTTPS protocol to HTTP by changing the variables in the casconfig_usermods.lua file that control the protocol and port used in the CAS Server Monitor link accessed from within SAS Studio. Change the following variables:

  ```
  env.CAS_VIRTUAL_HOST = 'external.mycompany.com'
  env.CAS_VIRTUAL_PORT = 443
  env.CAS_VIRTUAL_PROTO = 'https'
  ```

- Make sure the host name for the CAS_VIRTUAL_HOST variable is the same as the Common Name used in the server identity certificate that the Apache HTTP Server is using.

Enable or Disable TLS Using Port Families

Overview

By default, almost all network connections are secured using Transport Layer Security (TLS). Most ports are secured by enabling port families by default. Information included in this sections describes what port families are configured by default and how those default port family configurations can be changed.

Additional items that can be configured for TLS security.

Note: The following do not use the port families to enable or disable TLS.

- SAS Secret Manager ports are always secured.
- SAS Configuration Server ports are controlled by settings in the vars.yml file. The SECURE_CONSUL setting enables port 8501 with HTTPS. The DISABLE_CONSUL_HTTP_PORT setting disables port 8500. SeeSpecify Security Settings.

- SAS Cloud Analytic Services inter-node communications are not secured by default, but can be enabled through configuration changes. See "Configure TLS Between CAS Worker Nodes" on page 24.
- For encryption in-motion with the SAS Embedded Process, the configuration on the CAS side is complete by default. However, the SAS Embedded Process configuration, on either Hadoop or Teradata must be updated to enable it. The DCTCPMENCRYPT option defines if encryption is used. See "Encrypt Data Transfer When Using the SAS Data Connect Accelerator" on page 32..

For more information, see "Enable Required Ports" in *SAS Viya for Linux: Deployment Guide*.

Port Families

The ports shown in Table 4 on page 22are secured using TLS by default.. You can enable or disable network security traffic for TLS using categories (families) of ports.

Table 4 *TLS Port Families - Enabled by Default*

Family Name	Description	Ports that can be controlled
databaseTraffic	Port family that needs to control traffic to database servers that might be located on different network segments.	SAS Infrastructure Data Server (PostgresSQL), EP Data Connectors
sasData	Port family that controls traffic transporting data to SAS servers. The SAS Workspace Server and the SAS Object Spawner use this port family to enable and disable AES encryption at start up.	CAS Client, SAS Compute Server, SAS/CONNECT Server, SAS/CONNECT Spawner, SAS Event Stream Processing (ESP) Server, CAS Server Monitor SAS Workspace Server, SAS Object Spawner
serverControl	Port family that controls traffic sent between clustered servers to maintain the cluster.	SAS Launcher Server
web	Port family for any network associated with machines running web applications	Apache HTTP Server, all web apps and microservices, SAS Cache Locator (Apache Geode), SAS Message Broker (RabbitMQ), CAS Rest, ESP App, SAS Studio, CAS Server Monitor

Use SAS Environment Manager to Enable TLS Port Families

The SAS Configuration Server (consul) settings are controlled by the configuration service and SAS Environment Manager. To alter the settings in SAS Environment manager, change the sas.security network settings.

1 From the side menu , select **SAS Environment Manager**.

2 In the navigation bar, click

The Configuration page is an advanced interface. It is available to only SAS Administrators.

3 The default view is **Basic Services**. Select **Definitions** from the drop-down box.

4 In the **Definitions** list, filter on **sas.security**. Select sas.security.

5 If the definition has no properties configured, complete the following:

a In the top right corner of the window, click .

b In the New sas.security Configuration dialog box, select **network.web.enabled**. This is the property for the port family that you want to update to turn off web-enabled TLS.

false Disables TLS for this property.

true Enables TLS.

For a description of the properties, see "Configuration Properties: Reference (Applications)" in *SAS Viya Administration: Configuration Properties*.

c Click **Save**.

Note: The system will take a few minutes to recognize the new key before starting to use the new key.

6 Restart all services on all machines

```
sas-viya-all-services stop
sas-viya-all-services start
```

Programmatically Enable TLS Port Families

Using the SAS Environment Manager to disable or enable TLS on port families is the preferred method. However, you can enable or disable the TLS ports programmatically by using the following commands. To alter the settings in SAS Environment manager, change the sas.security network settings.

Log on to the SAS Configuration Server as a user with root or sudo privileges.

```
opt/sas/viya/home/bin/sas-bootstrap-config
--token-file /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tokens/consul/default/client.token kv write --force --site-default
config/application/sas.security/network.databaseTraffic.enabled false
/opt/sas/viya/home/bin/sas-bootstrap-config
--token-file /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tokens/consul/default/client.token kv write --force --site-default
config/application/sas.security/network.sasData.enabled false
/opt/sas/viya/home/bin/sas-bootstrap-config  --token-file
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tokens/consul/default/client.token kv write --force --site-default
config/application/sas.security/network.serverControl.enabled false
```

```
/opt/sas/viya/home/bin/sas-bootstrap-config --token-file
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tokens/consul/default/client.token kv write --force --site-default
config/application/sas.security/network.web.enabled false
```

Use Ansible to Enable/Disable TLS Port Families

Prior to deployment, add the following to the playbook sitedefault.yml file:

```
config:
    application:
        sas.security:
            network.web.enabled: false
            network.sasData.enabled: false
            network.databaseTraffic.enabled: false
            network.serverControl.enabled: false
```

Individual servers and spawners can be disabled by changing their instance key. For example, a connect spawner can be changed using the following code:

```
config/connect-spawner-${SASDEPLOYID}/sas.security/network.sasData.enabled
```

Each application must check configuration service entries that record the default settings for TLS. Customers can use SAS Environment Manager to override the default behavior by altering the settings shown above. The new settings are picked up the next time the application starts. Each application/service (including third-party applications) should use best practices for secure configuration.

Configure TLS Between CAS Worker Nodes

CAS supports TLS encrypted connections between the worker nodes. When configured, any data sent between worker nodes is sent over a TLS connection. The SAS Cloud Analytic Services inter-node communication is not secured by default. This is due to the large performance impact of enabling the CAS inter-node encryption. But, this can be enabled through configuration changes.

To enable CAS Internode TLS, a customer will be required to add a new Consul property by way of EV. They will have to know (from reading the document) that the property name is: config/cas-{$SASTENANT}-{$SASCASINSTANCE}/sas.security/network.casInternode.enabled (example: config/cas-shared-default/sas.security/network.casInternode.enabled). They will have to create this key, set its value to true, and then restart CAS. Said property will not be readily surfaced in EV. In other words, a customer cannot accidentally find it by just clicking around in EV.

In order to enable TLS encryption, every node must have a certificate file and a private key file. Having some nodes configured to encrypt data, and not having others encrypt data is not permitted. The certificates used for CAS node-to-node communication need to be signed by a trusted certificate authority (CA) or can be self-signed.

> **TIP** Every node needs its private key password.

CAUTION

Encryption has perfromance costs Encryption will degrade your performance, and increase the amount of CPU time that is required to complete any action. Actions that move large amounts of data are

penalized the most. Session start-up time is also impacted negatively. On tests that move large blocks of data between nodes, elapsed times can increase by a factor of ten.

Determine the type of certificate to use (site-signed, third-party-signed, or self-signed certificate). You can generate your own site-signed or third-party-signed certificates. To generate site-signed or third-party-signed certificates in .pem format or to generate your own self-signed or root CA certificates using OpenSSL, see "Manage Certificates" on page 41.

At installation, SAS lays down a Mozilla trusted CA bundle of certificates in the directory **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts**.

SAS recommends the following best practices for managing certificates and securing your private keys. These directories are referenced in the TLS environment variables below and point to the keys and certificates that you are using for configuring TLS.

- Place your server identity certificates for CAS in the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs** directory.

  Intermediate certificates are added to the server identity certificate in a certificate chain. The server identity certificate must be the first certificate in the chain. The intermediate certificate must be second. This order is important to allow validation with the private key to be successful.
- Place your root certificates and trusted CA certificates in the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts** directory.
- Place your private server keys in the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private** directory.
- Encrypt your private key when possible.
- Password-protect your private key file.
- Edit the keys.lua file in the **/opt/sas/viya/config/etc/cas/default** directory. Set the CAS_INTERNODE_SSL_KEYPW= environment variable in this configuration file.

Configure TLS between the worker nodes.

1. Log on to the CAS controller machine as a user with root or sudo privileges.
2. If you have a CAS session running, cancel it.
3. For the CAS Worker nodes, edit the existing keys.lua file and take the following steps to secure your private key file and the private key password.

   a. If the private key does not have an encrypted password, you can use the following commands to encrypt it.

   ```
   openssl rsa -aes128 -in /opt/sas/viya/config/etc/
   SASSecurityCertificateFramework/private/private.key
   -out /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
   private/tempprivate.key -passout pass:password
   mv opt/sas/viya/config/etc/SASSecurityCertificateFramework/
   private/tempprivate.key /opt/sas/viya/config/etc/
   SASSecurityCertificateFramework/private/private.key
   ```

   b. Ensure that the worker nodes private key file is readable only by CAS services. Use chmod to change the permissions:

   ```
   chmod 400 private.key
   ```

   When you list the file, you see the permissions are Read Only by all, -r------.

c Change the keys.lua file located at **/opt/sas/viya/config/etc/cas/default** to set the CAS_INTERNODE_SSL_KEYPW environment variable with the new encrypted password.

```
env.CAS_INTERNODE_SSL_KEYPW="encryptedpassword"
```

4 For the CAS worker nodes, add the following lines to the casconfig.lua file at **/opt/sas/viya/config/etc/cas/default**.

```
env.CAS_INTERNODE_DATA_SSL=true
env.CAS_INTERNODE_SSL_CERT="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/CASServer.crt"
env.CAS_INTERNODE_SSL_KEY="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/private.key"
env.CAS_INTERNODE_SSL_CA_LIST="/opt/sas/viya/config
/etc/SASSecurityCertificateFramework/cacerts/trustedcerts.pem"
```

Note: A certificate chain file needs to be specified for the CAS_INTERNODE_SSL_CERT= environment variable. The file needs to include the server identity certificate first, and then the signing intermediate CA certificates in the order in which they were signed. The root CA does not need to be included in this chain file.

For information about the environment variables, see “CAS TLS Environment Variables” on page 94..

5 Restart the cascontroller service.

```
sudo service sas-viya-cascontroller-default restart
```

SAS Cloud Analytic Services inter-node Encryption

SAS Cloud Analytic Services inter-node encryption is not enabled by default Copyright © SAS Inst i tute Inc. Al l right s reserved. 79 SAS Cloud Analytic Services inter-node encryption

- • There is a significant performance impact to using inter-node encryption
- • All the TLS certificates and private keys are deployed by default
- • Only the configuration option enabling inter-node encryption is set to false
- • In the file: /opt/sas/viya/config/etc/cas/default/casconfig_deployment.lua
- • The default option is: -- TLS enablement for internode port

```
env.CAS_INTERNODE_DATA_SSL = 'false'
```

To enable inter-node encryption, you can use use SAS EV, casconfig_usermods.lua, or update the vars.yml file as following.

- In the following file on all nodes: **/opt/sas/viya/config/etc/cas/default/casconfig_usermods.lua**
- Add the following text: -- TLS enablement for internode port env.CAS_INTERNODE_DATA_SSL = 'true'
- Restart the SAS Cloud Analytic Services Controller
- Alternatively, update the vars.yml and add the following to the CAS_CONFIGURATION section: env: CAS_INTERNODE_DATA_SSL: 'true'

Note: In the vars.yml remember to use space and not tabs to correctly line up options.

- Re-run the playbook

Using the Environment Manager, you have to know the configuration to create? config/cas-CAS_TENANT_NAME-CAS_INSTANCE_NAME/sas.security/network.casInternode.enabled

A customer can enable CAS Internode TLS in EV by knowing the configuration to create. Example: config/cas-shared-default/sas.security/network.casInternode.enabled -- with value true.

Configure SAS Viya to Connect to LDAPS Provider

Lightweight Directory Access Protocol (LDAP) connections can be established in a TLS session so that all data that is sent between the LDAP client and LDAP server is encrypted. LDAP over SSL/TLS is known as LDAPS.

To connect to an LDAPS provider, SAS Viya needs access to the CA certificate used by the LDAPS provider. To configure TLS between SAS Viya and the LDAPS provider, use the following instructions to add the CA certificates to the trustedcerts files on every machine in the deployment (as a best practice). See "Use Best Practices to Manage Certificates" on page 41.

Note: Only LDAP-based identity providers are supported. These instructions assume that you have basic familiarity with LDAP administration.

1 Log on to your machine as a user with root, SAS Admin, or sudo privileges.

2 If your LDAPS provider's CA certificate is not already included in the Mozilla bundle of trusted CA certificates, append the root certificate to the trustedcerts files in the **`/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/`** directory.

   - To add the root certificate to the trustedcerts.pem file, just include the root certificate at the end of the trustedcerts.pem file.
   - To add the root certificate to the trustedcerts.jks file, import the file using the keytool command.

   See "Add Your Certificates to the Trust List or to a Certificate Chain" on page 42 for information about adding your certificates to the trustedcerts files.

   Note: Do not delete the trustedcerts files.

   Note: Add your root certificate to the trustedcerts.pem and trustedcerts.jks files on every machine in the deployment.

3 Use the SAS Environment Manager to set the configuration property sas.identities.providers.ldap.connection. Specify an LDAPS port number (by default LDAPS is 636) and specify *LDAPS* in the **url** field. You can also use the port value 3269 (Global Catalog) for LDAPS.

   a If the Configuration page of SAS Environment Manager is not already displayed, select **Resources** ⇨ **Configuration** from the side menu ☰.

   b Select **Basic Services** from the list, and then select the **Identities service** from the list of services.

c In the **sas.identities.providers.ldap.connection** section, click  . In the Edit sas.identities.providers.ldap.connection Configuration window, do the following:

1 Update values for the **port** field, adding an LDAPS port value. Update the **url** field to specify *LDAPS*. For the remaining fields, review the default values and make changes as necessary. The default values are appropriate for most sites.

2 Click **Save**.

For additional configuration instructions, see "Configure Security" in *SAS Viya for Linux: Deployment Guide*. For details about the sas.identities.providers.ldap.connection property, see "Configuration Properties: Reference (Services)" in *SAS Viya Administration: Configuration Properties*.

4 If needed, restart the SAS Logon Manager service by running the following command:

```
sudo service sas-viya-saslogon-default restart
```

Note: It might take several minutes to restart SAS Logon Manager.

If needed, restart the Identifies service.

```
sudo service sas-viya-identities-default restart
```

Replace Certificates for LDAPS

By default, the Apache HTTP Server has been configured to serve as a reverse proxy to connect to the SAS Viya Web Application. SAS Viya installation provides certificates and configures TLS options in the Apache HTTP server configuration. However, SAS recommends replacing the default certificates with custom certificates. After the certificates have been replaced in the Apache HTTP server, the truststore can be rebuilt using the following instructions.

Note: See "Secure Apache HTTP Server" on page 4.

1 Locate the certificates that you would like to remove from the sitedefault.yml file, located at **`/opt/sas/viya/config/etc/consul.d/default/sitedefault.yml`**. For our example, we are removing sascaroot and sassha2rootca certificates from sitedefault.yml.

2 Remove certificates from Consul using `sas-bootstrap-config` commands as follows:

Note: The following code is shown on more than one line for display purposes only. This command may need to be on one line and should not contain line breaks.

```
/opt/sas/viya/home/bin/sas-bootstrap-config --token-file
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tokens/consul/default/client.token kv delete cacerts/sascaroot

/opt/sas/viya/home/bin/sas-bootstrap-config --token-file
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens
/consul/default/client.token kv delete cacerts/sassha2rootca
```

3 Ensure that the certificates have been removed.

```
/opt/sas/viya/home/bin/sas-bootstrap-config --consul
https://your-configuration-server:8501
--token 75bef370-cc93-44bb-a290-82833f6c4ddf kv read
--recurse cacerts
```

4 Stop the microservices.

```
sas-viya-all-services stop
```

5 Using Ansible, run the utility play rebuild-trust-stores.yml to rebuild the truststores.

```
ansible-playbook -i inventory.ini utility/rebuild-trust-stores.yml
```

6 Restart all the services so that they now reference the new truststores.

```
sudo sas-viya-all-services start
```

Restart the identities service.

```
sudo service sas-viya-identities-default restart
```

7 Sign in to LDAPS.

SAS Studio HTTP and HTTPS Access

SAS Studio is set up for HTTP and HTTPS during initial deployment. In SAS Viya, SAS Studio is configured to work with the Apache HTTP server.

To access SAS Studio in a full deployment, open a web browser and enter the URL in the address field: https://*reverse-proxy-server*/SASStudio/. By default, SAS Studio is secured.

In a programming-only deployment, SAS Viya end users must use HTTP to connect to SAS Studio or CAS Server Monitor because the Apache HTTP Server does not support HTTPS. To access SAS Studio in a programming-only deployment, open a web browser and enter the URL in the address field: http://*reverse-proxy-server*/SASStudio/ (Insecure Communication)

Log on using the credentials for your operating system account.

HTTPS Access to SAS Message Broker

Note: This section is applicable only if you have a full deployment. If you have a programming-only deployment, skip this section.

By default, SAS provides self-signed certificates and keys to secure the deployment. The URLs available to access SAS Message Broker for HTTP and HTTPS post installation are as follows:

- https://*RabbitMQ-IP-address*:15672/#/ (Encrypted Communications)
- http://*RabbitMQ-IP-address*:15672/#/ (Insecure Communication)

If after clicking on the HTTPS link shown above to access the Message Broker, you receive a certificate error, you might need to import the SAS Viya trusted CA certificate into the browser trust store. One way to perform this task follows.

1 From the Internet Explorer browser, enter https://*RabbitMQ-IP-address*:15672/#/

2 Import your SAS Viya intermediate CA.

   a In the RabbitMQ login window, click "Certificate error".

   b In the **Untrusted Certificate** dialog, select **View Certificates**.

   c In the Certificate window, click the **Install Certificate** button.

   d From the Certificate Import Wizard dialog, select Store Location **Current User**. Select **Next**.

   e For Certificate Store, I select **Place all certificates in the following store**. Browse and select **Trusted Root Certification Authorities**. Click **OK**.

   f Click **Next**. Thenf select **Finish**. You should receive a message that your "Import was successful".

Sign On to a SAS/CONNECT Spawner Using TLS

Use SAS/CONNECT as a bridge to access data across environments. You can use TLS to secure that bridge when you sign on to the SAS/CONNECT spawner from the SAS/CONNECT client. The sign-on command starts a SAS/CONNECT server.

In a full deployment of SAS Viya, TLS is on by default for SAS/CONNECT. No additional configuration is required.

In a SAS Viya programming-only deployment, you need to configure SAS/CONNECT by performing the following steps.

You can use the certificates provided by SAS, add custom certificates, or generate your own certificates to use TLS to secure SAS/CONNECT.

- If you are using a certificate whose root CA is not already in the Mozilla Trusted CA Certificate bundle, you need to add the root CA certificate to the Mozilla bundle by editing the trustedcerts.pem file. See "Add Your Certificates to the Trust List or to a Certificate Chain" on page 42.
- To create site-signed or third-party-signed certificates, see "Generate Site-Signed or Third-Party-Signed Certificates in PEM Format" on page 45.
- To create self-signed certificates, see "Generate Self-Signed Certificates" on page 49.

Note: If you are using custom certificates or generating your own certificates, use Best Practices on page 41 for securing your certificates and keys.

To configure SAS/CONNECT, perform the following steps:

1 Sign in with administrator privileges to the machine containing the SAS/CONNECT spawner.

2 In the connect_usermods.sh file located in `/opt/sas/viya/config/etc/connect/default`, set up TLS by adding the SSL encryption options. Edit the connect_usermods.sh file, and add the following encryption options to the USERMODS= line to encrypt the connection for the SAS/CONNECT spawner. Note that this file needs to have global Read permissions: -rw-r--r--

In the following code example, the names of the certificate file and the private key file are just example names. These would be the names of the files that you placed in the **/opt/sas/viya/config** directories.

Note: The options are enclosed in double quotation marks.

```
USERMODS="-netencrypt          /* a */
-netencryptalgorithm ssl             /* b */
-sslcertloc /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tls/certs/server.crt                 /* c */
-sslpvtkeyloc /config/etc/SASSecurityCertificateFramework/private/
private.key                           /* d */
-sslpvtkeypass 'password'"             /* e */"
```

1 The NETENCRYPT option specifies that encryption is required.

2 The NETENCRYPTALGORITHM= option specifies that the spawner is started using TLS.

3 The SSLCERTLOC= option specifies the location of a file that contains a digital certificate for the machine's public key. This is used by the server to send to clients for authentication.

   Note: If the certificate is not self-signed, the file specified by the SSLCERTLOC= option needs to be a certificate chain file that starts with the server identity certificate and includes the signing intermediate CA certificates. The root CA certificate does not need to be included in the certificate chain.

4 The SSLPVTKEYLOC= option specifies the location of the file that contains the private key that corresponds to the digital certificate that was specified by the SSLCERTLOC= option.

5 The SSLPVTKEYPASS= option specifies the password that TLS requires to decrypt the private key. The private key is stored in the file that was specified by the SSLPVTKEYLOC= option.

Note: SAS first looks for CA certificates in a file named trustedcerts.pem, located in the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts** directory. Therefore, the SSLCALISTLOC= system option is not required if you are storing your trusted certificates in the default location. However, if you choose not to use the default location to store certificates, you need to specify the SSLCALISTLOC= option with a location for the certificates for the SAS/CONNECT client and spawner. For each of the preceding examples, the default location is used.

3 Start the SAS/CONNECT spawner.

```
sudo service sas-viya-connect-default start
```

4 The SAS/CONNECT spawner runs the connectserver.sh script, which runs the connectserver_usermods.sh script. The connectserver_usermods.sh script is located in **/opt/sas/viya/config/etc/connectserver/default**. Edit the connectserver_usermods.sh file, and add the following encryption options to the USERMODS_OPTIONS= line. Note that this file needs to have global Read permissions: -rw-r--r--

Note: The options are enclosed in double quotation marks.

```
USERMODS_OPTIONS="-sslcertloc /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/Server.crt
-sslpvtkeyloc /config/etc/SASSecurityCertificateFramework/
private/private.key
-sslpvtkeypass 'password'"
```

Note: The certificates specified above are your server certificates.

