Paper 426-2017

Transport Layer Security (TLS) Configuration for SAS® 9.4 and SAS® Viya™ Components Made Easy

Heesun Park, SAS Institute Inc.

ABSTRACT

Transport Layer Security (TLS) configuration for SAS® components is essential to protect data in motion. All necessary encryption arrangement is established through a TLS handshake between the client and the server side. Many SAS® 9.4 and SAS® Viya™ components can be a client side, a server side, or both. SAS documentation primarily provides how-to steps for the configuration. This paper examines the X.509 certificate and the TLS handshake protocol, which are the basic building blocks of the secure communication. The paper focuses on the logic behind the setup and how various types of certificates are used in the configuration. Many unique client and server combinations of SAS components are illustrated and explained with the best-practice suggestions.

INTRODUCTION TO PUBLIC KEY CRYPTOGRAPHY AND THE RSA ALGORITHM

The Public Key Cryptography (PKC) was first introduced in 1978 by the paper published by Rivest, Shamir, and Adelman (later known as RSA) of MIT [1]. The first practical use of PKC was implemented by the SSL (Secure Socket Layer) handshake protocol by Netscape. The term SSL is being replaced by TLS (Transport Layer Protocol) in recent days. We will use SSL and TLS interchangeably in this paper.

To properly set up and manipulate TLS configuration and certificate-related materials, it is important, in my opinion, to have a solid understanding of the fundamentals of Public Key Cryptography (PKC). From the historical perspective, PKC is the major revolution in encryption technology in recent years. It is asymmetric encryption technology, meaning that we encrypt with one key and decrypt with other key (public key and private key pair). The RSA algorithm is the most widely used asymmetric encryption algorithm of PKC. As well, understanding how the x.509 certificate, which is the implementation of the RSA algorithm, is used in TLS protocol is important. During the TLS handshake stage, PKC is used, but not in subsequent data transmission. One of the major objectives of the TLS handshake is to agree upon the symmetric encryption algorithm to use and to generate the encryption key securely and dynamically using PKC.

At our discretion, we can check our level of understanding of PKC and TLS technology by answering the questions below. Once we understand it, things become relatively simple and easy.

- Mathematically, what constitutes the public key and private key (based on RSA algorithm)?
- Can we identify the modulus and the exponent from the public key in the x.509 certificate?
- What is the logic embedded in the x.509 certificate to prove the authenticity of the certificate?
- What is the purpose of the hash value and hash algorithm in the x.509 certificate?
- What does it mean to digitally sign the certificate?
- Why do we need a Certificate Authority (CA) certificate in the trusted keystore?
- When and where do we actually use a CA's public key?
- What are the three major goals of the SSL/TLS handshake?
- How does the encryption key get created for the agreed upon cipher suite (Symmetric encryption algorithm)?
- When do we actually use the public key from the server identity certificate?

The addition of the TLS protocol to the HTTP protocol has made secure communication on the web possible, and to a large extent, has made today's e-commerce a reality. The TLS protocol works in two stages [2].

The first stage is the public key cryptography-based handshake during which the two parties that want to communicate agree upon a traditional symmetric encryption algorithm to use in the subsequent communication. The parties also exchange session specific information (a “pre-master secret”) in order to generate the encryption key (“session key”) that each side will use with the mutually agreed upon encryption algorithm. Because the session key is dynamically generated for each session and is destroyed once the session is terminated, the security exposure of the session key is minimal.

The second stage is the use of symmetric encryption with the dynamically generated session key for the rest of the session.

The public key cryptography used in the first stage is an asymmetric encryption scheme that makes use of a public key and a private key. The public key and its matching private key are calculated based on prime number theory. The content encrypted with the public key can only be decrypted with the matching private key, and vice versa. The delivery mechanism of the public key is the X.509 TLS certificate.

ANATOMY OF X.509 CERTIFICATE

X.509 is the very widely used TLS certificate standard that defines the fields in a certificate and the uses of those fields. There are three types of TLS certificates: server identity, client, or personal certificates (we use these terms interchangeably) and Certificate Authority (CA) certificates. To ensure the authenticity of the certificates, server identity certificates and client certificates must be “signed” by a Certificate Authority (CA). Signing the certificate means that a one-way hash of the data in the certificate is encrypted with a CA’s private key. To decrypt and validate the certificate, the consumer of the certificate needs the CA’s certificate (that contains its public key) in its “trusted signer” area. These days, most browsers come with well-established CA certificates.

Among other things, an X.509 certificate contains the following fields.

- The “subject” field is used to identify the owner of the certificate.
- The “issuer” field indicates the name of the CA who has signed the certificate.
- The “Subject Public Key Info” field contains the public key algorithm used and the public key.
- The “signature” is the message digest (or the hash value of the certificate) encrypted with CA’s private key.
- The “subject” field is used for user identification in the case the client certificate is required for authentication for two-way TLS. It contains an LDAP like tree structure and typically works well when the application server’s user registry is an LDAP server.

The creation of TLS certificates has become relatively easy with open-source tools like OpenSSL and the Java keytool from Oracle. Most enterprise level companies these days have the Certificate Authority capability to sign their own server identity certificate and client certificates. We refer to that process as 3rd party signed certificate. Self-signed certificates should not be used in external facing production deployment.

SERVER IDENTITY CERTIFICATE GENERATION WITH ACTIVE DIRECTORY CERTIFICATE SERVICE (ADCS)

For enterprise level TLS configuration, server identity certificates used in the configuration need to be created and signed by an enterprise CA. For Windows domain networks, the most popular certificate management tool is Active Directory Certificate Service (ADCS). There are many ways to provide certificate requests to ADCS. One can enter certificate information directly from ADCS or create a request using Java keytool or OpenSSL and submit them to ADCS for signing. When we create certificates directly from ADCS, typically we create one in PCKS12 format or in PKCS7 format. Certificates in PKCS12 (.p12 or .pfx suffix) format also include a private key inside of the certificate. Certificates in PKCS12 format can be used as is for some TLS configurations or the key and the certificate can be separated using Java keytool and OpenSSL utilities. Figure 1 shows how to create a PCKS12 format certificate.