5 After a spawner is started on a SAS/CONNECT server, a SAS/CONNECT client can connect to it. The following example shows how to connect a client to a spawner that is running on a SAS/CONNECT server:

```
options netencryptalgorithm=SSL;
%let myserver=<myHost.myDomain.com> <port>;
SIGNON myserver user=sasdemo passwd="password";
```

If the spawner requires TLS encryption (NETENCRYPTALGORITHM=SSL), the SAS/CONNECT client needs to locate the root CA certificate to validate the spawner's certificate. For a Linux client, SAS first looks for the root CA certificate in the trustedcerts.pem file in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/default**. Otherwise, you need to specify the location of the root CA certificate by using system options SSLCALISTLOC= or SSLCACERTDIR=.

For a Windows SAS/CONNECT client, import the trusted root CA certificate into the Windows trusted root certificate store.

6 If you need a SAS 9.4 client to work with SAS Viya, see "Configure SAS 9.4 Clients to Work with SAS Viya" on page 39.

See Also

- "SAS System Options for Encryption" on page 74
- SAS Viya: Overview

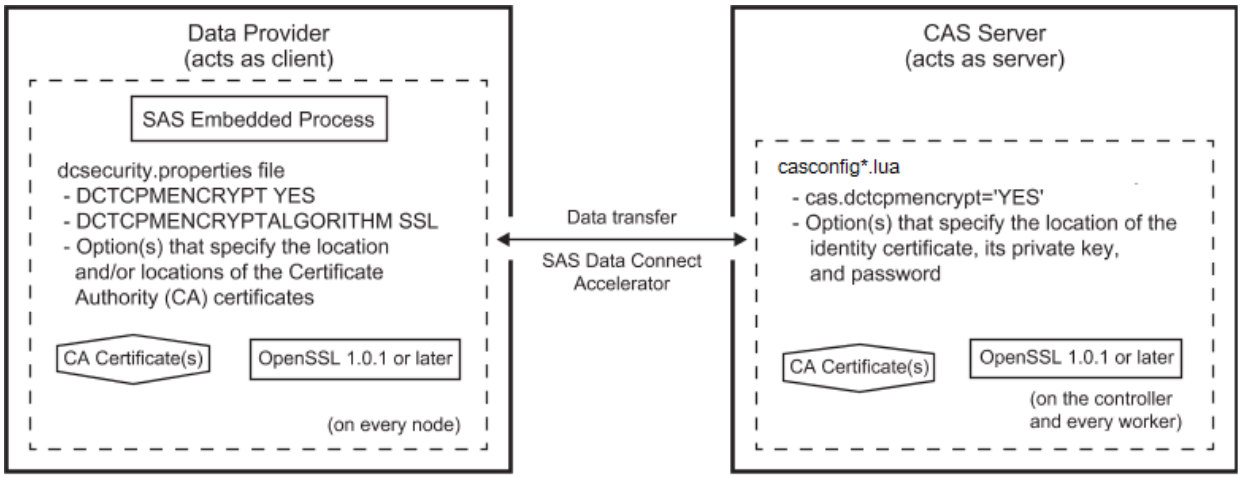
Encrypt Data Transfer When Using the SAS Data Connect Accelerator

If you are using a SAS Data Connect Accelerator, the data that is transferred between the data provider and the CAS server is not encrypted by default. However, SAS Viya does support TLS encryption between the data provider and the CAS server, and you can take steps to enable that encryption. It should be noted that performance can be affected when TLS encryption is enabled and large amounts of data are being transferred.

Overview of SAS Data Connect Accelerator Encryption

When data is transferred between a data provider and CAS, the data provider acts as the client and the CAS server acts as a server.

Figure 1 *Data Connect Accelerator Encryption and the CAS Server*



When the SAS Embedded Process is deployed on the data provider, a dcsecurity.properties file and a **certs** directory are created in the ***SAS-Embedded-Process-home*/security** directory. The **certs** directory will hold the TLS security certificates. The dcsecurity.properties file must be updated to enable data connector encryption.

When Viya 3.3 is deployed, TLS is enabled and configured on the CAS server (server side). The deployment process provides a default level of encryption for data in motion. Options are set in the vars.yml file and defined in the casconfig_deployment.lua file to enable data connector encryption and to provide the location of the TLS private key and password.

The prerequisites and process for enabling TLS encryption on the data provider is different for each data provider.

Note: A TLS private key and certificate are required for each CAS host.

Prerequisites When Enabling Encryption for the SAS Data Connect Accelerator for Teradata (on SAS Viya)

Here are the prerequisites for enabling encryption for the SAS Data Connect Accelerator for Teradata (on SAS Viya).

- Upgrade the OpenSSL package on all Teradata nodes to 1.0.1g or later to support TLS.

  The 64-bit OpenSSL library package that is most likely being used at your site is `libopenssl0_9_8-0.9.8j-0.50.1`. The required version is `libopenssl1_0_0-1.0.1g-0.37.1` or later. This package update is available on the Teradata patch server. Contact Teradata Customer Services to get this package updated. If you plan to use TLS now or in the future, it is best to upgrade the OpenSSL package before you install the SAS In-Database Technologies for Teradata (on SAS Viya).

  Note: The old version, **openssl0_**, and new version, **openssl1_0_0** (or later), can coexist.

- Install SAS/ACCESS Interface to Teradata (on SAS Viya) and SAS In-Database Technologies for Teradata (on SAS Viya).

These offerings include the SAS Embedded Process, the SAS Data Connect Accelerator for Teradata (on SAS Viya), and the SAS Embedded Process support functions. For more information, see *SAS Viya for Linux: Deployment Guide*.

- Obtain TLS identity certificates (site-signed, third-party-signed, or self-signed) from the CAS controller machine. These certificates are located in the trustedcerts.pem file. Corresponding certificate authority (CA) certificates must be installed on the Teradata nodes. If you use externally signed identity certificates in the CAS server, the Mozilla bundle of CA certificates that are provided by SAS can be deployed on the Teradata nodes.

  For more information about the location of the trustedcerts.pem file, see "(Optional) Deploy TLS Certificates" in *SAS Viya for Linux: Deployment Guide*.

  For more information about configuring CAS, see "Configure CAS TLS to Use Custom Certificates (programming-only deployment)" on page 15 and "Configure CAS TLS to Use SAS Default Certificates (programming-only deployment)" on page 18.

  Certificates, keys, and passwords produced for authenticating to the SAS Embedded Process for Teradata might coincide with those produced for other clients of the CAS server. However, they do not need to coincide. For information about generating certificates, see the appropriate topic in "Manage Certificates" on page 41.

- When you installed the SAS Embedded Process, the following file and directory were created:

  ```
  /opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/dcsecurity.properties
  /opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/certs
  ```

  - All directories and files should have the `owner:group = tdatuser:tdatudf` setting.
  - The **`/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/`** directory should have `drwxr-xr-x` permissions.
  - The **`/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/certs`** directory should have `drwxr-xr-x` permissions.
  - The dcsecurity.properties file should have `-rwxr-xr-x` permissions.

Enable Encryption for the SAS Data Connect Accelerator for Teradata (on SAS Viya)

Follow these steps to encrypt data transfer between Teradata and the CAS server using the SAS Data Connect Accelerator for Teradata (on SAS Viya).

1. On Teradata, modify the dcsecurity.properties file to enable SAS Data Connect Accelerator encryption.

   a. Navigate to the **`/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/`** directory.

   b. Change the DCTCPMENCRYPT option in the dcsecurity.properties file as follows.

   ```
   -DCTCPMENCRYPT YES
   ```

   CAUTION
   The DCTCPMENCRYPT option is set on both the CAS server and on the data provider. How the option is set on both sides determines whether the data being transferred is encrypted or not. For more information, see "DCTCPMENCRYPT Option Setting Interaction" on page 38.

c Add either the DCSSLCACERTDIR or DCSSLCALISTLOC option to the dcsecurity.properties file to specify either the location of the trusted certificate authorities or the public certificate(s) for trusted certificate authorities.

Here is an example.

```
-DCSSLCALISTLOC /opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/certs/certs-filename.pem
```

For more information about the options, see "dcsecurity.properties File Options for Data Transfer with the SAS Data Connect Accelerator" on page 105.

2 Copy the necessary TLS CA certificates to the **/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/certs** directory.

- If your CA certificates already exists on the Teradata server, copy the CA certificates to this directory.
- If your CA certificates exist on the CAS server, using a method of your choice, copy the CA certificates to this directory on the Teradata server. Here is an example.

```
scp CASCA1.pem tdatuser@teramach1:/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/certs
```

For more information about the location of the CA certificates on the CAS server, see "(Optional) Deploy TLS Certificates" in *SAS Viya for Linux: Deployment Guide*.

Note:

- The CA certificates on the Teradata server must authorize the identity certificates that are specified on the CAS server.
- All Teradata files and directories should have the `owner:group = tdatuser:tdatudf` setting.
- The **/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/** directory should have `drwxr-xr-x` permissions.
- The **/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/certs** directory should have `drwxr-xr-x` permissions.
- The dcsecurity.properties file should have `-rwxr-xr-x` permissions.

3 Copy the contents of the **/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security/** directory to all nodes on the Teradata cluster.

a Navigate to the **/opt/SAS** directory.

```
cd /opt/SAS/
```

b Create a compressed archive file.

```
tar cvof /tmp/sasep_security.tar/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security
```

c Do a parallel file transfer to push the files to all nodes.

```
pcl -send /tmp/sasep_security.tar /tmp
```

d Use the parallel shell command to extract the TAR file.

```
psh tar xvof /tmp/sasep_security.tar
```

e Create a backup copy of the TAR file. Keep a backup copy.

```
cp /tmp/sasep_security.tar /root/sasep_security.tar
```

f Remove the TAR file from the nodes.

```
psh rm /tmp/sasep_security.tar
```

4 Restart the SAS Embedded Process.

a Disable the SAS Embedded Process to stop new queries from being started.

```
CALL SQLJ.SERVERCONTROL ('SAS', 'disable', :A);
```

b Query the status of the SAS Embedded Process until the status returns this message: `Hybrid Server is disabled with no UDFs running.`

```
CALL SQLJ.SERVERCONTROL ('SAS', 'status', :A);
```

c Shutdown the SAS Embedded Process.

```
CALL SQLJ.SERVERCONTROL ('SAS', 'shutdown', :A);
```

d Enable the SAS Embedded Process.

```
CALL SQLJ.SERVERCONTROL ('SAS', 'enable', :A);
```

e Test the SAS Embedded Process. The SAS Embedded Process will start when the next SAS query that uses the SAS Embedded Process is sent to the database.

For more information about stopping and starting the SAS Embedded Process for Teradata, see Controlling the SAS Embedded Process.

Prerequisites When Enabling Encryption for the SAS Data Connect Accelerator for Hadoop (on SAS Viya)

Here are the prerequisites for enabling encryption for the SAS Data Connect Accelerator for Hadoop (on SAS Viya).

- Upgrade the OpenSSL package on all Hadoop nodes to 1.0.1g or later to support TLS.
- Install SAS/ACCESS Interface to Hadoop (on Viya) and SAS In-Database Technologies for Hadoop (on SAS Viya).

  These offerings include the SAS Embedded Process and the SAS Data Connect Accelerator for Hadoop (on SAS Viya). For more information, see *SAS Viya for Linux: Deployment Guide*.
- Obtain TLS identity certificates (site-signed, third-party-signed, or self-signed) from the CAS controller machine. These certificates are located in the trustedcerts.pem file. Corresponding certificate authority (CA) certificates must be installed on the Hadoop nodes. If you use externally signed identity certificates in the CAS server, the Mozilla bundle of CA certificates that are provided by SAS can be deployed on the Hadoop nodes.

  For more information about the location of the trustedcerts.pem file, see "(Optional) Deploy TLS Certificates" in *SAS Viya for Linux: Deployment Guide*.

  For more information about configuring CAS, see "Configure CAS TLS to Use Custom Certificates (programming-only deployment)" on page 15 and "Configure CAS TLS to Use SAS Default Certificates (programming-only deployment)" on page 18. Certificates, keys, and passwords produced for authenticating to the SAS Embedded Process for Hadoop might, but do not need to coincide with such produced for other clients of the CAS server.
- When you installed the SAS Embedded Process, the following file and directory were created:

  EPInstallDir/sasexe/SASEPHome/security/dcsecurity.properties
  EPInstallDir/sasexe/SASEPHome/security/certs

- The ***EPInstallDir*/sasexe/SASEPHome/security/** directory should have `drwxr-xr-x` permissions.
- The ***EPInstallDir*/sasexe/SASEPHome/security/certs** directory should have `drwxr-xr-x` permissions.
- The dcsecurity.properties file should have `-rwxr-xr-x` permissions.

Enable Encryption for the SAS Data Connect Accelerator for Hadoop (on SAS Viya)

Follow these steps to encrypt data transfer between Hadoop and the CAS server using the SAS Data Connect Accelerator for Hadoop (on SAS Viya).

1 On Hadoop, modify the dcsecurity.properties file to enable SAS Data Connect Accelerator encryption.

   a Navigate to the ***EPInstallDir*/sasexe/SASEPHome/security/** directory.

   b Change the DCTCPMENCRYPT option in the dcsecurity.properties file as follows.

   ```
   -DCTCPMENCRYPT YES
   ```

   CAUTION
   The DCTCPMENCRYPT option is set on both the CAS server and on the data provider. How the option is set on both sides determines whether the data being transferred is encrypted or not. For more information, see "DCTCPMENCRYPT Option Setting Interaction" on page 38.

   c Add either the DCSSLCACERTDIR or DCSSLCALISTLOC option to the dcsecurity.properties file to specify either the location of the trusted certificate authorities or the public certificate(s) for trusted certificate authorities.

   Here is an example.

   ```
   -DCSSLCALISTLOC EPInstallDir/sasexe/SASEPHome/security/certs/certs-filename.pem
   ```

   For more information about the options, see "dcsecurity.properties File Options for Data Transfer with the SAS Data Connect Accelerator" on page 105.

2 Copy the necessary TLS CA certificates to the ***EPInstallDir*/sasexe/SASEPHome/security/certs** directory.

   - If your CA certificates already exist on the Hadoop cluster, copy the TLS CA certificates to this directory.
   - If your CA certificates exist on the CAS server, using a method of your choice, copy the CA certificates to this directory on the Hadoop cluster. In the following example, hdpclus1 is the name of the Hadoop cluster.

   ```
   scp CASCA1.pem username@hdpclus1: EPInstallDir/sasexe/SASEPHome/security/certs
   ```

   For more information about the location of the trustedcerts.pem file, see "(Optional) Deploy TLS Certificates" in *SAS Viya for Linux: Deployment Guide*.

   Note:

- The CA certificates on the Hadoop cluster must authorize the identity certificates that are specified on the CAS server.
- The ***EPInstallDir*/sasexe/SASEPHome/security/** directory should have `drwxr-xr-x` permissions.
- The ***EPInstallDir*/sasexe/SASEPHome/security/certs** directory should have `drwxr-xr-x` permissions.
- The dcsecurity.properties file should have `-rwxr-xr-x` permissions.

3 Use the sasep-admin.sh script to copy the contents of the ***EPInstallDir*/sasexe/SASEPHome/security/** directory to all nodes on the Hadoop cluster.

a Navigate to the ***EPInstallDir*/sasexe/SASEPHome/bin** directory.

```
cd EPInstallDir/sasexe/SASEPHome/bin
```

b Run the sasep-admin.sh script with the -security deploy argument.

```
./sasep-admin.sh -security deploy
```

This script deploys the SAS Data Connect Accelerator security settings to all nodes on the cluster. For more information, see "SASEP-ADMIN.SH Script" in *SAS Viya in Linux: Deployment Guide*.

Note: You can use `sasep-admin.sh –security deploy -force` to overwrite the current settings.

DCTCPMENCRYPT Option Setting Interaction

The DCTCPMENCRYPT option must be set for both the CAS server and the data provider. How the option is set on both sides determines whether the data being transferred is encrypted, the data is sent in plaintext, or the data transfer fails. The following table describes the interaction.

DCTCPMENCRYPT	CAS setting - YES	CAS setting - NO	CAS setting - OPT
Data provider setting - YES	Data transfer - encrypted	Data transfer - fails	Data transfer - encrypted
Data provider setting - NO	Data transfer - fails	Data transfer - plaintext	Data transfer - plaintext
Data provider setting - OPT	Data transfer - encrypted	Data transfer - plaintext	Data transfer - encrypted

You might want to use the OPT setting is if you have more than one cluster set up as a client. If you want one cluster to use encrypted data transfer and one cluster to use plaintext, you would set the DCTCPMENCRYPT option of the first cluster to YES and the DCTCPMENCRYPT option of the second cluster to NO. You would then set the DCTCPMENCRYPT option of the CAS server to OPT.

Note: During deployment of Viya 3.3, the DCTCPMENCRYPT option is set to OPT on the CAS server. You can change CAS server settings in the casconfig_usermods.lua file.

Updating the CAS Configuration File Options for Data Transfer

You can check the current run-time data transfer encryption settings of the CAS server by using the CAS Server Monitor. The settings are on the Runtime Environment panel of the System State page. For more information about the CAS Server Monitor, see "Using CAS Server Monitor" in *SAS Viya Administration: SAS Cloud Analytic Services*.

CAS server options are stored in a configuration file. During deployment, the casconfig_deployment.lua is created in the **/opt/sas/viya/config/etc/cas/default** directory from content provided in the vars.yml flie. When the sas-viya-casconrtroller-default service is started, the options in the lua file are processed.

Changes to data transfer encryption options such as the DCTCPMENCRYPT option should be made in the casconfig_usermods.lua file.

For a complete list of options, see "CAS Configuration File Options for Data Transfer with the SAS Data Connect Accelerator" on page 103.

Updating the dcsecurity.properties File Options for Data Transfer

On Hadoop or Teradata, the file options for data transfer encryption are located in the dcsecurity.properties file. The dcsecurity.properties file is located in the following directory on your cluster.

- For Teradata, **/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security**
- For Hadoop, ***EPInstallDir*/sasexe/SASEPHOME/security**

After you update the dcsecurity.properties file, copy the file to all nodes of the cluster.

- For Teradata, do a parallel file transfer to push the dcsecurity.properties file to all nodes.
- For Hadoop, Use the sasep-admin.sh script to copy the contents of the ***EPInstallDir*/sasexe/SASEPHome/security/** directory to all nodes on the Hadoop cluster. Run this command from the ***EPInstallDir*/sasexe/SASEPHome/bin** directory.

  ```
  ./sasep-admin.sh -security deploy
  ```

For a complete list of options, see "dcsecurity.properties File Options for Data Transfer with the SAS Data Connect Accelerator" on page 105.

Configure SAS 9.4 Clients to Work with SAS Viya

Configure a SAS 9.4 client to work with SAS Viya.

Note: You must have SAS administrator privileges to import certificates from SAS Viya.

1 Obtain the CA certificate that was used to sign the certificate that the CAS Server is using. On most SAS Viya deployments, the files can be found as follows:

- In a SAS Viya full deployment, the file needed is the vault-ca.crt file, located at **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts**. The vault-ca.crt file contains two certificates. The first certificate is the SAS Viya root CA certificate issued by Vault and the second certificate is the SAS Viya intermediate CA certificate issued by Vault.
- In a programming-only SAS Viya deployment, the trustedcerts.pem and trustedcerts.jks files are located at **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/trustedcerts**.

2 On a Windows client, you need to import the SAS Viya root CA certificate and the intermediate certificate into the Window's certificate stores. These files have to be imported into the Window's truststore one at a time. Because the vault-ca.crt file contains two files, the SAS Viya root CA certificate and the SAS Viya intermediate CA certificate, you need to create two unique files, one containing the root CA and the other containing the intermediate CA certificates. Use a text editor and cut-and-paste as appropriate.

   Each certificate in the file is denoted with a -----BEGIN CERTIFICATE----- and an -----END CERTIFICATE----- . Include the -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- header and footer in each of the two new files.

   For our example, we named our new certificate files example_root.cer and example-intermediate-ca.cer. Save these two files on your Windows machine and then add your certificates to the Windows CA stores.

3 If you have a Linux 9.4m5 client connecting to a CAS Server that is TLS enabled, perform the following steps:

   a Copy the SAS Viya CA certificates (vault-ca.crt or the trustedcerts files) to a location on your SAS 9.4 deployment where you can access the certificates. The directory structure where the SAS 9.4 trusted CA certificates (trustedcerts.pem or trustedcerts.jks) are found is at **<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem**.

   Note: Do not overwrite the existing trustedcerts files.

   b Append the contents of the SAS Viya vault-ca.crt file (or the SAS Viya trustedcerts file) to the end of the SAS 9.4 trustedcerts file. There are various ways to add your certificates to the trustedcerts files:

   - Use the SAS Deployment Manager to add your certificates to the trusted CA bundle.
   - Use a text editor to add your certificates to the trustedcerts file.

   c On the Linux server, set environment variable CAS_CLIENT_SSL_CA_LIST= to the trust list that the client uses to connect to the server.

   ```
   export CAS_CLIENT_SSL_CA_LIST=
   '<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem'
   ```

   If your SAS 9.4 client is SAS Studio, you can add the export statement to the sasenv_local file that is located at **/SASHome/SASFoundation/bin**.

   Note:

   In the December 2017 release of SAS 9.4M5, the CAS_CLIENT_SSL_CA_LIST= environment variable does not need to be set.

Manage Certificates

Use Best Practices to Manage Certificates

SAS recommends the following best practices for managing certificates and securing your private keys.

- Place your server identity certificate chained to any intermediate certificates in the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs** directory.

  Intermediate certificates need to be added to the server identity certificate in a certificate chain. The server identity certificate must be the first certificate in the chain. The intermediate certificate must be second. This order is important to allow validation with the private key to be successful.

- If your custom root certificate is site-signed or is not already included in the Mozilla bundle of trusted CA certificates, then you need to manually add the root certificate to the trustedcerts files under the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts directory**.