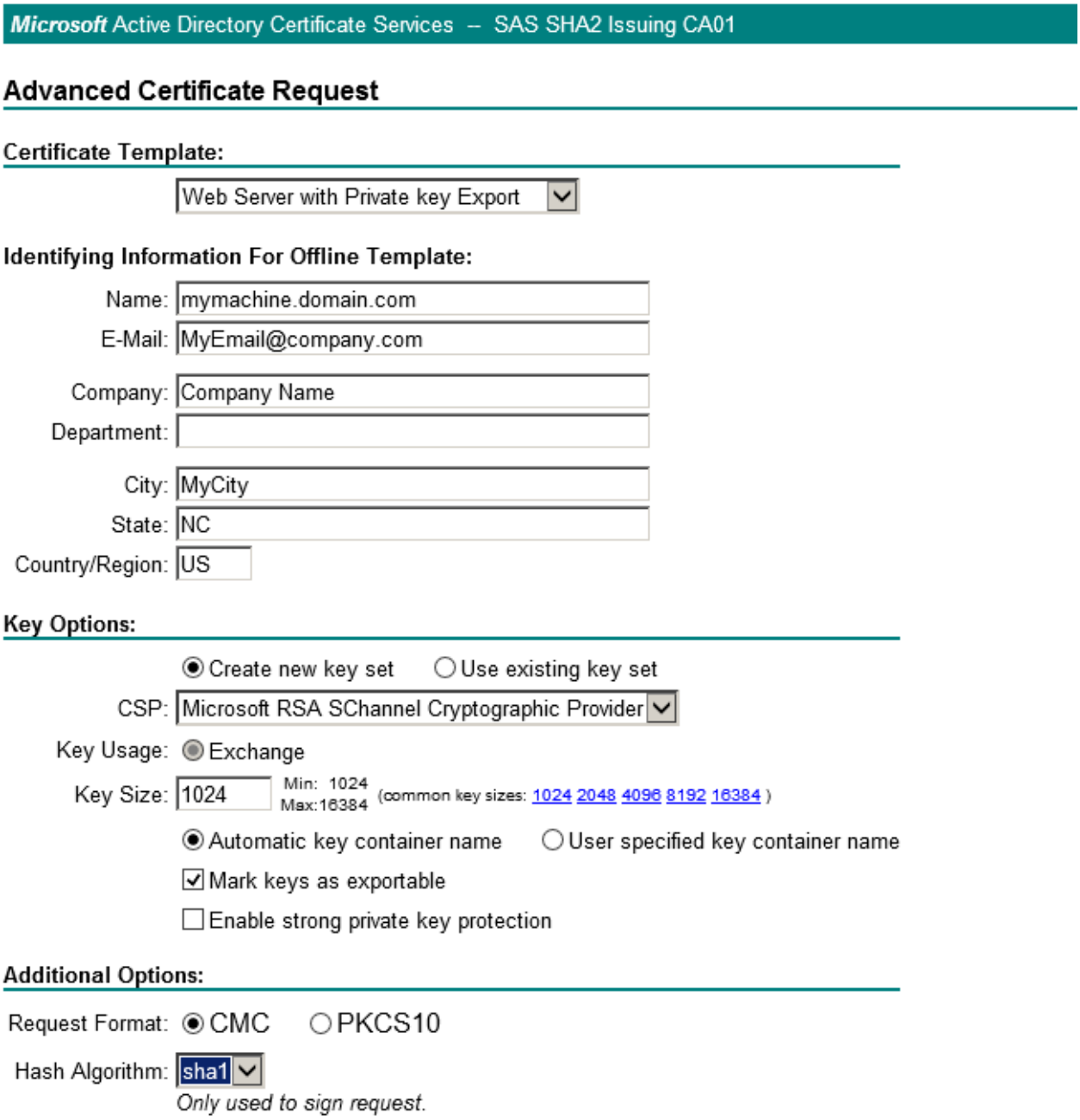

Figure 1. PKCS12 (PFX) Format Certificate Generation with ADCS

If we want to create a certificate request using Java keytool or OpenSSL and get it signed by our enterprise CA managed by ADCS, we can use the "Submit a Certificate Request or Renewal Request" option from ADCS. For example, we can create a certificate request for the SAS Studio server machine using the keytool command:

```
keytool -certreq -alias studio -keystore /opt/certs/studioStore.jks -file /opt/certs/studio.csr
```

Then from the Linux command line, use the Linux cat command on the studio.csr file and copy the block into ADCS certificate generation area and submit it. Figure 2 below shows the sample screen shot.