  You should also place a copy of the root certificate that you are adding to the trustedcerts files in the same directory. The root certificate should have a .crt file extension. This ensures that if the playbook needs to be rerun to update the installation, then this root certificate is automatically included in the regeneration of the trustedcerts files.

  For information, see "Add Your Certificates to the Trust List or to a Certificate Chain" on page 42.

  Note: Do not delete the trustedcerts.jks and the trustedcerts.pem files.

  Note: Add your root certificate to the trustedcerts.pem and trustedcerts.jks files on every machine in the deployment.

- Place your private server key in the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private** directory.
- Encrypt your private key when possible.
- Password-protect your private key file.
- Place your password in the encryption.key file as the first line of the file. The encryption key file is located at **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/**. This action ensures that if the playbook needs to be rerun to update the installation, the password in the encryption.key file will be used in the keys.lua file.

Managing Certificates Using Ansible Play Utilities

When using the Ansible playbook, the following utilities can be used to manage certificates on a SAS Viya full deployment. These utilities are run from the **sas_viya_playbook** directory.

rebuild-trust-stores.yml

On an Ansible controller machine, from the **/viya/sas_viya_playbook/** directory, run the rebuild-trust-stores.yml play to incorporate the customer CA bundle of trusted certificates (from Consul configuration) into the various truststore files (trustedcerts.pem and trustedcerts.jks) on each machine in the SAS Viya deployment.

```
ansible-playbook -i inventory.ini ./utility/rebuild-trust-stores.yml
```

distribute-httpd-certs.yml

This Ansible play adds your new custom certificate to **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts**. The play distributes copies of the certificate chain file to all machines with a name of httpproxy-**inventory name**-ca.crt. The play then rebuilds the trustedcerts.pem and trustedcerts.jks files and distributes them to every machine in the deployment.

On an Ansible controller machine, from the **sas_viya_playbook** directory, you can run the distribute-httpd-certs.yml play to distribute new certificates. On the Ansible controller machine, locate the utility file in the **/viya/sas_viya_playbook/utility** directory.

```
ansible-playbook -i inventory.ini ./utility/distribute-httpd-certs.yml
```

Note: This utility works on a full and programming-only deployment.

renew-security-artifacts.yml

IMPORTANT The renew-security-artifacts.yml playbook is not intended for general debugging. Please do not run this playbook. Instead, contact SAS Technical Support to obtain an updated version of this playbook and instructions to ensure a successful outcome.

On the Ansible controller machine, you can run the renew-security-artifacts.yml play to refresh the Vault CA certificates, tokens, keys, and server certificates. This play is located in the **/viya/sas_viya_playbook** directory.

```
ansible-playbook -i inventory.ini renew-security-artifacts.yml
```

Add Your Certificates to the Trust List or to a Certificate Chain

Add Certificates to the Trustedcerts File

SAS provides a trusted list of root CA certificates at installation. This trusted list includes the Mozilla bundle of CA certificates, the default Apache httpd certificates, and the CA certificates issued by Vault (only in a SAS Viya full deployment). There are two files named trustedcerts that contain the trusted list of certificates, trustedcerts.pem and trustedcerts.jks.

- In a SAS Viya deployment, the trusted certificates are found at **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts**.
- In a SAS 9.4 deployment, the trusted certificates are found at **<SASHome>/SASSecurityCertificateFramework/1.1/cacerts**.

Note: To ensure that the additional trusted certificates are not overwritten when you update your deployment, place your trusted root CA certificates in the directories shown above.

You can add additional root CA certificates to the trust list of certificates for the file format needed, JKS or PEM.

The following steps show how to add your CA root certificates, self-signed certificates, or your chained certificates to the trustedcerts.pem file.

1 You can use a text editor to add your certificates in any order to the trustedcerts.pem file. Here is an example template of certificates that a trustedcerts.pem file might contain.

```
<PEM encoded sascaroot>
-----BEGIN CERTIFICATE-----
certificate string
-----END CERTIFICATE-----
<PEM encoded sassha2rootca>
-----BEGIN CERTIFICATE-----
certificate string
-----END CERTIFICATE-----
<PEM encoded digicertrootca>
-----BEGIN CERTIFICATE-----
certificate string
-----END CERTIFICATE-----
<PEM encoded custom_ca_chain>
-----BEGIN CERTIFICATE-----
certificate string
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
certificate string
-----END CERTIFICATE-----
```

The content of the digital certificate in this example is represented as `<PEM encoded certificate>`. The content of each digital certificate is delimited with a `- - - - -BEGIN CERTIFICATE- - - - -` and `- - - - -END CERTIFICATE- - - - -` pair. All text outside the delimiters is ignored. Therefore, you might not want to use delimited lines for descriptive comments.

You can also concatenate the certificate authority files. For example, you can concatenate a root authority certificate file and a primary certificate file into a single PEM file. Here are two examples of concatenating certificates:

```
cat custom_ca_chain.pem >> trustedcerts.pem
cat vault-ca.crt >> trustedcerts.pem
```

Note: You can place these files in any order.

2 Because the digital certificate is encoded, it is unreadable. To view the file contents, you can use the following OpenSSL commands for your file type:

```
openssl x509 -in /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/cacerts/trustedcerts -text -noout
```

The following steps show how to add certificates to the trustedcerts.jks file (the Java truststore). Use the `keytool` command to add the certificates to the Java truststore. In this example, we assume that you have obtained a certificate from a CA not included in the truststore.

1 Locate the default truststore for your Java applications.

   - In a SAS Viya deployment, the trusted certificates are found at **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/trustedcerts.jks**.
   - In a SAS 9.4 deployment, the trusted certificates are found at **<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.jks**.

2 Import the CA certificate into the default truststore. In our example, we are assuming that the file root_ca.pem contains the CA's certificate. Use the following commands to import a root CA certificate (root_ca.pem in our example) into the default truststore.

```
$ keytool -importcert -file /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/cas/shared/default/custom.crt
-alias CAroot -keystore
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/
cacerts/trustedcerts.jks -storepass changeit
```

Note: Do not delete the trustedcerts files.

Note: Add your root certificate to the trustedcerts.pem and trustedcerts.jks files on every machine in the deployment.

For more information about how to manage your certificates and protect your keys, see "Manage Certificates" on page 41.

Add Certificates to the Certificate Chain

The list of TLS certificates, from the root certificate to the end-user certificate, represents the TLS certificate chain. We take the server identity certificates and chain the intermediate certificates with them. In a SAS Viya deployment, these chained certificate files are placed in the**/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs** directory.

For detailed information about how to manage your certificates and protect your keys, see "Manage Certificates" on page 41.

The following steps show how to add your intermediate certificates to the server identity certificates file to create a chain:

1 You can use a text editor to add your certificates to a file in PEM format called server.crt.

   Chaining files contain the server certificate first, the intermediate certificate that validates the server certificate next, the intermediate certificate that validates the first intermediate certificate next, and so on, down the chain all the way to the root certificate. The root certificate does not need to be in this file.

   Here is a template of what a chained server.crt file might contain.

```
(Your Server Id Certificate)
 - - - - -BEGIN CERTIFICATE- - - - -
<PEM encoded certificate>
- - - - -END CERTIFICATE- - - - -
```

```
(Intermediate Certificate(s))
 - - - - -BEGIN CERTIFICATE- - - - -
<PEM encoded certificate>
  - - - - -END CERTIFICATE- - - - -
(Intermediate Certificate(s))
 - - - - -BEGIN CERTIFICATE- - - - -
<PEM encoded certificate>
  - - - - -END CERTIFICATE- - - - -
```

The content of the digital certificate in this example is represented as `<PEM encoded certificate>`. The content of each digital certificate is delimited with a `- - - - -BEGIN CERTIFICATE- - - - -` and `- - - - -END CERTIFICATE- - - - -` pair. All text outside the delimiters is ignored. Therefore, you might not want to use delimited lines for descriptive comments.

You can also concatenate the certificate files. For example, you can concatenate an intermediate authority certificate file and a server certificate file into a single PEM file. Here is an example of concatenating certificates:

```
cat int_ca.crt >> server_id.crt
```

2 Because the digital certificate is encoded, it is unreadable. To view the file contents, you can use the following OpenSSL commands for your file type:

```
openssl x509 -in /config/etc/SASSecurityCertificateFramework/
tls/certs/server_id.crt -text -noout
```

Generate Site-Signed or Third-Party-Signed Certificates in PEM Format

You need to create two files, a private key file and a certificate file.

private key
: This private key is in RSA format and is saved in ASCII (Base64-encoded) PEM (Privacy Enhanced Mail) format.

third-party-signed certificate
: A certificate authority (CA) is a trusted third party. This certificate contains the CA's public key in X.509 certificate form and is saved in ASCII (Base64-encoded) PEM format.

SAS recommends the following best practices for managing certificates and securing your private keys for the CAS server.

- Place your server identity certificates in the **/config/etc/SASSecurityCertificateFramework/tls/certs** directory.

  Intermediate certificates need to be added to the server identity certificate in a certificate chain. The server identity certificate must be the first certificate in the chain. The intermediate certificate must be second. This order is important to allow validation with the private key to be successful.

- If your custom root certificate is site-signed or is not already included in the Mozilla bundle of trusted CA certificates, then you need to manually add the root certificate to the trustedcerts files under the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts directory**.

You should also place a copy of the root certificate that you are adding to the trustedcerts files in the same directory. The root certificate should have a .crt file extension. This ensures that if the playbook needs to be rerun to update the installation, then this root certificate is automatically included in the regeneration of the trustedcerts files.

For information, see "Add Your Certificates to the Trust List or to a Certificate Chain" on page 42.

Note: Do not delete the trustedcerts.jks and the trustedcerts.pem files.

Note: Add your root certificate to the trustedcerts.pem and trustedcerts.jks files on every machine in the deployment.

- Place your private server keys in the **/config/etc/SASSecurityCertificateFramework/private** directory structure and reference this directory location in the environment variables that you are setting.
- Encrypt your private key when possible.

Note: This example is one way of possibly several to generate certificates for use with TLS. Consult your administrator for details about what is required for your site.

Generate site-signed or third-party-signed certificates in PEM format.

1 Decide which type of CA to use at your site.
   - site-signed
   - third-party-signed

2 Change the directory to where your OpenSSL commands reside. For example:

```
cd /usr/bin
```

3 Use the following OpenSSL command to generate a new private key in RSA format and a CA certificate signing request in PEM format. Store your private key in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private**.

```
openssl req -new -out /opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs/
certreq.csr-newkey rsa:2048 -keyout /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/private.key -nodes
```

It is recommended that you supply an encrypted password on the key file. To do so, submit the following request.

```
openssl rsa -aes128 -in /opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/
private.key -out /opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/
tempprivate.key -passout pass:password
```

```
mv opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/tempprivate.key /opt/sas/
viya/config/etc/SASSecurityCertificateFramework/private/private.key
```

4 Verify your Certificate Signing Request (CSR).

```
openssl req -noout -text -in /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/certreq.csr
```

5 Submit your CSR file (certreq.csr) to your CA. This CA can be a CA at your site or a third party. You should receive the following certificates from your CA.

- signed certificate (containing the CA's public key)
- CA root certificate
- One or more CA intermediate certificates

6 Store the signed certificates from your CA in **opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs**.

7 Add your site-signed root CA certificates to the truststore. See "Add Your Certificates to the Trust List or to a Certificate Chain" on page 42.

See Also

For an example of using OpenSSL to generate site-signed or third-party-signed certificates in PEM format, see "Create Site-Signed or Third-Party-Signed Certificates in PEM Format" on page 107.

Generate Site-Signed or Third-Party-Signed Certificates in Java Keystore Format

The following steps create site-signed or third-party-signed certificates in Java keystore (JKS) format. Details of each step are shown after this summary.

1 Create the machine's keystore.

2 Create a certificate signing request (CSR).

3 Submit a .csr file to a CA.

4 Receive a signed certificate, CA root certificate, and one or more CA intermediate certificates.

5 Add the server's identity certificate to the keystore.

6 Add the CA intermediate certificate to the keystore.

Note: This example is one way of possibly several to generate certificates for use with TLS. Consult your administrator for details about what is required for your site.

The keystore contains private keys and certificates used by TLS servers to authenticate themselves to TLS clients. By convention, such files are referred to as keystores.

SAS recommends the following best practices for managing certificates for Java.

- The signed certificate and private key are contained in one JKS format file. Add your certificates to the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/java/jks** directory.
- Password-protect the private key.
- Password-protect the keystore. In the following example the keystore file is named keystore.jks.
- Make the keystore file readable only by members of the appropriate group.
- Make the file where the keystore password is referenced readable only by members of the appropriate group. For example, you might make the init_usermods.properties file (where the password is referenced by a keystore password property)) readable only by members of the appropriate group.

You can obtain site-signed or third-party-signed certificates using the Java Keytool. In the following scenario that we are using a certificate authority (CA) as our third party.

1 Log on to your machine as a user with root or sudo privileges.

2 Change the directory to where your `keytool` command resides. For example:

```
cd $JAVA_HOME/bin
```

3 Use the `keytool` command to create a new private key and keystore and store the information in the keystore file named keystore.jks. In the following example, we are first generating a private key server.key. We are also using alias server.

```
keytool -genkey -alias server -keyalg RSA -keystore
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/
java/jks/keystore.jks -storepass changeit -keypass password
-validity 360 -keysize 2048
```

The keystore password (which protects the keystore as a whole) and the key password (which protects the private key stored in the server entry) are set using the -storepass and -keypass options respectively.

Change the permissions on the keystore file (keystore.jks) to be readable only by members of the appropriate group. Use chmod or sudo to change the permissions.

```
chmod 600 keystore.jks
```

When you list the file, you see the permissions are Read/Write only -rw------- .

4 To query the contents of your Java keystore file, you can use the following command:

```
keytool -list -v -keystore /opt/sas/viya/config/etc
/SASSecurityCertificateFramework/java/jks/keystore.jks
-storepass changeit -keypass password
```

5 Use the `keytool` command to create certificate signing request (CSR) for an existing keystore. Here is an example command:

```
keytool -certreq -alias server -keystore
/opt/sas/viya/config/etc/SASSecurityCertificateFramework
/java/jks/keystore.jks -file
/opt/sas/viya/config/etc/SASSecurityCertificateFramework
/java/jks/server.csr -storepass changeit -keypass password
```

This command generates the CSR and stores it in a file called server.csr.

6 Submit your CSR file to your CA. For our example, we have provided a name for each of the signed certificates that we might receive, server_ca.pem, root_ca.pem, and int_ca.pem. You should receive the following from your CA:

- signed identity certificate (server_ca.pem)
- CA root certificate (root_ca.pem)
- one or more CA intermediate certificates (int_ca.pem)

7 After you have submitted your CSR to the CA and received the CA's reply (containing the signed certificate), import the reply into your keystore, located at **`/opt/sas/viya/config/etc/SASSecurityCertificateFramework/java/jks`**, using the following `keytool` options.

This step imports the signed server identity certificate and one or more intermediate certificates in PEM format into the keystore.

a Add the server identity certificate to your keystore. In this example, server_cert.pem is the server identity certificate.

```
keytool -importcert -file /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/java/jks/server_ca.pem
-keystore /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
java/jks/keystore.jks -storepass changeit -keypass password
-trustcacerts -alias server_ca
```

b If your server certificate is signed by an intermediate CA, import the intermediate certificate into your keystore file. In this example, int_ca.pem is the CA intermediate certificate.

```
keytool -importcert -file /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/java/jks/int_ca.pem -keystore
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/java/
jks/keystore.jks -storepass changeit -keypass password
-trustcacerts -alias int_ca
```

c Verify that the certificates that you added to your keystore are present.

```
keytool -v -list -keystore /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/java/jks/keystore.jks
-storepass changeit -keypass password
```

Generate Self-Signed Certificates

Self-signed certificates are signed by your own private key, rather than by an external CA. You can generate self-signed certificates or root certificates in PEM format using RSA or HMAC encryption or in Java keystore format.

A private key file and a self-signed certificate are needed.

private key
: This private key is in RSA format and is saved in ASCII (Base64-encoded) PEM format.

self-signed certificate
: This certificate contains a public key in X.509 certificate form and is saved in ASCII (Base64-encoded) PEM format.

SAS recommends the following best practices for managing certificates and securing your private keys.

- Place your server identity certificates in the **/config/etc/SASSecurityCertificateFramework/tls/certs** directory.

  Intermediate certificates are added to the server identity certificate in a certificate chain. The server identity certificate must be the first certificate in the chain. The intermediate certificate must be second. This order is important to allow validation with the private key to be successful.

- Place your root certificates and trusted CA certificates in the **/config/etc/SASSecurityCertificateFramework/cacerts** directory.

- Place your private server keys in the **/config/etc/SASSecurityCertificateFramework/private** directory structure and reference this directory location in the environment variables that you are setting.

- Encrypt your private key when possible.
- For Java, store your private key and signed certificates in **opt/sas/viya/config/etc/SASSecurityCertificateFramework/java/jks**.

Generate self-signed certificates or root certificates in PEM format using RSA encryption.

Note: This example is one of several possible ways to generate certificates for use with TLS. Consult your administrator for details about what is required for your site.

1 Change the directory to the directory where your OpenSSL commands reside. For example:

```
cd /usr/bin
```

2 Use the following OpenSSL command to generate a new private key using RSA encryption and a self-signed certificate:

```
openssl req -x509 -newkey rsa:2048 -keyout
/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/private.key
-out /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tls/certs/certreq.csr -days 1000
```

It is recommended that you supply an encrypted password on the key file. To do so, submit the following request:

```
openssl rsa -aes128 -in /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/private.key
-out /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
private/tempprivate.key -passout pass:password
mv opt/sas/viya/config/etc/SASSecurityCertificateFramework/
private/tempprivate.key /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/private.key
```

3 If you need to add your certificates to the trusted list of certificates, see "Add Your Certificates to the Trust List or to a Certificate Chain" on page 42..

Generate self-signed certificates in Java keystore format.

Note: This example is one of several possible ways to generate certificates for use with TLS. Consult your administrator for details about what is required for your site.

1 For servers based on Java, generate a self-signed certificate using keytool -genkeypair. This command creates a public/private key pair and wraps the public key into a self-signed certificate. For example, the following command creates a self-signed test certificate for the host and stores it in a keystore. For this example, we are using alias **javahost**.

```
$ keytool -genkeypair -keystore
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/
java/jks/javahost.jks -keyalg RSA -alias javahost
-dname "CN=javahost.example.com,O=Hadoop" -storepass changeit
-keypass password -validity 1000
```

Note: By default, self-signed certificates are valid for only 90 days. To increase this period, replace the previous command's `-validity <val_days>` parameter to specify the number of days for which the certificate should be considered valid.

2 If you need to add your certificates to the trusted list of certificates, see"Add Your Certificates to the Trust List or to a Certificate Chain" on page 42.

Convert Digital Certificate File Formats Using OpenSSL

In OpenSSL, you can use many parameters to convert between the different digital certificate file formats. The following are some examples of a few ways to convert files from one format to another. See OpenSSL Commands for more commands that can be used.

Convert DER to PEM File Format

Many certificate authorities provide certificates in DER format. If you have a DER formatted file, but need a PEM formatted file, you can convert the DER formatted file to PEM format using OpenSSL.

Note: You must convert a DER formatted file to PEM format before you can include it in a trust list on Linux.

Here is an example of how to convert a server digital certificate from DER input format to PEM output format:

```
x509 -inform DER -outform PEM -in certificate.cer -out certificate.pem
```

Convert PEM Encoded Certificate to DER File Format

If you have a PEM formatted file, but need a DER formatted file, you can convert the PEM formatted file to DER using OpenSSL.

Here is an example of how to convert a server digital certificate from PEM input format to DER output format:

```
x509 -outform der -in certificate.pem -out certificate.der
```

Convert PEM to PK12 File Format

If you are using a Java application that accepts only PKCS#12 format, you might need to convert your PEM formatted file that includes certificates and the separate key file to one file that includes both the certificate and the key file.

If you have a PEM formatted certificate file, but need a PK12 formatted file, you can convert the PEM format certificate to a PK12 format using OpenSSL. Here is one way of converting a PEM to a PK12 formatted file for non-FIPS (Federal Information Processing Standard) libraries.

```
pkcs12 -export -out path/certificate.p12 -inkey path/pivatekey.pem
-in path/certificate.pem -certfile certs.pem
```

```
pkcs12 -export -in server.cer -inkey server.key -out keystore.p12
-name server
```

Secure Credentials in the CAS Server (cas.servicesbaseurl)

Note: This section is applicable only if you have a full deployment. If you have a programming-only deployment, skip this section.

The URL that enables a CAS server to use SAS Viya services is set using the cas.SERVICESBASEURL= option. For example, CAS client credentials are passed to the SASLogon service at the address specified in the cas.SERVICESBASEURL= option in order to obtain an OAuth token.

This option is set in the casconfig.lua file located at **/opt/sas/viya/config/etc/cas/default/**.

1 In the casconfig.lua file, ensure that the HTTPS URL is used to access the Apache HTTP server machine.

```
cas.servicesbaseurl='https://webserver-host-name'
```

Note: The host name in the URL is the same as the Common Name used in the server identity certificate that Apache HTTP Server is using.

Note: In a SAS Viya full deployment, the cas.SERVICESBASEURL= option defaults to port 443 for HTTPS access.

2 When you set the cas.SERVICESBASEURL= option to use HTTPS, the CAS_CALISTLOC= environment variable needs to be set in the casconfig_usermods.lua file to point to the CA certificates that the Apache HTTP Server is using.

```
env.CAS_CALISTLOC=
'/path-to-CA-chain-used-for-Apache-HTTP-Server-certificate'
```

Note: If the CA certificates are already imported in the OpenSSL truststore, setting the env.CAS_CALISTLOC= environment variable is not necessary.

3 If you are setting the CAS_CALISTLOC= environment variable, you should copy the change made to this environment variable to the vars.yml file. This change ensures that your settings are not changed when upgrades are made to the deployment.

Note: See "Modify the vars.yml File" in *SAS Viya for Linux: Deployment Guide* for more details.

Add the following highlighted variables and their respective values:

```
CAS_CONFIGURATION:
   env:
     #CAS_DISK_CACHE: /tmp
     CAS_CLIENT_SSL_REQUIRED: 'true'
```

```
    CAS_CALISTLOC: path-to-CA-chain-used-for-Apache-HTTP-Server-certificate
  cfg:
    #gcport: 5580
    #httpport: 8777
    #port: 5570
    #colocation: 'none'
    servicesbaseurl: 'https://http-proxy-host-name'
```

Save and close the vars.yml file.

For information about using cas.SERVICESBASEURL=, see See "Configuration File Options Reference" in *SAS Viya Administration: SAS Cloud Analytic Services*.

Manage Tokens and Create JWT Signing Keys

Generate Signing Keys for JSON Web Tokens

Overview

A JSON Web Token (JWT) is a JSON object that is defined in RFC 7519 as a safe way to pass a set of information between two parties. Access tokens issued by SAS Logon Manager are also OpenID Connect ID tokens, which are JWTs.

The token consists of three parts: a header, claims, and a signature. All of these parts are base64 encoded. Here is what an example token might look like. Each part is separated by a period to create header.claims.signature.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

In a new Viya deployment, SAS Logon Manager generates RSA keys and CAS gets the public key that it needs from SAS Logon Manager automatically if the cas.SERVICESBASEURL= property is set. You can configure your own signing keys, overriding the SAS Logon Manager behavior. You can also supply a passphrase that is then hashed using HMAC.

To configure your JWT keys, the following configuration file options and properties need to be set for CAS and SAS Logon Manager.

cas.OAUTHSIGNINGKEY=
: The signing key must be either a Base64-encoded RSA private key that is used to digitally sign tokens, or a passphrase. See "Configuration File Options" in *SAS Viya Administration: SAS Cloud Analytic Services*.

  Note: CAS gets the public key that it needs from SAS Logon Manager automatically if the cas.SERVICESBASEURL= is set.

cas.OAUTHSIGNINGCERTIFICATE=
: The RSA public key corresponding to the RSA private key being used to digitally sign tokens. See "Configuration File Options" in *SAS Viya Administration: SAS Cloud Analytic Services*.

sas.logon.jwt
The set of properties that are used to secure JSON web tokens with RSA digital signatures or hashed message authentication codes (HMACs). For a description of the properties, see "Configuration Properties: Reference (Applications)" in *SAS Viya Administration: Configuration Properties*.

Generate JWT Signing Key

The following example uses OpenSSL to generate an RSA signing key.

Note: This example is one way of many to generate RSA signing keys. Consult your administrator for details about what is required for your site.

1 Change the working directory to the directory where SAS stores keys. SAS stores keys in the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework** directory structure. For example:

```
cd /opt/sas/viya/config/etc/SASSecurityCertificateFramework
```

2 Use the following OpenSSL command to generate a new RSA private key.

```
openssl genrsa -out /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/jwt-private.key 2048
```

3 Create a public directory.

```
mkdir public/
```

4 Extract the RSA public key. Submit the following request:

```
openssl rsa -in "./private/jwt-private.key" -out
"./public/jwt-public.key" -pubout/public/jwt-public.key
```

5 Copy the public key to the signingKey property using SAS Environment Manager. See "Configure the SAS Logon Manager with New JWT Signing Key" on page 55.

6 You can add the following environment variables to casconfig_usermods.lua file.

- Add cas.OAUTHSIGNINGCERTIFICATE= to the casconfig_usermods.lua file at **/opt/sas/viya/config/etc/cas/default**.

  ```
  cas.oauthsigningcertificate="/opt/sas/viya/config/etc/
  SASSecurityCertificateFramework/public/jwt-public.key"
  ```

- Add cas.OAUTHSIGNINGKEY= to the keys.lua file located at **/opt/sas/viya/config/etc/cas/default**. The keys.lua file has limited access and is more secure.

  ```
  cas.oauthsigningkey='zAcSGqF23Fu85e7qz7ZN2U4ZRhfV3W\
  pwPAoE3Z7kBw&SsiodoUaIvY8ltyTt5jkRh4J50vUPWVHaR7YPi5jC'
  ```

  Ensure that the permissions on the keys.lua file is 600 and is readable only by CAS service account.

  ```
  chmod 600 keys.lua
  ```

For information about the CAS configuration file options, see "Configuration File Options" in *SAS Viya Administration: SAS Cloud Analytic Services*.

The following example uses a simple passphrase. The server will use HMAC to generate the digital signature from the passphrase.

```
cas.oauthsigningkey="57443a4c052350a44638835d64fd66822f813319"
```

Configure the SAS Logon Manager with New JWT Signing Key

Use the SAS Environment Manager to set configuration properties that are used by the SAS Logon Manager. If you created a new JWT signing key, paste the key into the signingKey property.

1 From the side menu, select **SAS Environment Manager**.

2 In the navigation bar, click

The Configuration page is an advanced interface. It is available to only SAS Administrators.

3 The default view is **Basic Services**. Select **Definitions** from the drop-down box.

4 In the **Definitions** list, select **sas.logon.jwt**.

5 If the definition has no properties configured, complete the following:

a In the top right corner of the window, click .

b In the New sas.logon.jwt Configuration dialog box, paste the PEM-encoded RSA private key or passphrase into the value for the signingKey property.

For a description of the properties, see "Configuration Properties: Reference (Applications)" in *SAS Viya Administration: Configuration Properties*.

c Click **Save**.

Note: The system will take a few minutes to recognize the new key before starting to use the new key.

Obtain an Access Token to Register a New Client ID

You can use a curl command to obtain a token. You can then use that token to register a new client ID.

Consul tokens can be found at **`/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default`** and is named client.token.

An example curl command to request a registration token for a new client follows. In this example, the client is named APP.

```
curl -X POST "http://localhost/SASLogon/oauth/
clients/consul?callback=false&serviceId=app"
-H "X-Consul-Token: 29c4700f-ea89-41cd-8bc4-4198ccaa5bf9"
```

Note:

This request must pass a `callback=false` query string parameter and authenticate directly by passing a SAS Configuration Server (Consul) token. If the Consul token is valid, SASLogon returns the registration token in the response.

By default, tokens are valid for 12 hours. When you register the client ID, you can configure the amount of time that the token is valid. See how to register a client ID in "Authentication: How To" in *SAS Viya Administration: Authentication*.

For more information about using curl, see Curl Documentation.

Replace Tokens for SAS Configuration Server (Consul)

Overview

At installation, tokens are generated and placed in the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default** directory. Client, encryption, and management tokens are provided. The owner and group of these files is SAS.

client.token
: is the ACL client token that is used by all services to access values in the Key/Value store.

management.token
: is the ACL management token (acl_master_token) that is used to administer the ACLs.

encryption.token
: specifies the secret key that is used for encryption of Consul network traffic. Used for gossip communication.

You must use the value of an ACL token that is of type management to administer Consul ACLs. The value of this management token is created by the Ansible playbook and stored in the management.token file at **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default**.

Replace ACL Tokens

You must use the value of an ACL token that is of type management to administer Consul ACLs. In the following example, the **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default/management.token** file contains the ID of a management ACL that we want to change.

We are using the sas-bootstrap-config CLI to replace ACL tokens.

Note: You can also use the Consul ACL HTTP CLI to manage ACL tokens. For more information, see ACL HTTP Endpoint.

1 Source the /etc/profile.d/lang.sh to set the LANG environment variable. It will be set to a value such as en_US.UTF-8

```
source /etc/profile.d/lang.sh
```

2 The **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default/management.token** file contains the ID of a management ACL that we want to change.

```
sudo cat /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tokens/consul/default/management.token
0329addc-bb72-489c-9f0a-5421890dd2fb
```

3 Create a backup copy of the original management.token.

```
sudo cp /opt/sas/viya/config/etc/
```

```
SASSecurityCertificateFramework/tokens/consul/default/
management.token /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tokens/consul/default/
management.token.OLD
```

4 List the ACLs using the sas-bootstrap-config CLI.

```
sudo /opt/sas/viya/home/bin/sas-bootstrap-config
--token-file /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tokens/consul/default/
management.token acl list
```

These are the ACLs listed.


```
 {
        "CreateIndex": 4,
        "ModifyIndex": 4,
        "ID": "0329addc-bb72-489c-9f0a-5421890dd2fb",
        "Name": "Master Token",
        "Type": "management",
        "Rules": ""
    },
    {
        "CreateIndex": 3,
        "ModifyIndex": 64718,
        "ID": "anonymous",
        "Name": "Anonymous Token",
        "Type": "client",
        "Rules": "{\"service\":{\"\":{\"Policy\":\"read\"}}}"
    },
    {
        "CreateIndex": 19,
        "ModifyIndex": 64702,
        "ID": "eaa6de8a-3824-4c8f-a73a-dbd835c5cc97",
        "Name": "client",
        "Type": "client",
        "Rules": "{\"key\":{\"\":{\"Policy\":\"write\"}},\"service\":
        {\"\":{\"Policy\":\"write\"}},\"event\":{\"\":{\"Policy\":\"write\"}},
        \"query\":{\"\":{\"Policy\":\"write\"}}}"
    }
```


5 Clone the management token using the following command. The c43b7d1a-ccee-3792-a1d8-9576a9dbe7d2 ID is returned by the execution of the following code. This ID is the new value that is inserted into the management.token file at **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default**.

```
sudo /opt/sas/viya/home/bin/sas-bootstrap-config --token-file
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tokens/consul/default/management.token acl clone --acl-id
$(sudo cat /opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tokens/consul/default/management.token)
{
    "ID": "c43b7d1a-ccee-3792-a1d8-9576a9dbe7d2"
}
```

6 List the ACLs again to verify that the new management ACL has been created.

```
sudo /opt/sas/viya/home/bin/sas-bootstrap-config --token-file
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/
```

```
consul/default/management.token acl list
```

Here are the ACLs listed now.

```
{
        "CreateIndex": 4,
        "ModifyIndex": 4,
        "ID": "0329addc-bb72-489c-9f0a-5421890dd2fb",
        "Name": "Master Token",
        "Type": "management",
        "Rules": ""
    },
    {
        "CreateIndex": 3,
        "ModifyIndex": 64899,
        "ID": "anonymous",
        "Name": "Anonymous Token",
        "Type": "client",
        "Rules": "{\"service\":{\"\":{\"Policy\":\"read\"}}}"
    },
    {
        "CreateIndex": 64927,
        "ModifyIndex": 64927,
        "ID": "c43b7d1a-ccee-3792-a1d8-9576a9dbe7d2",
        "Name": "Master Token",
        "Type": "management",
        "Rules": ""
    },
    {
        "CreateIndex": 19,
        "ModifyIndex": 64897,
        "ID": "eaa6de8a-3824-4c8f-a73a-dbd835c5cc97",
        "Name": "client",
        "Type": "client",
        "Rules": "{\"key\":{\"\":{\"Policy\":\"write\"}},
        \"service\":{\"\":{\"Policy\":\"write\"}},\"event\":
        {\"\":{\"Policy\":\"write\"}},\"query\":{\"\":{\"Policy\":\"write\"}}}"
    }
```

7 Replace the value in the management.token file with the value that was returned from the clone command.

```
sudo bash -c 'echo c43b7d1a-ccee-3792-a1d8-9576a9dbe7d2 >
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/
consul/default/management.token'
```

8 Destroy the old management ACL.

```
sudo /opt/sas/viya/home/bin/sas-bootstrap-config
--token-file /opt/sas/viya/config/etc/SASecurityCertificateFramework/
tokens/consul/default/management.token acl destroy --acl-id $(sudo cat
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/
tokens/consul/default/management.token.OLD)
```

9 After the new ACLs have been created in Consul and the management.token and client.token files have been updated with the new values, copies of the original .token files can be deleted.

Replace ACL Tokens Using the sas-crypto-management Tool

You can use the sas-crypto-management application located at **/opt/sas/viya/home/SASSecurityCertificateFramework/bin/** to generate a value that can be used as the ID for an ACL.

```
sudo /opt/sas/viya/home/bin/SASSecurityCertificateFramework/
bin/sas-crypto-management uuid --out-file /opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tokens/consul/default/client.token
```

You can then use the value of the ACL ID that was generated using the sas-crypto-management tool (instead of the value that Consul generates using its clone command as shown in "Replace ACL Tokens" on page 56). Then, use the `create` command to specify the ID that should be used.

Replace an Encryption Token

All Consul agents that are running as servers or clients need to have an encryption key. The Consul agent supports encrypting all of its network traffic. The SASSecurityCertificateFramework provides the encryption token that is used for gossip communication. Enabling gossip encryption requires only that you set an encryption key when starting the Consul agent.

The Consul RPM start script generates a file named config-gossip.json in **/opt/sas/viya/config/etc/consul.d**. The consul RPM uses the value obtained from the gossip.token file in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default**. You can see the type of information contained in the file by submitting the following command:

```
sudo cat /opt/sas/viya/config/etc/consul.d/config-gossip.json
```

The generated file contains encryption information that looks like the following. The encryption key is 16-bytes and Base64 encoded.

```
{     "encrypt": "y/k+KRpeZZVmzHCVrvbR6A==" }
```

The -encrypt option specifies the secret key to use for encryption of Consul network traffic. This key must be 16-bytes that are Base64-encoded. All nodes within a cluster must share the same encryption key to communicate. The provided key is automatically persisted to the data directory and loaded automatically whenever the agent is restarted. More information about this option can be found at Consul Configuration Command-line Options.

There are situations when the encryption token might need to be replaced.

1. Sign on to the machine that runs the SAS Configuration Server (Consul) as the SAS install user (sas) or with sudo privileges.
2. Consul provides the consul `keygen` command to generate a new key.

   ```
   consul keygen
   ```
3. Copy the value that is generated (for example, X4SYOinf2pTAcAHRhpj7dA==) and open config-gossip.json located at **/opt/sas/viya/config/etc/consul.d/**.

   Use the copied string as the value for the encrypt parameter:

   ```
   "encrypt": "X4SYOinf2pTAcAHRhpj7dA=="
   ```
4. Stop Consul.

   ```
   sudo service sas-viya-consul-default stop
   ```
5. Delete the local.keyring and remote.keyring files in **/opt/sas/viya/config/data/consul/serf**.

All nodes within a cluster must share the same encryption key to communicate. The provided key is automatically persisted to the data directory and loaded automatically whenever the agent is restarted. This option is provided on each agent's initial start-up sequence. The value of this secret key is persisted to the **/opt/sas/viya/config/data/consul/serf** directory to files local.keyring and remote.keyring.

Note: If a key is provided after Consul has been initialized with an encryption key, then the provided key is ignored and a warning is displayed.

6 Restart SAS Configuration Server (Consul).

```
sudo service sas-viya-consul-default restart
```

When Consul is restarted, the Consul RPM start script regenerates the config-gossip.json file and Consul reads this value and re-creates the local.keyring and remote.keyring files.

You can read about how this is done for Consul at Encryption for Consul.

Concepts

SAS/SECURE

SAS/SECURE Overview

Refer to "NETENCRYPT System Option" on page 74 and "NETENCRYPTALGORITHM= System Option" on page 75 for details. You can specify various encryption algorithms as well as TLS to secure data in motion.

Linux supports the following encryption algorithms:

- RC2
- RC4
- DES
- TripleDES
- AES

Refer to "Encryption Algorithms" on page 72 for more information about encryption algorithms supported for use with SAS/SECURE.

SAS/SECURE Software Availability

For software delivery purposes, SAS/SECURE is a product within the SAS System. SAS/SECURE is included with the SAS Viya software. In prior releases, SAS/SECURE was an add-on product that

was licensed separately. This change makes strong encryption available in all deployments (except where prohibited by import restrictions).

SAS/SECURE Export Restrictions

For U.S. export purposes, SAS designates each product based on the encryption algorithms and the product's functional capability. SAS/SECURE is available to most commercial and government users inside and outside the U.S. However, some countries (for example, Russia, China, and France) have import restrictions on products that contain encryption, and the U.S. prohibits the export of encryption software to specific embargoed or restricted destinations.

SAS/SECURE for Linux includes the following encryption algorithms.

- RC2 using up to 128-bit keys
- RC4 using up to 128-bit keys
- DES using up to 56-bit keys
- TripleDES using up to 168-bit keys
- AES using 256-bit keys

SAS/SECURE Installation and Configuration

SAS/SECURE is installed and delivered on every installation. Whether SAS/SECURE is used depends on the options that are set.

To use encryption provided by SAS/SECURE for communications and networking, specify the NETENCRYPT system option and set the NETENCRALG= system option to a value of RC2, RC4, DES, TRIPLEDES, AES, or SSL. Refer to "NETENCRYPT System Option" on page 74 and "NETENCRYPTALGORITHM= System Option" on page 75.

Transport Layer Security (TLS)

Transport Layer Security (TLS) Overview

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that are designed to provide communication security. TLS and SSL are protocols that provide network data privacy, data integrity, and authentication.

Note: All discussion of TLS is also applicable to the predecessor protocol, Secure Sockets Layer (SSL).

TLS uses X.509 certificates and hence asymmetric cryptography to assure the party with whom they are communicating and to exchange a symmetric key. As a consequence of choosing X.509 certificates, certificate authorities and a public key infrastructure are necessary to verify the relation

between a certificate and its owner, as well as to generate, sign, and administer the validity of certificates. For information about certificates, see “Certificates” on page 64.

In addition to providing encryption services, TLS performs client and server authentication, and it uses message authentication codes to ensure data integrity. The client requests a certificate from the server, which it validates against the public certificate of the certificate authority used to sign the server certificate. The client then verifies the identity of the server and negotiates with the server to select a cipher (encryption method). The cipher that is selected is the first match between the ciphers that are supported on both the client and the server. All subsequent data transfers for the current request are then encrypted with the selected encryption method.

TLS System Requirements

SAS supports TLS on the Linux operating environment.

TLS Software Availability

SAS Viya supports TLS version 1.2 on Linux.

The default minimum protocol for OpenSSL is TLS 1.2.

SAS Viya uses the OpenSSL libraries provided for the operating systems on your system and OpenSSL libraries installed on your machine.

TLS Installation and Configuration

SAS Viya supports TLS on Linux using the operating system’s OpenSSL libraries. However, in SAS Viya, the Apache HTTP Server is used as the web server and uses mod_ssl and the sas-ssl.conf file provided by SAS to provide TLS configuration.

In SAS Viya, the deployment provides the following support for TLS configuration.

- On the Apache HTTP Server, the module called mod_ssl provides TLS support. This module relies on OpenSSL to provide the cryptography engine. In addition, SAS Viya includes customizations to support SAS internal standards for developing software that protects data in motion.
- The Apache HTTP Server (web server) has localhost certificate and key files that allow HTTPS access to SAS Studio, CAS Server Monitor, and SAS Visual Analytics.
- Each machine in the deployment has a Mozilla bundle of trusted CA certificates in the SASSecurityCertificateFramework that is used by SAS and Java processes if TLS for CAS is turned on.
- The SASSecurityCertificateFramework also generates encrypted self-signed certificates for the CAS controller machine in the deployment that can be used to turn on TLS for CAS. These self-signed certificates are part of the Mozilla bundle of trusted CA certificates (trustedcerts files).

For more information about Certificates and how SAS Viya uses them with TLS, see “Certificates” on page 64.

For information about configuring TLS, see “Configure TLS and HTTPS ” on page 4.

TLS Terminology

The following concepts are fundamental to understanding TLS:

certificate authorities (CAs)
: Cryptography products provide security services by using digital certificates, public-key cryptography, private-key cryptography, and digital signatures. Certificate authorities (CAs) create and maintain digital certificates, which also help preserve confidentiality.

  Various commercial CAs, such as VeriSign and Thawte, provide competitive services for the e-commerce market. You can also develop your own CA by using products from companies such as RSA Security and Microsoft or from the Open-Source Toolkit OpenSSL.

digital signatures
: A digital signature affixed to an electronic document or to a network data packet is like a personal signature that concludes a hand-written letter or that validates a credit card transaction. Digital signatures are a safeguard against fraud. A unique digital signature results from using a private key to encrypt a message digest. A document that contains a digital signature enables the receiver of the document to verify the source of the document. Electronic documents are said to be verified if the receiver knows where the document came from, who sent it, and when it was sent.

  Another form of verification comes from message authentication codes (MAC), which ensure that a signed document has not been changed. A MAC is attached to a document to indicate the document's authenticity. A document that contains a MAC enables the receiver of the document (who also has the secret key) to know that the document is authentic.

digital certificates
: Digital certificates are electronic documents that ensure the binding of a public key to an individual or an organization. Digital certificates provide protection from fraud.

  Usually, a digital certificate contains a public key, a user's name, and an expiration date. It also contains the name of the certificate authority (CA) that issued the digital certificate and a digital signature that is generated by the CA. The CA's validation of an individual or an organization allows that individual or organization to be accepted at sites that trust the CA.

public and private keys
: Public-key cryptography uses a public and a private key pair. The public key can be known by anyone, so anyone can send a confidential message. The private key is confidential and known only to the owner of the key pair, so only the owner can read the encrypted message. The public key is used primarily for encryption, but it can also be used to verify digital signatures. The private key is used primarily for decryption, but it can also be used to generate a digital signature.

symmetric key
: In symmetric key encryption, the same key is used to encrypt and decrypt the message. If two parties want to exchange encrypted messages securely, they must both have a copy of the same symmetric key. Symmetric key cryptography is often used for encrypting large amounts of data because it is computationally faster than asymmetric cryptography. Typical algorithms include DES, TripleDES, RC2, RC4, and AES.

asymmetric key
: Asymmetric or public key encryption uses a pair of keys that have been derived together through a complex mathematical process. One of the keys is made public, typically by asking a CA to publish the public key in a certificate for the certificate-holder (also called the subject). The

private key is kept secret by the subject and never revealed to anyone. The keys work together where one is used to perform the inverse operation of the other: If the public key is used to encrypt data, only the private key of the pair can decrypt it. If the private key is used to encrypt, the public key must be used to decrypt. This relationship allows a public key encryption scheme where anyone can obtain the public key for a subject and use it to encrypt data that only the user with the private key can decrypt. This scheme also specifies that when a subject encrypts data using its private key, anyone can decrypt the data by using the corresponding public key. This scheme is the foundation for digital signatures.