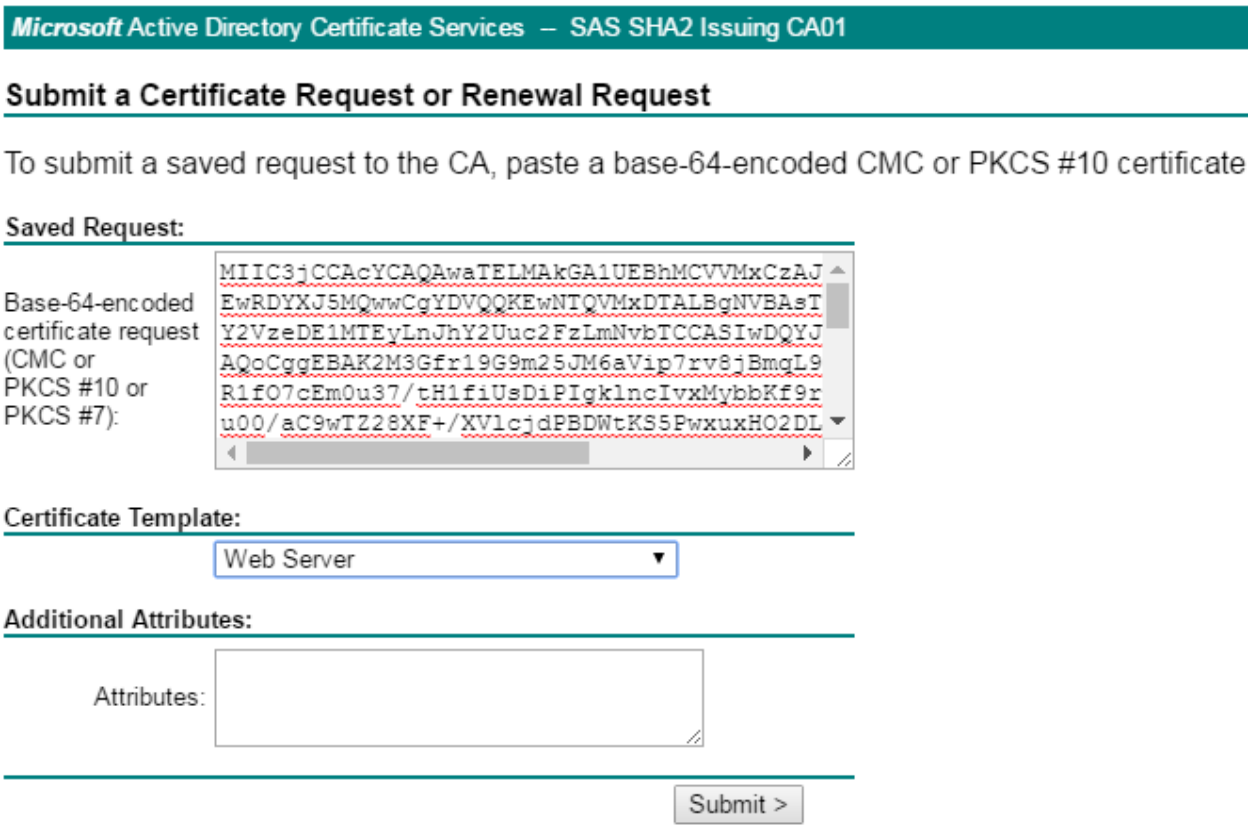

Figure 2. PKCS7 Format Certificate Generation with ADCS

Once we have certificates created, we need to maintain them in proper format based on the nature of the deployment environment. In SAS Studio or in other software that is based on Java, certificates are placed in the keystore (server certificates) and truststore (CA certificate) using the Java keytool command. In other deployment environments such as CAS Server Monitor or SAS Foundation, PEM (Private Enhance Mail) format certificates are used for server certificates and CA certificates. There are two major certificate formats: DER (Distinguished Encoding Rule) and PEM (Private Enhance Mail). We can designate the format when we create a certificate. Certificates can be converted to either format relatively easily using OpenSSL.

IDENTIFYING TLS CLIENT AND SERVER COMPONENTS IN SAS CONFIGURATION

Before we start TLS configuration for SAS components [3], we need to review the major entry points to the configuration and assess the risks involved. Clearly, all entry points that are exposed to external access should be protected first. The protection of SAS internal client and server communication should be assessed based on your organization's security policy. In some cases, it should be protected, in others, it could be optional.

The new major component of SAS® Viya architecture is the SAS® Cloud Analytic Services (CAS) Server. SAS Viya products include other SAS® 9.4 components such as SAS® Studio and SAS/CONNECT® along with microservice based implementation of SAS® Visual Analytics. SAS Viya architecture and implementation allows seamless integration with SAS 9.4 based components. But first, we need to understand the overall picture and identify TLS client and server side for CAS Server access.

All resources in the CAS Server are accessible from CAS Server Monitor. The CAS Server Monitor is a version of web server that handles all HTTP and REST API based CAS action requests.

The following diagram depicts the TLS configuration for CAS Server Monitor (CSM). A browser is the TLS client side and the CAS Server Monitor functions as the TLS server side. For server side TLS, a server identity certificate and a private key and its password need to be defined for the CAS Server Monitor (or web server). The CAS Server configuration file (casconfig.lua) contains environment variable definitions that feed those parameters to the web server.