Certificates

About Certificates

Certificates are required for configuring TLS and HTTPS.

Digital certificates are used in a network security system to guarantee that the two parties exchanging information are really who they claim to be. Certificates are used to authenticate a server process or a human user. Digital certificates are issued by a certificate authority (CA).

A CA is an organization that verifies the information or the identity of computers on a network and issues digital certificates of authenticity and public keys. As part of a public key infrastructure (PKI), a CA checks with a registration authority to verify information provided by the requestor of a digital certificate. If the registration authority verifies the requestor's information, the CA can then issue a certificate.

There are three types of certificates that can be used to authenticate entities.

- third-party-signed

  You can go to a commercial third-party certificate authority (VeriSign, GeoTrust, Thawte, DigiCert, Comodo, and so on), or a company can create their own CA and then use it to generate server and client certificates.

- site-signed

  You go to the IT department at your site to obtain a certificate.

- self-signed

  You serve as your own certificate authority.

After generating a digital certificate for the CA, the server, and the client (optional), you must identify for the client application one or more CAs that are to be trusted. This list is called a *trust list* or certificate chain.

A certificate chain is a sequence of certificates, where each certificate in the chain is signed by the subsequent certificate. The purpose of a certificate chain is to establish a chain of trust from a peer certificate to a trusted CA certificate. The CA vouches for the identity in the peer certificate when it signs it. If the CA is one that you trust (a copy of the CA certificate is in your root certificate directory), you can trust the signed peer certificate as well.

Mozilla Trusted Certificate Authority Certificate Bundle

SAS ships Viya with a default list of certification authority (CA) certificates from Mozilla that are known as the Mozilla trusted certificate authority (CA) certificate bundle. These are root certificates. The purpose of the root certificate is to establish a digital chain of trust. The root is the trust anchor.

These Mozilla trusted CA certificates are located in **`/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts`**. There are two files that contain the trusted list of certificates. These are the trustedcerts.pem and the trustedcerts.jks files. These files are located on every machine in the SAS Viya deployment.

The Mozilla trusted bundle of CA Certificates is the basis for the trusted list of certificates. The trustedcerts files include the following certificates:

- Mozilla trusted certificate authority (CA) certificate bundle
- the Vault issued CA certificates
- the SAS generated certificates
- any certificate chain pointed to by HTTPD_CERT_PATH in the vars.yml file.

Your web browser will inherently trust all certificates that have been signed by any root that has been embedded in the browser itself or in an operating system on which it relies.

Certificates Issued by Vault (Full Deployment)

In a full deployment of SAS Viya, HashiCorp Vault is used to generate and sign root and intermediate TLS certificates. These TLS certificates are used to secure communication between various SAS Viya processes. Vault provides a point of contact for services requiring certificates needed to maintain secured communication.

Note: SAS recommends installing a full deployment, which includes the product visual interfaces and microservices.

Vault provides certificates that are part of the secured deployment. They are signed by a CA root and CA intermediate certificate created by Vault. Certificates issued by Vault and key files are placed on the CAS Controller, SAS/CONNECT Server, SAS Configuration Server (Consul), SAS Launcher Server, SAS Message Broker (RabbitMQ), and SAS Infrastructure Data Server (PostgreSQL). For more information about servers in a deployment, see "Infrastructure Servers: Overview" in *SAS Viya Administration: Infrastructure Servers*.

Note: All the microservices have certificates signed by the Vault CA that are stored in Vault.

Certificate files and key files provided at deployment are as follows:

vault-ca.crt
: The file vault-ca.crt contains the CA's certificates. It contains two certificates: The CA's Root certificate, and the CA's Intermediate certificate. This file is placed on all machines in the deployment to allow those machines to trust this machine when they connect to it.

The vault.ca.crt file is located at the following location./**opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/vault-ca.crt**

sas_encrypted.crt
: A root CA certificate issued by Vault for the deployment. Sometimes referred to as the machine certificate. Each machine in a deployment has its own root CA certificate. This file is placed on all other machines in the deployment to allow those machines to trust this machine when they connect to it. This certificate file has a plaintext private key contained in file sas_encrypted.key. This file is located at **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs/sas_encrypted.crt**

Note: CAS uses a certificate that resides in the following directory. **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs/cas/default/sas_encrypted.crt**

sas_encrypted.key
: The RSA private key associated with the public key. This key is embedded within file sas_encrypted.crt. This RSA private key is encrypted. Its decryption key is the contents of file encryption.key. This file is located at the following location./**opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/cas/default/sas_encrypted.key**

encryption.key
: The passphrase (or key) used to encrypt and decrypt the RSA private key in file sas_encrypted.key. This file is located at the following location./**opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/cas/default/encryption.key**

In SAS Environment Manager, the interface for managing certificates in Vault is the SAS Secret Manager. The SAS Secret Manager is based on HashiCorp Vault 0.6.4. SAS Secret Manager uses Vault to store and generate secrets such as Transport Layer Security (TLS) certificates.

Note: A programming-only deployment does not use SAS Secret Manager. SAS Secret Manager is installed on the same machines where SAS Configuration Server resides. SAS Configuration Server must be running for SAS Secret Manager to be operational.

For information about the configuration properties, see "Configuration Properties: Reference (Services)" in *SAS Viya Administration: Configuration Properties*.

For more information, see "SAS Secret Manager" in *SAS Viya Administration: Infrastructure Servers*.

Default Certificates Provided for Apache HTTPD

SAS Viya uses an Apache HTTP server as a reverse proxy server to secure your environment. Apache provides default security settings using mod_ssl to secure the server with self-signed certificates. By default, the SAS Viya deployment installs Apache httpd on the machines that you designate as targets for the HTTP proxy installation unless the proxy server has already been installed.

SAS recommends that you install Apache httpd and configure the Apache HTTP Server to use certificates (your custom certificates) that comply with the security policies at your enterprise before you start the deployment process. This task can be performed pre-deployment or post-deployment of SAS Viya. If you configure Apache httpd pre-deployment and use your custom certificates, when you run the Ansible playbook to deploy SAS Viya, the custom certificates are automatically distributed to secure the server.

If you choose to use the Apache httpd default settings, Apache provided certificates are used to bring up the server. These settings are reasonably secure, but they are not compliant with SAS security standards. SAS recommends replacing the default certificates with custom certificates that comply with the security policies at your enterprise. If you do not add compliant certificates and instead keep the default security settings and certificates, end users will see a standard web browser warning message. SAS recommends replacing the certificates before giving end users access to SAS Viya.

Note: The default security is the only security that is available in a programming-only deployment.

When the Ansible playbook runs at installation, it inspects any existing certificates and the CA chain to determine whether they comply with SAS security requirements. If compliant certificates are found, the certificates are used without changes. If only the default mod_ssl is found, the playbook generates self-signed certificates and configures mod_ssl to use it. You can add your own certificates after the completion of the deployment process, which will require a brief outage.

The default Self-signed certificates and key files provided by default for the Apache HTTP Server are specified in the ssl.conf file. The location of the ssl.conf file is **/etc/httpd/conf.d/**. The certificate and key files are specified using the following directives. The default filenames are localhost.crt and localhost.key.

- `SSLCertificateFile /etc/pki/tls/certs/localhost.crt`
- `SSLCertificateKeyFile /etc/pki/tls/private/localhost.key`

SAS Issued Certificates

In a programming-only deployment of SAS Viya, self-signed certificates are provided for configuring TLS with CAS. These certificates are provided for machines in the deployment as follows:

- Server identity certificates are placed in the sas_encrypted.crt file **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs**. These are encrypted and unencrypted certificates for the CAS controller and unencrypted certificates for the CAS worker nodes.
- Private keys are placed in the sas_encrypted.key file in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private**. These are encrypted and unencrypted keys for the CAS controller and unencrypted keys for the CAS worker nodes.
- CA certificates are placed in the trustedcerts.pem file (Mozilla bundle of trusted certificates) in **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts**. There are unencrypted certificates and encrypted certificates. Only encrypted certificates are provided for the CAS controller. These certificates are copied to all of the machines in the deployment.

Self-signed certificates and key files are also provided for the Apache HTTP Server. The location and filenames are associated with the following directives in the ssl.conf file located at **/etc/httpd/conf.d/**. Certificates are located in localhost.crt and in localhost.key files.

- `SSLCertificateFile /etc/pki/tls/certs/localhost.crt`
- `SSLCertificateKeyFile /etc/pki/tls/private/localhost.key`

Certificate File Formats

There are many file formats used to structure certificates. Here are some of them:

- encodings (also used as extensions)

  PEM
  : Privacy Enhanced Email (.pem) is a container format (Base64 Encoded x.509). The .pem extension is used for different types of X.509v3 files, which contain ASCII (Base64) armored data prefixed with a "—– BEGIN ..." line.

    Examples are CA certificate files or an entire certificate chain. This file can contain an issued public certificate, a public key, a private key, and intermediate and root certificates.

    The PEM file format is preferred by open-source software. It can have a variety of extensions (.pem, .key, .cer, .cert, and so on). For information about converting between file formats, see "Convert Digital Certificate File Formats Using OpenSSL" on page 51.

  DER
  : Distinguished Encoding Rules (.der) is used for binary DER encoded certificates. A PEM file is just a Base64-encoded DER file. OpenSSL can convert these to PEM. DER supports storage of a single certificate. These files can also bear the .cer extension or the .crt extension. For information about converting between file formats, see "Convert Digital Certificate File Formats Using OpenSSL" on page 51.

  JKS
  : JKS is a file format that is specific to Java. It is the Java keystore implementation. A keystore is a storage facility for cryptographic keys and certificates. Keytool is a key and certificate management utility that uses JKS as the file format of the key and certificate databases (KeyStore and TrustStores).

  PKCS12 .P12
  : Public-Key Cryptography Standards (.pkcs12) is a file format that has both public and private keys in the file and all certificates in a certification path. This container file is fully encrypted with a password-based symmetric key. PFX is a predecessor to PKCS#12.

    Note: The PKCS#12 format is the only file format that can be used to export a certificate and its private key.

    For information about converting between file formats, see "Convert Digital Certificate File Formats Using OpenSSL" on page 51.

- common extensions

  CRT
  : The CRT extension is used for certificates. It supports storage of a single-certificate. The certificates can be encoded as binary DER or as ASCII PEM. The CER and CRT extensions are nearly synonymous.

    Note: The only time CRT and CER can safely be interchanged is when the encoding type can be identical. For example, PEM-encoded CRT is the same as PEM-encoded CER.

CSR
: This is a certificate signing request. Some applications can generate these for submission to certificate authorities. It includes some of the key details of the requested certificate, such as subject, organization, and state, as well as the public key of the certificate that will be signed. These are signed by the CA and a certificate is returned. The returned certificate is the public certificate. Note that this public certificate can be in a couple of formats.

KEY
: The KEY extension is used both for public and private keys. The keys can be encoded as binary DER or as ASCII PEM.

SSH (Secure Shell)

SSH (Secure Shell) Overview

SSH is an abbreviation for Secure Shell. SSH is a protocol that enables users to access a remote computer via a secure connection. SSH is available through various commercial products and as freeware. OpenSSH is a free version of the SSH protocol suite of network connectivity tools.

Although SAS software does not directly support SSH functionality, you can use the tunneling feature of SSH to enable data to flow between a SAS client and a SAS server. Port forwarding is another term for tunneling. The SSH client and SSH server act as agents between the SAS client and the SAS server, tunneling information via the SAS client's port to the SAS server's port.

Linux operating systems can access an OpenSSH server on another Linux system. To access an OpenSSH server, Linux systems require OpenSSH software.

Windows systems require PuTTY software.

Currently, SAS supports the OpenSSH client and server that supports protocol level SSH-2 in Linux environments. Other third-party applications that support the SSH-2 protocol currently are untested. Therefore, SAS does not support these applications.

To understand the configuration options that are required for the OpenSSH and PuTTY clients and the OpenSSH server, it is recommended that you have a copy of the book *SSH, the Secure Shell: The Definitive Guide* by Daniel J. Barrett, Richard E. Silverman, and Robert G. Byrnes. This book is an invaluable resource when you are configuring the SSH applications, and it describes in detail topics that include public key authentication, SSH agents, and SSHD host keys.

SSH System Requirements

SAS supports SSH in the Linux operating environment.

SAS supports SSH in these operating environments:

- UNIX
- Windows
- z/OS

SSH Software Availability

OpenSSH supports SSH protocol versions 1.3, 1.5, and 2.0.

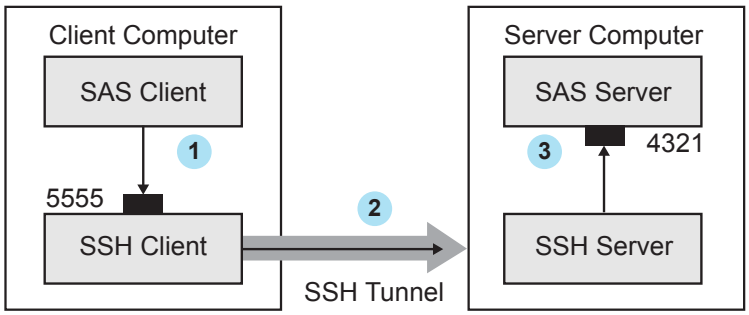
To build the OpenSSL software, refer to the following resources:

- www.openssh.com
- www.ssh.com
- PuTTY Download Page
- Barrett, Daniel J., Richard E. Silverman, and Robert G. Byrnes. 2005. *SSH, the Secure Shell: The Definitive Guide.* Sebastopol, CA: O'Really.

SSH Tunneling Process

An inbound request from a SAS client to a SAS server is shown as follows:

Figure 2 *SSH Tunneling Process*



1 The SAS client passes its request to the SSH client's port 5555.

2 The SSH client forwards the SAS client's request to the SSH server via an encrypted tunnel.

3 The SSH server forwards the SAS client's request to the SAS server via port 4321.

Outbound, the SAS server's reply to the SAS client's request flows from the SAS server to the SSH server. The SSH server forwards the reply to the SSH client, which passes it to the SAS client.

SSH Tunneling: Process for Installation and Setup

SSH software must be installed on the client and server computers. Exact details about installing SSH software at the client and the server depend on the particular brand and version of the software that is used. See the installation instructions for your SSH software.

The process for setting up an SSH tunnel consists of the following steps:

1 SSH tunneling software is installed on the client and server computers. Details about tunnel configuration depend on the specific SSH product that is used. On Linux, you use OpenSSH software to access your Linux OpenSSH server.

2 The SSH client is started as an agent between the SAS client and the SAS server.

3 The components of the tunnel are set up. The components are a listen port, a destination computer, and a destination port. The SAS client accesses the listen port, which is forwarded to the destination port on the destination computer. SSH establishes an encrypted tunnel that indirectly connects the SAS client to the SAS server.

Encrypting ODS Generated PDF Files

You can use ODS to generate PDF output. When these PDF files are not password protected, any user can use Acrobat to view and edit the PDF files. You can encrypt and password-protect your PDF output files by specifying the PDFSECURITY= system option. Valid security levels for the PDFSECURITY= option are NONE or HIGH. SAS encrypts PDF documents using a 128-bit encryption algorithm. With PDFSECURITY=HIGH, at least one password must be set using the PDFPASSWORD= system option. A password is required to open a PDF file that has been generated with ODS.

Table 5 *PDF System Options*

Task	System Option
Specifies whether text and graphics from PDF documents can be edited.	PDFACCESS | NOPDFACCESS
Controls whether PDF documents can be assembled.	PDFASSEMBLY | NOPDFASSEMBLY
Controls whether PDF document comments can be modified.	PDFCOMMENT | NOPDFCOMMENT
Controls whether the contents of a PDF document can be changed.	PDFCONTENT | NOPDFCONTENT
Controls whether text and graphics from a PDF document can be copied.	PDFCOPY | NOPDFCOPY
Controls whether PDF forms can be filled in.	PDFFILLIN | NOPDFFILLIN
Specifies the page layout for PDF documents.	PDFPAGELAYOUT=
Specifies the page viewing mode for PDF documents.	PDFPAGEVIEW=

Task	System Option
Specifies the password to use to open a PDF document and the password used by a PDF document owner.	PDFPASSWORD=
Controls the resolution used to print the PDF document.	PDFPRINT=
Controls the printing permissions for PDF documents.	PDFSECURITY=

Encryption Algorithms

The following encryption algorithms are provided with SAS Viya:

SAS Proprietary for SAS data set encryption with passwords
: is a cipher that uses parts of the passwords that are stored in the SAS data set as part of the 32-bit rolling key encoding of the data. This encryption provides a medium level of security. With the speed of today's computers, it could be subjected to a brute force attack on the 2,563,160,682,591 possible combinations of valid password values, many of which must produce the same 32-bit key.

SAS Proprietary Encryption for communications
: is a cipher that provides basic fixed encoding services under all operating environments that are supported by SAS. The algorithm expands a single message to approximately one-third by using 32-bit fixed encoding. This encoding is used for passwords in configuration files, login passwords, internal account passwords, and so on.

RC2
: is a block cipher that encrypts data in blocks of 64 bits. A *block cipher* is an encryption algorithm that divides a message into blocks and encrypts each block. The RC2 key size ranges from 8 to 256 bits. SAS/SECURE uses a configurable key size of 40 or 128 bits. (The NETENCRYPTKEYLEN system option is used to configure the key length.) The RC2 algorithm expands a single message by a maximum of 8 bytes. RC2 is an algorithm developed by RSA Data Security, Inc.

RC4
: is a stream cipher. A *stream cipher* is an encryption algorithm that encrypts data one byte at a time. The RC4 key size ranges from 8 to 2048 bits. SAS/SECURE uses a configurable key size of 40 or 128 bits. (The NETENCRYPTKEYLEN system option is used to configure the key length.) RC4 is an algorithm developed by RSA Data Security, Inc.

DES (Data Encryption Standard)
: is a block cipher that encrypts data in blocks of 64 bits by using a 56-bit key. The algorithm expands a single message by a maximum of 8 bytes. DES was originally developed by IBM but is now published as a U.S. Government Federal Information Processing Standard (FIPS 46-3).

TripleDES
: is a block cipher that encrypts data in blocks of 64 bits. TripleDES executes the DES algorithm on a data block three times in succession by using a single 56-bit key. This has the effect of

encrypting the data by using a 168-bit key. TripleDES expands a single message by a maximum of 8 bytes. TripleDES is defined in the American National Standards Institute (ANSI) X9.52 specification.

AES (Advanced Encryption Standard)
is a block cipher that encrypts data in blocks of 128 bits by using a 256-bit key. AES expands a single message by a maximum of 16 bytes. Based on its DES predecessor, AES has been adopted as the encryption standard by the U.S. Government. AES is one of the most popular algorithms used in symmetric key cryptography. AES is published as a U.S. Government Federal Information Processing Standard (FIPS 197).

DSA (Digital Signature Algorithm)
The Digital Signature Algorithm (DSA) is a public-key (or assymetric-key) cryptography algorithm. A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or document. A DSA is used to compute and verify digital signatures. Essentially, the DSA helps verify that data has not been changed after it is signed, thus providing message integrity.

In 1994, the National Institute of Standards and Technology (NIST) issued a Federal Information Processing Standard for digital signatures, known as the DSA or DSS. This was adopted as FIPS 186 in 1993.

Elliptic Curve (ECC)
is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. ECC requires smaller keys compared to non-ECC cryptography (based on plain Galois fields) to provide equivalent security. Elliptic curves are applicable for encryption, digital signatures, pseudo-random generators and other tasks.

MD5 (Message Digest)
is a series of byte-oriented algorithms that produce a 128-bit hash value from an arbitrary-length message. It is an algorithm used for hashing. It was developed by Rivest.

Note: This algorithm is not FIPS 140-2 compliant.

RSA (Rivest-Shamir-Adleman)
RSA is a public-key (or assymetric-key) cryptography algorithm and is widely used for secure data transmission. It is used for both encryption and authentication. Encryption and decryption are carried out using two different keys, the public key and the private key. A public-key system means that the algorithm for encrypting a message is publicly known, but the algorithm to decrypt the message is only privately known. In RSA, the public key is a large number that is a product of two primes, plus a smaller number. The private key is a related number.

SHA-1 (Secure Hash Algorithm)
produces a 160-bit (20-byte) hash value. A SHA-1 hash value is typically rendered as a hexadecimal number 40 digits long. This algorithm was developed by the U.S. National Security Agency (NSA) and published in 2001 by the NIST as a U.S. Federal Information Processing Standard (FIPS) PUB 180-1.

SHA-256 (Secure Hash Algorithm)
is essentially a 256-bit block cipher algorithm that encrypts the intermediate hash value using the message block as key. This algorithm was developed by the U.S. National Security Agency (NSA) and published in 2001 by the NIST as a U.S. Federal Information Processing Standard (FIPS) PUB 180-4.

SHA-384 (Secure Hash Algorithm)
: SHA384 is a truncated version of SHA512. Ir is essentially a 384-bit block cipher algorithm that encrypts the intermediate hash value using the message block as key. SHA-512 uses 64-bit words.

SHA-512 (Secure Hash Algorithm)
: is essentially a 512-bit block cipher algorithm that encrypts the intermediate hash value using the message block as key. SHA-512 uses 64-bit words.

Reference

SAS System Options for Encryption

This section contains the SAS System options that can be used to configure encryption. These options can be specified in a number of different ways: in configuration files, in properties files, in SAS programs in the OPTIONS statement, on the SAS/CONNECT spawner command line, and in the SAS System Options window in SAS 9.

These system options are used for SAS/CONNECT and workspace servers. In a SAS Viya programming deployment, these system options are still used. In a SAS Viya deployment, TLS is configured by default. TLS is now turned on and off by enabling TLS Ports.

NETENCRYPT System Option

Specifies whether encryption is required for the connection.

NETENCRYPT | NONETENCRYPT

NETENCRYPT
: specifies that encryption is required.

NONETENCRYPT
: specifies that encryption is not required, but is optional.

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default	NONETENCRYPT

Linux specifics	Linux
See	“NETENCRYPTALGORITHM= System Option” on page 75

The default for this option specifies that encryption is used if the NETENCRYPTALGORITHM option is set and if both the client and the server are capable of encryption. If encryption algorithms are specified but either the client or the server is incapable of encryption, then encryption is not performed.

Encryption might not be supported at the client or at the server in these situations:

- You are using a release of SAS (prior to SAS 8) that does not support encryption.
- Your site (the client or the server) does not have a security software product installed.
- You specified encryption algorithms that are incompatible in SAS sessions on the client and the server.

NETENCRYPTALGORITHM= System Option

Specifies the algorithm or algorithms to be used for encrypted client/server data transfers.

NETENCRYPTALGORITHM= algorithm | (“algorithm-1”... “algorithm-n”)

algorithm | (“algorithm-1”... “algorithm-n”)
specifies the algorithm or algorithms that can be used for encrypting data that is transferred between a client and a server across a network. These algorithms are specified on the Server.

When you specify two or more encryption algorithms, use a space or a comma to separate them, and enclose the algorithms in parentheses.

The following algorithms can be used:

- AES
- DES
- RC2
- RC4
- TripleDES
- SASProprietary
- SSL

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications

Alias	NETENCRALG=
Default	No algorithim is defined
Linux specifics	Linux
Tip	The NETENCRYPTALGORITHM= option must be specified in the server session.
See	“NETENCRYPT System Option” on page 74
Example	options netencryptalgorithm=(ssl);

Use the NETENCRYPTALGORITM option to specify one or more encryption algorithms that you want to use to protect the data that is transferred across the network. If more than one algorithm is specified, the client session negotiates the first specified algorithm with the server session. If the client session does not support that algorithm, the second algorithm is negotiated, and so on.

If either the client session or the server session specifies the NETENCRYPT option (which makes encryption mandatory) but a common encryption algorithm cannot be negotiated, the client cannot connect to the server.

If the NETENCRYPTALGORITHM= option is specified in the server session only, then the server's values are used to negotiate the algorithm selection. If the client session supports only one of multiple algorithms that are specified in the server session, the client can connect to the server.

There is an interaction between either NETENCRYPT or NONETENCRYPT and the NETENCRYPTALGORITHM option.

Table 6 *Client/Server Connection Outcomes*

Server Settings	Client Settings	Connection Outcome
NONETENCRYPT NETENCRALG=*alg*	No settings	If the client is capable of encryption, the client/server connection is encrypted. Otherwise, the connection is not encrypted.
NETENCRYPT NETENCRALG=*alg*	No settings	If the client is capable of encryption, the client/server connection is encrypted. Otherwise, the client/server connection fails.
No settings	NONETENCRYPT NETENCRALG=*alg*	A client/server connection is not encrypted.
No settings	NETENCRYPT NETENCRALG=*alg*	A client/server connection fails.
NETENCRYPT or NONETENCRYPT	NETENCRALG=*alg-2*	Regardless of whether NETENCRYPT or

Server Settings	Client Settings	Connection Outcome
NETENCRALG=*alg-1*		NONETENCRYPT is specified, a client/server connection fails.

NETENCRYPTKEYLEN= System Option

Specifies the key length that is used by the encryption algorithm for encrypted client/server data transfers.

NETENCRYPTKEYLEN= 0 | 40 | 128

0
specifies that the maximum key length that is supported at both the client and the server is used.

40
specifies a key length of 40 bits for the RC2 and RC4 algorithms.

128
specifies a key length of 128 bits for the RC2 and RC4 algorithms. If either the client or the server does not support 128-bit encryption, the client cannot connect to the server.

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias	NETENCRKEY=
Default	0
Linux specifics	Linux

The NETENCRYPTKEYLEN= option supports only the RC2 and RC4 algorithms. The SAS Proprietary, DES, TripleDES, SSL, and AES algorithms are not supported.

By default, if you try to connect a computer that is capable of only a 40-bit key length to a computer that is capable of both a 40-bit and a 128-bit key length, the connection is made using the lesser key length. If both computers are capable of 128-bit key lengths, a 128-bit key length is used.

Using longer keys consumes more CPU cycles. If you do not need a high level of encryption, set NETENCRYPTKEYLEN=40 to decrease CPU usage.

SSLCACERTDIR= System Option

Specifies the location of the trusted certificate authorities (CA) found in OpenSSL format.

SSLCACERTDIR="file-path"

"file-path"

specifies the location where the public certificates for all of the trusted certificate authorities (CA) in the trust chain are filed. There is one file for each CA. Each CA certificate file must be PEM-encoded (base64). For more information, see "Certificate File Formats" on page 68.

The names of the files are the value of a hash that OpenSSL generates.

Note: OpenSSL generates different hash values for each OpenSSL version. For example, OpenSSL 0.9.8 generates different hash values than does OpenSSL 1.0.2.

OpenSSL looks up the CA certificate based on the x509 hash value of the certificate. SSLCACERTDIR= requires that the certificates are located in the specified directory where the certificate names are the value of a hash that OpenSSL generates.

If you are upgrading from a version of OpenSSL that is older than 1.0.0, you need to update your certificate directory links. Starting with code base 1.0.0, SHA hashing is used instead of MD5. You can use the OpenSSL C_REHASH utility to re-create symbolic links to files named by the hash values.

You can discover the hash value for a CA and then create a link to the file named after the certificate's hash value. Note that you must add ".0" to the hash value.

```
ln -s cacert1.pem 'openssl x509 -noout -hash -in
/u/myuser/sslcerts/cacert1.pem'.0
```

If you list the CA file, you see the link between the file named after the certificate's hash value and the CA file.

```
lrwxrwxrwx 1 myuser rnd 10 Apr 7 14:42 6730c6a9.0 -> cacert1.pem
```

To verify the path of the server certificate file (cacert1.pem for our example), use the following OpenSSL command:

```
openssl verify -CApath /u/myuser/sslcerts cacert1.pem
```

Client	Optional
Server	Optional
Valid in	Configuration file, SAS invocation, SAS/CONNECT spawner start-up, connectserver_usermods.sh script
Categories	Communications: Networking and Encryption System Administration: Security
Default	The default file and location for certificates is **/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/trustedcerts.pem**. You can point to a different file and location using the SSLCALISTLOC= system option or the SSLCACERTDIR system option. There is one trusted certificate file pointed to by

the SSLCALISTLOC= system option. By contrast, the SSLCACERTDIR system option allows the customer to specify a location where multiple certificate files reside. See "SSLCALISTLOC= System Option" on page 79.

Linux specifics

Linux

Examples

The SSLCACERTDIR system option points to the directory where the CA certificate is located. Export the environment variable on Linux hosts for the Bourne Shell:

```
export SSLCACERTDIR=/u/myuser/sslcerts/
```

Set the environment variable on Linux hosts for the C Shell directory where the CA certificates are located:

```
SETENV SSLCACERTDIR /u/myuser/sslcerts/
```

Set the environment variable at SAS invocation for Linux hosts:

```
-set "SSLCACERTDIR=/u/myuser/sslcerts/"
```

For Foundation Servers such as workspace servers and stored process servers, if certificates are used, SAS searches for certificates in the following order:

For Foundation Servers such as workspace servers and stored process servers, if certificates are used, SAS searches for certificates in the following order:

1. SAS looks for SAS system option SSLCALISTLOC= to find the file trustedcerts.pem.
2. SAS looks for the SSLCALISTLOC= environment variable to find the file trustedcerts.pem.
3. If the SSLCALISTLOC= system option or environment variable is not used, the trustedcerts.pem file located in */opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts* is used as the default.
4. If trustedcerts.pem exists, and the SSL_CERT_DIR and SSLCACERTDIR environment variables are set, SAS checks trustedcerts.pem first before it searches the directory.
5. If trustedcerts.pem does not exist, but the certificates are in the directory defined by SSL_CERT_DIR or SSLCACERTDIR, then SAS ignores SSLCALISTLOC=.
6. If trustedcerts.pem does not exist, and the SSL_CERT_DIR and SSLCACERTDIR environment variables are not set, SAS reports an error.

Note: A trusted CA certificate is required at the client in order to validate a server's digital certificate. The trusted CA certificate must be from the CA that signed the server certificate.

SSLCALISTLOC= System Option

Specifies the location of the public certificate(s) for trusted certificate authorities (CA).

SSLCALISTLOC="file-path"

"file-path"
specifies the location of a single file that contains the public certificate(s) for all of the trusted certificate authorities (CA) in the trust chain.

Note: Specify this option on the client. Optionally, specify this option on the server.

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Notes	If the SSLCALISTLOC= system option is not specified, SAS defaults to a file named trustedcerts.pem located in `/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts` The trustedcerts.pem file contains the list of trusted CA certificates provided by SAS at installation. If you use this option, it must be specified on the client, but does not have to also be specified on the server.

The SSLCALISTLOC= system option specifies the location of a single file that contains the public certificate(s) for all of the trusted certificate authorities (CA) in the trust list. The CA file must be PEM-encoded (base64).

The location of the trusted certificate file specified by the SSLCALISTLOC= system option or SSLCACERTDIR= system option or the trustedcerts.pem file is needed on the spawner to verify the certificate from the SAS/CONNECT server.

The default path set for the SSLCALISTLOC= system option on the workspace server is **`/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/trustedcerts.pem`**. By default, the trustedcerts.pem file contains a managed set of trusted root certificates (Mozilla bundle of certificates and others) provided at SAS installation.

Note: The SSLCACERTDIR= system option can be used instead of using the SSLCALISTLOC= system option. SSLCACERTDIR= points to a directory that contains all of the public certificate file(s) of all CA(s) in the trust list. One file exists for each CA in the trust list. For more information, see "SSLCACERTDIR= System Option" on page 78.

For Foundation Servers such as workspace servers and stored process servers, if certificates are used, SAS searches for certificates in the following order:

1. SAS looks for SAS system option SSLCALISTLOC= to find the file trustedcerts.pem.
2. SAS looks for the SSLCALISTLOC= environment variable to find the file trustedcerts.pem.

3 If the SSLCALISTLOC= system option or environment variable is not used, the trustedcerts.pem file located in */opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts* is used as the default.

4 If trustedcerts.pem exists, and the SSL_CERT_DIR and SSLCACERTDIR environment variables are set, SAS checks trustedcerts.pem first before it searches the directory.

5 If trustedcerts.pem does not exist, but the certificates are in the directory defined by SSL_CERT_DIR or SSLCACERTDIR, then SAS ignores SSLCALISTLOC=.

6 If trustedcerts.pem does not exist, and the SSL_CERT_DIR and SSLCACERTDIR environment variables are not set, SAS reports an error.

Note: A trusted CA certificate is required at the client in order to validate a server's digital certificate. The trusted CA certificate must be from the CA that signed the server certificate. The SSLCALISTLOC= option is required at the server only if the SSLCLIENTAUTH option is also specified at the server.

Note: Unless the SSLCACERTDIR= system option is set or the default trustedcerts.pem file is used, the SSLCALISTLOC= system option is needed on the spawner to verify the certificate from the SAS/CONNECT server.

SSLCACERTDATA= System Option

Specifies the name of the issuer of the digital certificate that TLS should use.

SSLCACERTDATA="*encoded-string*"

"*encoded-string*"
specifies the base64-encoded x509 text that represents a single certificate authority (CA) certificate. This string is in PEM format. The text string starts with the line "-----BEGIN CERTIFICATE-----" and ends with the line "-----END CERTIFICATE-----".

This option provides a way to programmatically specify a CA certificate rather than having to point to a file that contains the certificate information. The certificate must be PEM-encoded (base64) format.

Here is an example of how you might use the SSLCACERTDATA= system option to specify a certificate.

```
data _null_;
  length certInfo $3200.;
  input txt $67.;
  retain certInfo;

  if _N_ = 1 then
    certInfo=txt;
  else
    certInfo=catx('0a'x,certInfo,txt);
  call symput('certInfo',trim(left(certInfo)));
  datalines;
-----BEGIN CERTIFICATE-----
MIICbzCCAfagAwIBAgIJAP7q5/tk7+laMAoGCCqGSM49BAMCMHYxCzAJBgNVBAYT
```

```
AlVTMQswCQYDVQQIDAJOQzENMAsGA1UEBwwEQ2FyeTEWMBQGA1UECgwNU0FTIElu
c3RpdHV0ZTEMMAoGA1UECwwDSURCMSUwIwYDVQQDDBxkZW1vUm9vdENBLUVDRFNB
LVAzODQtU0hBMjU2MB4XDTE2MTEwNDE4MDMzMVoXDTI2MTEwMjE4MDMzMVowdjEL
MAkGA1UEBhMCVVMxCzAJBgNVBAgMAk5DMQ0wCwYDVQQHDARDYXJ5MRYwFAYDVQQK
DA1TQVMgSW5zdGl0dXRlMQwwCgYDVQQLDANJREIxJTAjBgNVBAMMHGRlbW9Sb290
Q0EtRUNEU0EtUDM4NC1TSEEyNTYwdjAQBgcqhkjOPQIBBgUrgQQAIgNiAAQghfjE
5iiiPQtb/Ors/GeNuLRXWnUhqnPWw4X0veIQT5rXFWZmiwReIjaYt9KChhmFkPno
cQ1m3HpdVnP86cPLPpLSvcAG/dO6o2W2SakiOWa1cA1UKsRhy/kUMnTSGJSjUDBO
MB0GA1UdDgQWBBSXhRRVQTNHpe1A9NsdUa+Y/IxhTTAfBgNVHSMEGDAWgBSXhRRV
QTNHpe1A9NsdUa+Y/IxhTTAMBgNVHRMEBTADAQH/MAoGCCqGSM49BAMCA2cAMGQC
MFJf5/2+eRSwCxrOyVjgyI4Teiofggrji5StKyQzHhDnXPljdYRss0WxxhbdBcxo
8wIwDjX8Yx61lY52U/h0q8ZkuNJWu0gJ8ZmrOVttkUBYUU0D1Cer6pdl4gQd6mUz
oXrB
-----END CERTIFICATE-----
   ;
run;

options SSLCACERTDATA="&certInfo";
```

SSLCERTLOC= System Option

Specifies the location of the digital certificate for the machine's public key. This is used for authentication.

SSLCERTLOC="file-path"

"file-path"
specifies the location of a file that contains a digital certificate for the machine's public key. The certificate must be PEM-encoded (base64). This is used by servers to send to clients for authentication.

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Note	If you use this option, it must be specified on the server, but does not have to also be specified on the client.

The SSLCERTLOC= option is required for a server. It is required at the client only if the SSLCLIENTAUTH option is specified at the server. In order for a TLS connection to succeed, the SAS/CONNECT server needs to be started with the -SSLCERTLOC= and -SSLPVTKEYLOC= system options set in the SAS/CONNECT spawner. Alternatively, the -SSLPKCS12LOC= system option can be used.

In SAS Viya, set the SSL options on the spawner and the server in the connectserver_usermods.sh file (**/opt/sas/viya/config/etc/connectserver/default**) and in the connect_usermods.sh file (**/opt/sas/viya/config/etc/connect/default**). For configuration information, see "Sign On to a SAS/CONNECT Spawner Using TLS" on page 30.

SSLCIPHERLIST= System Option

Specifies the ciphers that can be used on Linux for OpenSSL.

SSLCIPHERLIST=*openssl_cipher_list*

openssl-cipher-list

The SSLCIPHERLIST= system option specifies the ciphers that can be used on Linux for OpenSSL. Refer to the OpenSSL Ciphers document to see how to format the *openssl-cipher-list* and for a complete list of the ciphers that work with your TLS version. The OpenSSL Ciphers information can be found at https://docs.openssl.org/3.3/man1/openssl-ciphers.

Note: SAS does not support CAMELLIA, IDEA, MD2, and RC5 ciphers.

Note: The protocol and cipher information for the actual connection can be seen by setting *dumpCurrentCipherInfo* at the SAS DEBUG level.

Note: If you set a minimum protocol that does not allow some ciphers, you might get an error.

Client	Optional
Server	Optional
Valid in	Configuration file, command line
Categories	Communications: Networking and Encryption System Administration: Security
Restriction	If the SSLMODE= option is set, this option is ignored.
Linux specifics	Linux
Notes	This option can also be specified as an environment variable. This system option must be set before TLS is loaded. It cannot be changed after TLS is loaded. You must set the environment variable before the SAS/CONNECT spawner is started and before SAS is started on the client.
Example	Specify the system option: `-SSLCIPHERLISTS= HIGH`

SSLCLIENTAUTH System Option

Specifies whether a server should perform client authentication.

SSLCLIENTAUTH | NOSSLCLIENTAUTH

SSLCLIENTAUTH
specifies that the server should perform client authentication. Server authentication is always performed, but the SSLCLIENTAUTH option enables a user to control client authentication. This option is valid only when used on a server.

> **TIP** If you enable client authentication, a certificate for each client is needed.

NOSSLCLIENTAUTH
specifies that the server should not perform client authentication.

Default NOSSLCLIENTAUTH is the default.

Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Note	If you use this option, it is specified on the server.

SSLCRLCHECK System Option

Specifies whether a Certificate Revocation List (CRL) is checked when a digital certificate is validated.

SSLCRLCHECK | NOSSLCRLCHECK

SSLCRLCHECK
specifies that CRLs are checked when digital certificates are validated.

NOSSLCRLCHECK
specifies that CRLs are not checked when digital certificates are validated.

Client	Optional
Server	Optional

Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Note	If you use this option, it can be specified on the client and server.
See	"SSLCRLLOC= System Option" on page 85

A certificate revocation list (CRL) is published by a certificate authority (CA) and contains a list of revoked digital certificates. The list contains only the revoked digital certificates that were issued by a specific CA.

The SSLCRLCHECK option is required at the server only if the SSLCLIENTAUTH option is also specified at the server. Because clients check server digital certificates, this option is relevant for the client.

SSLCRLLOC= System Option

Specifies the location of a certificate revocation list (CRL).

SSLCRLLOC="file-path"

"file-path"
specifies the location of a file that contains a certificate revocation list (CRL).

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Notes	If you use this option, it can be specified on the client and server. The SSLCRLLOC= option is required only when the SSLCRLCHECK option is specified.
See	"SSLCRLCHECK System Option" on page 84

SSLMINPROTOCOL= System Option

Specifies the minimum TLS or SSL protocol that can be negotiated when using OpenSSL.

SSLMINPROTOCOL=*protocol*

protocol
specifies the minimum TLS or SSL protocol version that is negotiated between Linux servers when using OpenSSL. SAS Viya supports specifying TLS1.2 and TLSv1.2. The following other values can be specified, but are less secure: SSL3, SSLV3, TLS, TLS1, TLSV1, TLS1.0, TLSV1.0, TLS1.1, and TLSV1.1.

CAUTION
TLS versions 1.0 and 1.1 are insecure. It is highly recommended that you use TLS 1.2 or later.

Note: A message is written to the SAS log when an invalid value is specified.

During the first TLS handshake attempt, the highest supported protocol version is offered. If this handshake fails, earlier protocol versions are offered instead. TLS1.2 is the default minimum OpenSSL protocol. By default, the SSLMODE= option is set to SSLMODESP800131A, which uses TLS 1.2 to negotiate between client and servers. You can specify an earlier fallback value, but it is not recommended.

See the SAS SAS Statement Regarding OpenSSL Security Advisories for the most current information about the versions of OpenSSL used in SAS products and about the advisories under consideration.

Client	Optional
Server	Optional
Valid in	Configuration file, command line, SAS/CONNECT spawner start-up if this option is used as an environment variable
Categories	Communications: Networking and Encryption System Administration: Security
Default	TLS 1.2.
Restriction	If the SSLMODE= option is set, this option is ignored.
Linux specifics	Linux
Notes	This option can also be specified as an environment variable. This environment variable must be set before TLS or SSL are loaded. It cannot be changed after TLS or SSL are loaded. You must set the environment variable before the SAS/CONNECT spawner is started and before SAS is started on the client.

Example Specify the system option as follows:

```
-SSMINPROTOCOL="TLS1.2"
```

SSLMODE= System Option

Sets the allowed TLS version and cipher suites to be used for TLS.

SSLMODE=*ssl-mode*

ssl-mode

SSLMODESUITEB128
is the mode of operation that uses the cipher-suites specified in the NIST Suite B Cryptography using 128 AES encryption.

SSLMODESUITEB192
is the mode of operation that uses the cipher-suites specified in the NIST Suite B Cryptography using 192 AES encryption.

SSLMODESP800131A
is the DEFAULT configuration mode for TLS communication.

SSLMODEDEPRECATED
is the mode of operation that uses the cipher-suites specified in the NIST Special Publication 800-131A.

When system option SSLMODE= is set, system option SSLMINPROTOCOL= is ignored. If SSLMODE= is not set, SAS checks the SSLMINPROTOCOL= system option and uses the protocol set. If neither system option is set, SAS uses the default cipher mode SSLMODESP800131A.

Client Optional

Server Optional

Valid in Configuration file, command line, SAS/CONNECT spawner start-up if this option is used as an environment variable, connectserver_usermods.sh script

Categories Communications: Networking and Encryption

System Administration: Security

Default SSLMODESP800131A

Restriction If the SSLMODE= option is set, this option is ignored.

Interaction When system option SSLMODE= is set, system option SSLMINPROTOCOL= is ignored.

Linux specifics Linux

See For a list of ciphers that are supported for each of the modes that can be specified for the SSLMODE= system option, see SSLMODE= System Option Supported Ciphers.

Example Specify the system option as follows:

```
-sslmode SSLMODESP800131A
```

SAS uses the National Institute of Standards and Technology (NIST) Special Publication 800-131A (SP800-131A) as the minimum compliance standard for TLS and to extend the FIPS standards. TLS version 1.2 is the default version of TLS that SAS supports. However, SAS does provide the ability to specify less secure TLS 1.1 if needed (SSLMODEDEPRECATED). For details of SP800-131A, see NIST Special Publication 800-131A, Revision 1.

Suite B cryptography allows TLS client and server applications to specify a profile compliant with Suite B Cryptography as defined in RFC 5430: Suite B Profile for Transport Layer Security (TLS). Suite B cryptography specifies the cryptographic algorithms that can be used in a "Suite B Compliant" TLS V1.2 session. Suite B requires the key establishment and authentication algorithms that are used in TLS V1.2 sessions to be based on Elliptic Curve Cryptography, and the encryption algorithm to be AES.

For a list of ciphers that are supported for each of the modes that can be specified for the SSLMODE= system option, see SSLMODE= System Option Supported Ciphers.