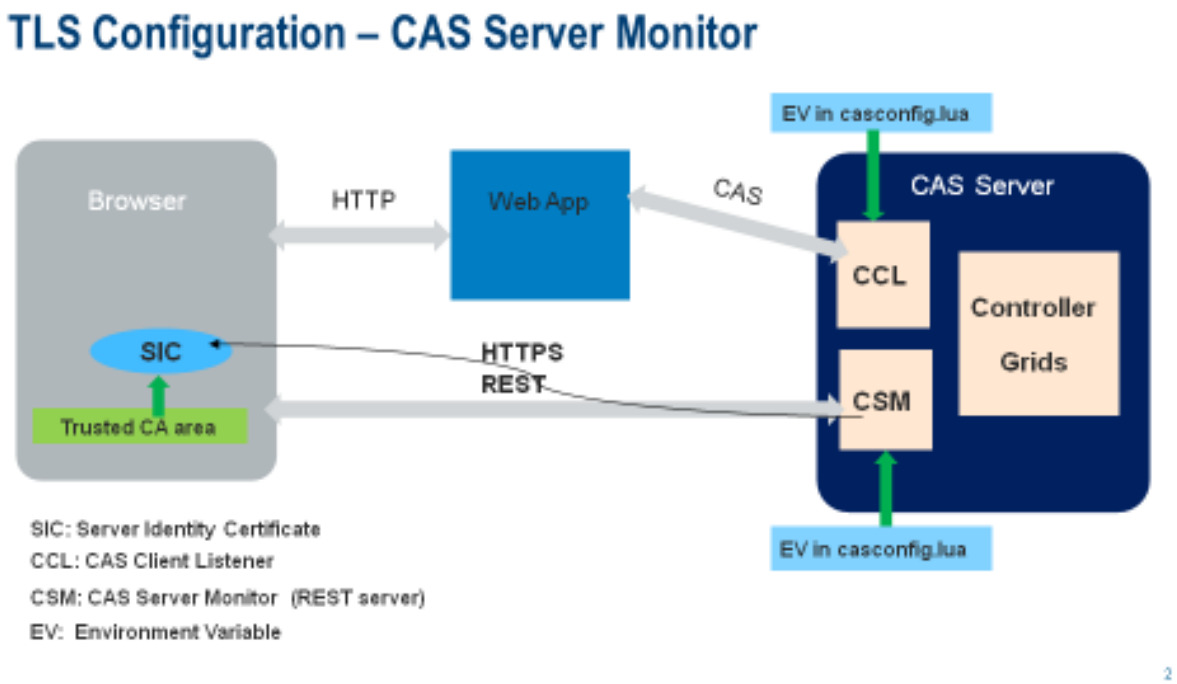

Figure 3. TLS Client and Server Component for CAS Server Monitor

Figure 3 shows the TLS configuration for the CAS Server Monitor. For the web server, a server identity certificate and its private key are specified in PEM format. In this paper, we will create server identity certificate using Java keystore first (as described in previous section), which can be used directly for SAS Studio TLS configuration. In my opinion, it is easier to convert a certificate created by keystore to PEM format than vice versa. Once we create a server identity certificate using Java keytool and keystore, it needs to be converted to PEM format. For example, for CAS Server Monitor TLS configuration, we need to convert the certificate to PEM format. The trick here is that we treat PKCS12 (.p12 or .pfx) format certificate as a form of keystore. Refer to the following example to create a PKCS12 format certificate from keystore and separate out the private key from the PKCS12 certificate:

```
keytool -importkeystore -srckeystore studioStore.jks -destkeystore
questvm07.p12 deststoretype PKCS12 -srcalias studio
```

Then export the certificate (certificate only) into PEM format:

```
openssl pkcs12 -in questvm07.p12 -nokeys -out questvm07.pem
```

Also export unencrypted private key in PEM format from the same PKCS12 certificate:

```
openssl pkcs12 -in questvm07.p12 -nodes -nocerts -out questvm07_key.pem
```