SSLPKCS12LOC= System Option

Specifies the location of the PKCS #12 encoding package file.

SSLPKCS12LOC="file-path"

"file-path"
specifies the location of the PKCS #12 DER encoding package file that contains the certificate and the private key.

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Notes	If you use this option, it can be specified on the client and server. You must specify both the -SSLPKCS12LOC option and the -SSLPKCS12PASS option together.
See	"SSLPKCS12PASS= System Option" on page 89

If the SSLPKCS12LOC= option is specified, the PKCS #12 DER encoding package must contain both the certificate and private key. The SSLCERTLOC= and SSLPVTKEYLOC= options are ignored.

You must specify both the -SSLPKCS12LOC option and the -SSLPKCS12PASS option in order for the SAS/CONNECT server to access the appropriate server scripts. In SAS Viya, set the SSL options on

the spawner and the server in the connectserver_usermods.sh file (**/opt/sas/viya/config/etc/connectserver/default**) and in the connect_usermods.sh file (**/opt/sas/viya/config/etc/connect/default**). For configuration information, see "Sign On to a SAS/CONNECT Spawner Using TLS" on page 30.

SSLPKCS12PASS= System Option

Specifies the password that TLS requires for decrypting the private key.

SSLPKCS12PASS=password

password
specifies the password that TLS requires in order to decrypt the PKCS #12 DER encoding package file. The PKCS #12 DER encoding package is stored in the file that is specified by using the SSLPKCS12LOC= option.

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Notes	If you use this option, it can be specified on the client and server. You must specify both the -SSLPKCS12LOC option and the -SSLPKCS12PASS option together.
See	"SSLPKCS12LOC= System Option" on page 88

The SSLPKCS12PASS= option is required only when the PKCS #12 DER encoding package is encrypted.

You must specify both the -SSLPKCS12LOC option and the -SSLPKCS12PASS option in order for the SAS/CONNECT server to access the appropriate server scripts. In SAS Viya, set the SSL options on the spawner and the server in the connectserver_usermods.sh file (**/opt/sas/viya/config/etc/connectserver/default**) and in the connect_usermods.sh file (**/opt/sas/viya/config/etc/connect/default**). For configuration information, see "Sign On to a SAS/CONNECT Spawner Using TLS" on page 30.

SSLPVTKEYLOC= System Option

Specifies the location of the private key that corresponds to the digital certificate.

SSLPVTKEYLOC="file-path"

"file-path"
specifies the location of the file that contains the private key that corresponds to the digital certificate that was specified by using the SSLCERTLOC= option.

Client	Optional
Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Notes	If you use this option, it can be specified on the client and server. You must specify the -SSLCERTLOC option if you specify the -SSLPVTKEYLOC option. -SSLPVTKEYPASS is required only when the private key is encrypted.
See	"SSLCERTLOC= System Option" on page 82and "SSLPVTKEYPASS= System Option" on page 90.

The SSLPVTKEYLOC= option is required at the server only if the SSLCERTLOC= option is also specified at the server.

The key must be PEM-encoded (base64). For more information, see "Certificate File Formats" on page 68.

You must specify both the -SSLCERTLOC option and the -SSLPVTKEYLOC option in order for the SAS/CONNECT server to access the appropriate server scripts. In SAS Viya, set the SSL options on the spawner and the server in the connectserver_usermods.sh file (**`/opt/sas/viya/config/etc/connectserver/default`**) and in the connect_usermods.sh file (**`/opt/sas/viya/config/etc/connect/default`**). For configuration information, see "Sign On to a SAS/CONNECT Spawner Using TLS" on page 30.

SSLPVTKEYPASS= System Option

Specifies the password that TLS requires for decrypting the private key.

SSLPVTKEYPASS="password"

"password"
specifies the password that TLS requires in order to decrypt the private key. The private key is stored in the file that is specified by using the SSLPVTKEYLOC= option.

Client	Optional

Server	Optional
Valid in	Configuration file, OPTIONS statement, SAS System Options window in SAS 9, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Notes	If you use this option, it can be specified on the client and server. You must specify the -SSLCERTLOC= option if you specify the -SSLPVTKEYLOC= option. -SSLPVTKEYPASS= is required only when the private key is encrypted.
See	"SSLCERTLOC= System Option" on page 82and "SSLPVTKEYPASS= System Option" on page 90.

The SSLPVTKEYPASS= option is required only when the private key is encrypted. OpenSSL performs key encryption.

Note: No SAS system option is available to encrypt private keys.

SSLREQCERT= System Option

Specifies what checks to perform on server certificates in a TLS session.

SSLPVTKEYLOC=ALLOW | DEMAND | NEVER | TRY

ALLOW
specifies that the client requests a server certificate, but the session proceeds normally even if no certificate is provided or an invalid certificate is provided.

DEMAND
specifies that a server certificate is requested, and if no valid certificate is provided, the session terminates. DEMAND is the default setting.

NEVER
specifies that the authentication server does not ask for a certificate.

TRY
specifies that the client requests a server certificate, and if no certificate is provided, the session proceeds normally. If an invalid certificate is provided, the session terminates.

If you do not add the SSLREQCERT= option to your configuration file, then the default value is DEMAND. If you specify SSLREQCERT=, then the value of SSLREQCERT= applies to all of your authentication providers.

Note: To ensure proper security, SSLREQCERT=*DEMAND* should be specified.

Client	Optional
Server	Optional
Valid in	Configuration file, SAS invocation, SAS/CONNECT spawner command line, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Linux specifics	Linux
Example	export SSLREQCERT=DEMAND

SSLSNIHOSTNAME= System Option

Enables the client to specify the Server Name Indication (SNI) in the TLS handshake that identifies the server name that it is trying to connect to.

SSLSNIHOSTNAME= "*hostname*"

specifies the host name that is used for the Server Name Indication (SNI) TLS extension. If it is not specified, the target host name is used. The client uses SNI in the first message of the TLS handshake (connection setup) to identify the server name that it is trying to connect to.

The client uses SNI in the TLS handshake to identify the server name that it is trying to connect to. When making a TLS connection, the client requests a digital certificate from the web server. After the server sends the certificate, the client examines it and compares the name that it was trying to connect to with the name or names included in the certificate. If a match is found, the connection proceeds as normal.

Client	Optional
Server	Optional
Valid in	Configuration file, SAS invocation, SAS/CONNECT spawner start-up if this option is used as an environment variable, connectserver_usermods.sh script
Category	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default	The default is the name of the host being contacted.
Linux specifics	Linux
Notes	This option can also be specified as an environment variable. The TLS SNI extension is always sent to the web server.

Example — Specify the system option as follows:

```
-SSLSNIHOSTNAME="www.example.org"
```

SAS Environment Variables for Encryption

Overview of Environment Variables

Linux environment variables are variables that apply to both the current shell and to any subshells that it creates. The way in which you define an environment variable depends on the shell that you are running. For more information, see Defining Environment Variables in UNIX Environments.

SSL_USE_SNI Environment Variable

Disables the use of Server Name Indication (SNI) in the TLS handshake for the client.

SSL_USE_SNI

Linux clients and servers support TLS Server Name Indication (SNI). The client uses SNI in the first message of the TLS handshake (connection setup) to identify the server name that it is trying to connect to.

The client uses SNI in the TLS handshake to identify the server name that it is trying to connect to. When making a TLS connection, the client requests a digital certificate from the web server. After the server sends the certificate, the client examines it and compares the name that it was trying to connect to with the name or names included in the certificate. If a match is found, the connection proceeds as normal.

Client — Optional

Server — Optional

Valid in — SAS invocation, configuration file

Categories — Communications: Networking and Encryption

System Administration: Security

Default — By default, the TLS SNI extension is sent as part of the TLS handshake.

Restriction — System option SSLSNIHOSTNAME= is used to specify the Server Name Indication (SNI) that identifies the server name that it is trying to connect to. This environment variable is now used to turn off SNI which is sent by default.

Linux specifics — Linux

Examples — Export the environment variable on Linux hosts for the Bourne Shell:

```
export SSL_USE_SNI=1
```

Set the environment variable at SAS invocation for Linux hosts:

```
SETENV SSL_USE_SNI
```

CAS TLS Environment Variables

CAS server options are stored in configuration files. During deployment, casconfig_deployment.lua is created in the **/opt/sas/viya/config/etc/cas/default** directory. This file contains CAS configuration settings that are created during deployment by Ansible from vars.yml.

The casconfig_usermods.lua file is now used by the SAS administrator to add CAS configuration options. During redeployments or upgrades, casconfig_usermods.lua is not modified. Therefore, to preserve your CAS configurations, always use casconfig_usermods.lua for changes.

When you start the server with the `service sas-viya-cascontroller-default start` command, the casconfig_deployment.lua, casconfig.lua, and the casconfig_usermods.lua configuration files are processed.For more information, see "SAS Cloud Analytic Services: Reference" in *SAS Viya Administration: SAS Cloud Analytic Services*.

These are the configuration options that can be used for configuring TLS on CAS servers and clients.

env.CAS_CALISTLOC=<'*path/CA-list-file*'>

Specifies the location of a single file that contains the public certificate(s) for all of the trusted certificate authorities (CA) in the trust list. This is the CA list location when CAS is acting as a client

Client	Optional
Valid in	Server configuration file, cas.settings file, cas configuration files, and operating system command line
Used by	CAS Server
Category	Security
Requirement	The certificate files and the key files being referenced by these environment variables must be PEM-encoded (Base64 ASCII).
Example	`env.CAS_CALISTLOC='/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/trustedcerts.pem'`

env.CAS_CERTLOC=<'*path/certificate-file*'>

Specifies the path and filename of the file that contains the PEM-formatted certificate to be used for TLS communications.

This environment variable can also point to a certificate chain that starts with the server identity certificate and includes one or more intermediate CA certificates in the order in which they were signed.

Valid in	Server configuration file, cas.settings file, cas configuration files, and operating system command line
Used by	CAS REST API
Category	Security

Requirement Use with env.CAS_PVTKEYLOC on page 100 and env.CAS_PVTKEYPASS on page 101.

Example
```
env.CAS_CERTLOC='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/cas/shared/
default/sas_encrypted.crt'
```

env.CAS_CLIENT_SSL_CA_LIST=<'*path/certificates-file*'>

Specifies the path and filename of the file that contains the list of trusted certificate authorities (CAs). This environment variable can be used by the CAS server or by the client connecting to the CAS server. For the server, this environment variable points to the trust list used to accept connections to the server. For the client, this environment variable points to the trust list that the client uses to connect to the server.

Client Optional

Server Optional

Valid in Server configuration file, cas.settings file, CAS Lua configuration files, and operating system command line

Used by CAS client, Lua client, Python Client, CAS server, SAS 9.4 client

Category Security

Example
```
export CAS_CLIENT_SSL_CA_LIST='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/cacerts/trustedcerts.pem'
export CAS_CLIENT_SSL_CA_LIST='/opt/sas/viya/
config/etc/SASSecurityCertificateFramework/cacerts/vault-ca.crt
export <SASHome>/SASSecurityCertificateFramework/1.1/
cacerts/trustedcerts.pem
```

env.CAS_CLIENT_SSL_CERT=<'*path/certificate-file*'>

Specifies the path and filename of the file that contains the certificate that the client uses to connect to the server for TLS communications. This environment variable is used when accepting connections to the CAS server.

This environment variable can also point to a certificate chain that starts with the server identity certificate and includes one or more intermediate CA certificates in order that they are signed.

Server Optional

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS server

Category Security

Notes Environment Variables CAS_CLIENT_SSL_KEY=, CAS_CLIENT_SSL_KEYPW=, CAS_CLIENT_SSL_CERT= are specified together.

The contents of this file are not confidential.

Example
```
env.CAS_CLIENT_SSL_CERT='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/sas_encyrpted.crt'
env.CAS_CLIENT_SSL_CERT='/opt/sas/viya/config/etc/
```

```
SASSecurityCertificateFramework/tls/certs/cas/default/sas_encrypted.crt'
```

env.CAS_CLIENT_SSL_CLIENT_CERT=<'*path/certificate-file*'>

Specifies the path and filename of the file that contains the certificate that the client uses to connect to the server for TLS communications.

This environment variable is specified when the client presents a certificate to the server. In most configurations, only the server presents a certificate to the client.

This environment variable can also point to a certificate chain that starts with the server identity certificate and includes one or more intermediate CA certificates in order that they are signed.

Client Optional

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS client

Category Security

Notes The contents of this file are not confidential.

Environment Variables CAS_CLIENT_SSL_CLIENT_KEY=, CAS_CLIENT_SSL_CLIENT_KEYPW=, CAS_CLIENT_SSL_CLIENT_CERT= are specified together.

Example
```
env.CAS_CLIENT_SSL_CLIENT_CERT='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/CASClient.crt'
env.CAS_CLIENT_SSL_CERT='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/cas/default/CASClient.crt'
```

env.CAS_CLIENT_SSL_CLIENT_KEY=<'*path/key-file*'>

Specifies the path and filename of the file that contains the private key for the client to use to connect to the server for TLS communications.

This environment variable is specified when the client presents a certificate to the server. In most configurations, only the server presents a certificate to the client.

Client Optional

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS client

Category Security

Note Environment Variables CAS_CLIENT_SSL_CLIENT_KEY=, CAS_CLIENT_SSL_CLIENT_KEYPW=, CAS_CLIENT_SSL_CLIENT_CERT= are specified together.

Tip The contents of this file should be kept confidential.

Example
```
env.CAS_CLIENT_SSL_CLIENT_KEY='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/CASClient.key'
env.CAS_CLIENT_SSL_CLIENT_KEY='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/cas/default/CASClient.key'
```

env.CAS_CLIENT_SSL_CLIENT_KEYPW=<'*password*'>

Specifies the password for the client's private key. The password in this variable should match the password used to generate the private key file specified by the CAS_CLIENT_SSL_CLIENT_KEY= environment variable.

Note: This password is not encoded.

This password should be set in a Lua configuration file that is readable only by the CAS service account.

Client	Optional
Valid in	Server configuration file, cas.settings file, and operating system command line
Used by	CAS client
Category	Security
Note	Environment Variables CAS_CLIENT_SSL_CLIENT_KEY=, CAS_CLIENT_SSL_CLIENT_KEYPW=, CAS_CLIENT_SSL_CLIENT_CERT= are specified together.
Example	`env.CAS_CLIENT_SSL_CLIENT_KEYPW='encryptedpassword'`

env.CAS_CLIENT_SSL_KEY=<*path/key-file*>

Specifies the path and filename of the file that contains the private key for the client to be used for TLS communications. This key is used when accepting connections to the server.

Server	Optional
Valid in	Server configuration file, cas.settings file, and operating system command line
Used by	CAS server
Category	Security
Note	Environment Variables CAS_CLIENT_SSL_KEY=, CAS_CLIENT_SSL_KEYPW=, CAS_CLIENT_SSL_CERT= are specified together.
Tip	The contents of this file should be kept confidential.
Example	`env.CAS_CLIENT_SSL_KEY='/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/sas_encrypted.key'` `env.CAS_CLIENT_SSL_KEY='/opt/sas/viya/config/etc/SASSecurityCertificateFramework/private/cas/default/sas_encrypted.key'`

env.CAS_CLIENT_SSL_KEYPW=<'*password*'>

Specifies the password for the server's private key. The password in this variable should match the password used to generate the private key file specified by the CAS_CLIENT_SSL_KEY= environment variable.

Note: This password is not encoded.

Server Optional

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS server

Category Security

Note Environment Variables CAS_CLIENT_SSL_KEY=, CAS_CLIENT_SSL_KEYPW=, CAS_CLIENT_SSL_CERT= are specified together.

Example `env.CAS_CLIENT_SSL_KEYPW='password'`

env.CAS_CLIENT_SSL_KEYPWLOC==<'*path/certificate-file*'>

Specifies the location of the password file for the server's private key. The password in this variable should match the password used to generate the private key file specified by the CAS_CLIENT_SSL_KEY= environment variable.

Server Optional

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS server

Category Security

Note Environment Variables CAS_CLIENT_SSL_KEY= and CAS_CLIENT_SSL_CERT= are specified together.

Example
```
env.CAS_CLIENT_SSL_KEYPWLOC='/opt/sas/viya/
SASSecurityCertificateFramework/private/cas/default/encryption.key'
env.CAS_CLIENT_SSL_KEYPWLOC='/opt/sas/viya/
SASSecurityCertificateFramework/private/encryption.key'
```

env.CAS_CLIENT_SSL_REQUIRED=<'true' | 'false' >

Determines whether encryption is used between the client and the server.

Valid in Server configuration file, cas.settings file, CAS Lua config files, and operating system command line

Used by CAS server

Category Security

Example `env.CAS_CLIENT_SSL_REQUIRED='true'`

env.CAS_INTERNODE_DATA_SSL=<true | false>

Enables encryption for the analytics cluster when set to **`true`**. This value must be the same on every node in the cluster.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example `env.CAS_INTERNODE_DATA_SSL=true`

env.CAS_INTERNODE_SSL_CA_LIST=<'*path/keystore*'>
Specifies the path and filename of the file that contains the list of trusted certificate authorities (CAs). This setting is likely to be the same for all nodes in the grid.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example
```
env.CAS_INTERNODE_SSL_CA_LIST="/opt/sas/viya/config
/etc/SASSecurityCertificateFramework/cacerts/trustedcerts.pem"
```

env.CAS_INTERNODE_SSL_CERT=<'*path/certificate-file*'>
Specifies the path and filename of the file that contains the certificate to be used for TLS communications for the certificate specific to the node being configured.

This environment variable can point to a certificate chain that starts with the server identity certificate and includes the intermediate CA certificates in the order in which they are signed.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example
```
env.CAS_INTERNODE_SSL_CERT="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/sas_encrypted.crt"
env.CAS_INTERNODE_SSL_CERT="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/cas/default/sas_encrypted.crt"
```

env.CAS_INTERNODE_SSL_KEY=<'*path/key-file*'>
Specifies the path and filename of the file that contains the private key used to sign the certificate specific to the CAS node being configured. This setting is likely to be different on every machine.

Valid in server configuration file, cas.settings file, and operating system command line

Category Security

Tip The contents of this file should be kept confidential.

Example
```
env.CAS_INTERNODE_SSL_CERT="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/cas/default/sas_encrypted.key"
env.CAS_INTERNODE_SSL_CERT="/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/sas_encrypted.key'
```

env.CAS_INTERNODE_SSL_KEYPW=<'*password*'>
Specifies the password for the private key.

The setting is the password for the encrypted private key used to sign the certificate specific to the node being configured.

Note: This password is not encoded.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example `env.CAS_INTERNODE_SSL_KEYPW='encryptedpassword'`

env.CAS_INTERNODE_SSL_KEYPWLOC =<'*path/certificate-file*'>
Specifies the location of the password/key file for the encrypted private key used to sign the certificate specific to the node being configured.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example
```
env.CAS_INTERNODE_SSL_KEYPWLOC='opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/cas/default/encryption.key'
env.CAS_INTERNODE_SSL_KEYPWLOC='opt/sas/viya/config/
SASSecurityCertificateFramework/private/encryption.key'
```

env.CAS_PKCS12LOC=<'*path/certificate-file*'>
Specifies the path and filename of the PKCS12 (DER formatted binary) file that contains the certificate and private key.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example
```
env.CAS_PKCS12LOC='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/sas_encrypted.p12'
env.CAS_PKCS12LOC='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/tls/certs/cas/default/sas_encrypted.p12'
```

env.CAS_PKCS12PASS=<'*path/password-file*'>
Specifies the password for the private key specified by env.CAS_PKCS12LOC.

Note: This password is not encoded.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example `env.CAS_PKCS12PASS='password'`

env.CAS_PVTKEYLOC=<'*path/key-file*'>
Specifies the path and filename of the file that contains the private key that corresponds to the digital certificate.

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS REST API

Category Security

Tip The contents of this file should be kept confidential.

Example
```
env.CAS_PVTKEYLOC='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/sas_encrypted.key'
env.CAS_PVTKEYLOC='/opt/sas/viya/config/etc/
```

```
SASSecurityCertificateFramework/private/cas/default/sas_encrypted.key'
```

env.CAS_PVTKEYPASS=<'*password*'>
Specifies the password for the private key specified by env.CAS_PVTKEYLOC.

Note: This password is not encoded.

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS REST API

Category Security

Example `env.CAS_PVTKEYPASS='password'`

env.CAS_PVTKEYPASSLOC=<'*path/key-file*'>
Specifies the path and filename of the file that contains the private key that corresponds to the digital certificate.

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS REST API

Category Security

Tip The contents of this file should be kept confidential.

Example
```
env.CAS_PVTKEYLOC='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/encryption.key'
env.CAS_PVTKEYLOC='/opt/sas/viya/config/etc/
SASSecurityCertificateFramework/private/cas/default/encryption.key'
```

env.CAS_SSLCLIENTAUTH=<true>
When set to any value, causes client certificates to be validated when TLS connections are initiated.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example `env.CAS_SSLCLIENTAUTH=true`

env.CAS_SSLCRLCHECK=<false>
When set to any value, causes the certificate revocation list (CRL) to be checked when TLS connections are initiated.

Valid in Server configuration file, cas.settings file, and operating system command line

Category Security

Example `env.CAS_SSLCRLCHECK='false'`

env.CAS_SSLNAMECHECK=<true>

When set to any value, causes the name of the server to be checked against the host name specified in the server identity certificate pointed to by env.CAS_CERTLOC to validate the server's identity.

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS REST API

Category Security

Example `env.CAS_SSLNAMECHECK=true`

env.CAS_SSLREQCERT=<"NEVER | ALLOW | TRY | DEMAND">

Specifies what the client should do with the information sent by the server.

The variable env.CAS_SSLREQCERT must specify one of the following values:

- **DEMAND**

  The client asks for a server certificate. For the connection to continue, the server must provide a certificate, and the certificate must pass validation.

CAUTION

For security purposes, DEMAND is the setting that should be specified.

- **NEVER**

  The client never asks the CAS server for a certificate.

- **ALLOW**

  The client asks the server for a certificate. If the server does not provide a certificate, or if the certificate does not pass validation, the TLS connection continues.