Now the server identity certificate and its private key can be used in the CAS Server Monitor configuration file. Refer to the Encryption section of SAS Viya 3.1 Administration [4] for detail description of the process. The environment variables that are set are as follows:

```
env.CAS_USE_HTTPS='1'
env.CAS_CERTLOC='<cert_path>/questvm07.pem'
env.CAS_PVTKEYLOC='<cert_path>/questvm07_key.pem'
```

```
env.CAS_PVTKEYPASS=''
env.CAS_SSLREQCERT="NEVER"
```

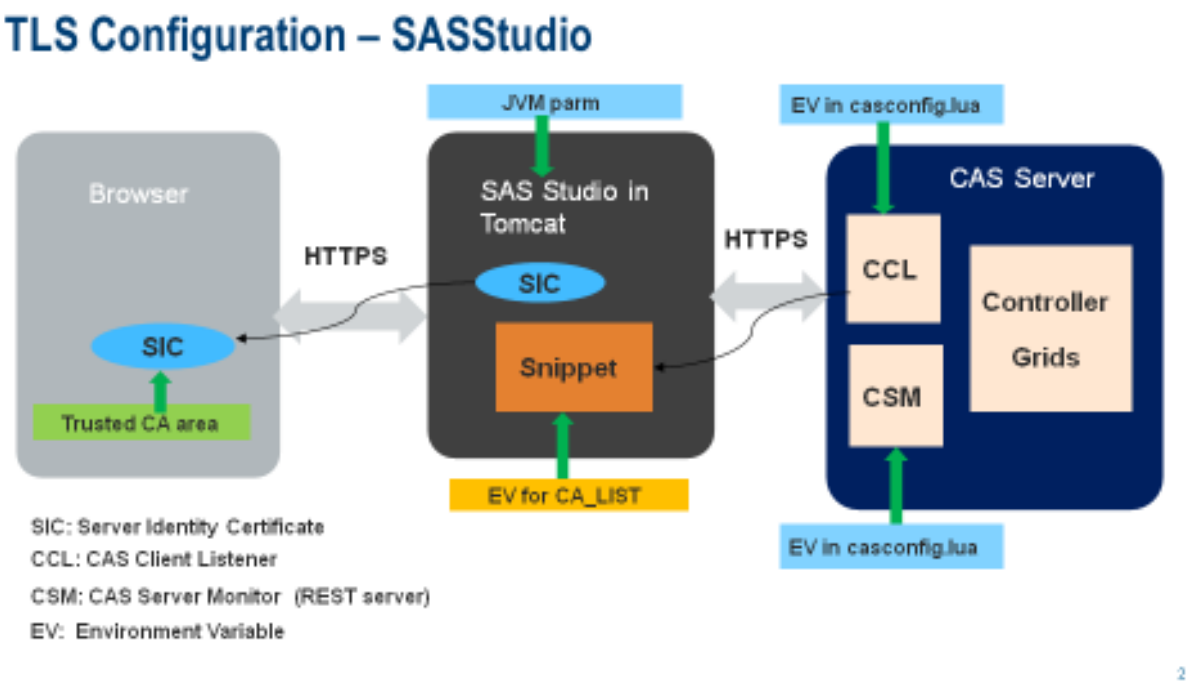

Figure 4. TLS Client and Server Components for SAS Studio

SAS Studio is a major CAS Server access mechanism from the web interface and the most popular way to run SAS code that accesses the CAS Server. Figure 4 shows the TLS client and server side combination for browser, SAS Studio, and CAS Server.

SAS Studio is a web application deployed in a web application container (ApacheTomcat®). Clearly a browser is the TLS Client side and the Java Virtual machine (JVM) that houses the web application container becomes TLS server side. The server identity certificate for the JVM is kept in the keystore specified through the JVM parameter. The following properties are specified in the init_usermods.properties file located in /opt/sas/viya/config/etc/sasstudio/default folder.

These properties are converted to JVM parameters for the web application server.

Based on the keystore (studioStore.jks) created in the previous section, the SAS Studio configuration file can be updated as follows:

```
sasstudio.appserver.https.keyalias=studio
sasstudio.appserver.https.keystorepass=<password>
sasstudio.appserver.https.keystorefile=<studioStore.jks>
```

Optionally, communication can be configured for TLS between SAS Studio snippet (component representing client side) that submits a CAS action request and the CAS Client Listener (CCL – server side) on the CAS Server. Because the optional configuration is communication between internal components, the TLS configuration is not required. If your organization's security policy requires this configuration, it can be configured. When SAS Studio snippet becomes a TLS client, the list of CA

certificates is defined through Linux environment variables in the sasenv_local file located in /opt/sas/viya/home/SASFoundation/bin.

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

Meanwhile, the server identity certificate, private key, and optionally private key password for CAS Client Lister (CCL) are defined using the following environment variables in the CAS server configuration file (casconfig.lua).

```
env.CAS_CLIENT_SSL_CA_LIST="/</SASSecurityCertificateFramework/cacerts/trus
tedcerts.pem">
env.CAS_CLIENT_SSL_CERT=<"/SASSecurityCertificateFramework/tls/certs/questv
m07.pem">
env.CAS_CLIENT_SSL_KEY=<"/SASSecurityCertificateFramework/private/questvm07
_key.pem">
env.CAS_CLIENT_SSL_REQUIRED=true
env.CAS_CLIENT_SSL_KEYPW=""
```

Note that CAS_CLIENT_SSL_* options refer to TLS “server” side parameters for the CAS Client Listener component.

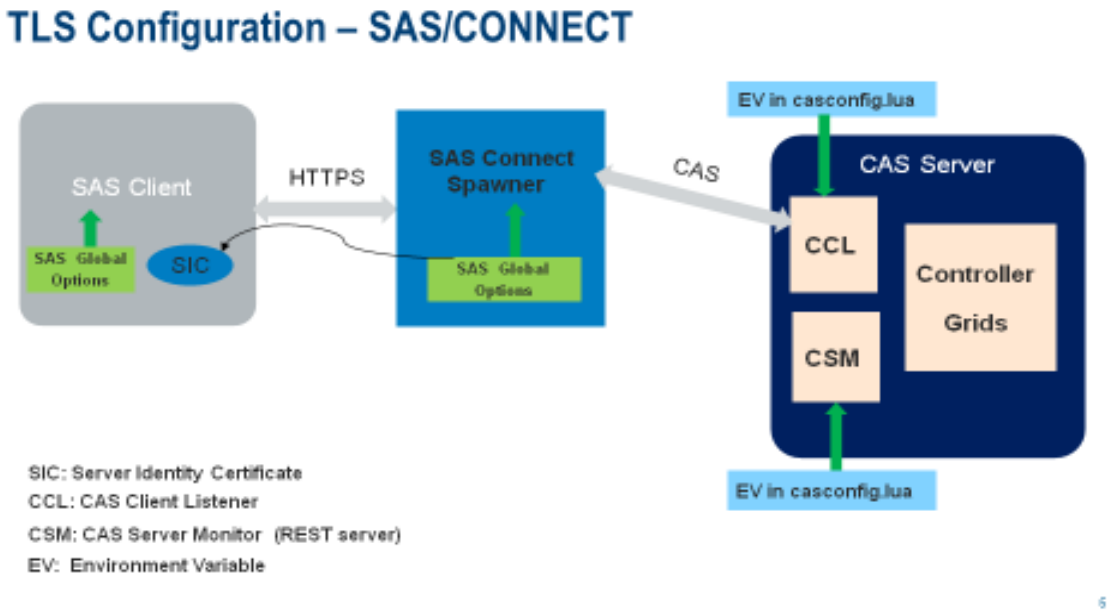

Figure 5. TLS Client and Server Components for SAS/CONNECT

The CAS server can be accessed from the SAS/CONNECT spawner. A SAS client session can connect to the SAS/CONNECT spawner through a TLS connection. In this scenario (Figure 5), the SAS/CONNECT spawner is TLS server side and the SAS client session is the TLS client side. For the SAS/CONNECT spawner, a server identity certificate, private key, and CA certificates are defined through SAS global options. These options are defined in the /opt/sas/viya/config/etc/connect/default/connect_usermods.sh file as shown below. PEM format certificates are used in the SAS environment.

```
USERMODS="-netencrypt -netencryptalgorithm ssl -sslcertloc
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs/questvm0
7_int.pem -sslpvtkeyloc
/config/etc/SASSecurityCertificateFramework/private/questvm07_key.pem -
sslpvtkeypass password "
```