- **TRY**

  The client asks the server for a certificate. If the server does not provide a certificate, the TLS connection continues. However, if the certificate does not pass validation, the TLS connection fails.

Valid in Server configuration file, cas.settings file, and operating system command line

Used by CAS REST API

Category Security

Example `env.CAS_SSLREQCERT='DEMAND'`

env.CAS_USE_HTTPS_ALL=<'TRUE' | 'FALSE'>

When set to TRUE, causes connections using the CAS REST API to use HTTPS.

Valid in Server configuration file, cas.settings file, CAS Lua config files, and operating system command line

Used by CAS REST API

Category Security

Default false

Example `env.CAS_USE_HTTPS="FALSE"`

Configuration File Options for Data Transfer

CAS Configuration File Options for Data Transfer with the SAS Data Connect Accelerator

CAS server options are stored in a configuration file. During deployment, this configuration file, casconfig_deployment.lua, is created in the **/opt/sas/viya/config/etc/cas/default** directory. When you start the server with the `service sas-viya-cascontroller-default start` command, the options are read.

For more information about the CAS server configuration files, see see, "Understanding Configuration Files and Start-up Files" in *SAS Viya Administration: SAS Cloud Analytic Services*.

These are the configuration options that can be used for data transfer encryption with the SAS Data Connect Accelerator. For a complete list of CAS configuration file options, see *SAS Viya Administration: SAS Cloud Analytic Services*.

cas.DCSSLCERTISS='*issuer-of-digital-certificate*'
Specifies the name of the issuer of the digital certificate that TLS should use.

cas.DCSSLCERTLOC='*pathname*'
Specifies the location of a file that contains the digital certificate for the machine's public key also known as the identity certificate. This is used by servers to send to clients for authentication.

Requirement The certificate file must be PEM-encoded (base64).

See "Certificate File Formats" on page 68

cas.DCSSLCERTSERIAL='*serial-number*'
Specifies the serial number of the digital certificate that TLS should use.

cas.DCSSLCERTSUBJ='*subject-name*'
Specifies the subject name of the digital certificate that TLS should use.

cas.DCSSLPKCS12LOC='*pathname*'
Specifies the location of the PKCS #12 DER encoding package file that contains the identity certificate and the private key.

Requirement If the cas.DCSSLPKCS12LOC= option is specified, the PKCS #12 DER encoding package must contain both the certificate and private key. The cas.DCSSLCERTLOC= and cas.DCSSLPVTKEYLOC= options are ignored.

See "Certificate File Formats" on page 68

cas.DCSSLPKCS12PASS on page 104

cas.DCSSLPKCS12PASS=*password*

Specifies the password that TLS requires in order to decrypt the PKCS #12 DER encoding package file.

Interaction The PKCS #12 DER encoding package is stored in the file that is specified by using the cas.DCSSLPKCS12LOC= option.

Note The cas.DCSSLPKCS12PASS= option is required only when the PKCS #12 DER encoding package is encrypted.

See cas.DCSSLPKCS12LOC on page 103

cas.DCSSLPVTKEYLOC='*pathname*'

Specifies the location of the file that contains the private key that corresponds to the digital certificate that was specified by using the cas.DCSSLCERTLOC= option.

Requirement The key must be PEM-encoded (base64).

Note The cas.DCSSLPVTKEYLOC= option is required at the server only if the cas.DCSSLCERTLOC= option is also specified at the server.

See "Certificate File Formats" on page 68

cas.DCSSLCERTLOC on page 103

cas.DCSSLPVTKEYPASS on page 104

cas.DCSSLPVTKEYPASS=*password*

Specifies the password that TLS requires in order to decrypt the private key.

Interaction The private key is stored in the file that is specified by using the cas.DCSSLPVTKEYLOC= option.

Note The cas.DCSSLPVTKEYPASS= option is required only when the private key is encrypted. OpenSSL performs key encryption.

See cas.DCSSLPVTKEYLOC on page 104

cas.DCSSLPVTKEYPASSLOC='*pathname*'

Specifies the location of the file that contains the password that TLS requires in order to decrypt the private key.

Interactions The private key is stored in the file that is specified by using the cas.DCSSLPVTKEYLOC= option.

If the DCSSLPVTKEYPASS option is specified, it is used. Otherwise, the DCSSLPVTKEYPASSLOC option is used.

See cas.DCSSLPVTKEYPASS on page 104

cas.DCTCPMENCRYPT='YES' | 'NO' | 'OPT'

Specifies whether encryption is required for the connection.

'YES' means that data encryption is required.

'NO' means that data will be sent as plaintext.

'OPT' means that data encryption is preferred but not required.

Aliases: 'REQ' or 'REQUIRED' for 'YES'

'OPTIONAL' for 'OPT'

Default: No value. However, if you specify cas.DCTCPMENCRYPTALGORITHM='SSL' and cas.DCTCPMENCRYPT= is not specified, cas.DCTCPMENCRYPT= defaults to 'YES'.

Requirement: The option values must be uppercase.

Interactions: Encryption is determined by the setting of this option on both the client (data provider) and server (CAS) side. For more information, see "DCTCPMENCRYPT Option Setting Interaction" on page 38.

If you specify cas.DCTCPMENCRYPTALGORITHM='SSL' and cas.DCTCPMENCRYPT is not specified, cas.DCTCPMENCRYPT defaults to 'YES'.

Note: If you have multiple clusters and you set the DCTCPMENCRYPT= option on the client (data provider) side to YES for one cluster and NO for another cluster, you might want to set the server (CAS) side cas.DCTCPMENCRYPT= option to 'OPT'.

cas.DCTCPMENCRYPTALGORITHM='SSL'

Specifies the algorithm to be used for encrypted data transfers using the SAS Data Connect Accelerator.

Default: SSL

Requirement: The option value, SSL, must be uppercase.

Interaction: If you specify cas.DCTCPMENCRYPTALGORITHM='SSL' and cas.DCTCPMENCRYPT= is not specified, cas.DCTCPMENCRYPT= defaults to 'YES'.

Note: SSL (TLS) is the only algorithm available at this time for encrypted data transfers using the SAS Data Connect Accelerator.

dcsecurity.properties File Options for Data Transfer with the SAS Data Connect Accelerator

You can set the following options in the dcsecurity.properties file. The dcsecurity.properties file is located in the following directory on your cluster.

- For Teradata, **`/opt/SAS/SASTKInDatabaseServerForTeradata/14.00000/security`**
- For Hadoop, ***`EPInstallDir`*`/sasexe/SASEPHOME/security`**

The syntax for setting the properties is as follows:

```
-option-name
option-setting
```

Here is an example.

```
-DCTCPMENCRYPTALGORITHM SSL
```

These are the options that you can set in the dcsecurity.properties file options for data transfer encryption with the SAS Data Connect Accelerator.

DCSSLCACERTDIR '*pathname*'
Specifies the directory where the public certificates for all of the trusted certificate authorities (CA) in the trust list are filed.

Requirement: Each CA certificate file must be PEM-encoded (base64).

Interaction: The DCSSLCALISTLOC= option can be used instead of or in conjunction with the DCSSLCACERTDIR= option.

Note: Different versions of OpenSSL generate different hash values. For example, OpenSSL 0.9.8 generates different hash values from those generated by OpenSSL 1.x.

See "Certificate File Formats" on page 68

DCSSLCALISTLOC '*pathname*'
Specifies the location of a single file that contains the public certificate(s) for all of the trusted certificate authorities (CA) in the trust list.

Requirement: The CA file must be PEM-encoded (base64).

Interaction: The DCSSLCACERTDIR= option can be used instead of or in conjunction with the DCSSLCALISTLOC= option.

See "Certificate File Formats" on page 68

DCTCPMENCRYPT YES | NO | OPT
Specifies whether encryption is required for the connection.

YES means that data encryption is required.

NO means that data will be sent as plaintext.

OPT means that data encryption is preferred but not required.

Aliases: REQ or REQUIRED for YES

OPTIONAL for OPT

Default: NO. However, if you specify the DCTCPMENCRYPTALGORITHM option and DCTCPMENCRYPT is not specified, DCTCPMENCRYPT defaults to YES.

Requirement: The option values must be uppercase.

Interaction: Encryption is determined by the setting of this option on both the client (data provider) and server (CAS). For more information, see "DCTCPMENCRYPT Option Setting Interaction" on page 38.

DCTCPMENCRYPTALGORITHM SSL
Specifies the algorithm to be used for encrypted data transfers using the SAS Data Connect Accelerator.

Default: SSL

Requirement The option value, SSL, must be uppercase.

Note TLS is the only algorithm available at this time for encrypted data transfers using the SAS Data Connect Accelerator.

Examples

Create Site-Signed or Third-Party-Signed Certificates in PEM Format

Generate a Private Key in RSA Format and a Certificate Signing Request

The tasks that you perform to request a digital certificate for the CA, the server, and the client are similar. However, the values that you specify are different.

In this example, Proton, Inc. is the organization that is applying to become a CA. A certificate request is sent to a certificate authority to get it signed, thereby becoming a CA. After Proton, Inc. becomes a CA, it can serve as a CA for issuing other digital certificates to clients and servers on its network. The certificates generated by the Proton, Inc. CA are considered site-signed certificates.

Note: You can also sign the certificate yourself if you have your own certificate authority or create a self-signed certificate.

To create a site-signed certificate using OpenSSL, first you need to generate a private key in RSA format. This file is not protected with a passphrase and is saved in the ASCII (Base64-encoded) PEM format.

1 Edit your existing openssl.cnf file or create an openssl.cnf file. OpenSSL by default looks for a configuration file in **`/usr/lib/ssl/openssl.cnf`**. It is good practice to add `-config ./openssl.cnf` to the commands OpenSSL CA or OpenSSL REQ to ensure that OpenSSL is reading the correct file.

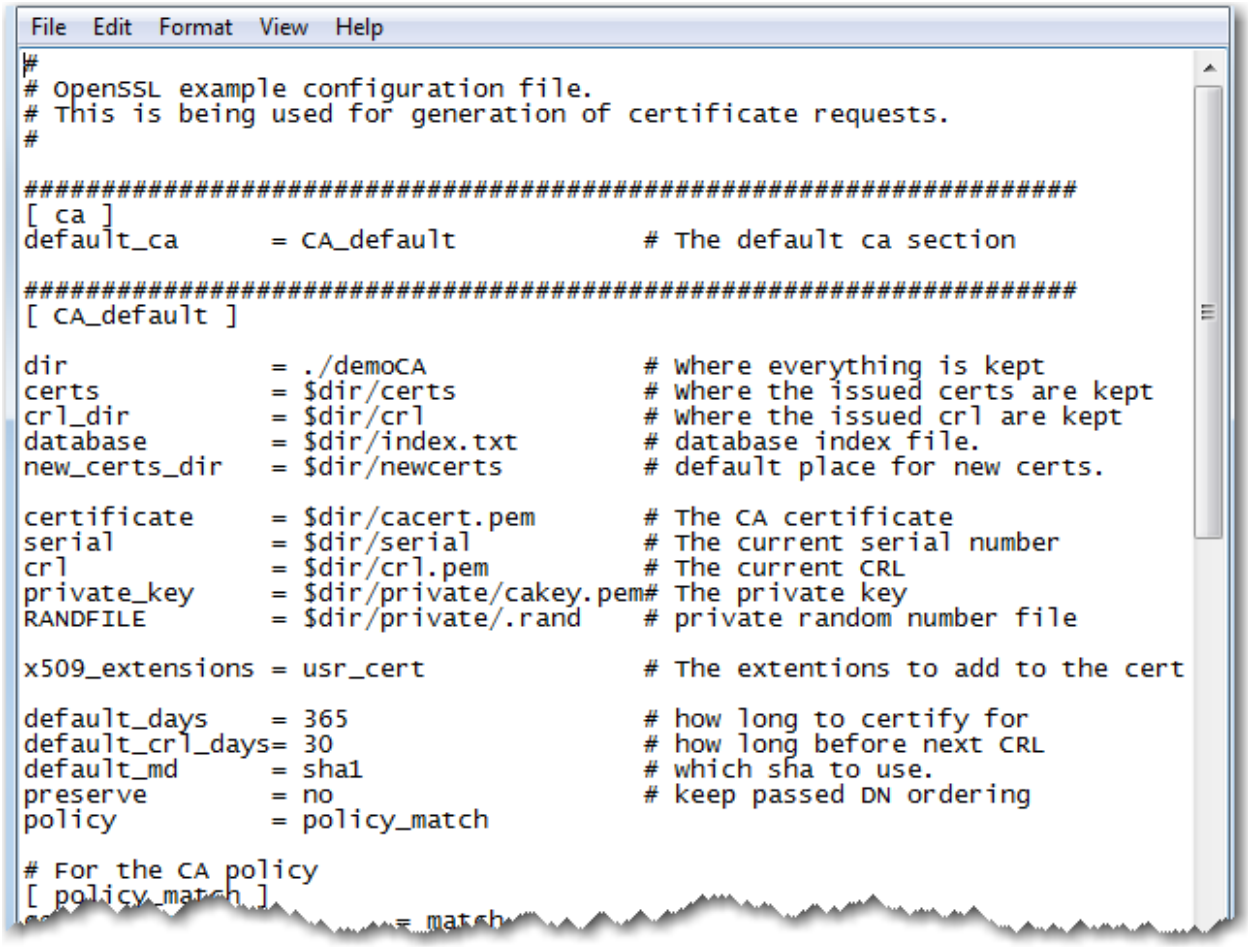
   Note: You can find where the openssl.cnf file is located by submitting the OpenSSL command:

   ```
   openssl version -d
   ```

   .

   Here is an example of some of the information that can be specified in the openssl.cnf file. You need to specify where OpenSSL should look for information. Here is a partial file example. Much more information about certificates can be specified.

Figure 3 *Example of an OpenSSL.cnf File*



```
File  Edit  Format  View  Help
#
# OpenSSL example configuration file.
# This is being used for generation of certificate requests.
#

##################################################################
[ ca ]
default_ca      = CA_default            # The default ca section

##################################################################
[ CA_default ]

dir             = ./demoCA              # where everything is kept
certs           = $dir/certs            # where the issued certs are kept
crl_dir         = $dir/crl              # where the issued crl are kept
database        = $dir/index.txt        # database index file.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE        = $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert

default_days    = 365                   # how long to certify for
default_crl_days= 30                    # how long before next CRL
default_md      = sha1                  # which sha to use.
preserve        = no                    # keep passed DN ordering
policy          = policy_match

# For the CA policy
[ policy_match ]
                = match
```

2 Select the `apps` subdirectory of the directory where OpenSSL was built.

3 Initialize OpenSSL.

```
$ openssl
```

4 Issue the appropriate command to request a digital certificate. In the example below, we are creating an RSA private key and generating a certificate signing request all at once.

Table 7 *OpenSSL Commands for Requesting a Private Key*

Recipient of Certificate Request	OpenSSL Command
CA	`req -config ./openssl.cnf -new -out ca.csr -newkey rsa:2048 -keyout cakey.pem -sha256`
Server	`req -config ./openssl.cnf -new -out server.csr -newkey rsa:2048 -keyout serverkey.pem -sha256`
Client	`req -config ./openssl.cnf -new -out client.csr -newkey rsa:2048 -keyout clientkey.pem -sha256`

Table 8 *Arguments and Values Used in OpenSSL Commands*

OpenSSL Arguments and Values	Functions
`req`	Requests a certificate.
`-config ./openssl.cnf`	Specifies the storage location for the configuration details for the OpenSSL program.
`-new`	Identifies the request as new.
`-out ca.csr`	Specifies the storage location for the certificate request.
`-newkey rsa:2048`	Generates a new private key along with the certificate request that is 2048 bits in length using the RSA algorithm.
`-keyout cakey.pem`	Specifies the storage location for the private key.
`-nodes`	Prevents the private key from being encrypted. Not Recommended. For best practice, encrypt the private key.
-sha256	Specifies the SHA-256 hash algorithm be used. Without this option, the default is SHA-1.

5 Informational messages are displayed and prompts for additional information appear according to the specific request.

To accept a default value, press the Enter key. To change a default value, type the appropriate information and press the Enter key.

Note: Unless the -NODES option is used in the OpenSSL command when creating a digital certificate request, OpenSSL prompts you for a password before allowing access to the private key.

Here is an example of a request for a digital certificate:

```
OpenSSL> req -config ./openssl.cnf -new -out ca.req -newkey rsa:2048
-keyout privkey.pem -nodes
Using configuration from ./openssl.cnf
Generating a 2048 bit RSA private key
.........................++++++
.........................................++++++
writing new private key to 'cakey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
```

```
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [North Carolina]:
Locality Name (city) [Cary]:
Organization Name (company) [Proton Inc.]:
Organizational Unit Name (department) [IDB]:
Common Name (YOUR name) []: proton.com
Email Address []:Joe.Bass@proton.com

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

The request for a digital certificate is complete.

Note: For the server, the Common Name must be the name of the computer that the server runs on. In our examples, we are using proton.com.

Generate a Public Certificate

1 Issue the appropriate command to generate a public certificate from the certificate signing request.

Table 9 *OpenSSL Commands for Generating Digital Certificates*

Recipient of Generated Certificate	OpenSSL Command
CA	`x509 -req -in ca.csr -signkey cakey.pem -out cacert.pem -sha256` **Note:** This command generates a self-signed certificate.
Server	`ca -config ./openssl.cnf -in server.csr -out server.pem -md sha256` **Note:** This command creates certificates signed by the CA. These are defined in the openssl.cnf file.
Client	`ca -config ./openssl.cnf -in client.csr -out client.pem -md sha256` **Note:** This command creates certificates signed by the CA. These are defined in the openssl.cnf file.

Table 10 *Arguments and Values Used in OpenSSL Commands to Generate a Certificate*

OpenSSL Arguments and Values	Functions
`x509`	Identifies the certificate display and signing utility. Typically used to generate a self-signed certificate.
`-req`	Specifies that a certificate be generated from the request.
`ca`	Identifies the Certificate Authority utility.
`-config ./openssl.cnf`	Specifies the storage location for the configuration details for the OpenSSL utility.
`-in filename.csr`	Specifies the storage location for the input for the certificate request.
`-out filename.pem`	Specifies the storage location for the certificate.
`-signkey cakey.pem`	Specifies the private key that is used to sign the certificate that is generated by the certificate request.
-md sha256	Specifies the SHA-256 hash algorithm be used. Without this option, the default is SHA-1.

2 Informational messages are displayed and prompts for additional information appear according to the specific request.

To accept a default value, press the Enter key. To change a default value, type the appropriate information, and press the Enter key.

Here is a sample of the messaging from a CSR for a server digital certificate:

Note: The password is for the CA's private key.

```
Using configuration from ./openssl.cnf
Enter PEM pass phrase: password
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'US'
stateOrProvinceName   :PRINTABLE:'NC'
localityName          :PRINTABLE:'Cary'
organizationName      :PRINTABLE:'Proton, Inc.'
```

```
organizationalUnitName:PRINTABLE:'IDB'
commonName            :PRINTABLE:'proton.com'
Certificate is to be certified until April 16 17:48:27 2016 GMT (365 days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries Data Base Updated
```

The subject's Distinguished Name is obtained from the digital certificate request.

The generation of a digital certificate is complete.

Check Your Digital Certificate Using OpenSSL

To check a digital certificate, issue the following command:

```
openssl> x509  -text -in filename.pem
```

A digital certificate contains data that was collected to generate the digital certificate timestamps, a digital signature, and other information. However, because the generated digital certificate is encoded (usually in PEM format), it is unreadable.

Create a Certificate Chain in PEM Format Using OpenSSL

After generating a digital certificate for the CA, the server, and the client (optional), you must identify for the OpenSSL client application one or more CAs that are to be trusted. This list is called a *chain of trust*. This chain includes a set of certificates, where each one has been signed by the one that comes after it.

On the client, if there is only one CA to trust, specify in the client application the name of the file that contains the OpenSSL CA digital certificate. If multiple CAs are to be trusted, you can copy and paste into a new file the contents of all the digital certificates of CAs to be trusted by the client application. These CAs can be primary, intermediate, or root certificates. Add the root CAs to the client's truststore.

For the server, do not include the Root CA in the server's certificate chain.

To manually create a new chain of trust, use the template shown below.

```
(Your Server Certificate - ssl.crt)

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----

(Your Intermediate CA Certificate(s))

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----

(Your Root CA Certificate)
```

```
-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----
```

The content of the digital certificate in this example is represented as `<PEM encoded certificate>`. The content of each digital certificate is delimited with a `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` pair. All text outside the delimiters is ignored. Therefore, you might not want to use delimited lines for descriptive comments.

Generally, OpenSSL returns .pem files, CAs return .crt files (Microsoft returns .cer files). Instead of manually cutting and pasting these files together (regardless of your file extension), you can also concatenate the certificate authority files. For example, you can take an intermediate authority certificate file, a root authority certificate file, and a primary certificate file and concatenate them into a single PEM file. Here is an example of concatenating certificates:

```
cat server.pem > certchain.pem
cat intermediateCA.pem >> certchain.pem
cat rootCA.pem >> certchain.pem
```

Because the digital certificate is encoded, it is unreadable. You will see a string of hexadecimal characters. To view the file contents, you can use the following OpenSSL commands for your file type:

```
openssl x509 -in cert.pem -text -noout
openssl x509 -in cert.cer -text -noout
openssl x509 -in cert.crt -text -noout
```

Use the following OpenSSL command to view a DER-encoded certificate:

```
openssl x509 -in certificate.der -inform der -text -noout
```

Note: If you are including a digital certificate that is stored in DER format into your certificate chain, you must first convert it to PEM format. For more information, see "Convert DER to PEM File Format" on page 51.

Verify Certificates in the Trust Chain Using OpenSSL

Clients and servers exchange and validate each other's digital certificates. All of the CA certificates that are needed to validate a server certificate compose a trust chain. All CA certificates in a trust chain have to be available for server certificate validation.

You can use the following OpenSSL command to verify that certificates are signed by a recognized certificate authority (CA):

```
openssl verify -verbose -CAfile <your-CA_file>.pem <your-server-cert>.pem
```

If your local OpenSSL installation recognizes the certificate or its signing authority and everything checks out (dates, signing chain, and so on), you get a simple OK message.