Notice that the file specified by the SSLCERTLOC= option (questvm07_int.pem) needs to be a certificate chain file that starts with the server identity certificate and includes the signing intermediate CA certificates. It can be easily created by concatenating the server identity certificate with Intermediate CA certificates as follows (intermediate CA certificate is sasca01.pem in this example):

```
cd /opt/sas/viya/config/etc/SASSecurityCertificateFramework/tls/certs
cp questvm07.pem questvm07_int.pem
cat sasca01.pem >>  questvm07_int.pem
```

If the SAS/CONNECT spawner runs in TLS mode (as configured above), the SAS/CONNECT client needs to locate the root CA certificate to validate the SAS/CONNECT spawner's server identity 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, we need to specify the location of the root CA certificate by using system option SSLCALISTLOC=.

TLS PROBLEM DEBEGGING TIPS

Even though we are attentive to the TLS configuration process, from time to time, the TLS configuration might not work properly on the first attempt. Turning trace options on in SAS components is one way to debug the problem. However, in many cases, the problem stems from the certificates themselves. We can then use the OpenSSL s_server and the OpenSSL s_client command line tool to verify that the server identity certificates that we generated are working correctly. We can bring up a simple version of the web server with our server identity certificate using the following one line command (assuming that certificate and private key are available in the current directory):

```
openssl s_server -key questvm07_key.pem -cert questvm07.pem -accept 25624
-www
```

Because this brings up the web server, we can point to it from the browser and see whether the TLS handshake works using the server identity server we supplied as follows:

https://questvm07.unx.sas.com:25624

If the certificate arrangements are correct, it will display TLS handshake information such as cipher suites available on both sides, the TLS protocol version, synchronous encryption algorithm (cipher) selected, and much more.

We can also use the OpenSSL s_client command to point to our SAS TLS server component and see whether the TLS handshake works as intended. If it works, the TLS handshake sequence and all certificate information is displayed. If it fails, error information that shows the nature of the problem is displayed. Here is one example of the s_client command:

```
openssl s_client -connect questvm07.unx.sas.com:8777 -CAfile
/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/trustedcer
ts.pem -debug
```

This tool is very powerful. We showed only one example of many to debug any certificate or TLS handshake issue. It is worthwhile to keep these debug tools handy and use them if needed.

CONCLUSION

In this paper, we discussed the basic building blocks of the TLS configuration such as Public Key Cryptography (PKC), RSA algorithm, X.509 certificate, certificate management, and the TLS handshake process. Once we understand these fundamentals and properly identify TLS client and server side of the SAS configuration, the TLS configuration process becomes relatively easy. Based on the implementation of each component, for example, Java code or C code, configuration parameters might take different formats but the basic logic stays the same. It is also essential to learn and become familiar with certificate management tools like OpenSSL, keytool, and ADCS (Active Directory Certificate Service).

REFERENCES

[1] Rivest, R.; A. Shamir; L. Adleman (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Communications of the ACM*, 21 (2): pp.120–126.

[2] Heesun Park and Stan Redford. 2007. "Client Certificate and IP Address-based Multi-Factor Authentication for J2EE Web Applications." *Proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research* (CASCON). New York, NY. Available at http://dl.acm.org/citation.cfm?id=1321229

[3] Heesun Park. 2015. "SSL Configuration Best Practices for SAS® Visual Analytics 7.1 Web Applications and SAS® LASR™ Authorization Service." *Proceedings of SAS Global Forum 2015.* Available at http://support.sas.com/resources/papers/proceedings15/SAS1541-2015.pdf

[4] SAS Institute Inc. 2016. *SAS® Viya™ 3.1 Administration*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/onlinedoc/viya/index.html.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Heesun Park
SAS Institute Inc.
Email: Heesun.Park@sas.com